IN THE UNITED STATES DISTRICT COURT
FOR THE WESTERN DISTRICT OF TEXAS
MIDLAND/ODESSA DIVISION

NEURAL AI, LLC, Plaintiff, v. NVIDIA CORPORATION, Defendant.	Case No. 7:24-cv-00221-ADA-DTG **JURY TRIAL DEMANDED** **HIGHLY CONFIDENTIAL – ATTORNEYS' EYES ONLY**

DEFENDANT NVIDIA CORPORATION'S PRELIMINARY INVALIDITY CONTENTIONS

TABLE OF CONTENTS

Page

I. INTRODUCTION 1
- A. Priority Date 2
- B. Claim Construction 3
- C. Ongoing Discovery and Supplementation 4
- D. Prior Art Identification and Citations Thereto 6
- E. No Patentable Weight 8

II. STATE OF THE ART PRIOR TO THE ASSERTED PATENTS 8
- A. GPUs for General Purpose Computing 8
  - 1. History of Graphics Processing Units 9
  - 2. Development of GPUs for General Purpose Computing 11
- B. Known Features of GPUs Prior to the Asserted Patents 14
  - 1. Prior Art Parallel Computing and "Controllers" 14
  - 2. The Stream Programming Model of GPUs 18
  - 3. APIs and Programming GPUs 19
  - 4. Known Memory Management Techniques 22
    - a. Using Pointers for Memory Management 22
    - b. Using Outputs as Inputs and Pointer Swapping 23
    - c. Transferring Data Between System Memories 26
  - 5. Artificial Neural Networks and Other Applications of GPGPU 27
  - 6. Real-Time Data Input 30
  - 7. Task and Resource Management on a GPU 31
- C. Merrimac and Brook 33
- D. NVIDIA Technology Pre-Dates Neural AI and the Alleged Inventions 35
  - 1. The NV1 Chip and Edge 3D Board 36
  - 2. GeForce 256 Graphics Processing Unit 38
  - 3. Kelvin through Curie Architectures 39
  - 4. Tesla Architecture 42
  - 5. Fermi Architecture 45
  - 6. NVIDIA CUDA [NVIDIA Confidential] 46

III. INVALIDITY UNDER 35 U.S.C. §§ 102 AND 103 48
A. Legal Standards 48
1. Anticipation 48
2. Public Use/Sale Bar 48
3. Obviousness 49
B. Prior Art Invalidity of U.S. Patent No. 8,648,867 51
1. Identification of Prior Art 51
a. Patents and Patent Publications 51
b. Printed Publications 52
c. Prior Art Products and Systems 55
d. Prior Invention Under § 102(g) 55
2. Anticipation 57
3. Obviousness 58
a. Level of Ordinary Skill in the Art 59
a. Scope and Content of the Prior Art 59
b. Motivations to Combine Prior Art 62
c. Secondary Considerations of Non-Obviousness 78
4. Priority Date 78
a. Provisional Application 79
b. Asserted Conception, Reduction to Practice, and Diligence 79
C. Prior Art Invalidity of U.S. Patent No. RE48,438 80
1. Identification of Prior Art 80
a. Patents and Patent Publications 80
b. Printed Publications 81
c. Prior Art Products and Systems 84
d. Prior Invention Under § 102(g) 84
2. Anticipation 86
3. Obviousness 87
a. Level of Ordinary Skill in the Art 88
b. Scope and Content of the Prior Art 88
c. Motivations to Combine Prior Art 91
d. Secondary Considerations of Non-Obviousness 118

4. Priority Date 119
a. Provisional Application 119
b. Asserted Conception, Reduction to Practice, and Diligence 119
D. Prior Art Invalidity of U.S. Patent No. RE49,461 120
1. Identification of Prior Art 120
a. Patents and Patent Publications 121
b. Printed Publications 121
c. Prior Art Products and Systems 124
d. Prior Invention Under § 102(g) 124
2. Anticipation 126
3. Obviousness 127
a. Level of Ordinary Skill in the Art 128
b. Scope and Content of the Prior Art 128
c. Motivations to Combine Prior Art 131
d. Secondary Considerations of Non-Obviousness 149
4. Priority Date 149
a. Provisional Application 150
b. Asserted Conception, Reduction to Practice, and Diligence 150
IV. INVALIDITY UNDER 35 U.S.C. § 112 151
A. Legal Standards 152
1. Invalidity Under Pre-AIA 35 U.S.C. § 112 ¶ 1 (Written Description / Non-Enablement) 152
2. Invalidity Under Pre-AIA 35 U.S.C. § 112 ¶ 2 (Indefiniteness) 152
3. Invalidity for Improper Recapture 153
4. Invalidity Under the Original Patent Doctrine 153
B. Section 112 Invalidity of U.S. Patent No. 8,648,867 154
1. Lack of Written Description / Enablement 154
2. Indefiniteness 154
C. Section 112 Invalidity of U.S. Patent No. RE48,438 156
1. Lack of Written Description / Enablement 156
2. Indefiniteness 158
3. Improper Recapture 160
4. Invalidity Under the Original Patent Rule 161

D. Section 112 Invalidity of U.S. Patent No. RE49,461 161
1. Lack of Written Description / Enablement 161
2. Indefiniteness 162
3. Improper Recapture 163
4. Invalidity Under the Original Patent Rule 164
V. INEQUITABLE CONDUCT 164
A. Legal Standards 165
1. Inequitable Conduct 165
a. Materiality 165
b. Intent 166
c. Infectious Unenforceability 168
2. Reviving Abandoned Patent Applications 169
B. Inequitable Conduct Based on Withheld References 170
C. Inequitable Conduct Based on a False Declaration Regarding Abandonment 172
VI. ACCOMPANYING DOCUMENT PRODUCTION 174

Pursuant to the Court's Scheduling Order dated January 27, 2025, and the Standing Order Governing Proceedings 4.4 for Patent Cases, Defendant NVIDIA Corporation ("Defendant" or "NVIDIA") hereby provides its Preliminary Invalidity Contentions ("Invalidity Contentions") concerning the invalidity of U.S. Patent Nos. 8,648,867 ("the '867 Patent"), RE48,438 ("the '438 Patent"), and RE49,461 ("the '461 Patent") (collectively, the "Asserted Patents").[1]

I. INTRODUCTION

On January 14, 2025, Plaintiff Neural AI LLC ("Plaintiff" or "Neural AI") served its Disclosure of Asserted Claims and Preliminary Infringement Contentions ("Infringement Contentions") and asserted the following claims (collectively, the "Asserted Claims"):

Asserted Patent	Asserted Claims
'867 Patent	16–19
'438 Patent	1–10, 12–14, 16–18, 20–32, 40–53[2]
'461 Patent	21–31

NVIDIA contends that each of the Asserted Claims is invalid under at least 35 U.S.C. §§ 102, 103, and/or 112. The prior art and invalidity arguments identified in these Invalidity Contentions is identified with respect to the currently asserted claims only. NVIDIA reserves the right to supplement these contentions and rely on additional references with respect to claims later identified by Plaintiff. NVIDIA further reserves the right to modify, amend, or supplement these

[1] NVIDIA also incorporates by reference Patent Trial and Appeal Board Case Nos. IPR2025-00606 and -00608 (already filed) and -00609 and -00610 (yet to be filed), including the petitions, supporting declarations, and exhibits, into these Invalidity Contentions, along with any other IPR petitions or *ex parte* reexamination requests and supporting materials filed with respect to any of the Asserted Patents.

[2] In the cover document to its Infringement Contentions, Plaintiff purports to assert claim 54 of the '438 Patent, but none of the charts accompanying the Infringement Contentions include claim 54. NVIDIA thus understands that Plaintiff is not asserting claim 54. To the extent Plaintiff is permitted to amend its contentions to assert claim 54 of the '438 Patent, NVIDIA will supplement these Invalidity Contentions to identify prior art to claim 54.

Invalidity Contentions to the extent the Court permits Plaintiff to amend, supplement, or otherwise modify its Infringement Contentions, including by modifying the claims asserted, and to the full extent permitted by the Local Patent Rules or the Standing Order Governing Proceedings 4.4 for Patent Cases.

A. Priority Date

In its Infringement Contentions, Plaintiff contends that the Asserted Claims of all Asserted Patents "are entitled to a priority date corresponding to the conception of the claimed inventions, which occurred no later than February 25, 2005," and that "[t]he conception of the inventions . . . was followed by continuous diligence and actual reduction to practice of the claimed inventions no later than December 14, 2005." The Invalidity Contentions are based on the alleged priority date identified by Plaintiff in its Infringement Contentions.

Nothing in these Invalidity Contentions shall be understood as an agreement that any Asserted Patent is entitled to claim priority to the dates identified by Plaintiff, or to any provisional priority document. NVIDIA does not agree that any of the evidence produced with Plaintiff's Infringement Contentions or shortly after service of Plaintiff's Infringement Contentions demonstrates conception by February 25, 2005, and/or actual reduction to practice by December 14, 2005. NVIDIA reserves the right to amend these Invalidity Contentions and identify additional prior art references if Plaintiff is permitted by the Court to later assert an even earlier priority date or to later produce documents purporting to further evidence conception and reduction to practice.

Any reference to an "asserted priority date" in these Invalidity Contentions refers to the priority date identified in Plaintiff's Infringement Contentions. Reference to a "priority date" or an "asserted priority date" should not be construed to mean that NVIDIA agrees that any of the Asserted Patents are in fact entitled to such priority date, or that Plaintiff has provided proper notice as to its contentions for a priority date. To the extent Plaintiff alleges that any prior art

reference or system relied on in these Invalidity Contentions does not actually qualify as prior art to an Asserted Patent, NVIDIA reserves the right to rebut those allegations (*e.g.*, by demonstrating an earlier date for the challenged prior art and/or a later priority date for a particular Asserted Patent and/or Asserted Claim).[3] Likewise, to the extent Plaintiff successfully establishes an invention date before any of the prior art references relied on by NVIDIA, then those references serve as evidence of secondary considerations of obviousness, particularly, contemporaneous invention by others.

B. Claim Construction

The Invalidity Contentions are based on (1) NVIDIA's present understanding of the Asserted Claims, and (2) the claim constructions Plaintiff appears to be using based on the Infringement Contentions, all without regard to whether NVIDIA agrees with Plaintiff's apparent or expressed claim constructions. NVIDIA reserves the right to supplement or otherwise amend these Invalidity Contentions in response to any court-ordered clarifications on claim constructions, any report from any expert witness for Plaintiff regarding the scope of the claims, any briefing filed by Plaintiff relating to the scope of the claims, and any position taken by Plaintiff concerning claim scope, infringement, or invalidity.

NVIDIA takes no position on any matter of claim construction in these Invalidity Contentions. If NVIDIA's disclosures herein are consistent with any explicit, apparent, or implied claim constructions in Plaintiff's Infringement Contentions, no inference is intended and no inference should be drawn that NVIDIA agrees with any of Plaintiff's claim constructions. Any statement herein describing or tending to describe any claim element is provided solely for the

[3] NVIDIA reserves the right to rely on additional documents and evidence to rebut any efforts by Plaintiff to allege any reference was not publicly available or otherwise available as prior art.

purpose of understanding and/or applying the cited prior art. In addition, to the extent that these Invalidity Contentions rely on or otherwise embody a particular order in which the steps of method claims are performed, NVIDIA does not necessarily propose that the method claims must be limited to such order, although NVIDIA may later propose such an order.

Nothing herein should be read to suggest that NVIDIA agrees that any particular claim term is sufficiently definite, enabled or supported by the written description to meet the requirements of 35 U.S.C. § 112. Likewise, nothing herein should be read to suggest that NVIDIA agrees that the preamble of any claim is or is not limiting.

Prior art not included in these Invalidity Contentions, whether or not now known to NVIDIA, might become relevant depending on the claim constructions proposed by the parties and/or the Court's claim construction rulings. NVIDIA reserves all rights to supplement or modify the positions and information in these Invalidity Contentions, including, without limitation, the prior art and grounds of invalidity set forth herein, after the Court has construed any term of the Asserted Claims.

C. Ongoing Discovery and Supplementation

NVIDIA bases these Invalidity Contentions on its current knowledge and understanding of the Asserted Patents, Plaintiff's Infringement Contentions, and other facts and information available as of the date of these contentions. Plaintiff's Infringement Contentions are deficient and fail to provide sufficient allegations as to how the accused products infringe each and every claim limitation; rather, they simply parrot the claim language and then cite various documents—and in many circumstances, point to no evidence at all. Plaintiff therefore has prejudiced NVIDIA's ability to understand, for purposes of preparing these Invalidity Contentions, what Plaintiff alleges to be the scope of the Asserted Claims. Contemporaneously with the Invalidity Contentions, NVIDIA is producing technical documents sufficient to show operation of the

accused products. NVIDIA expects that and Plaintiff is required to "seasonably amend" its Infringement Contentions—pursuant to the Court's Scheduling Order and the Standing Order Governing Proceedings 4.4 for Patent Cases—well in advance of the deadline for final infringement contentions, including to provide clearer theories of infringement, in view of NVIDIA's production. NVIDIA's Invalidity Contentions generally are set forth with equal or greater specificity, and in no instance with less specificity, than Plaintiff's Infringement Contentions. NVIDIA reserves the right to amend these Invalidity Contentions if and when Plaintiff provides further infringement contentions as to each Asserted Claim.

NVIDIA's investigation into prior art—including prior art identified in these disclosures, third-party prior art (including system art and related evidence), documents, and knowledgeable witnesses—is ongoing. Furthermore, NVIDIA anticipates seeking and obtaining discovery, including third-party discovery, that further evidences and supports the invalidity of the Asserted Claims. Thus, NVIDIA expects to present additional information and evidence in support of these Invalidity Contentions, and to revise, amend, and/or supplement these Invalidity Contentions accordingly, including through its Final Invalidity Contentions, in a manner consistent with the Federal Rules of Civil Procedure and the Court's rules and applicable orders.

NVIDIA further reserves the right to rely on statements or admissions from those owing a duty of candor, such as named inventors, prosecution counsel, and others involved in the prosecution of the patent applications or related applications, concerning the scope of the prior art relevant to the Asserted Patents found in, *inter alia*, the following: the respective prosecution histories of the Asserted Patents and related patent applications; any deposition testimony of the named inventors; and the papers filed and any evidence submitted by Plaintiff in connection with this litigation or any other litigation or invalidity proceeding involving the Asserted Patents.

NVIDIA reserves the right to modify and/or supplement these Invalidity Contentions with an explanation of why the Asserted Claims are invalid under 35 U.S.C. § 102(f)–(g) in the event NVIDIA obtains additional evidence that the named inventors did not invent (either alone or in conjunction with others) the subject matter claimed or that the claimed subject matter was invented by others.

Prior art not identified in this disclosure, whether known or not known to NVIDIA, may become relevant, including prior art concerning the state of the art at the time of invention, as well as simultaneous or near-simultaneous independent invention by others. For instance, NVIDIA is currently unaware of the extent, if any, to which Plaintiff will contend that limitations of the Asserted Claims are not disclosed in the prior art identified by NVIDIA. To the extent that such an issue arises, NVIDIA reserves the right to identify other references that would disclose, practice, or render obvious the allegedly missing limitation(s) of the disclosed subject matter.

D. Prior Art Identification and Citations Thereto

In these Invalidity Contentions, NVIDIA identifies specific, exemplary portions of prior art references that disclose the elements of the Asserted Claims. While NVIDIA has identified exemplary disclosures for each element, they do not necessarily identify every disclosure of the same element in each prior art reference. A person of ordinary skill in the art would read a prior art reference as a whole and in the context of other publications, literature, and general knowledge in the field and would rely upon other information including other publications and general scientific or engineering knowledge. NVIDIA therefore reserves the right to rely upon other unidentified portions of the prior art references and on other publications and prior art products and expert testimony to provide context and to aid in the understanding and interpretation of the identified portions of the prior art.

NVIDIA also reserves the right to rely upon (1) other portions of the cited prior art references, other publications, prior art products, and the testimony of experts to establish that the alleged inventions would have been obvious to a person of ordinary skill in the art, including on the basis of modifying or combining certain cited references; (2) admissions relating to prior art in the Asserted Patents or related patents, the prosecution histories of the Asserted Patents or related patents; (3) counterparts of any U.S. or foreign patents identified in these Invalidity Contentions; and (4) any prior art references, other publications, prior art products, and the testimony of experts to establish that the alleged inventions would have been obvious to a person of ordinary skill in the art used during any proceeding at the Patent Office involving the Asserted Patents. For prior art identified as system art or as prior invention, NVIDIA reserves the right to rely on embodiments of those systems (whether in physical, digital, or other form) to demonstrate its invalidity arguments. Where a prior art reference includes citations to other references, those other references are considered incorporated by reference for context.

The references discussed in the claim charts may disclose the elements of the Asserted Claims explicitly and/or inherently, and/or they may be relied upon to show the state of the art in the relevant time frame. The suggested obviousness combinations are provided in the alternative to NVIDIA's anticipation contentions and are not to be construed to suggest that any reference included in the combinations is not by itself anticipatory.

Where NVIDIA identifies in the claim charts a particular figure in a prior art reference, the identification should be understood to encompass the caption and description of the figure as well as any text relating to the figure in the specification and prosecution history in addition to the figure itself. Similarly, where an identified portion of text refers to a figure or other material, the identification should be understood to include the referenced figure or other material as well.

NVIDIA's identification of a prior art document or system as a "primary" (or "base") reference, or a "secondary" (or "supplemental") reference, is for purposes of convenience only, and nothing prevents NVIDIA from relying on any reference disclosed in these Invalidity Contentions in any way in a combination of prior art references. *Schwendimann v. Neenah, Inc.*, 82 F.4th 1371, 1384 (Fed. Cir. 2023) ("[C]haracterizing references "as 'primary' and 'secondary' is merely a matter of presentation with no legal significance.").

E. No Patentable Weight

NVIDIA reserves the right to argue that various portions of the Asserted Claims, such as an intended use or result, nonfunctional descriptive material, and certain preamble language, are entitled to no patentable weight. Mapping of a portion of an Asserted Claim to a prior art reference does not represent that such portion of the claim is entitled to patentable weight when comparing the claimed subject matter to the prior art.

II. STATE OF THE ART PRIOR TO THE ASSERTED PATENTS

Prior to the Asserted Patents, the concepts claimed in the patents were well known and developed in the art, and features accused of infringing had already been developed by NVIDIA. This includes performing numerical simulations and computations on GPUs, such as simulating artificial neural networks. The sections below describe the state of the art at the time of the Asserted Patents and NVIDIA's development of relevant technologies, which confirm the invalidity (including obviousness) of the Asserted Claims. This state of the art further demonstrates motivations to combine or modify prior art, and NVIDIA reserves the right to rely on these facts with respect to obviousness combinations and motivations to combine.

A. GPUs for General Purpose Computing

As discussed in further detail below, in the late 1990s, NVIDIA developed the first consumer-level graphics card that could offload the entire graphics pipeline from the CPU,

dubbing it a "Graphics Processing Unit," or "GPU." Led in part by NVIDIA, GPUs were then used for other parallel processing tasks, including for general-purpose computing.

1. History of Graphics Processing Units

A GPU is an integrated circuit with computer hardware designed for graphics. GPUs were developed in the 1990s by NVIDIA to accelerate graphics rendering. The Asserted Patents recognize that "[v]arious brand[s] of GPU [we]re relevant" at the time, including "GPU's based on the GeForce series from NVIDIA Corporation."[4] NVIDIA released the world's first programmable GPU, the GeForce 256, in 1999. At the time of release, the GeForce 256 had the equivalent parallel processing performance of a "maximum configuration 256-processor Cray T3D," one of the leading parallel processing systems as the time.[5] NVIDIA developed and released several other GPUs prior to 2005, including the GeForce 2 (2000), GeForce 3 (2001), GeForce 4 (2002), GeForce FX (2003), and GeForce 6 (2004) GPUs.[6]

Unlike CPUs, which were optimized to handle tasks sequentially, GPUs were designed with parallel processing in mind. GPUs were originally intended to leverage their parallel processing architecture to accelerate graphics rendering in desktop computers, particularly real-time rendering of three-dimensional environments. At a high level, traditional GPUs processed "textures" (stored as data arrays) by using "shaders" (programs) to modify pixel data and create graphics for display.[7] Shaders operated in a highly parallel manner, executing mathematical

---

[4] *E.g.*, '867 Patent at 4:40–43.

[5] NVIDIA-NAI-0003861 (GeForce256 Press Release).

[6] *E.g.*, NVIDIA-NAI-0002604 (GPU Gems 2: Programming Techniques for High-Performance Graphics and General-Purpose Computation (2005)) at 471–91 ("GeForce 6 Series GPUs provide the GPU programmer with unparalleled flexibility and performance . . . ."); *see also* NVIDIA-NAI-0005887 (Cg Tutorial) at 11–14 (describing the "generations" of GPUs).

[7] NVIDIA-NAI-0002604 (GPU Gems 2: Programming Techniques for High-Performance Graphics and General-Purpose Computation (2005)) at 500–01.

operations on individual texture elements (sometimes called texels or pixels) to perform tasks like shading, filtering, lighting computations, and blending. These operations were ideal for parallelism because the computations for each element were typically independent of others, allowing GPUs to process thousands of elements simultaneously.

GPUs are fundamentally different processors from CPUs. CPUs are intended as general purpose processors that act as the "brain" of a computer system. In contrast, at the time of the alleged invention of the Asserted Patents, GPUs were designed to accelerate highly parallel processes, which included both graphics applications and general computing applications. The following figure shows an example of the type of parallel processing operations that occur in the "graphics hardware pipeline" of GPUs:

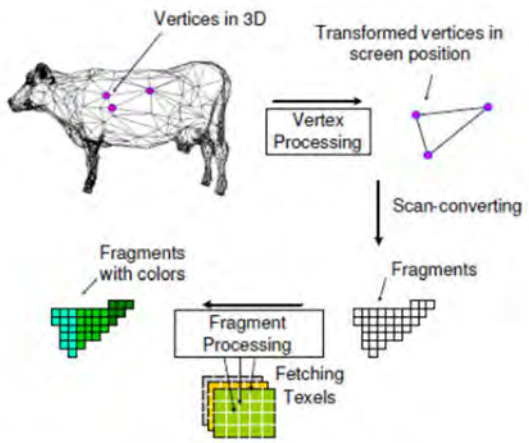


Figure 1: A simplified illustration of the graphics hardware pipeline.

GPUs are dependent upon, and require a CPU, to operate. No system can include only a GPU. Whereas CPUs are designed for low-latency, sequential code performance for only a few threads, GPUs are designed for rapidly switched, massively data parallel operations for many threads, requiring a different memory and registry architecture. Whereas CPUs employ up to a small number of general-purpose processing cores in, *e.g.*, a "multi-core" CPU, GPUs contains

hundreds or thousands of special purpose cores. These features allowed GPUs to handle any parallelizable tasks, including both graphics and non-graphics computing tasks.

As noted above, when first released in the 1990s, GPUs had a fixed pipeline for processing graphics in specific ways. GPUs quickly evolved to include programmable shaders, which gave developers the ability to write custom programs and enabled them to manipulate graphics at a much more granular level. For example, programmable shaders allowed for more complex visual effects, such as per-pixel lighting, procedural textures, and realistic surface details, revolutionizing graphics rendering in video games, simulations, and visual effects. These innovations also allowed for general-purpose computing on GPUs (GPGPU), discussed in more detail below, enabling GPUs to handle non-graphics tasks like scientific and numerical simulations and machine learning.

2. Development of GPUs for General Purpose Computing[8]

GPUs were used almost immediately to perform parallelized computations for non-graphics applications, such as numerical simulations and artificial neural networks. In the early 2000s, this type of computing was often referred to as "general-purpose computing on GPUs," sometimes referred to as "GPGPU." Because graphics processing is an inherently parallelizable task, GPUs (including those developed by NVIDIA) were designed with parallelism baked into their structure.[9] Additionally, GPUs did not need to support as many different sorts of functions and datatypes as general-purpose CPUs; by omitting the circuity needed to support these features, GPU circuits were able to fit more processing power onto a single chip die than an equivalent CPU.

---

[8] *See generally* NVIDIA-NAI-0000888 (Survey of General-Purpose Computation on Graphics Hardware).

[9] NVIDIA-NAI-0002604 (GPU Gems 2: Programming Techniques for High-Performance Graphics and General-Purpose Computation (2005)) at 457–70.

GPGPU works by using "textures" to store data and repurposing "shaders" to perform computations on the data.[10] The techniques enabled GPUs to handle tasks like scientific computing, numerical simulations, and neural network training. NVIDIA and others led GPGPU research and development. For example, NVIDIA not only improved hardware and software for programmable GPUs, but also actively supported the GPGPU community through educational efforts. NVIDIA published books like GPU Gems 2, which provided practical guidance and techniques for using GPUs in general-purpose applications, helping developers harness the parallel processing power of GPUs for a wide range of fields.[11] One recognized benefit of GPGPU applications was that, with the architecture described above, all calculations occurring in parallel on the GPU, they "free up the CPU for other tasks."[12]

For example, NVIDIA released the GeForce 3 (NV20) based on the Kelvin architecture in 2001 and GeForceFX (NV30) based on the Rankine architecture in 2002. These products were the first to add programmable shader engines to GPUs, providing a vertex shader in 2001, and in 2002, programmable fragment shaders. The incorporation of a GPU-hardware-based programmable shaders within the GeForce 3 (NV20) in 2001 and GeForce FX in 2002 was another NVIDIA pioneering innovation. The programmability was delivered by a "[s]imple to use programming model" that allowed the developer to "manipulate vertex transforms."[13] The field of

---

[10] NVIDIA-NAI-0002604 (GPU Gems 2: Programming Techniques for High-Performance Graphics and General-Purpose Computation (2005)) at 497–502.

[11] NVIDIA-NAI-0002604 (GPU Gems 2: Programming Techniques for High-Performance Graphics and General-Purpose Computation (2005)) at 451 ("General-Purpose Computation on GPUs: A Primer").

[12] NVIDIA-NAI-0002604 (GPU Gems 2: Programming Techniques for High-Performance Graphics and General-Purpose Computation (2005)) at 650.

[13] *See* https://web.archive.org/web/20030103124007/http://developer.nvidia.com/docs/IO/1310/SUPP/page07.jpg.

general purpose GPU computing emerged from the "combination of high floating-point performance and programmability" found within NVIDIA's GPUs.

One example of NVIDIA GPU optimized for GPGPU operations is NVIDIA's NV40 GPUs based on the Curie architecture. In such architectures, the programmable vertex engine and programmable fragment engine allowed the GPU to be "seen as a large amount of programmable floating-point horsepower and memory bandwidth that can be exploited for compute-intensive applications completely unrelated to computer graphics. . . . When used for non-graphics applications, it can be viewed as two programmable blocks that run serially: the vertex processor and the fragment processor, both with support for fp32 operands and intermediate values. . . . In addition, both the vertex and the fragment processor are highly computationally capable."[14]

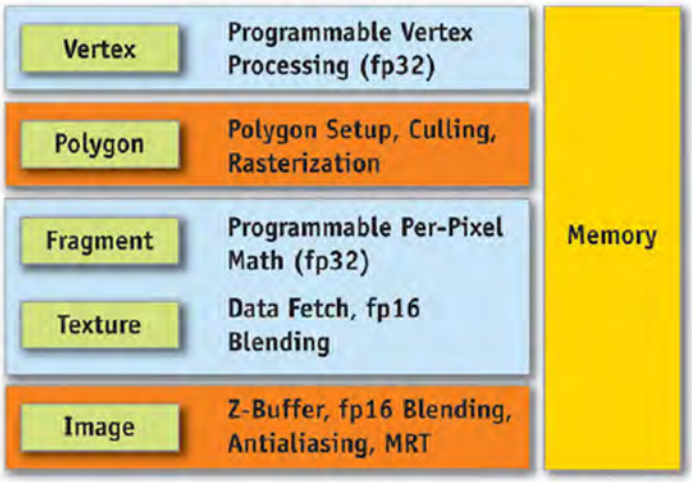


Figure 30-6. The GeForce 6 Series Architecture Viewed as a Graphics Pipeline

[14] NVIDIA-NAI-0002604 (GPU Gems 2: Programming Techniques for High-Performance Graphics and General-Purpose Computation (2005)) at 817–18.

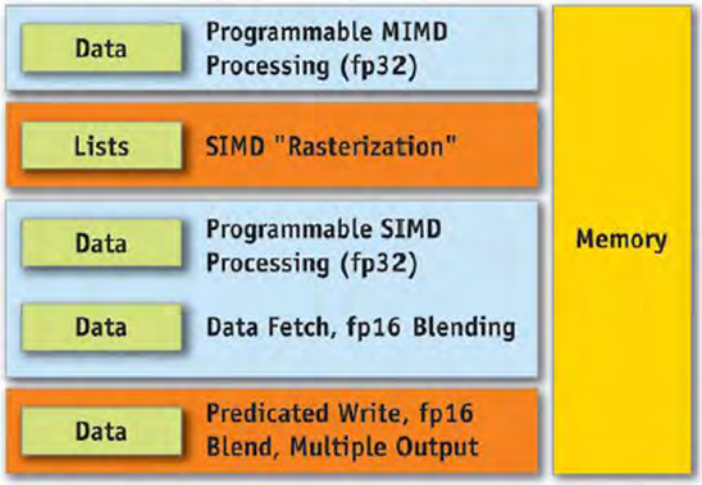


Figure 30-7. The GeForce 6 Series Architecture for Non-Graphics Applications [15]

Further details regarding NVIDIA's GPU architectures for GPGPU operations prior to the Asserted Patents are provided below.

B. Known Features of GPUs Prior to the Asserted Patents

The Asserted Patents describes several features and functions of GPGPU. As explained below, these features and functions were all well known to those skilled in the art prior to the Asserted Patents.

1. Prior Art Parallel Computing and "Controllers"

The principles of parallel processing were known prior to the Asserted Patents, and researchers and engineers in the field had applied these parallelization techniques to GPUs, which are particularly suited to perform parallel computations with their parallel processing engines. Researchers and engineers had also settled on effective designs for parallel processing systems, including a "controller" for managing the parallel computing process.

[15] NVIDIA-NAI-0002604 (GPU Gems 2: Programming Techniques for High-Performance Graphics and General-Purpose Computation (2005)) at Figs. 30-6, 30-7.

Parallel computing involves dividing a computational task among more than one processor. When appropriately configured, a system using many processors can perform challenging calculations more quickly than a single-processor system by instructing these many processors to work on different parts of the problem simultaneously.

Decades before the Asserted Patents, parallel computing was of particular interest to the scientific computing community, who expected that speed gains through parallelization would make new classes of experiments and computations possible.[16] It was found, however, that parallel processes lose efficiency when individual processors sit idle or when they compete for memory resources. Thus, parallel computing system designers proposed architectures to address such bottlenecks that may arise in such systems. A common solution involved using one set of computer hardware (*e.g.*, a controller) to coordinate the function and results of a second set of hardware that performs the core computations (*e.g.*, processors). By having the controller manage the computations of the parallel processors, as opposed to only a CPU, the CPU is free and available to perform other tasks.

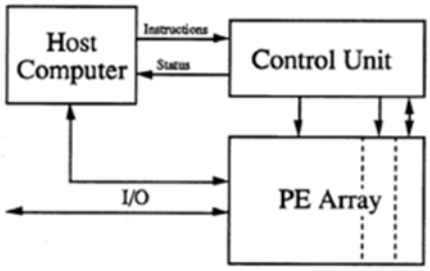
For example, one class of control architecture for parallel computation is Single-Instruction-Multiple-Data ("SIMD"), which has been known in the art since at least the 1970s.[17] There are many examples in the prior art of SIMD architectures (including for neural networks) with a host computer, controller, and parallel processors.[18]

---

[16] NVIDIA-NAI-0002604 (GPU Gems 2: Programming Techniques for High-Performance Graphics and General-Purpose Computation (2005)) at 365–67.

[17] NVIDIA-NAI-0004513 (Computer Organizations) at 1, 13.

[18] NVIDIA-NAI-0005047 (Dataflow Machine Architecture) at 375–89.



Host Computer - Processor Array Relationship

Figure SIMD-2b [19]

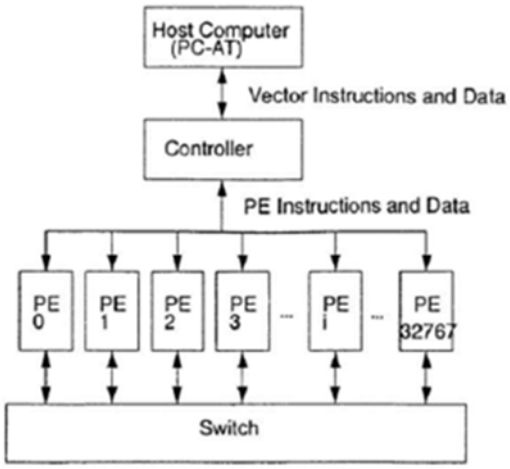


Figure 1: A block diagram of the MM32k. [20]

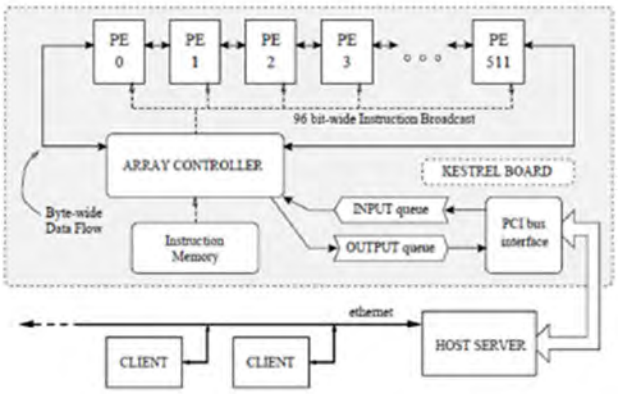


Fig. 3. Kestrel board architecture. [21]

[19] NVIDIA-NAI-0004437 (SIMD Processor Array Architectures) at 1–2.

[20] NVIDIA-NAI-0003517 (Massively-Parallel SIMD Processor) at 845 (SIMD processor for neural network computation).

[21] NVIDIA-NAI-0004674 (High-Performance SIMD) at 2.

Another class of control architecture for parallel processing is the Multiple Instruction Multiple Data ("MIMD") architecture.[22] In such a system, operations are split into units whose outputs do not depend on each other, and which might require different calculations.[23] Like SIMD, prior art MIMD systems used a controller.[24]

Beginning in the late 1990s, those skilled in the art recognized that the existing GPU pipeline could be adapted to known parallel processing paradigms, like SIMD and MIMD. Texture memory, originally designed to store a matrix of pixels with float values at each location, closely resembled the vector structures commonly used in numerical analysis. By reimagining its GPU architecture in the context of established parallel computing models, NVIDIA made its GPUs highly suitable for these frameworks. In the early 2000s, NVIDIA GPUs supported features like "MIMD branching, SIMD branching, and condition codes," all of which were well-understood architectures in the art of parallel computing.[25]

Companies like NVIDIA had also applied the well-known "controller" framework for directing parallel computations. For example, NVIDIA patents disclose improved GPU architectures with a "controller" to direct the distribution and execution of instructions in the processing cores of the GPU.[26]

---

[22] NVIDIA-NAI-0004513 (Computer Organizations) at 957–58.

[23] NVIDIA-NAI-0004513 (Computer Organizations) at 957–58; NVIDIA-NAI-0002604 (GPU Gems 2: Programming Techniques for High-Performance Graphics and General-Purpose Computation (2005)) at 547–48 ("[I]n a MIMD-parallel architecture, different processors may simultaneously execute different instructions.").

[24] NVIDIA-NAI-0004513 (Computer Organizations) at 957–58.

[25] NVIDIA-NAI-0002604 (GPU Gems 2: Programming Techniques for High-Performance Graphics and General-Purpose Computation (2005)) at 547–49.

[26] NVIDIA-NAI-0004975 (Nickolls); NVIDIA-NAI-0004792 (Kirk).

2. The Stream Programming Model of GPUs

Prior to the Asserted Patents, it was known in the art that GPUs operated based on the "stream programming model," where data is represented as a *stream*—an ordered set of elements of the same data type. Streams are processed by *kernels*, which is a program or function that "tak[es] one or more streams as inputs and produc[es] one or more streams as outputs."[27]

A key feature of this stream programming model is that kernels—the computational units in the pipeline—process entire streams of data rather than individual elements. This design ensures that computations on individual stream elements are entirely independent of one another, enabling what seems like a serial calculation to be executed efficiently on data-parallel hardware. Moreover, because the outputs of a kernel depend solely on its inputs, all the data required for execution is known upfront, allowing for highly optimized data storage, access, and management. This architecture aligned perfectly with the existing graphics pipeline in GPUs (including NVIDIA's), which relied on data parallelism and element independence.[28]

In the stream programming model, applications are built by chaining multiple kernels together. By the time of the Asserted Patents, GPUs featured high-performance data-parallel processors capable of implementing two essential kernels in the graphics pipeline: the vertex program and the fragment program. These programs could sustain high computation rates on user-defined tasks with the precision necessary for solving general-purpose computing challenges. In effect, they transformed GPUs into programmable stream processors, making them suitable for far

---

[27] NVIDIA-NAI-0002604 (GPU Gems 2: Programming Techniques for High-Performance Graphics and General-Purpose Computation (2005)) at 464 ("In the stream programming model, all data is represented as a stream—which we define as an ordered set of data of the same data type. That data type can be simple (a stream of integers or floating-point numbers) or complex (a stream of points or triangles or transformation matrices).").

[28] NVIDIA-NAI-0002604 (GPU Gems 2: Programming Techniques for High-Performance Graphics and General-Purpose Computation (2005)) at 464.

more than graphics rendering. Their architecture excelled in tasks with "high arithmetic intensity," such as solving systems of linear equations, making GPUs an attractive solution for a wide range of computationally demanding applications beyond their traditional graphics domain.[29]

3. APIs and Programming GPUs

Prior to the Asserted Patents, as adoption of GPGPU spread and demand grew, graphics hardware manufacturers such as NVIDIA developed improved software tools for programmable GPUs, such as advanced application programming interfaces (APIs) and programming languages. These tools facilitated interactions with GPU pipelines, including programmable shaders for GPGPU tasks. Graphics APIs act as a bridge between software applications and the graphics hardware by providing high-level, simple-to-use functions for developers while handling low-level communication with the hardware.

For example, programmers made use of NVIDIA's GPUs through graphics Application Programing Interfaces (APIs), such as Direct3D and OpenGL, as well as GPU languages such as HLSL, GLSL, and Cg.[30] On release, the GeForce 256 GPU used Microsoft's DirectX 7.0 API.[31] By 2004, NVIDIA's GPUs supported OpenGL 2.1, then developed by Silicon Graphics, Inc. NVIDIA worked in tandem with API designers to enable programmers to make better use of their GPUs.

---

[29] NVIDIA-NAI-0002604 (GPU Gems 2: Programming Techniques for High-Performance Graphics and General-Purpose Computation (2005)) at 464–67, 495.

[30] NVIDIA-NAI-0002604 (GPU Gems 2: Programming Techniques for High-Performance Graphics and General-Purpose Computation (2005)) at xxxii; '461 Patent at 2:20–22 ("A GPU on a conventional video card is usually controlled through OpenGL, DirectX, or similar graphic application programming interfaces (APIs).").

[31] NVIDIA-NAI-0003861 (GeForce 256 Press Release) at 2.

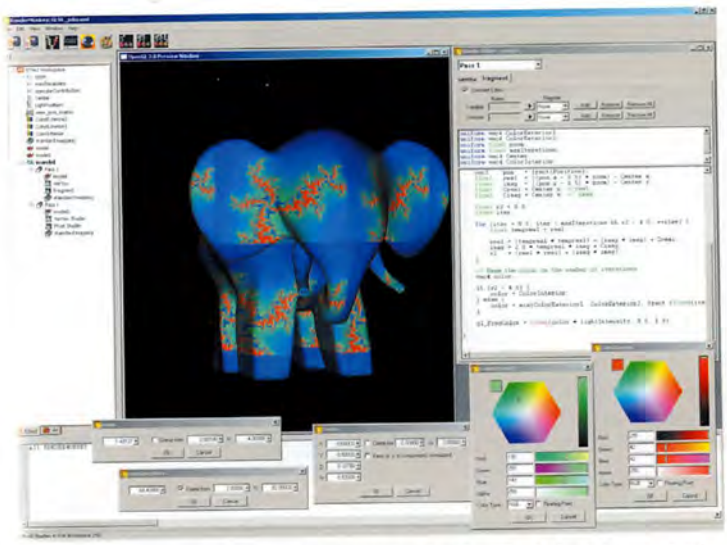
OpenGL, short for Open Graphics Library, is a cross-platform API developed in 1992 by Silicon Graphics to standardize the way developers program hardware to render graphics. It provided a hardware-agnostic interface, allowing developers to create complex graphical applications without worrying about the specifics of the underlying hardware. Initially designed for high-performance workstations used in industries like CAD, scientific visualization, and video production, OpenGL soon became a key tool in game development and real-time rendering. With the introduction of programmable shaders in GPUs during the late 1990s and early 2000s (discussed in further detail below), OpenGL evolved to support these features, enabling developers to write custom shader programs in GLSL (OpenGL Shading Language).[32]

Other high-level programming languages prior to the Asserted Patents included HLSL (High-Level Shader Language) and Cg (C for Graphics). HLSL, developed by Microsoft, was introduced as part of DirectX to provide developers with a tool for creating custom shaders on Windows platforms. Cg, developed by NVIDIA, aimed to simplify GPU programming by providing a C-like syntax, making it accessible to a broader range of developers.[33]

Visual tools were also created to facilitate GPU programming and interacting with programs such as modeling and simulation programs. For example, the book *OpenGL Shading Language* shows a graphical user interface (GUI) that allowed a user to interactively adjust parameters used by the GPU for shading an object.

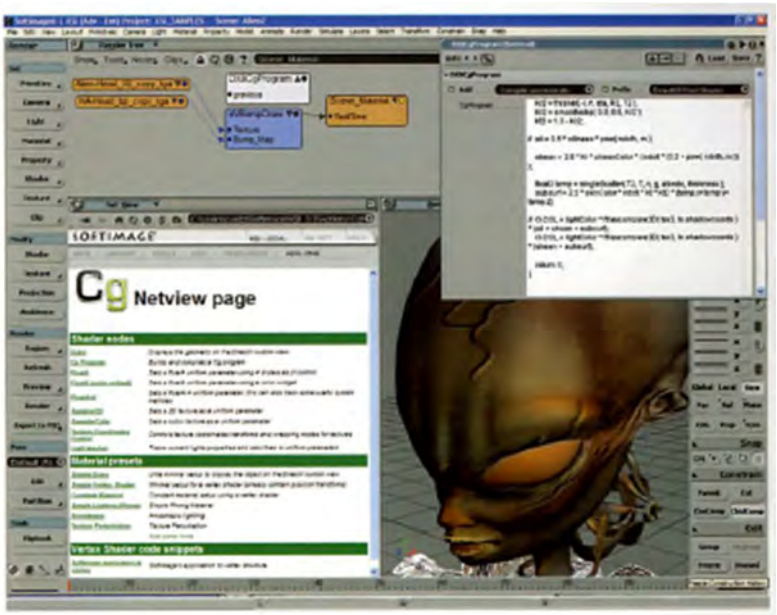
[32] NVIDIA-NAI-0005887 (Cg Tutorial) at 26–29.

[33] NVIDIA-NAI-0005887 (Cg Tutorial) at 4–5 ("Cg program can perform physical simulation, compositing, and other nonshading tasks.").



olor Plate 1 Screen shot of the RenderMonkey IDE user interface. The shader that is procedurally nerating a Julia set on the surface of the elephant is shown in the source code window. Color selection ols and user interface elements for manipulating user-defined uniform variables are also shown. [34]

Similarly, the book *Cg: The Cg Tutorial* describes and shows various Cg viewers that can be used to facilitate user interaction.



[35]

[34] NVIDIA-NAI-0003832 (OpenGL Shading) at 27.

[35] NVIDIA-NAI-0005887 (Cg Tutorial) at 39–41, 44.

4. Known Memory Management Techniques

As GPUs became more powerful, with increasing clock speeds and chip sizes, the amount of time it took to transfer data (cost of communication) increased compared to the amount of time it took to perform computations (cost of computation).[36] Thus, memory management techniques were already an area of focus for developers.

a. Using Pointers for Memory Management

Computers store information in memory. Memory is not an infinite resource, so programmers are careful to avoid storing data that will not be needed later. Further, memory is more limited the closer it is to the processor. A processor generally has a local memory cache with less space, and a larger memory (such as DRAM, which is main memory, SSD, which is Flash-based storage, or a hard drive) with more space. But transferring data to and from main memory or storage can be much slower than moving data in the local cache, and so programmers avoid unnecessary such transfers. Reading and writing data takes time, even in the processor cache, so programmers avoid unnecessary copying (which both reads and writes data).

The C programming language, first released in 1977, has long offered sophisticated tools to handle memory use and access. Data used during the operation of a program is called runtime data, and these are generally stored in memory cache. The C programming language includes high-level functions to allocate and deallocate memory in the cache for variables, and it uses automatic processes that deallocate local variables when a function closes.[37] To pass an array (*i.e.*, data) to a function, C instructs programmers to create a pointer variable which stores the address of the array in the memory, and then pass that address to the function. This approach allows the

---

[36] NVIDIA-NAI-0002604 (GPU Gems 2: Programming Techniques for High-Performance Graphics and General-Purpose Computation (2005)) at 459–60.

[37] NVIDIA-NAI-0001112 (C Programming Language) at 86–116.

function to read data from the space in memory directly and to write data back to that same space directly. Without using pointers, a system would have to copy into a memory location accessed by the function and then copy the resulting data back at the end of the function, which would require a lot of time and system resources. These design features reduce the number of unnecessary read/write cycles and reduce the risk that the processor cache will run out of memory.

Similar memory management features quickly became common among computer programming languages, particularly those developed for numerical simulation and computation. Fortran, for example, was one of the first abstract languages used in scientific computation, and it introduced pointer handling in 1990.[38]

b. Using Outputs as Inputs and Pointer Swapping

It was well known that in many numerical simulations and computations, the output of one computational step is used as the input for the next step. For example, in a simulation of a physical system over time, the result (output) of one time step is fed into the simulation as an input for the next time step, creating a feedback loop that allows the model to evolve accurately over time. Prior to the Asserted Patents, those skilled in the art had already developed effective solutions for implementing these kinds of computations in computers.

A simple approach is to copy the output data into a new array for use in the subsequent step, but this copying process incurs a performance cost due to the time and memory bandwidth required to duplicate potentially large datasets. A more efficient alternative—which was well known and used in the art prior to the Asserted Patents—is to avoid copying altogether by swapping the pointers of the input and output data arrays. By simply redirecting the pointer that

[38] NVIDIA-NAI-0002039 (Fortran Pointers) at 17–20 ("There are at least two reasons why pointers are needed. The most frequently mentioned reason is so that Fortran programmers can manage the memory required by their applications.").

indicates which array to use as input, the simulation can proceed without duplicating data, significantly reducing overhead. This approach not only saves time but also minimizes memory usage, making it especially valuable in high-performance computing applications where memory and processing power are critical resources.

Programming techniques for swapping pointers existed in computer programming long before the Asserted Patents. For example, one textbook relating to the C programming language teaches a "swap" function, which swaps *pointers* to values, rather than swapping the values themselves, for efficiency.

> The way to obtain the desired effect is for the calling program to pass *pointers* to the values to be changed:
>
> ```
> swap(
> ```
>
> Since the operator & produces the address of a variable, &a is a pointer to a. In swap itself, the parameters are declared as pointers, and the operands are accessed indirectly through them.
>
> ```
> void swap(int *px, int *py)  /* interchange *px and *py */
> {
>  int temp;
>
>  temp = *px;
>  *px = *py;
>  *py = temp;
> }
> ```
> [39]

Prior to the Asserted Patents, GPUs had been designed in ways to easily allow the above principles to be applied. For example, GPUs (such as NVIDIA's) allowed shaders to render results to a buffer (memory) rather than a display. It was known to use these buffers to perform iterative computations, using the output of one rendering step as the input to the next, until a final frame was ready to render to the display (in the case of graphics) or final result was passed back to a CPU and/or system memory (in the case of GPGPU). It was well understood that programmers could use these buffers to solve iterative numerical simulations with limited read-write cycles

[39] NVIDIA-NAI-0001112 (C Programming Language) at 88.

between CPU and GPU.[40] One known way to accomplish this was through "render-to-texture," through which the results of a program (shader) are written to local texture memory "so they can be used as input to future programs."[41]

The concept of "pointer swapping" was well known in the graphics and GPGPU space prior to the Asserted Patents as a way to leverage the output of one computation as the input to the next. For example, in the book GPU Gems, NVIDIA employee and author Mark Harris provides C++ code for simulating a reaction-diffusion system on a GPU that includes a "`swap()`" function at the end of each iteration that allows the output of one step of the simulation to become the input of the next step. The code defines one buffer (*i.e.*, portion of memory) as the input (`currentSource`) and another as the output (`currentTarget`). That is, the input data for a step is stored in a memory location pointed to by the variable `currentSource` and the output data is saved in a memory location pointed to by the variable `currentTarget`. At the end of computational cycle, the function `swap()` swaps the pointers for the `currentSource` buffer with that of the `currentTarget` buffer, so that the data pointed to by `currentTarget` becomes the input pointed to by `currentSource` for the next cycle.

[40] NVIDIA-NAI-0002604 (GPU Gems 2: Programming Techniques for High-Performance Graphics and General-Purpose Computation (2005)) at 501–08.

[41] NVIDIA-NAI-0002604 (GPU Gems 2: Programming Techniques for High-Performance Graphics and General-Purpose Computation (2005)) at 501–08.

Listing 31-2 *(continued)*. C++ Code to Set Up and Run a Reaction-Diffusion Simulation on the GPU

```
  // Initialize the state of the simulation with values in array
  pugInitBuffer(rdBuffer, array);
}

void update_rd(){ // Call this every iteration
  pugBindStream(rdProgram, "concentration", rdBuffer, currentSource);
  PUGRect range(0, width, 0, height);
  pugRunProgram(rdProgram, rdBuffer, range, currentTarget);

  std::swap(currentSource, currentTarget);
}
```

[42]

GPU Gems contains several other examples of pointer swapping during numerical computations.[43]

In this art, these techniques were sometimes known as "double buffering" or "ping-pong rendering." For example, OpenGL's "double buffering" used two framebuffers (which were dedicated areas of memory)—one for rendering and one for display—and swaps pointers between them at the end of each frame. In fact, by 2004, OpenGL 2.1 supported quadruple buffering.[44] DirectX 9 likewise supported double buffering at least by 2003.[45]

c. Transferring Data Between System Memories

Prior to the Asserted Patents, it was known that in numerical simulations and other computer applications, a user did not need to preserve all information from each time step. Rather, a user could elect when and how to store output data, including intermediate results of calculations.

---

[42] NVIDIA-NAI-0002604 (GPU Gems 2: Programming Techniques for High-Performance Graphics and General-Purpose Computation (2005)) at 508 (annotated).

[43] NVIDIA-NAI-0002604 (GPU Gems 2: Programming Techniques for High-Performance Graphics and General-Purpose Computation (2005)) at 44 ("Then, we swap the roles of the two textures and iterate the process . . . ."), 713 ("swap rho and newrho pointers" at the conclusion of each step of a "Conjugate Gradient Solver"), 743 ("To sort the entire field, we just call the next list and swap the buffers until we're done.").

[44] NVIDIA-NAI-0003831 (OpenGL 2.1 Reference) at 1.

[45] NVIDIA-NAI-0004483 (Advanced Image Processing) at 1, 18, 22–23; NVIDIA-NAI-0001953 (GPGPU Tutorials) at 17 (describing the "ping pong technique," in which the user "alternately use[s] the output of a given rendering pass as an input in the next one"); NVIDIA-NAI-0003913 (GPU Implementation of Neural Networks) at 323–24.

A user running a numerical simulation would understand that runtime data is primarily stored in the processor's cache as long as it fits within the available cache size. If the cache capacity is exceeded, data is transferred to other levels of memory, such as DRAM, which offers larger storage but with increased access latency. Since transferring data between different memory hierarchies incurs computational overhead in terms of time and resource management, the user must balance the need for storing output data with the cost of moving it. For example, the user might only require a steady-state solution at the final step of the simulation, minimizing storage needs. Alternatively, he or she might need to track the evolution of a concentration profile over time, choosing an appropriate step size to achieve a desired accuracy. In some cases, a variable step-size method might be employed to maintain numerical stability, while still ensuring output is recorded at regular intervals. In each scenario, the user configures the simulation to store output selectively, discarding or overwriting intermediate results as necessary. Ultimately, the user must strike an appropriate balance between data retention, computational efficiency, and storage constraints, based on the specific objectives of the simulation.[46]

5. Artificial Neural Networks and Other Applications of GPGPU

Long before the Asserted Patents, GPUs had been used for a variety of traditional scientific computing applications. This included, for example, solving systems of linear equations[47];

[46] NVIDIA-NAI-0003632 (Numerical Recipes in C) at 139–43 (showing a selection of step size according to tolerance for error and demand for speed).

[47] NVIDIA-NAI-0002604 (GPU Gems 2: Programming Techniques for High-Performance Graphics and General-Purpose Computation (2005)) at 703–18.

reaction-diffusion simulations[48]; protein folding and structure prediction[49]; and fluid dynamics and flow simulation,[50] among many others.[51]

One significant application that arose from early GPGPU advancements was the ability to simulate artificial neural networks on GPUs.[52] An artificial neural network is a computer-based model designed to mimic the way the human brain processes information. In the human brain, neurons are organized into layers that work together to process information. Neurons communicate by sending signals to each other, forming pathways that support thinking, learning, and memory. Information flows through layers of neurons, with each layer refining and interpreting the data. As pathways are used more frequently, they strengthen, allowing the brain to recognize patterns. The collection of layers of neurons is referred to as a "neural network."

An "artificial neural network" mimics a natural neural network in a computer. The artificial neural network is made up of layers of nodes—each acting like a neuron—that receive data, perform calculations, and pass the results to the next layer of nodes. Every node is connected to nodes in the next layer with parameters called weights. By adjusting these weights, or connections, between these layers of nodes during training, the network gradually learns to

---

[48] NVIDIA-NAI-0002604 (GPU Gems 2: Programming Techniques for High-Performance Graphics and General-Purpose Computation (2005)) at 505–08.

[49] NVIDIA-NAI-0000980 (Buck 2003) at 8; NVIDIA-NAI-0002604 (GPU Gems 2: Programming Techniques for High-Performance Graphics and General-Purpose Computation (2005)) at 695–702.

[50] NVIDIA-NAI-0000849 (3D Fluid Dynamics) at 1; NVIDIA-NAI-0002604 (GPU Gems 2: Programming Techniques for High-Performance Graphics and General-Purpose Computation (2005)) at 747–63.

[51] *See generally* NVIDIA-NAI-0002604 (GPU Gems 2: Programming Techniques for High-Performance Graphics and General-Purpose Computation (2005)) at Part VI ("Simulation and Numerical Algorithms").

[52] NVIDIA-NAI-0003913 (GPU Implementation of Neural Networks) at 322–24; NVIDIA-NAI-0000919 (Artificial Neural Network Computation) at 622.

recognize patterns and make decisions, much like how repeated experiences in the human brain strengthen neural pathways.

In one simple illustrative example, the inputs to a neural network are represented as a vector or matrix of values. Each input is then multiplied by a corresponding weight (representing the connection between layers of nodes). The sum of these weighted inputs is then calculated and that resulting vector is passed through an activation function that determines whether a node (neuron) should be activated (fire) based on the weighted sum of its inputs. Activation functions introduce non-linearity to the system, and enable the system to learn complex patterns. A bias term can be added to the weighted sum of inputs prior to passing through the activation function to shift the results to better fit the data. These computations are computed for all nodes in the layer, and the results are passed as inputs to the next layer.

An artificial neural network is usually trained by feeding a model large amounts of labelled data. For example, to identify handwritten numbers, the network would be provided with a large number of labelled images of numbers (positive examples) and other objects that are not numbers (negative examples). The computer simulating the neural network would then make predictions (*i.e.*, what number is represented in the image) by running the training data through its layers of nodes. The predictions are compared to the labels (the correct answers) to calculate an error. This error is propagated backward through the network—a process called backpropagation—which adjusts the internal weights to improve accuracy. Because this process requires performing millions of complex calculations (like matrix multiplication) across many layers and iterations, it is computationally intensive.

Prior to the Asserted Patents, it was known that the parallel processing power of GPUs was crucial for efficiently training these networks, as GPUs are particularly well-suited for the matrix

operations and numerical calculations required for neural network training.[53] For example, before the Asserted Patents, GPUs were already being utilized to power artificial neural networks in a variety of applications, including tasks such as image and pattern recognition,[54] object detection,[55] medical diagnostics,[56] and checkers.[57] The rapid progress in this field owed much to the computational capabilities provided by early GPGPU innovations, including innovations from NVIDIA and others.[58]

6. Real-Time Data Input

Prior to the Asserted Patents, the idea of using GPUs to process real-time streams of input, such as video data from a camera, was being explored in numerous research and development contexts. For example, GPU-based computer vision was well developed, with "[t]he single-instruction, multiple-data (SIMD) capability of the GPU mak[ing] it suitable for running computer vision tasks."[59]

It was thus well understood at the time that the parallel processing capabilities of GPUs made them particularly suitable for real-time inputs. Researchers and engineers had been experimenting with receiving real-time inputs, like live video streams from cameras, and sending

---

[53] NVIDIA-NAI-0003913 (GPU Implementation of Neural Networks) at 322–23 ("[M]any inner product operations can be replaced with matrix multiplication, which is more appropriate for GPU implementation.").

[54] NVIDIA-NAI-0003913 (GPU Implementation of Neural Networks).

[55] NVIDIA-NAI-0000919 (Artificial Neural Network Computation) at 622.

[56] NVIDIA-NAI-0000859 (ANN in Medical Intervention).

[57] NVIDIA-NAI-0003443 (Game Programming Gems) at 6.

[58] *See generally* NVIDIA-NAI-0002604 (GPU Gems 2: Programming Techniques for High-Performance Graphics and General-Purpose Computation (2005)).

[59] NVIDIA-NAI-0002604 (GPU Gems 2: Programming Techniques for High-Performance Graphics and General-Purpose Computation (2005)) at 649.

them to GPUs for tasks such as image processing, object tracking, and motion detection.[60] As one example, researchers had used artificial neural networks in a GPU to track a soccer ball in real time for robot soccer.[61]

7. Task and Resource Management on a GPU

GPGPU involves running many tasks concurrently within a single stage of processing and coordinating dependencies between stages of processing. Thus, efficient management of tasks assigned to the GPU and of system resources is critical for optimal utilization of a GPU's parallel processing capabilities. The operating system is the central software that manages hardware components in a computer, including CPUs and GPUs, ensuring that they work together smoothly and efficiently. One key part of this management is handled by the "scheduler," a mechanism built into the operating system that decides which tasks—whether they are individual threads or complete processes—get to use the computer's resources at any given time.[62] The scheduler works by assigning priorities and time slices to tasks based on factors like urgency, resource demands, and overall system load, ensuring that critical operations run promptly while also balancing the needs of less urgent tasks.

Schedulers enable shared resource utilization by determining which tasks should run at any given moment and making necessary computational resources available when required. For instance, once all the resources needed for a specific task are ready, the scheduler can prioritize and select that task for execution. The scheduler also shifts access to and control over particular

---

[60] NVIDIA-NAI-0003913 (GPU Implementation of Neural Networks); NVIDIA-NAI-0000919 (Artificial Neural Network Computation).

[61] NVIDIA-NAI-0000919 (Artificial Neural Network Computation) at 622.

[62] *E.g.*, NVIDIA-NAI-0000919 (Artificial Neural Network Computation); NVIDIA-NAI-0004886 (Gosalia); NVIDIA-NAI-0004877 (Lindholm).

resources based on the task. Scheduling policies govern how these decisions are made and take into account factors like task priority, resource demands, and timing. These policies are designed to meet various objectives, such as minimizing latency or response time, maximizing throughput, or meeting deadlines in real-time systems.

Prior to the Asserted Patents, one of the most widely used scheduling strategies was priority scheduling, where each task was assigned a priority value.[63] In the situation that a set of two or more tasks have the same priority value, the scheduling policy can shift to using round robin scheduling for that set of tasks, assigning time slices to each task in equal portions and in circular order so that each task has an equal chance of getting resources.[64] The scheduler would then use these priority levels to allocate resources, giving preference to higher-priority tasks. In systems that supported what is known as "preemption," higher-priority tasks could interrupt lower-priority ones, temporarily suspending their execution. When a task is preempted, the resources it was using are freed up for the higher-priority task. This dynamic allocation ensures that critical tasks are executed promptly while maintaining system responsiveness.

The concept of queuing tasks for executing was fundamental to task scheduling.[65] When a resource required for a particular task is currently in use, the scheduler will queue the task for later execution. Schedulers queue tasks as they come in by assigning each task a position in the queue based on a scheduling algorithm. For example, if a GPU is currently executing a computation, the scheduler can queue the task by placing it in a specific position in the queue based on factors such as task priority and resource availability. The queuing of tasks allowed

---

[63] NVIDIA-NAI-0004877 (Lindholm) at 4:11–5:62.

[64] NVIDIA-NAI-0004877 (Lindholm) at 4:11–5:62.

[65] *See, e.g.*, NVIDIA-NAI-0004733 (Buck '977).

operating systems to handle multitasking effectively, balancing the load and ensuring that high-priority tasks received timely attention. By managing tasks that require access to the same data, computational resources currently in-use, or have data dependencies, schedulers optimize efficiency, balance computational loads, and improve data coherency in a system.

C. Merrimac and Brook

Some of the allegedly infringing functionalities evolved from the prior art Brook streaming computing language and software, which was designed to run on a high-performance systems with a plurality of processing units, one of which was known as "Merrimac."

Merrimac and Brook (collectively, "Merrimac") were developed at Stanford, and the stream computing hardware and software concepts developed for Merrimac/Brook were applied to the CUDA computing architecture accused by Plaintiff. The Merrimac hardware was not built. Rather, the Merrimac designers developed its software, including the Brook stream computing language, and simulated the proposed hardware in computer simulations. Merrimac's proposed hardware architecture was described in detail in a number of printed publications ("Merrimac Publications") most of which were collected on the project's public website at http://merrimac.stanford.edu before the earliest alleged priority date of the Asserted Patents.

The Merrimac website is currently viewable through the Internet Archive. These Merrimac Publications describe different aspects of Merrimac's hardware and software, making it obvious for a POSITA to combine them in order to learn the system's architecture and specific features. Additionally, the simulation of the proposed hardware for Merrimac anticipates the Asserted Claims. References to "Merrimac" are shorthand for the description of various hardware and software aspects of the Merrimac project in the Merrimac Publications.

The Merrimac Publications comprise a selection from the following documents collected and publicly available on the Merrimac project's website at http://merrimac.stanford.edu:

- Conference and journal papers regarding Merrimac collected at http://merrimac.stanford.edu/publications/
- Merrimac Chapter in the 2003 CITS Annual Report, Streaming Supercomputer Whitepaper Brook specifications collected at http://merrimac.stanford.edu/publications/whitepapers.html
- Brook Specifications collected at http://merrimac.stanford.edu/brook/

Each of these documents describes the Merrimac supercomputing project and all of the documents were collected together on the project website.

The Brook computer language was an integral part of the Merrimac project because the proposed Merrimac hardware was designed with Brook in mind. A 2004 presentation given at the SIGGRAPH Conference discusses adaptation of Brook to a GPU architecture.[66] Likewise, a paper by the same presenters was also presented at SIGGRAPH 2004 directed to the development of Brook for GPUs describes Brook's features, evolution, and performance.[67]

A 2006 doctoral dissertation further discusses work on Merrimac and the extension of Brook to the GPU architecture.[68] Merrimac comprised specialized stream computing processing elements discussed in more detail below. This hardware utilized a large number of such processing elements for parallel computations. The Brook language (from which CUDA was developed) and Merrimac, both of which pre-date the Asserted Patents, discloses or renders obvious the alleged inventions, at least under Plaintiff's theories of infringement. For the avoidance of doubt, all of the analysis regarding Merrimac is presented under Plaintiff's apparent view of claim scope and meaning, and proposed claim constructions.

---

[66] NVIDIA-NAI-0000925 (SIGGRAPH 2004 Presentation).

[67] NVIDIA-NAI-0000961 (Brook for GPU 2004).

[68] NVIDIA-NAI-0000980 (Dissertation).

D. NVIDIA Technology Pre-Dates Neural AI and the Alleged Inventions

NVIDIA was founded on April 5, 1993, and focused on the development and introduction of new families of industry-defining accelerators that accelerate 2-D and 3D graphics and video in PC platforms. NVIDIA, among others in the industry, recognized that a general-purpose CPU was not efficient at handling audio and graphics data, especially the "repetitive geometry-processing operations."[69] At the time of NVIDIA's founding, it was clear that system performance would benefit from offloading these graphics related tasks from the CPU to a dedicated graphics chip.

As explained below, technologies and functionalities from NVIDIA that are accused of infringement (or precursors thereto) were developed prior to the alleged inventions in the Asserted Patents. For example, the programming of the GPU in C with CUDA built on the DirectX, OpenGL and Cg programmability of earlier architectures, as well as on Brook streaming computing language developed at Stanford.

Table 1. NVIDIA GPU technology development.

Date	Product	Transistors	CUDA cores	Technology
1997	RIVA 128	3 million	—	3D graphics accelerator
1999	GeForce 256	25 million	—	First GPU, programmed with DX7 and OpenGL
2001	GeForce 3	60 million	—	First programmable shader GPU, programmed with DX8 and OpenGL
2002	GeForce FX	125 million	—	32-bit floating-point (FP) programmable GPU with Cg programs, DX9, and OpenGL
2004	GeForce 6800	222 million	—	32-bit FP programmable scalable GPU, GPGPU Cg programs, DX9, and OpenGL
2006	GeForce 8800	681 million	128	First unified graphics and computing GPU, programmed in C with CUDA
2007	Tesla T8, C870	681 million	128	First GPU computing system programmed in C with CUDA
2008	GeForce GTX 280	1.4 billion	240	Unified graphics and computing GPU, IEEE FP, CUDA C, OpenCL, and DirectCompute
2008	Tesla T10, S1070	1.4 billion	240	GPU computing clusters, 64-bit IEEE FP, 4-Gbyte memory, CUDA C, and OpenCL
2009	Fermi	3.0 billion	512	GPU computing architecture, IEEE 754-2008 FP, 64-bit unified addressing, caching, ECC memory, CUDA C, C++, OpenCL, and DirectCompute

[70]

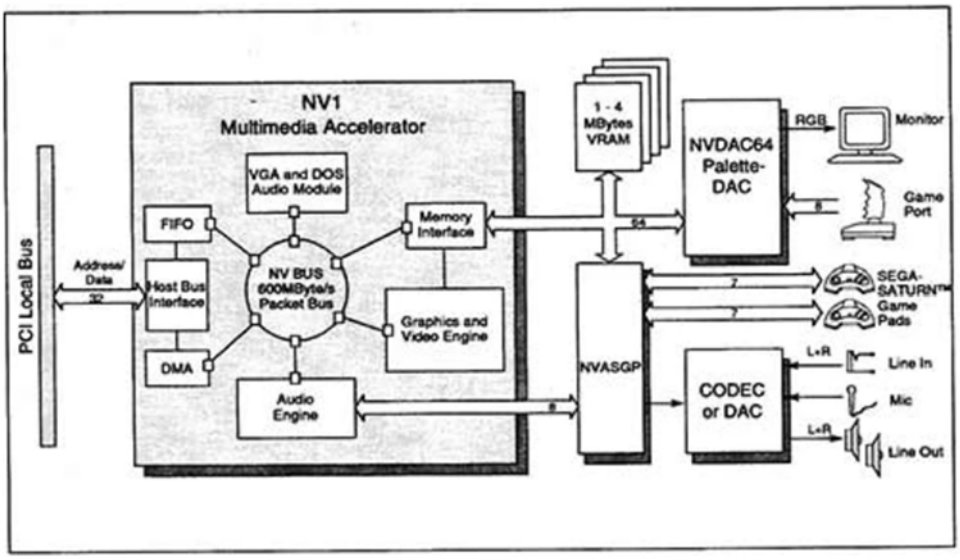
[69] NVIDIA-NAI-0005324 (PC 3D Prepares for PlayStation) at 2.

[70] NVIDIA-NAI-0004660 (Nickolls, Dally, "The GPU Computing Era," 2010).

1. The NV1 Chip and Edge 3D Board

NVIDIA worked for multiple years to create its first product, the NV1.[71] "The NV1 was a multimedia accelerator that provided 3D graphics, video and audio for interactive multimedia."[72] When NVIDIA introduced the NV-1 multimedia accelerator in 1995, the NV1 was the first multimedia accelerator to offer real-time 3D graphics in a single chip.[73] In doing so, NVIDIA defined a new product category.[74]

The NV1 chip offloaded several types of work from a computer CPU, including separate blocks that offloaded direct memory access (DMA), graphics and video, and audio tasks. These tasks were accomplished using, for example, an Audio Engine, Graphics and Video Engine and Host Bus Interface, shown the diagram below:[75]



[71] NVIDIA-NAI-0003585 (NVIDIA Corporate Timeline).

[72] NVIDIA-NAI-0003524 (NVIDIA 10-K Filed 1999) at 539.

[73] NVIDIA-NAI-0003822 (Jon Peddie, Nvidia's Quadratic Processor, the NV1, IEEE Computer Society (Jan. 18, 2021)).

[74] https://www.cecs.uci.edu/~papers/mpr/MPR/ARTICLES/100203.pdf.

[75] NVIDIA-NAI-0003822 (Jon Peddie, Nvidia's Quadratic Processor, the NV1, IEEE Computer Society (Jan. 18, 2021)).

The chip uses a separate engine for graphics and video.[76] The audio engine included a 50 MHz DSP that could perform 350 million operations per second.[77] These separate processing engines were interconnected through NV BUS, a 600Mbyte/s packet bus."[78] Additionally, NV1 provided a virtualized programming interface that provided for a programming architecture.[79] NVIDIA provided an SDK that was object-oriented with class-types (for example, for graphics, video, audio, etc.) that were written directly into programming registers on the NV1 chip.[80] Much of this architecture remains part of NVIDIA's products today.

The NV1 chip was incorporated into several models of add in boards (AIBs), including Edge 3D by Diamond Multimedia Systems.[81] In addition to the NV1 chip, the Edge 3D included additional circuitry to offload joypad related tasks from the CPU, that previously "occup[ied] up to 10% of the already straining CPU during use."[82]

---

[76] https://www.cecs.uci.edu/~papers/mpr/MPR/ARTICLES/100203.pdf.

[77] https://www.cecs.uci.edu/~papers/mpr/MPR/ARTICLES/100203.pdf.

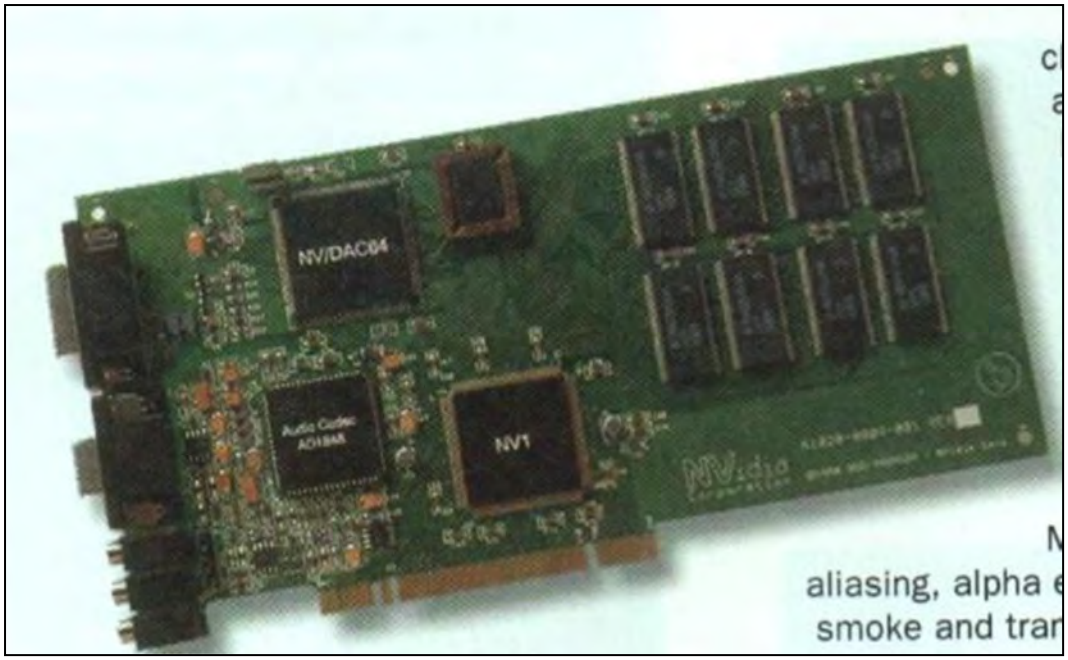
[78] NVIDIA-NAI-0003822 (Jon Peddie, Nvidia's Quadratic Processor, the NV1, IEEE Computer Society (Jan. 18, 2021)).

[79] NVIDIA-NAI-0003822 (Jon Peddie, Nvidia's Quadratic Processor, the NV1, IEEE Computer Society (Jan. 18, 2021)).

[80] NVIDIA-NAI-0003822 (Jon Peddie, Nvidia's Quadratic Processor, the NV1, IEEE Computer Society (Jan. 18, 2021)).

[81] *See* https://forums.anandtech.com/threads/history-of-nvidia-graphics-cards-vol-1.2586920/.

[82] NVIDIA-NAI-0005301 ("nVidia delivers fast 3D for PC," Edge, 2/29/1996).



2. GeForce 256 Graphics Processing Unit

NVIDIA released the GeForce 256 in 1999. The GeForce 256 was the first product marketed by NVIDIA as a graphics processing unit, or GPU.[83] The GeForce 256 included an NV10 chip that included the integrated transform and lighting (T&L) engine, triangle setup/clipping engines, and rendering engines in a single chip.[84] The T&L engine was itself made up of a vertex shader and a geometry translator.[85] The GeForce 256 was enabled to use DirectX 7.0 and OpenGL 1.2.1.[86][87] This functionality was also the basis for the Quadro 256 board.[88]

[83] NVIDIA-NAI-0002029 (Evolution of the Graphics Processing Unit (GPU)) at 42.

[84] NVIDIA-NAI-0002029 (Evolution of the Graphics Processing Unit (GPU)) at 42–43; NVIDIA-NAI-0003813 (NVIDIA's GeForce 256).

[85] NVIDIA-NAI-0003813 (NVIDIA's GeForce 256).

[86] NVIDIA-NAI-0004660 (Nickolls, Dally, "The GPU Computing Era," 2010).

[87] NVIDIA-NAI-0003813 (NVIDIA's GeForce 256).

[88] NVIDIA-NAI-0003813 (NVIDIA's GeForce 256).

This product garnered acclaim from the industry, including many technical awards.[89] The GeForce 256 doubled the number of graphics rendering pipelines from two in the RIVA TNT to four in GeForce256.[90,91] The GeForce 256 also combined the fragment (or pixel) computation engine with vertex computations that "requir[ed] significant floating-point performance" on the GPU.[92] NVIDIA's GeForce 256 delivered performance on the scale of 50 gigaFLOPS, or GFLOPS.[93] Generally, floating-point numbers are a form of mathematical notation often used with scientific problems where the number is represented by a value number that is raised to an exponent number. Computer performance is often measured by how many floating-point mathematical operations a computer can perform in a second, indicated by the shorthand FLOPS. Because computers can typically do many millions of FLOPS, it is common to refer to a million FLOPS as megaFLOPS, a billion FLOPS as gigaFLOPS, and a trillion FLOPS as teraFLOPs, indicated by the shorthand TFLOPS. By this time, NVIDIA had "become the largest independent graphics-chip supplier."[94]

3. Kelvin through Curie Architectures

The abilities of NV10 and the GeForce 256 evolved into programmable shaders within the Kelvin and Rankine architectures.[95] NVIDIA released the GeForce 3 (NV20) based on the Kelvin architecture in 2001 and GeForce FX (NV30) based on the Rankine architecture in 2002.[96] "[T]he

---

[89] NVIDIA-NAI-0005324 (PC 3D Prepares for PlayStation).

[90] https://www.tomshardware.com/picturestory/464-nvidia-graphics-cards.html.

[91] NVIDIA-NAI-0005324 (PC 3D Prepares for PlayStation).

[92] NVIDIA-NAI-0002029 (Evolution of the Graphics Processing Unit (GPU)).

[93] NVIDIA-NAI-0005290.

[94] NVIDIA-NAI-0005324 (PC 3D Prepares for PlayStation).

[95] NVIDIA-NAI-0004660 (GPU Computing Era); NVIDIA-NAI-0000703 (Evolution of GPUs).

[96] *See, e.g.*, NVIDIA-NAI-0082852 (NVIDIA GPU Architecture (2005)).

GeForce 3 introduced the first programmable vertex processor that executed vertex shader programs, along with a configurable 32-bit floating point pixel-fragment pipeline, programmed with OpenGL and [DirectX8.]"[97]

These products were the first to add programmable shader engines to GPUs, providing "a vertex shader in 2001, and in 2002, . . . introduced programmable fragment shaders."[98] The incorporation of GPU-hardware-based programmable shaders within the GeForce 3 (NV20) in 2001 and GeForce FX in 2002 was another NVIDIA pioneering innovation. The programmability was delivered by a "[s]imple to use programming model" that allowed the developer to "manipulate vertex transforms."[99] The field of general-purpose GPU computing emerged from the "combination of high floating-point performance and programmability" found within NVIDIA's GPUs.[100]

With respect to NVIDIA's NV40 GPUs based on the Curie architecture, it was explained that the programmable vertex engine and programmable fragment engine allowed the GPU to be "seen as a large amount of programmable floating-point horsepower and memory bandwidth that can be exploited for compute-intensive applications completely unrelated to computer graphics. . . . When used for non-graphics applications, it can be viewed as two programmable blocks that run serially: the vertex processor and the fragment processor, both with support for fp32 operands

---

[97] NVIDIA-NAI-0004660 (GPU Computing Era).

[98] NVIDIA-NAI-0002029 (Evolution of the Graphics Processing Unit (GPU)) at 43.

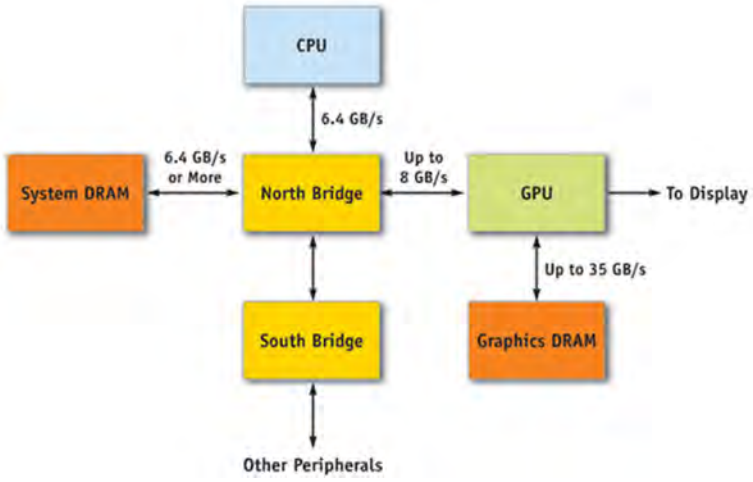
[99] https://web.archive.org/web/20030103124007/http://developer.nvidia.com/docs/IO/1310/SUPP/page07.jpg.

[100] NVIDIA-NAI-0002029 (Evolution of the Graphics Processing Unit (GPU) at 43–44.

and intermediate values. . . . In addition, both the vertex and the fragment processor are highly computationally capable."[101]

In 2004, NVIDIA released the GeForce 6800 and Quadro FX Curie graphics cards with NV45 graphics chips, also based on the Curie GPU architecture. Compared to the NV40, the NV45 GPU added an AGP-to-PCI Express bridge giving the card support for "Scalable Link Interface," known as "SLI."[102] In Multi-GPU mode, "the driver configures both boards as a single device: the two GPUs look like a single logical device to all graphics applications."[103] NV40/NV45 supported DirectX 9.0, OpenGL 2.0, and Cg.

The diagram below shows the overall system architecture of an NV40/NV45 GPU, such as a GeForce 6800:



[104]

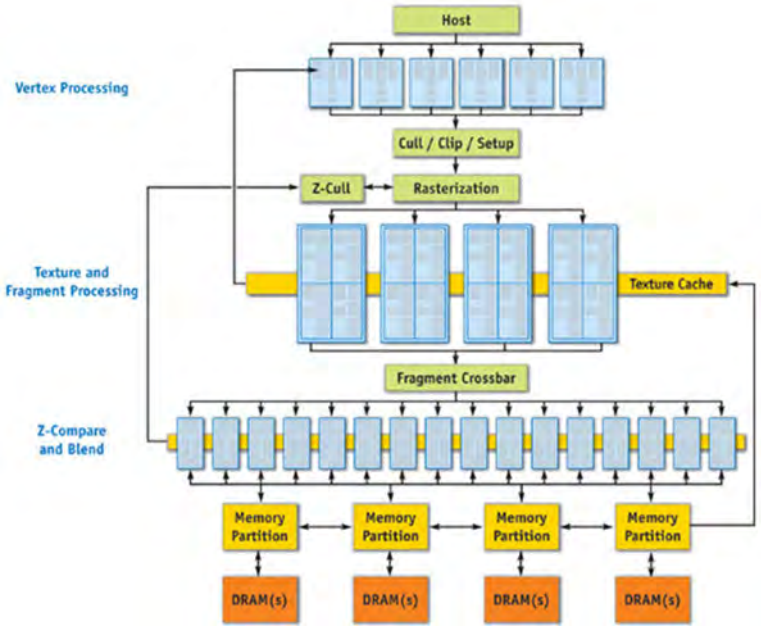
[101] NVIDIA-NAI-0002604 (GPU Gems 2: Programming Techniques for High-Performance Graphics and General-Purpose Computation (2005)) at 817–18.

[102] NVIDIA-NAI-0000624 (2005 GPU Programming Guide) at 59–60.

[103] NVIDIA-NAI-0000624 (2005 GPU Programming Guide) at 59.

[104] NVIDIA-NAI-0002604 (GPU Gems 2: Programming Techniques for High-Performance Graphics and General-Purpose Computation (2005)) at 473.

Below is a functional diagram of the graphics pipeline in an NV40/NV45 GPU, such as a GeForce 6800:



[105]

In 2005, NVIDIA released the G70 GPU, which was also based on the Curie architecture. This GPU was also known as the GeForce 7900 series. It contained more pixel shaders, vertex shaders, and texture mapping units than the other GPUs based on the Curie architectures, providing better performance and more detailed graphics.[106]

4. Tesla Architecture

In 2006, NVIDIA released the G80 GPU based on the Tesla architecture. The G80 (also referred to internally at NVIDIA for portions of development as the NV50) included the CUDA compute architecture, which, as discussed above, was an extension of prior GPU programming languages such as the Brook language.[107] The GeForce 8 GPU or G80 used "streaming

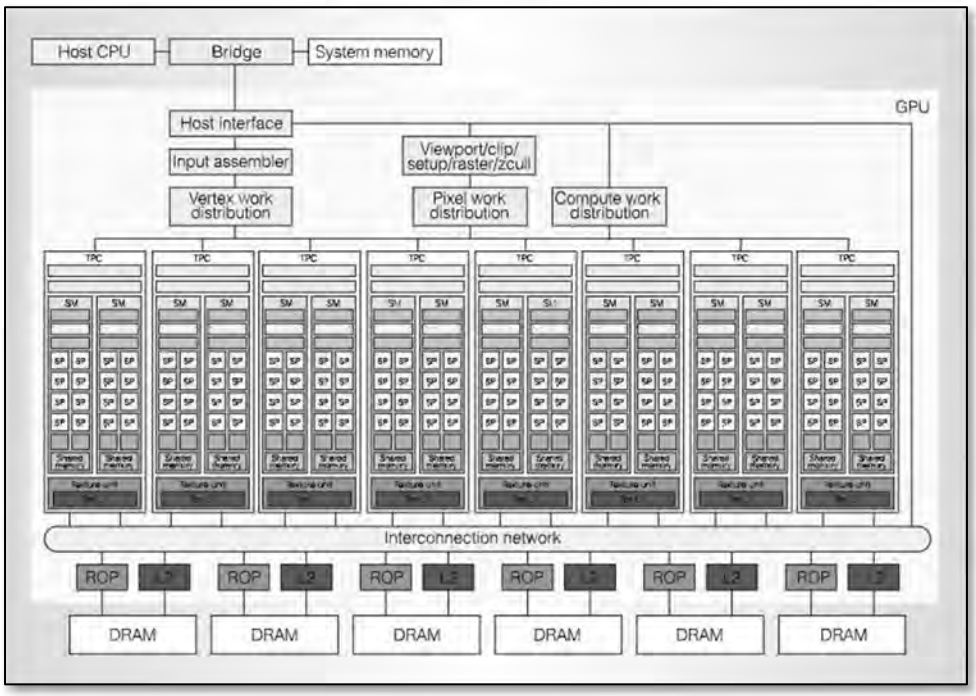
[105] NVIDIA-NAI-0002604 (GPU Gems 2: Programming Techniques for High-Performance Graphics and General-Purpose Computation (2005)) at 474.

[106] https://www.techpowerup.com/gpu-specs/nvidia-g70.g40.

[107] NVIDIA-NAI-0002046 (From Brook to CUDA).

multiprocessors (SMs) that were used to run both vertex and fragment shaders. These SMs could also run 'compute shaders' independent of the graphics pipeline."[108]

The G80 included 8 texture processor clusters with each having two streaming multiprocessors, SMs.[109] In the G80, each SM included 8 streaming processors. Double precision floating point support was also added to address the needs of scientific and high-performance computing (HPC) applications.[110] The Tesla GPU included a unified graphics and computing GPU architecture as shown in the block diagram:[111]



The G80 GPU supported a global pool of memory that was sharable by all threads in the compute context.[112] During the design of G80, NVIDIA recognized that the architecture would

[108] NVIDIA-NAI-0002029 (Evolution of the Graphics Processing Unit (GPU)) at 43.

[109] NVIDIA-NAI-1049091 (Tesla Compute) at 126.

[110] NVIDIA-NAI-0005302 (Fermi Compute Architecture).

[111] NVIDIA-NAI-0003594 (NVIDIA Tesla) at 41.

[112] NVIDIA-NAI-1049091 (Tesla Compute) 107.

provide a foundation for "[s]callable clusters of multi-threaded processors" using a "[s]hared local memory" and a "[s]hared global memory."[113] "A shared memory facilitated communication between the threads in an array."[114] During the design of G80, NVIDIA also recognized the ability of the G80 to perform FFT and BLAS calculations.[115] Further, the G80 Tesla design included Atomic operations, for example enabling "various RMW associative operation in global memory."[116] The atomic operations were added in the revision of the Tesla architecture.[117] The G80 GPU was capable of running CUDA 1.0 applications.[118] Tesla version 2 added dedicated hardware within the SM for double precision operations.[119]

The G80 GPU with the Tesla architecture included single-instruction, multiple-thread (SIMT) functionality. "The SM's SIMT multithreaded instruction units creates, manages, schedules and executes threads in groups of 32 parallel threads called warps. … Each SM manages a pool of 24 warps, with a total of 768 threads."[120]

At the release of G80, NVIDIA's GPUs included numerous processing engines, an interconnection network and at least one memory controller to perform parallel computing.[121]

---

[113] NVIDIA-NAI-1049018 (NVCompute Overview (July 22, 2005)) at 031.

[114] NVIDIA-NAI-0002029 (Evolution of the Graphics Processing Unit (GPU)) at 43.

[115] NVIDIA-NAI-1049018 (NVCompute Overview (July 22, 2005)) at 040.

[116] NVIDIA-NAI-1049091 (Tesla Compute) at 114.

[117] NVIDIA-NAI-1049127 (Tesla2 and Fermi Compute) at 129.

[118] NVIDIA-NAI-0001472 (NVIDIA CUDA Programming Guide) at 1–2.

[119] NVIDIA-NAI-1049127 (Tesla2 and Fermi Compute) at 130.

[120] NVIDIA-NAI-0003594 (NVIDIA Tesla) at 43–44.

[121] NVIDIA-NAI-0001472 (NVIDIA CUDA Programming Guide) at 1–5.

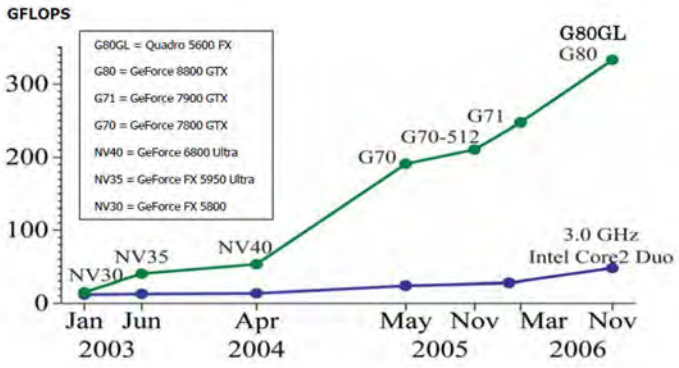


Figure 1-1. Floating-Point Operations per Second for the CPU and GPU

Independent sources reported that the single-precision floating-point performance of the G80 was 345.6 GFLOPS[122] and 518 GFLOPS.[123] The performance improved to 933 GFLOPS in the Tesla 2.0 in a GTX280.[124]

5. Fermi Architecture

While NVIDIA's Fermi architecture was launched in 2010, its development began before the Asserted Patents. It included a set of Streaming Multiprocessors with each SM having 32 streaming processors.[125] In addition to adding CUDA cores, Fermi supported a unified address space for scratchpad memory, the graphics processing unit memory, and the system memory.[126] Fermi also expanded the capabilities of atomic operations.[127] The Fermi architecture delivered 8x the peek double precision floating point performance over the GT200.[128]

---

[122] NVIDIA-NAI-0003590 (NVIDIA GeForce 8800 GTX).

[123] NVIDIA-NAI-0004698.

[124] NVIDIA-NAI-0004698.

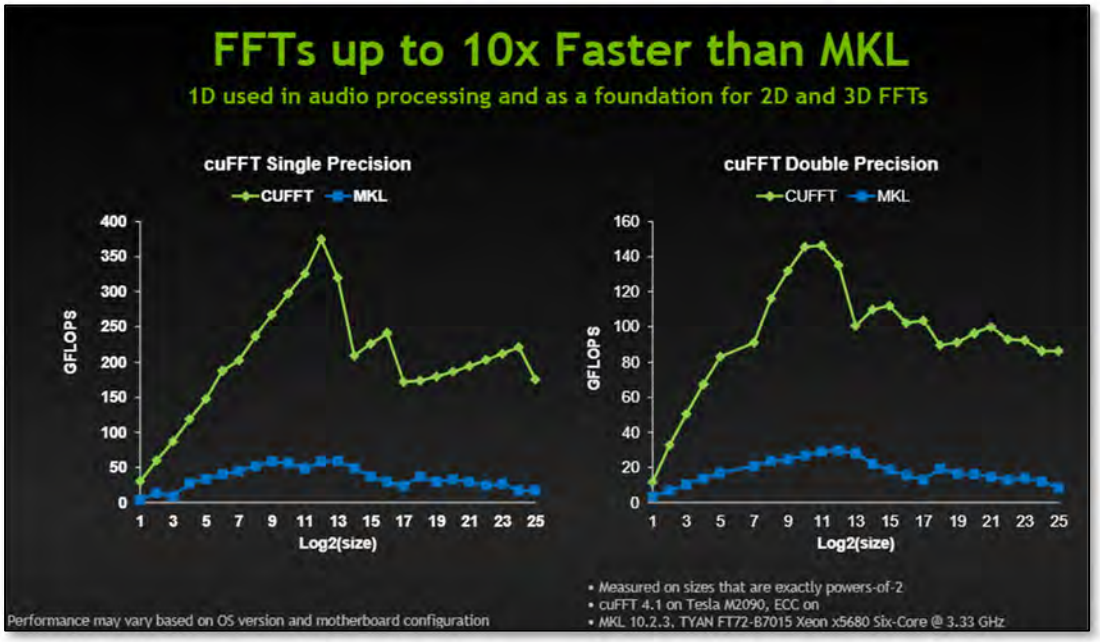
[125] NVIDIA-NAI-0005302 (Fermi Compute Architecture) at 8.

[126] NVIDIA-NAI-0005079 (The Top 10 Innovations in the New NVIDIA Fermi Architecture, and the Top 3 Next Challenges (2009)).

[127] NVIDIA-NAI-0005079 (The Top 10 Innovations in the New NVIDIA Fermi Architecture, and the Top 3 Next Challenges (2009)).

[128] NVIDIA-NAI-0005302 (Fermi Compute Architecture).

Comparison of the performance of a M2090 GPU using the Fermi architecture against an Intel six-core Xeon processor demonstrates that the Fermi GPU architecture results is significant performance advantages over the Intel architecture, and these improvements are not tied to the use of a GPU-GPU interconnect.[129]



6. NVIDIA CUDA [NVIDIA Confidential]

Throughout its history, NVIDIA has supported its products with software developed by NVIDIA, and this effort has expanded over decades. In addition to providing drivers for NVIDIA GPUs, as early as 2001, NVIDIA also made Software Development Kits (SDKs) available, including the NVSDK 5.1 that provided "advancements in the OpenGL SDK bring effects such as vertex program refraction, detail normal maps, procedural texturing with dynamic normal maps, and updates to nvparse, along with a wealth of up-to-date documentation. We are also excited to release the beginnings of an Audio SDK, which will continue to expand in anticipation of our

[129] NVIDIA-NAI-0003477 (Languages, APIs & Development Tools) at 484.

nForce products."[130] By 2006, NVIDIA provided a developer site with software offering multiple SDKs for GPU deployment, performance analysis, and content creation, including the NVSDK, NVIDIA Scene Graph SDK, Cg Toolkit, among many others.[131]

In addition to these software offerings largely related to graphics, in November 2006, NVIDIA publicly announced CUDA, a general purpose parallel computing platform and programming model that leverages the parallel compute engine in NVIDIA GPUs to solve many complex computational problems. [redacted]

CUDA development at NVIDIA had been going on long before the Asserted Patents, including the development of functionalities from prior art systems like Merrimac/Brook and Cg, as discussed above. For example, CUDA was built on the foundations laid by earlier programming languages such as Brook and Cg.[134] As explained above, Brook was designed to leverage the power of GPUs for general-purpose computing, introducing a stream computing model that allowed for parallel processing of data. This approach resonated with programmers, offering higher-level parallel programming concepts that could be compiled to the GPU. CUDA applied

---

[130] https://web.archive.org/web/20010927122209/http://developer.nvidia.com/.

[131] https://web.archive.org/web/20060813135512/http://developer.nvidia.com/page/tools.html.

[redacted]

[134] NVIDIA-NAI-0002046 (From Brook to CUDA), https://www.nvidia.com/content/GTC/documents/1408_GTC09.pdf; https://developer.nvidia.com/blog/inside-the-programming-evolution-of-gpu-computing/.

these same principles, providing a flexible and powerful platform for GPU computing. By moving beyond the constraints of graphics-focused programming models, CUDA enabled developers to harness the full potential of GPUs for a wide range of applications, from scientific computing to artificial intelligence.

III. INVALIDITY UNDER 35 U.S.C. §§ 102 AND 103

NVIDIA contends that the Asserted Claims are invalid as anticipated by the prior art under 35 U.S.C. § 102 and/or as obvious in view of the prior art, the knowledge of a person having ordinary skill in the art, and/or secondary factors of obviousness under 35 U.S.C. § 103.

A. Legal Standards

1. Anticipation

Under pre-AIA 35 U.S.C. § 102, anticipation occurs when a prior art reference or system discloses each and every element of a claimed invention, either explicitly or inherently. This means that if a prior art reference or system describes the same invention as the claim, the claim is considered anticipated and therefore not patentable.

2. Public Use/Sale Bar

Under the public use bar, "[a] person shall be entitled a patent unless . . . the invention was . . . in public use . . . in this country, more than one year prior to the date of the application for patent in the United States."[135] "The public use bar is triggered 'where, before the critical date, the invention is [(1)] in public use and [(2)] ready for patenting.'"[136] "An invention is in public use if it is shown to or used by an individual other than the inventor under no limitation, restriction,

[135] 35 U.S.C. § 102(b).

[136] *Polara Eng'g Inc. v. Campbell Co.*, 894 F.3d 1339, 1348 (Fed. Cir. 2018) (citing *Invitrogen Corp. v. Biocrest Mfg., L.P.*, 424 F.3d 1374, 1379 (Fed. Cir. 2005)).

or obligation of confidentiality."[137] To determine whether this occurred, the court considers, "inter alia, the nature of and public access to activities involving the invention [and] confidentiality obligations imposed upon observers."[138]

3. Obviousness

Under pre-AIA 35 U.S.C. § 103, "[a] patent may not be obtained . . . if the differences between the subject matter sought to be patented and the prior art are such that the subject matter as a whole would have been obvious at the time the invention was made to a person having ordinary skill in the art to which said subject matter pertains." The analysis of obviousness looks at (1) the scope and content of the prior art; (2) the differences between the claims and the prior art; (3) the level of ordinary skill in the art; and (4) secondary considerations of non-obviousness.[139]

The U.S. Supreme Court decision in *KSR International Co. v. Teleflex Inc.*[140] ("*KSR*") reiterated that patents that are based on new combinations of elements or components already known in a technical field may be found to be obvious. The Supreme Court in *KSR* rejected a rigid application of the "teaching, suggestion, or motivation [to combine]" test.[141] "In determining whether the subject matter of a patent claim is obvious, neither the particular motivation nor the avowed purpose of the patentee controls. What matters is the objective reach of the claim."[142] "Under the correct analysis, any need or problem known in the field of endeavor at the time of invention and addressed by the patent can provide a reason for combining the elements in the

---

[137] *Am. Seating Co. v. USSC Grp., Inc.*, 514 F.3d 1262, 1267 (Fed. Cir. 2008).

[138] *Id.*; *Delano Farms Co. v. California Table Grape Comm'n*, 778 F.3d 1243, 1247 (Fed. Cir. 2015).

[139] *Graham v. John Deere Co. of Kansas City*, 383 U.S. 1, 17–18 (1966).

[140] 550 U.S. 398 (2007).

[141] *Id.* at 418.

[142] *Id.*

manner claimed."[143] In particular, the Supreme Court emphasized the principle that "[t]he combination of familiar elements according to known methods is likely to be obvious when it does no more than yield predictable results."[144] A key inquiry is whether the "improvement is more than the predictable use of prior art elements according to their established function."[145] If a person of ordinary skill can implement a predictable variation, Section 103 likely bars its patentability.[146] The rationale to combine or modify prior art references is significantly stronger when the references seek to solve the same problem, come from the same field, and correspond well.[147]

In view of *KSR*, the PTO issued a set of guidelines for evaluating whether a claim is obvious in view of prior art. *See* Examination Guidelines for Determining Obviousness Under 35 U.S.C. § 103 in View of the Supreme Court Decision in *KSR International Co. v. Teleflex Inc*.[148] These guidelines identified various rationales for finding a claim obvious, including:

(a) Combining prior art elements according to known methods to yield predictable results;

(b) Simple substitution of one known element for another to obtain predictable results;

(c) Use of known technique to improve similar devices (methods, or products) in the same way;

(d) Applying a known technique to a known device (method, or product) ready for improvement to yield predictable results;

---

[143] *Id.* at 420.

[144] *Id.* at 415.

[145] *Id.* at 416–17.

[146] *Id.* at 416–17.

[147] *In re Inland Steel Co.*, 265 F.3d 1354, 1362 (Fed. Cir. 2001) (allowing two references to be combined as invalidating art under similar circumstances where the art "focus[ed] on the same problem . . . c[a]me from the same field of art [and] . . . the identified problem found in the two references correspond[ed] well").

[148] 72 Fed. Reg. 57,526 (Oct. 10, 2007).

(e) "Obvious to try"—choosing from a finite number of identified, predictable solutions, with a reasonable expectation of success;

(f) Known work in one field of endeavor may prompt variations of it for use in either the same field or a different one based on design incentives or other market forces if the variations would have been predictable to a person of ordinary skill in the art; and

(g) Some teaching, suggestion, or motivation in the prior art that would have led a person of ordinary skill to modify the prior art reference or to combine prior art reference teachings to arrive at the claimed invention.

Certain of these rationales are discussed more specifically below for each of the asserted patents. The fact that others are not discussed more specifically should not be interpreted as an admission or concession that it does not apply. To the contrary, the discussion below simply provides more explanation of these specific rationales.

B. Prior Art Invalidity of U.S. Patent No. 8,648,867

1. Identification of Prior Art

NVIDIA identifies the following prior art now known to NVIDIA that anticipate or render obvious one or more of the Asserted Claims of the '867 Patent. Each of the following patents, publications, systems, and products is prior art under at least one or more of 35 U.S.C. § 102(a), (b), (e), (f), or (g) (pre-AIA) or § 102(a) (post-AIA), as applicable. NVIDIA's reliance on each prior art reference below includes the reference itself, anything incorporated by the reference, and any testimony by those with knowledge of the reference, such as named authors and inventors.

a. Patents and Patent Publications

Patent / Publication No.	First Named Inventor	Date of Issue / Publication
Japanese Patent Appl. H11-232247	Takahashi Nakamoto	Aug. 27, 1999
Japanese Patent Publ. H04-237388	Yoichi Tamura	Aug. 25, 1992
U.K. Patent Appl. No. 2291553	David Wilson	Jan. 24, 1996
U.S. Patent No. 6,853,382	James Van Dyke	Feb. 8, 2005
U.S. Patent No. 7,139,003	David Kirk	Nov. 21, 2006

Patent / Publication No.	First Named Inventor	Date of Issue / Publication
U.S. Patent No. 7,219,085	Ian Buck	May 15, 2007
U.S. Patent No. 7,256,788	Viet-Tam Luu	August 14, 2007
U.S. Patent No. 7,339,592	John Lindholm	Mar. 4, 2008
U.S. Patent No. 7,418,576	John Lindholm	Aug. 26, 2008
U.S. Patent No. 7,673,304	Anuj Gosalia	Mar. 2, 2010
U.S. Patent No. 7,681,060	John Nickolls	Dec. 28, 2010
U.S. Patent No. 7,694,304	Kevin Kissell	Apr. 6, 2010
U.S. Patent No. 7,747,070	Siddhartha Puri	Jun. 29, 2010
U.S. Patent No. 7,830,392	John Danskin	Nov. 9, 2010
U.S. Patent No. 7,899,995	Radoslav Danilak	Mar. 1, 2011
U.S. Patent No. 8,059,131	Donald Bittel	Nov. 15, 2011
U.S. Patent No. 8,730,249	John Danskin	May 20, 2014
U.S. Patent No. 9,262,129	Michael Santori	Feb. 16, 2016
U.S. Patent Publ. No. 2003/0140179	Nicholas Wilt	Jul. 24, 2003
U.S. Patent Publ. No. 2003/0195938	Kevin Howard	Oct. 16, 2003
U.S. Patent Publ. No. 2005/0197977	Ian Buck	Sept. 8, 2005
U.S. Patent Publ. No. 2006/0098019	David Tarditi	May 11, 2006

b. Printed Publications

Citation	Publication Date
A. Gorchetchnikov, *KInNeSS: A new software environment for simulations of neuronal activity*	2004
A. Lefohn, *GPU Memory Model Overview*	2005
A. Veen, *Dataflow Machine Architecture*	1986
B. Kernigham et al., The C Programming Language	1988
B. Khailany, *Exploring the VLSI Scalability of Stream Processors*	2003
B. Khailany, *The VLSI Implementation and Evaluation of Area- and Energy-Efficient Streaming Media Processors*	2003
B. Svensson, *SIMD Architectures*	1992
C. Davis, *Graphics Processing Unit Computation of Neural Networks*	2005
Cg Toolkit: User's Manual	2005
D. Steinkraus, *Using GPUs for Machine Learning Algorithms*	2005
Dominik Göddeke -- GPGPU::Basic Math Tutorial	2005

Citation	Publication Date
E. Chu, *FFT algorithms and their adaptation to parallel processing*	1998
E. Kilgariff, *The GeForce 6 Series GPU Architecture*	2006
Game Programming Gems 4	2004
General Purpose Computation on Graphics Hardware	2005
GPU Gems 2: Programming Techniques for High-Performance Graphics and General-Purpose Computation	2005
H. Demuth, *Neural Network Toolbox for Use with MATLAB*	2004
H. Esmaeilzadeh, *Neural Network Stream Processing Core (NnSP) for Embedded Systems*	2006
H. Esmaeilzadeh, *NnSP: Embedded Neural Network Stream Processor*	2005
I. Buck et al., *Brook for GPUs: Stream Computing on Graphics Hardware* (presentation)	2004
I. Buck et al., *Brook for GPUs: Stream Computing on Graphics Hardware*	2004
I. Buck et al., *Data Parallel Computation on Graphics Hardware*	2003
I. Buck, *Brook for GPUs*	2003
I. Buck, *Stream Computing on Graphics Hardware*	2005
J. Ahn, *Scatter-Add in Data Parallel Architecture*	2005
J. Chen, *Fast Fourier Transform with BrookGPU*	2006
J. Fung, *Computer Vision Signal Processing on Graphics Processing Units*	2004
J. Fung, *OpenVIDIA: Parallel GPU Computer Vision*	2005
J. Georgii, *Interactive Simulation of Deformable Bodies on GPUs*	2005
J. Leiterman, *Learn Vertex and Pixel Shader Programming with DirectX 9*	2004
J. Martin, *Fortran 90 Pointers vs. "Cray" Pointers*	1991
J. Montrym, *The GeForce 6800*	2005
J. Ohmer, *Implementation of Kernel Methods on the GPU*	2005
J. Owens, *A Survey of General-Purpose Computation on Graphics Hardware*	2005
K. Jung, *Neural network-based text location in color images*	2001
K. Moreland, *The FFT on a GPU*	2003
K. Oh, *GPU implementation of neural networks*	2004
M. Botnen et al., *Using Commodity Graphics Hardware for Medical Image Segmentation*	2005

Citation	Publication Date
M. Erez, *Analysis and Performance Results of a Molecular Modeling Application on Merrimac*	2004
M. Erez, *Fault Tolerance Techniques for the Merrimac Streaming Supercomputer*	2005
M. Flynn, *Some Computer Organizations and Their Effectiveness*	1972
M. Glover et al., *A Massively-Parallel SIMD Processor for Neural Network and Machine Vision Applications*	1993
M. Harris, *GPGPU: General-Purpose Computation on GPUs*	2004
M. Harris, *Mapping Computational Concepts to GPUs*	2005
M. Harris, *Real-Time Cloud Simulation and Rendering*	2003
M. Kilgard, *A Follow-up Cg Runtime Tutorial for Readers of The Cg Tutorial*	2005
M. Kilgard, *Cg – A System for Programming Graphics Hardware in a C-like Language*	2006
M. Kilgard, *Cg and NVIDIA*	2006
M. Kilgard, *Cg in Two Pages*	2003
M. Kilgard, *NVIDIA OpenGL 2.0 Support*	2005
M. Kilgard, *Re-implementing the Follow-up Cg Runtime Tutorial with CgFX*	2006
M. Segal, *The OpenGL Graphics System: A Specification*	2004
MATLAB: The Language of Technical Computing	2005
Merrimac: The Streaming Supercomputer	2003
NVIDIA GPU Programming Guide (Version 2.4.0)	2005
OpenGL Super Bible	1996
P. Lisboa, *A review of evidence of health benefit from artificial neural networks in medical intervention*	2002
R. Agarwal, *A High Performance Parallel Algorithm for 1-D FFT*	1994
R. Hughey, *The UCSC Kestrel high performance SIMD processor: present and future*	2001
R. Rost, *OpenGL Shading Language*	2004
S. Rixner, *Stream Processor Architecture*	2001
S. Vitabile, *Real-Time Road Signs Recognition on a SIMD Architecture*	2004
The Cg Tutorial: The Definitive Guide to Programmable Real-Time Graphics	2003
U. Kapasi, *Programmable Stream Processors*	2003
U. Kapasi, *The Imagine Stream Processor*	2002

Citation	Publication Date
W. Dally, *Merrimac: Supercomputing with Streams*	2003
W. Press, *Numerical Recipes in C* (2d ed.)	2002
W. Thies, *StreamIt: A Language for Streaming Applications*	2002
Y Liu et al., *Real-Time 3D Fluid Simulation on GPU with Complex Obstacles*	2004
Z. Fan, *GPU Cluster for High Performance Computing*	2004
Z. Luo, *Artificial Neural Network Computation on Graphic Process Unit*	2005

c. Prior Art Products and Systems

Product / System
NVIDIA – NV40 (*e.g.*, GeForce 6800)
NVIDIA – G70 (*e.g.*, GeForce 7900)
NVIDIA – NV50/G80
BrookGPU-Merrimac ("Brook")
NVIDIA – Cg
The Imagine Stream Processor
SANNDRA / KInNeSS

d. Prior Invention Under § 102(g)

Prior Invention	Inventor / Assignee
NV50/G80	NVIDIA

Additionally, as explained in Section I.C, Plaintiff's Infringement Contentions are deficient and fail to provide sufficient allegations as to how the accused products infringe each and every claim limitation of the '867 Patent. Indeed, for most limitations, Plaintiff's Infringement Contentions are vague and unclear as to which particular functionality Plaintiff contends satisfies a claim limitation, prejudicing NVIDIA's ability to fully investigate invalidity under Plaintiff's theories of infringement.

While NVIDIA does not agree that the functionalities accused of infringement in Plaintiff's Infringement Contentions meet the claim limitations, the functionalities that Plaintiff apparently

accuses of infringement were in development prior to the '867 Patent. Thus, to the extent those functionalities infringe the '867 Patent, they also invalidate it. *Peters v. Active Mfg. Co.*, 129 U.S. 530, 547 (1889) ("That which infringes, if later, would anticipate, if earlier."). For example, as explained in Section II.D.6, CUDA development at NVIDIA had been going on long before the priority date of the Asserted Patents, including functionalities from prior art systems like Merrimac/Brook and Cg, as discussed above. CUDA was built on the foundations laid by earlier programming languages such as Brook and Cg, and CUDA-related development documents show functionalities that Plaintiff may accuse of infringement as having existed prior to any asserted or actual priority date for the '867 Patent. For example, libraries such as cuFFT and cuBLAS were developed prior to the priority date of the '867 Patent. The following exemplary categories of documents evidence prior invention and development by NVIDIA:

- CUDA Internal Programming Guides
- CUDA Language Specifications
- CUDA API Specifications and Documentation
- CUDA Driver Compiler Internal Documentation
- cuFFT Internal Documentation
- cuBLAS Internal Documentation

NVIDIA reserves the right to rely on these and any other documents showing NVIDIA's prior invention. NVIDIA further incorporates by reference its charts on Brook/Merrimac and Cg, which (as explained above) were precursors to the accused CUDA functionality developed at NVIDIA.

Thus, to the extent Plaintiff clarifies its infringement allegations by identifying and explaining how specific CUDA or other functionality allegedly meets the claim limitations,

NVIDIA reserves the right to demonstrate that the same functionality is prior art under 35 U.S.C. § 102(g)(2).

2. Anticipation

NVIDIA has identified the following prior art known to NVIDIA as anticipating the Asserted Claims of the '867 Patent. Each of the references cited below anticipates the Asserted Claims, either expressly or inherently as understood by a person of ordinary skill in the art, depending on the Court's interpretation of the claims. In some instances, NVIDIA has treated certain prior art as anticipatory, where certain elements are present based on at least Plaintiff's apparent claim construction in its Infringement Contentions. Exhibits A1–A13 are charts setting forth where in the prior art references each element of the Asserted Claims of the '867 Patent are found. Where NVIDIA charts a prior art product or system based on patents and/or printed publications related to that product/system, NVIDIA reserves the right to rely on such patents and/or printed publications alone as anticipatory references to the Asserted Claims of the '867 Patent.

Exhibit	Reference
A1	NV40
A2	G70
A3	NV50/G80
A4	Cg
A5	Brook
A6	Imagine Stream Processor
A7	SANNDRA / KInNeSS
A8	U.S. Patent No. 7,681,060 (Nickolls)
A9	U.S. Patent No. 7,139,003 (Kirk)
A10	U.S. Patent No. 7,219,085 (Buck '085)
A11	GPU Gems 2: Programming Techniques for High-Performance Graphics and General-Purpose Computation

Exhibit	Reference
A12	Japanese Patent Application JPH11232247A (Nakamoto)
A13	Stream Processor Architecture (Rixner)

3. Obviousness

To the extent that Plaintiff argues that any of the above references do not anticipate the Asserted Claims of the '867 Patent, the claims are invalid as obvious under 35 U.S.C. § 103 in view of these references alone, in combination with the knowledge of a POSITA, and in combination with other references as described below. Exhibits A1–A13 and Appendix A further identify specific examples of art that alone or in combination render obvious the limitations of the Asserted Claims of the '867 Patent. Additional evidence of obviousness can be found in the reference identified in the section identifying prior art patents, publications, and systems.

While some of the above identified prior art may anticipate the asserted claims, this presents no obstacle to also relying on this reference as a basis for invalidity based on obviousness. It is established law that "a rejection for obviousness under § 103 can be based on a reference which happens to anticipate the claimed subject matter." *In re Meyer*, 599 F.2d 1026, 1031 (C.C.P.A. 1979). And to the extent any prior art item cited herein may not fully disclose a limitation of an asserted claim or is alleged by Plaintiff to lack disclosure of the limitation, such limitation is present and identified in another prior art item as shown in the attached claim charts. It would have been obvious to combine the disclosed claim elements based on the teachings of other art in existence at that time, the general knowledge of a person of ordinary skill in the art, and the recognition of similarities between the art and the claimed features. On such basis, on an element-by-element basis, NVIDIA expressly intends to combine one or more prior art items identified herein with each other to address any further contentions from Plaintiff that a particular prior art item supposedly lacks one or more elements of an asserted claim. In other words,

NVIDIA contends that each charted prior art item can be combined with other prior art items if a particular prior art item lacks or does not explicitly disclose an element or feature of an asserted claim.

a. Level of Ordinary Skill in the Art

A person of ordinary skill in the art at the time of the earliest effective filing date of the '867 Patent would have had a Bachelor's degree in electrical or computer engineering (or equivalent discipline) and at least two years' experience in research, design and/or development of computer systems, including experience with GPUs. Additional education could substitute for experience, and vice versa.

a. Scope and Content of the Prior Art

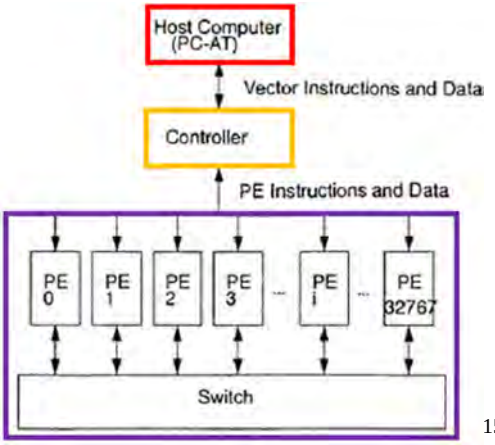
The '867 Patent is directed to a system for performing *general-purpose* computing (such as numerical simulations) on traditionally *special-purpose* GPUs. The '867 Patent does not claim to have invented performing general-purpose computing on GPUs or any specialized hardware for doing so.[149] Instead, the patent claims to have invented a novel architecture that offloads the setup and control of GPU computations from the host CPU to an "accelerator controller," freeing the CPU for other tasks.[150] The USPTO issued the '867 Patent solely on the basis that the use of the claimed "accelerator controller" was not disclosed in the prior art of record (which was only four references). As explained throughout these Invalidity Contentions, including in Exhibits A1–A13 and Appendix A, the use of an "accelerator controller" to perform numerical simulations was well known prior to the '867 Patent, including in NVIDIA's own patents (such as Nickolls and Kirk).

[149] '867 Patent at 1:21–30, 14:13.

[150] *Id.* at 2:9–20.

So too were the other concepts in the Asserted Claims of the '867 Patent well known by those skilled in the art.

As explained in Section II.B.1 (incorporated herein by reference), prior to the '867 Patent, those skilled in the art had created effective designs for parallel processing. One common feature was a "controller" to coordinate the function of the parallel processors. These systems included a host computer or processor (red), an accelerator (*e.g.*, a series of processors) (purple), and a controller (orange) separate from the CPU. It was well known that having the controller, rather than the CPU, manage the processors frees up the CPU to perform other functions—one of the benefits the inventors allege for the "accelerator controller" in the '867 Patent.[151] Indeed, it was understood that one benefit of GPGPU applications was that, with the architecture described above and with calculations occurring in parallel on the GPU, they "free up the CPU for other tasks."[152]

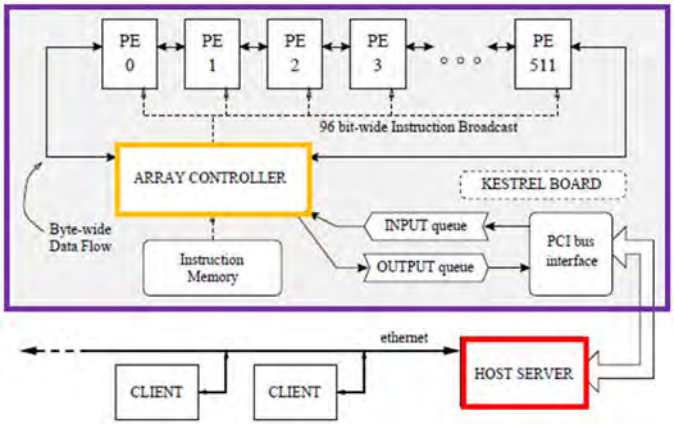


[153]

[151] '867 Patent at 2:19–20, 13:32–39.

[152] NVIDIA-NAI-0002604 (GPU Gems 2: Programming Techniques for High-Performance Graphics and General-Purpose Computation (2005)) at 650.

[153] NVIDIA-NAI-0003517 (Massively-Parallel SIMD Processor) at 845 (SIMD processor for neural network computation) (annotated).



[154]

The Asserted Claims of the '867 Patent also recite techniques such as using "pointers" to identify memory addresses and swapping pointers to output and input data addresses so the output of one computational cycle becomes the input of a subsequent cycle. As explained in Section II.A.4 (incorporated herein by reference), these concepts were well known prior to the '867 Patent. Prior to the '867 Patent, it was known that using pointers for memory addresses offers significant benefits in programming. Pointers allow functions to directly access and manipulate data stored in memory, which reduces the need for copying data between different memory locations. This approach minimizes read/write cycles, saving time and system resources. For example, as early as the C programming language, released in 1977, programming languages included functions to pass the address of an array to a function, which allowed efficient reading and writing of data.[155]

It was also well known that in numerical simulations and computations, the output of one step is often used as the input for the next step. To avoid the performance cost of copying data, programmers use pointer swapping, which redirects the pointer to the data rather than duplicating it. This technique saves time and minimizes memory usage, making it valuable in high-performance computing. GPUs, such as those from NVIDIA, have been designed to apply these

[154] NVIDIA-NAI-0004674 (High-Performance SIMD) at 2 (annotated).

[155] NVIDIA-NAI-0001112 (C Programming Language) at 86–116.

principles, using buffers to perform iterative computations with limited read-write cycles. Techniques like "render-to-texture" and "double buffering" were well known in the graphics and GPGPU space, allowing efficient use of memory and processing power. These techniques were described and used, for example, in NVIDIA's book GPU Gems 2, including in an example of a numerical simulation of a chemical reaction-diffusion.[156]

b. Motivations to Combine Prior Art

A person of ordinary skill in the art would have been motivated to combine the above identified prior art items. Here, the claimed elements of the '867 Patent are disclosed in the prior art, as established above in Exhibits A1–A13 and Appendix A. Further, any elements claimed in the Asserted Claims would have been obvious to a person of ordinary skill in the art and predictable solutions resulting from routine optimization. Certain of these rationales are discussed more specifically below. The fact that others are not discussed more specifically should not be interpreted as an admission or concession that it does not apply. To the contrary, the discussion below simply provides more explanation of these specific rationales.

As set forth below, it would have been obvious to one of person of ordinary skill in the art to combine the teachings of similar prior art references to render the Asserted Claims obvious. The references identified in the exemplary combinations set forth below are analogous prior art and for at least the reasons set forth below, a person of ordinary skill in the art would have had reasons for combining them. The various teachings, suggestions, and/or reasons to modify any of the references and/or to combine any two or more references come from various sources, including the prior art (specific and as a whole), common knowledge, common sense, predictability,

[156] NVIDIA-NAI-0002604 (GPU Gems 2: Programming Techniques for High-Performance Graphics and General-Purpose Computation (2005)) at 505–08.

expectations, industry trends, design incentives or need, market demand or pressure, market forces, the nature of the problem faced, and/or the knowledge possessed by one of ordinary skill in the art.

Furthermore, to the extent that Plaintiff argues any modification of a base reference is required to meet the claim elements, such modification was routine and predictable, within the skill of a person of ordinary skill in the art, and a person of ordinary skill in the art would have had a reasonable expectation of success in making such a combination.

The following are exemplary motivations to combine:

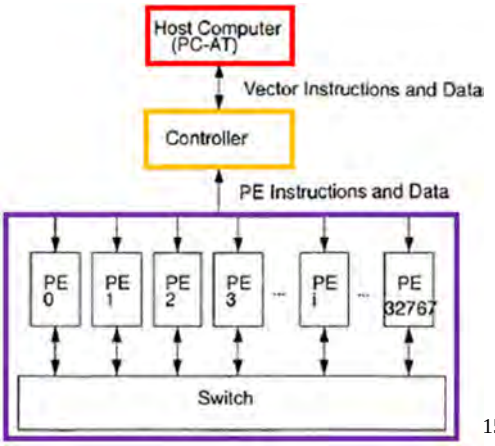
(1) Performing Operations by an "Accelerator Controller"

Some limitations of the Asserted Claims of the '867 Patent relate to operations performed by an "accelerator controller," including:

- "transferring, by an accelerator controller, the first input data into a first partition, referenced by first pointer, of an accelerator memory before a first computational cycle of the numerical simulation" (cl. 16);
- "storing, by the accelerator controller, the first output data into a second partition, referenced by a second pointer, of the accelerator memory" (cl. 16);
- "sending, by the accelerator controller, instructions for performing the at least one calculation to the at least one graphics processing unit" (cl. 17); and
- "transferring, by the accelerator controller, the first output data to the main memory during the second computational cycle" (cl. 19).

To the extent that any of the primary references do not expressly disclose these or any related limitations, a POSITA would have been motivated to modify any such references based on any of the references in Exhibits A1–A13 and Appendix A disclosing those limitations.[157]

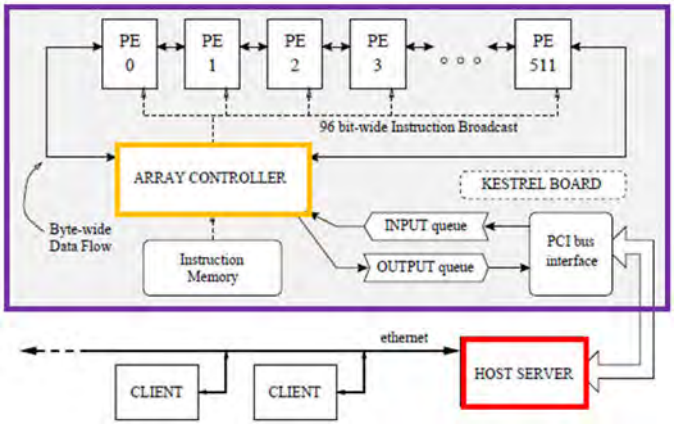
Prior to GPUs being used as "accelerators," it was well known that parallel processing systems (which a GPU is) were most effective with a "controller" that coordinated the function of the parallel processors, including transferring data and providing instructions for computations. As shown in the examples below, these "controllers" could be part of or separate from the accelerators themselves.



[158]

[157] The Asserted Claims of the '867 Patent do not specify whether the "accelerator controller" is part of the claimed "accelerator" or a separate component. In Figure 2 of the '867 Patent, controller 220 is part of accelerator 180. '867 Patent at Fig. 2. Either way, whether the "accelerator controller" is part of the "accelerator" or not, these limitations would have been obvious in view of the prior art cited in these Invalidity Contentions.

[158] NVIDIA-NAI-0003517 (Massively-Parallel SIMD Processor) at 845 (SIMD processor for neural network computation) (annotated).



159

These same principles had been applied where GPUs were used as accelerators, such as in the example below from Nickolls, where core interface 128 (yellow) (an "accelerator controller") controls operation of core 126 (an "accelerator"), including communicating with accelerator memory (blue) and providing instructions to the processing engines (red).[160]

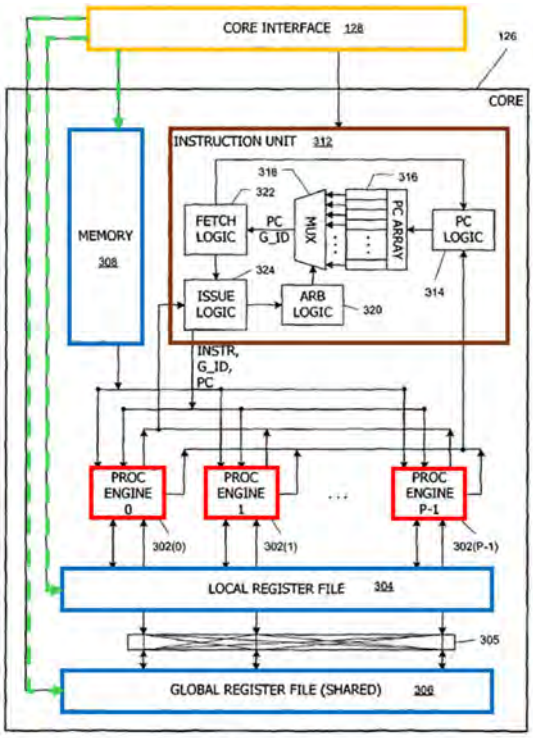


FIG. 3

[159] NVIDIA-NAI-0004674 (High-Performance SIMD) at 2 (annotated).

[160] NVIDIA-NAI-0004975 (Nickolls) at 6:47–55, Fig. 3 (annotated).

A POSITA would have been motivated to apply these principles to any of the primary references in Exhibits A1–A13. One of the primary known advantages of an accelerator controller was the offloading of tasks from the CPU to the GPU. GPUs are designed to handle parallel processing tasks more efficiently than CPUs, making them ideal for intensive computations such as graphics rendering, scientific simulations, and machine learning. By delegating the administration of these tasks to a controller separate from the CPU, the CPU is freed up to handle other critical system functions, leading to improved overall system performance and responsiveness.

Another motivation to using an "accelerator controller" is the enhanced efficiency in managing memory and data transfers. A controller dedicated to the accelerator can better optimize the flow of data between the CPU, GPU, and memory, reducing bottlenecks and ensuring that the GPU operates at peak efficiency. This is particularly important in high-performance computing applications where large datasets need to be processed quickly. A POSITA would have understood that with an accelerator controller separate from the CPU, there is better synchronization between the CPU and GPU, ensuring that tasks are executed in the correct order and that data integrity is maintained. This coordination is crucial for applications that require precise timing and accuracy, such as real-time simulations and complex data analysis.

In view of the knowledge demonstrated in the art in Exhibits A1–A13 and Appendix A, it would have been obvious for a POSITA to implement an "accelerator controller" as claimed, and he or she would have been motivated to do so to for the advantages described above.

(2) Storing Data in a Partition

Some limitations of the Asserted Claims of the '867 Patent relate to storing specific data (such as input or output data) in a "partition," including:

- "transferring, by an accelerator controller, the first input data into a first partition, referenced by first pointer, of an accelerator memory before a first computational cycle of the numerical simulation" (cl. 16);
- "storing, by the accelerator controller, the first output data into a second partition, referenced by a second pointer, of the accelerator memory" (cl. 16); and
- "partitioning the accelerator memory into the first partition, the second partition, a third partition to store internal variables, and a fourth partition to store data used as input at a particular computation cycle of the numerical simulation" (cl. 18).

To the extent that any of the primary references do not expressly disclose these or any related limitations, a POSITA would have been motivated to modify any such references based on any of the references in Exhibits A1–A13 and Appendix A disclosing those limitations.[161]

Prior to the '867 Patent, it was well known that memory could be sub-divided and partitioned, such that specific data is stored in a specific "partition" of memory. For example, references disclose storing data in "buffers" (which can be "partitions").[162] As another example, one reference describes "partition[ing] (physically or logically) [a frame buffer] into a number B of partitions 602, where B might be equal to or different from the number N of processing clusters 302. . . . Each cluster 302 can write pixel data to any one (or more) of the B partitions 602."[163]

---

[161] Plaintiff's Infringement Contentions apply a broad interpretation of "partition," apparently contending that any data stored in an addressable memory space is in its own "partition." *E.g.*, '867 Inf. Cont. Chart for Maxwell Architecture at 27–38. Under this interpretation of "partition," even primary reference would disclose the claimed "partitions."

[162] *E.g.*, NVIDIA-NAI-0002604 (GPU Gems 2: Programming Techniques for High-Performance Graphics and General-Purpose Computation (2005)) at 474 ("First, commands, textures, and vertex data are received from the host CPU through shared buffers in system memory or local frame-buffer memory."), 504–08, 511, 525–26.

[163] NVIDIA-NAI-0005033 (U.S. Patent No. 8,730,249) at 2:45–56, 11:19–12:37, Figs. 5, 6.

A POSITA would have been motivated to apply these principles to any of the primary references in Exhibits A1–A13. One of the primary benefits of a "partition" is the organization and isolation of data. By partitioning data, it becomes easier to manage and access specific datasets without interference from other data. This is particularly useful in complex systems where different types of data need to be processed simultaneously. Another advantage of using storing specific data in a partition is the improvement in performance and scalability. When data is partitioned, each partition can be processed independently, which allows for parallel processing. This means that multiple processors or threads can work on different partitions simultaneously, significantly speeding up data processing tasks.

In view of the knowledge demonstrated in the art in Exhibits A1–A13 and Appendix A, it would have been obvious for a POSITA to use the "partitions" as claimed, and he or she would have been motivated to do so to for the advantages described above.

(3) Referencing Data by a Pointer

Some limitations of the Asserted Claims of the '867 Patent relate to referencing data by a "pointer," including:

- "transferring, by an accelerator controller, the first input data into a first partition, referenced by first pointer, of an accelerator memory before a first computational cycle of the numerical simulation" (cl. 16);
- "storing, by the accelerator controller, the first output data into a second partition, referenced by a second pointer, of the accelerator memory" (cl. 16); and
- "swapping the first pointer with the second pointer at the end of the first computational cycle, such that the first output data becomes an input for a second computational cycle of the numerical simulation" (cl. 16).

To the extent that any of the primary references do not expressly disclose these or any related limitations, a POSITA would have been motivated to modify any such references based on any of the references in Exhibits A1–A13 and Appendix A disclosing those limitations.

As explained in Section II.A.4, it was well known prior to the '867 Patent (and, indeed, decade before in some programming languages) to use "pointers" for memory addresses, such that the pointers can be shared without copying the entirety of a memory location. As an example, the C programming language (released in the 1970s) has tools to handle memory use and access, including by using "pointers." In the C programming language, to pass an array (*i.e.*, data) to a function, programmers create a pointer variable which stores the address of the array in the memory, and then pass that address to the function.[164] This approach allows the function to read data from the space in memory directly and to write data back to that same space directly. Without using pointers, a system would have to copy into a memory location accessed by the function and then copy the resulting data back at the end of the function, which would require a lot of time and system resources. Fortran, another programming language, provided for pointers starting in 1990.[165]

A POSITA would have been motivated to apply these principles to any of the primary references in Exhibits A1–A13. As explained above and throughout these Invalidity Contentions, using pointers to share memory addresses provides significant efficiency and performance benefits. Pointers allow direct access to memory locations, enabling functions to manipulate data stored in memory without the need for copying it. This direct access reduces the number of read

[164] NVIDIA-NAI-0001112 (C Programming Language) at 86–116.

[165] NVIDIA-NAI-0002039 (Fortran Pointers) at 17–20 ("There are at least two reasons why pointers are needed. The most frequently mentioned reason is so that Fortran programmers can manage the memory required by their applications.").

and write operations, which are time consuming and resource intensive. By using pointers, programmers can pass the address of a variable or an array to a function, allowing the function to operate on the original data rather than a copy. This approach not only saves time but also conserves memory, as it eliminates the need to allocate additional space for duplicate data. Because pointers provide the address of data, rather than the value of the data, the use of pointers additionally ensures that the most up-to-date data is read and written. This approach maintains data coherency across the system by allowing access to the latest version of the data.

Pointers facilitate dynamic memory management, which is crucial for optimizing resource usage in complex applications. With pointers, programmers can allocate and deallocate memory as needed, ensuring that memory is used efficiently and that there is no wastage. This is particularly important in systems with limited memory resources. By using pointers for memory addresses in any of the primary references in Exhibits A1–A13, a POSITA would have achieved these known benefits, providing for efficiency in computations.

In view of the knowledge demonstrated in the art in Exhibits A1–A13 and Appendix A, it would have been obvious for a POSITA to use pointers to identify data in memory as claimed, and he or she would have been motivated to do so to for the advantages described above.

(4) Swapping Pointers

One limitation of the Asserted Claims of the '867 Patent relates to swapping pointers, such that output data from one computational cycle is used as input data for a next cycle:

- "swapping the first pointer with the second pointer at the end of the first computational cycle, such that the first output data becomes an input for a second computational cycle of the numerical simulation" (cl. 16).

To the extent that any of the primary references do not expressly disclose this or any related limitations, a POSITA would have been motivated to modify any such references based on any of the references in Exhibits A1–A13 and Appendix A disclosing those limitations.

Prior to the '867 Patent, it was well known that in many numerical simulations and computations, the output of one computational step is used as the input for the next step. For example, in a simulation of a physical system over time, the result (output) of one time step is fed into the simulation as an input for the next time step, creating a feedback loop that allows the model to evolve accurately over time. Copying output data into a new array for the next step can be slow and uses a lot of memory. A better option, known before the '867 Patent, is to avoid copying by swapping the pointers of the input and output arrays. Redirecting the pointer allows the simulation to continue without duplicating data, saving time and memory—essential in high-performance computing where resources are limited.

Programming techniques for swapping pointers existed in computer programming long before the '867 Patent. For example, one textbook relating to the C programming language teaches a "swap" function, which swaps *pointers* to values, rather than swapping the values themselves, for efficiency.

> The way to obtain the desired effect is for the calling program to pass *pointers* to the values to be changed:
>
> ```
> swap(
> ```
>
> Since the operator & produces the address of a variable, &a is a pointer to a. In swap itself, the parameters are declared as pointers, and the operands are accessed indirectly through them.
>
> ```
> void swap(int *px, int *py)  /* interchange *px and *py */
> {
     int temp;

     temp = *px;
     *px = *py;
     *py = temp;
> }
> ```
>
> [166]

[166] NVIDIA-NAI-0001112 (C Programming Language) at 88.

Prior to the '867 Patent, GPUs had been designed in ways to easily allow the above principles to be applied. For example, GPUs (such as NVIDIA's) allowed shaders to render results to a buffer (memory) rather than a display. It was known to use these buffers to perform iterative computations, using the output of one rendering step as the input to the next, until a final frame was ready to render to the display (in the case of graphics) or final result was passed back to a CPU and/or system memory (in the case of GPGPU). It was well understood that programmers could use these buffers to solve iterative numerical simulations with limited read-write cycles between CPU and GPU. One known way to accomplish this was through "render-to-texture," through which the results of a program (shader) are written to local texture memory "so they can be used as input to future programs."[167]

The concept of "pointer swapping" was well known in the graphics and GPGPU space prior to the '867 Patent as a way to leverage the output of one computation as the input to the next. For example, in the book GPU Gems 2, NVIDIA employee and author Mark Harris provides C++ code for simulating a reaction-diffusion system on a GPU that includes a "`swap()`" function at the end of each iteration that allows the output of one step of the simulation to become the input of the next step. The code defines one buffer as the input (`currentSource`) and another as the output (`currentTarget`). That is, the input data for a step is stored in a memory location pointed to by the variable `currentSource` and the output data is saved in a memory location pointed to by the variable `currentTarget`. At the end of computational cycle, the function `swap()` swaps the pointers for the `currentSource` buffer with that of the `currentTarget`

[167] NVIDIA-NAI-0002604 (GPU Gems 2: Programming Techniques for High-Performance Graphics and General-Purpose Computation (2005)) at 501–08.

buffer, so that the data pointed to by `currentTarget` becomes the input pointed to by `currentSource` for the next cycle.

Listing 31-2 *(continued)*. C++ Code to Set Up and Run a Reaction-Diffusion Simulation on the GPU

```
  // Initialize the state of the simulation with values in array
  pugInitBuffer(rdBuffer, array);
}

void update_rd(){ // Call this every iteration
  pugBindStream(rdProgram, "concentration", rdBuffer, currentSource);
  PUGRect range(0, width, 0, height);
  pugRunProgram(rdProgram, rdBuffer, range, currentTarget);

  std::swap(currentSource, currentTarget);
}
```

[168]

GPU Gems 2 contains several other examples of pointer swapping during numerical computations.[169] In this art, these techniques were sometimes known as "double buffering" or "ping-pong rendering," and had been applied to GPUs extensively.[170]

A POSITA would have been motivated to apply these principles to any of the primary references in Exhibits A1–A13. A POSITA would have understood the benefits of swapping pointers while performing numerical computations and would have been motivated to apply those techniques when writing programs to perform numerical computations on GPUs. As explained above, pointers allow processes to access data in the most efficient manner possible—using

[168] NVIDIA-NAI-0002604 (GPU Gems 2: Programming Techniques for High-Performance Graphics and General-Purpose Computation (2005)) at 501–08 (annotated).

[169] NVIDIA-NAI-0002604 (GPU Gems 2: Programming Techniques for High-Performance Graphics and General-Purpose Computation (2005)) at 44 ("Then, we swap the roles of the two textures and iterate the process . . . ."), 713 ("swap rho and newrho pointers" at the conclusion of each step of a "Conjugate Gradient Solver"), 743 ("To sort the entire field, we just call the next list and swap the buffers until we're done.").

[170] NVIDIA-NAI-0003831 (OpenGL 2.1 Reference) at 1; NVIDIA-NAI-0004483 (Advanced Image Processing) at 1, 18, 22–23; NVIDIA-NAI-0001953 (GPGPU Tutorials) at 17 (describing the "ping pong technique," in which the user "alternately use[s] the output of a given rendering pass as an input in the next one"); NVIDIA-NAI-0003913 (GPU Implementation of Neural Networks) at 323–24.

pointers eliminates the computational and communication overhead associated with copying, transmitting, and tracking the location of the data. By swapping pointers during computations, programmers can avoid the costly process of copying large datasets between memory locations. Instead, they simply redirect the pointer to the new data, allowing the computation to proceed without duplicating data. This approach saves time and minimizes memory usage, which is crucial in high-performance computing applications where processing power and memory are critical resources.

Moreover, pointer swapping enhances the overall system performance by reducing the overhead associated with data transfers. When data is copied, it incurs a performance cost due to the time and memory bandwidth required to duplicate potentially large datasets. By swapping pointers, this overhead is eliminated, allowing the system to operate more efficiently. This technique is particularly valuable in iterative computations, such as simulations of physical systems over time, where the output of one time step is fed into the next. Pointer swapping ensures that these computations can be performed quickly and accurately, without unnecessary delays or resource consumption.

Additionally, a POSITA would have been motivated to use pointer swapping because it was a well-established technique in the graphics and GPGPU space, where it is used to leverage the output of one computation as the input to the next. GPUs, such as those from NVIDIA, were designed to be able to apply these principles, using buffers to perform iterative computations with limited read-write cycles. Techniques like "render-to-texture" and "double buffering" allow efficient use of memory and processing power, making pointer swapping an attractive solution for a wide range of computing applications. Overall, the motivation to use pointer swapping in

numerical computations is driven by the need for improved performance, efficiency, and resource management.

In view of the knowledge demonstrated in the art in Exhibits A1–A13 and Appendix A, it would have been obvious for a POSITA to swap pointers at the end of a computational cycle, such that the output data becomes an input for another computational cycle as claimed, and he or she would have been motivated to do so to for the advantages described above.

(5) Transferring Data to Main Memory During a Computational Cycle

One limitation of the Asserted Claims of the '867 Patent relates to the timing of data transfer:

- "transferring, by the accelerator controller, the first output data to the main memory during the second computational cycle" (cl. 19).

To the extent that any of the primary references do not expressly disclose this or any related limitations, a POSITA would have been motivated to modify any such references based on any of the references in Exhibits A1–A13 and Appendix A disclosing those limitations.

Prior to the '867 Patent, it was known that in numerical simulations and other computer applications, a user did not need to preserve all information from all time steps. Rather, a user could elect when and how to store output data. A user running a numerical simulation would understand that runtime data is primarily stored in the processor's cache as long as it fits within the available cache size. If the cache capacity is exceeded, data is transferred to other levels of memory, such as RAM, which offers larger storage but with increased access latency. Since transferring data between different memory hierarchies incurs computational overhead in terms of time and resource management, the user must balance the need for storing output data with the cost of moving it. For example, the user might only require a steady-state solution at the final step

of the simulation, minimizing storage needs. Alternatively, he or she might need to track the evolution of a concentration profile over time, choosing an appropriate step size to achieve a desired accuracy. In some cases, a variable step-size method might be employed to maintain numerical stability, while still ensuring output is recorded at regular intervals. In each scenario, the user configures the simulation to store output selectively, discarding or overwriting intermediate results as necessary. Ultimately, the user must strike an appropriate balance between data retention, computational efficiency, and storage constraints, based on the specific objectives of the simulation.[171]

These considerations would have made it obvious to a POSITA to transfer output data to main memory during continued computational cycles. A POSITA would have been motivated to apply these principles to any of the primary references in Exhibits A1–A13. A POSITA would have understood that forcing processors to sit idle as they wait for another process to complete increases overall runtime, and that there were already well-known techniques for asynchronous data transfer. A POSITA would have further understood that users sometimes prefer to permanently store some intermediate as well as final results, for example to visualize how a physical process proceeds over time.

In view of the knowledge demonstrated in the art in Exhibits A1–A13 and Appendix A, it would have been obvious for a POSITA to transfer output data to main memory during a later computational cycle as claimed, and he or she would have been motivated to do so to for the advantages described above.

[171] NVIDIA-NAI-0003632 (Numerical Recipes in C) at 139–43 (showing a selection of step size according to tolerance for error and demand for speed).

(6) Performing Numerical Simulations on GPU Hardware

Some of the references described in these Invalidity Contentions provide programs and algorithms for numerical simulations, which are agnostic as to the GPU hardware on which they are executed. One such example is the diffusion-reaction simulation of GPU Gems 2.[172] Other references describe techniques such as "double buffering" or "ping-pong rendering" on GPUs, but again do not specify the specific GPU architecture on which those techniques are applied.[173] A POSITA would have been motivated to combine any of these GPU programming techniques with the GPU accelerator hardware disclosed in Exhibits A1–A13, many of which relate to NVIDIA GPUs. Indeed, a POSITA would have been motivated to implement GPU programming techniques on the most advanced GPUs at the time, which a POSITA would have known at the time were provided by NVIDIA.

Examples of the accelerator architectures on which these techniques could be applied are Nickolls,[174] Kirk,[175] NV40,[176] G70,[177] and NV50/G80[178]—all of which are from NVIDIA. These and the other GPU architectures charted provided advanced parallel processing capabilities and efficient data-sharing mechanisms. A POSITA would have understood that these systems would significantly reduce execution time compared to other processing architectures. Mathematical and

---

[172] NVIDIA-NAI-0002604 (GPU Gems 2: Programming Techniques for High-Performance Graphics and General-Purpose Computation (2005)) at 501–08.

[173] NVIDIA-NAI-0003831 (OpenGL 2.1 Reference) at 1; NVIDIA-NAI-0004483 (Advanced Image Processing) at 1, 18, 22–23; NVIDIA-NAI-0001953 (GPGPU Tutorials) at 17 (describing the "ping pong technique," in which the user "alternately use[s] the output of a given rendering pass as an input in the next one").

[174] *See* Exhibit A8.

[175] *See* Exhibit A9.

[176] *See* Exhibit A1.

[177] *See* Exhibit A2.

[178] *See* Exhibit A3.

scientific simulations often involve iterative calculations where results from one computation stage influence subsequent stages, and these systems disclose shared memory that would allow for the outputs of one stage of computations become inputs to subsequent stages.

Likewise, a POSITA would have understood the benefits of applying these GPU programming techniques, such as swapping pointers (as explained above). A POSITA would have understood that these combinations simply applied a known technique (*e.g.*, GPU programming techniques) to a known device (*e.g.*, GPU-accelerated parallel processing systems, such as that in Nickolls, Kirk, NV40, G70, NV50/G80, and the other primary references) ready for improvement to yield predictable results (*e.g.*, fast and efficient computation of complex numerical simulations).

c. Secondary Considerations of Non-Obviousness

A patentee bears the burden of production with respect to evidence of secondary considerations of non-obviousness. *ZUP, LLC v. Nash Mfg., Inc.*, 896 F.3d 1365, 1373 (Fed. Cir. 2018). As of the date of these Invalidity Contentions, Plaintiff has not identified any evidence of secondary considerations. NVIDIA reserves the right to present additional evidence and argument regarding secondary considerations of non-obviousness to the extent Plaintiff purports to identify any such evidence. In any event, as shown in these Invalidity Contentions, other companies and individuals described, built, and/or patented the exact same concepts in the '867 Patent before Plaintiff ever did—often many years before. Even where others' invention(s) occurred around the same time as the '867 Patent, such simultaneous invention demonstrates the obviousness of the '867 Patent.

4. Priority Date

U.S. Application No. 11/860,254 (hereinafter, "the '254 application"), which led to the '867 Patent, was filed on September 24, 2007. The '254 application purports to claim priority to provisional appl. no. 60/826,892 (hereinafter, "the '892 provisional"), which was filed on

September 25, 2006. To the extent that Plaintiff asserts that the '867 Patent is entitled to an earlier priority date based on any provisional application or documentation, Plaintiff has the burden of proof and has not demonstrated an earlier priority date.

a. Provisional Application

To the extent that Plaintiff asserts that the Asserted Claims of the '867 Patent are entitled to an earlier priority date based on the disclosure of the '892 provisional, Plaintiff has the burden of proof and has not demonstrated support for all claim limitations in the '892 provisional.

b. Asserted Conception, Reduction to Practice, and Diligence

In its Infringement Contentions, Plaintiff contends that the Asserted Claims of the '867 Patent "are entitled to a priority date corresponding to the conception of the claimed inventions, which occurred no later than February 25, 2005," and that "[t]he conception of the inventions . . . was followed by continuous diligence and actual reduction to practice of the claimed inventions no later than December 14, 2005." The documents that Plaintiff identifies in support of this contentions fail to demonstrate earlier conception, reduction to practice, or diligence.

The documents produced to date by Plaintiff fail to support earlier conception, let alone conception as early as February 25, 2005. Plaintiff's documents do not disclose several limitations of the Asserted Claims of the '867 Patent, including, without limitation, the "accelerator controller," referencing input data and output data by a "first pointer" and a "second pointer," respectively, and "swapping the first pointer with the second pointer at the end of the first computational cycle, such that the first output data becomes an input for a second computational cycle of the numerical simulation." For example, [redacted]

The documents produced to date also do not evidence an actual reduction to practice, let alone diligence from the alleged conception date until either an actual or constructive reduction to practice. The document produced by Plaintiff [redacted]

As such, the Asserted Claims of the '867 Patent are not entitled to an earlier priority date.

C. Prior Art Invalidity of U.S. Patent No. RE48,438

1. Identification of Prior Art

NVIDIA identifies the following prior art now known to NVIDIA that anticipate or render obvious one or more of the Asserted Claims of the '438 Patent. Each of the following patents, publications, systems, and products is prior art under at least one or more of 35 U.S.C. § 102(a), (b), (e), (f), or (g) (pre-AIA) or § 102(a) (post-AIA), as applicable. NVIDIA's reliance on each prior art reference below includes the reference itself, anything incorporated by the reference, and any testimony by those with knowledge of the reference, such as named authors and inventors.

a. Patents and Patent Publications

Patent / Publication No.	First Named Inventor	Date of Issue / Publication
Japanese Patent Appl. H11-232247	Takahashi Nakamoto	Aug. 27, 1999
Japanese Patent Publ. H04-237388	Yoichi Tamura	Aug. 25, 1992

Patent / Publication No.	First Named Inventor	Date of Issue / Publication
U.K. Patent Appl. No. 2291553	David Wilson	Jan. 24, 1996
U.S. Patent No. 6,853,382	James Van Dyke	Feb. 8, 2005
U.S. Patent No. 7,139,003	David Kirk	Nov. 21, 2006
U.S. Patent No. 7,219,085	Ian Buck	May 15, 2007
U.S. Patent No. 7,256,788	Viet-Tam Luu	August 14, 2007
U.S. Patent No. 7,339,592	John Lindholm	Mar. 4, 2008
U.S. Patent No. 7,418,576	John Lindholm	Aug. 26, 2008
U.S. Patent No. 7,673,304	Anuj Gosalia	Mar. 2, 2010
U.S. Patent No. 7,681,060	John Nickolls	Dec. 28, 2010
U.S. Patent No. 7,694,304	Kevin Kissell	Apr. 6, 2010
U.S. Patent No. 7,747,070	Siddhartha Puri	Jun. 29, 2010
U.S. Patent No. 7,830,392	John Danskin	Nov. 9, 2010
U.S. Patent No. 7,899,995	Radoslav Danilak	Mar. 1, 2011
U.S. Patent No. 8,059,131	Donald Bittel	Nov. 15, 2011
U.S. Patent No. 8,730,249	John Danskin	May 20, 2014
U.S. Patent No. 9,262,129	Michael Santori	Feb. 16, 2016
U.S. Patent Publ. No. 2003/0140179	Nicholas Wilt	Jul. 24, 2003
U.S. Patent Publ. No. 2003/0195938	Kevin Howard	Oct. 16, 2003
U.S. Patent Publ. No. 2005/0197977	Ian Buck	Sept. 8, 2005
U.S. Patent Publ. No. 2006/0098019	David Tarditi	May 11, 2006

b. Printed Publications

Citation	Publication Date
A. Gorchetchnikov, *KInNeSS: A new software environment for simulations of neuronal activity*	2004
A. Lefohn, *GPU Memory Model Overview*	2005
A. Veen, *Dataflow Machine Architecture*	1986
B. Kernigham et al., The C Programming Language	1988
B. Khailany, *Exploring the VLSI Scalability of Stream Processors*	2003
B. Khailany, *The VLSI Implementation and Evaluation of Area- and Energy-Efficient Streaming Media Processors*	2003
B. Svensson, *SIMD Architectures*	1992
C. Davis, *Graphics Processing Unit Computation of Neural Networks*	2005

Citation	Publication Date
Cg Toolkit: User's Manual	2005
D. Steinkraus, *Using GPUs for Machine Learning Algorithms*	2005
Dominik Göddeke -- GPGPU::Basic Math Tutorial	2005
E. Chu, *FFT algorithms and their adaptation to parallel processing*	1998
E. Kilgariff, *The GeForce 6 Series GPU Architecture*	2006
Game Programming Gems 4	2004
General Purpose Computation on Graphics Hardware	2005
GPU Gems 2: Programming Techniques for High-Performance Graphics and General-Purpose Computation	2005
H. Demuth, *Neural Network Toolbox for Use with MATLAB*	2004
H. Esmaeilzadeh, *Neural Network Stream Processing Core (NnSP) for Embedded Systems*	2006
H. Esmaeilzadeh, *NnSP: Embedded Neural Network Stream Processor*	2005
I. Buck et al., *Brook for GPUs: Stream Computing on Graphics Hardware* (presentation)	2004
I. Buck et al., *Brook for GPUs: Stream Computing on Graphics Hardware*	2004
I. Buck et al., *Data Parallel Computation on Graphics Hardware*	2003
I. Buck, *Brook for GPUs*	2003
I. Buck, *Stream Computing on Graphics Hardware*	2005
J. Ahn, *Scatter-Add in Data Parallel Architecture*	2005
J. Chen, *Fast Fourier Transform with BrookGPU*	2006
J. Fung, *Computer Vision Signal Processing on Graphics Processing Units*	2004
J. Fung, *OpenVIDIA: Parallel GPU Computer Vision*	2005
J. Georgii, *Interactive Simulation of Deformable Bodies on GPUs*	2005
J. Leiterman, *Learn Vertex and Pixel Shader Programming with DirectX 9*	2004
J. Martin, *Fortran 90 Pointers vs. "Cray" Pointers*	1991
J. Montrym, *The GeForce 6800*	2005
J. Ohmer, *Implementation of Kernel Methods on the GPU*	2005
J. Owens, *A Survey of General-Purpose Computation on Graphics Hardware*	2005
K. Jung, *Neural network-based text location in color images*	2001
K. Moreland, *The FFT on a GPU*	2003
K. Oh, *GPU implementation of neural networks*	2004

Citation	Publication Date
M. Botnen et al., *Using Commodity Graphics Hardware for Medical Image Segmentation*	2005
M. Erez, *Analysis and Performance Results of a Molecular Modeling Application on Merrimac*	2004
M. Erez, *Fault Tolerance Techniques for the Merrimac Streaming Supercomputer*	2005
M. Flynn, *Some Computer Organizations and Their Effectiveness*	1972
M. Glover et al., *A Massively-Parallel SIMD Processor for Neural Network and Machine Vision Applications*	1993
M. Harris, *GPGPU: General-Purpose Computation on GPUs*	2004
M. Harris, *Mapping Computational Concepts to GPUs*	2005
M. Harris, *Real-Time Cloud Simulation and Rendering*	2003
M. Kilgard, *A Follow-up Cg Runtime Tutorial for Readers of The Cg Tutorial*	2005
M. Kilgard, *Cg – A System for Programming Graphics Hardware in a C-like Language*	2006
M. Kilgard, *Cg and NVIDIA*	2006
M. Kilgard, *Cg in Two Pages*	2003
M. Kilgard, *NVIDIA OpenGL 2.0 Support*	2005
M. Kilgard, *Re-implementing the Follow-up Cg Runtime Tutorial with CgFX*	2006
M. Segal, *The OpenGL Graphics System: A Specification*	2004
MATLAB: The Language of Technical Computing	2005
Merrimac: The Streaming Supercomputer	2003
NVIDIA GPU Programming Guide (Version 2.4.0)	2005
OpenGL Super Bible	1996
P. Lisboa, *A review of evidence of health benefit from artificial neural networks in medical intervention*	2002
R. Agarwal, *A High Performance Parallel Algorithm for 1-D FFT*	1994
R. Hughey, *The UCSC Kestrel high performance SIMD processor: present and future*	2001
R. Rost, *OpenGL Shading Language*	2004
S. Rixner, *Stream Processor Architecture*	2001
S. Vitabile, *Real-Time Road Signs Recognition on a SIMD Architecture*	2004
The Cg Tutorial: The Definitive Guide to Programmable Real-Time Graphics	2003

Citation	Publication Date
U. Kapasi, *Programmable Stream Processors*	2003
U. Kapasi, *The Imagine Stream Processor*	2002
W. Dally, *Merrimac: Supercomputing with Streams*	2003
W. Press, *Numerical Recipes in C* (2d ed.)	2002
W. Thies, *StreamIt: A Language for Streaming Applications*	2002
Y Liu et al., *Real-Time 3D Fluid Simulation on GPU with Complex Obstacles*	2004
Z. Fan, *GPU Cluster for High Performance Computing*	2004
Z. Luo, *Artificial Neural Network Computation on Graphic Process Unit*	2005

c. Prior Art Products and Systems

Product / System
NVIDIA – NV40 (*e.g.*, GeForce 6800)
NVIDIA – G70 (*e.g.*, GeForce 7900)
NVIDIA – NV50/G80
BrookGPU-Merrimac ("Brook")
NVIDIA – Cg
The Imagine Stream Processor
SANNDRA / KInNeSS

d. Prior Invention Under § 102(g)

Prior Invention	Inventor / Assignee
NV50/G80	NVIDIA

Additionally, as explained in Section I.C, Plaintiff's Infringement Contentions are deficient and fail to provide sufficient allegations as to how the accused products infringe each and every claim limitation of the '438 Patent. Indeed, for most limitations, Plaintiff's Infringement Contentions are vague and unclear as to which particular functionality Plaintiff contends satisfies a claim limitation, prejudicing NVIDIA's ability to fully investigate invalidity under Plaintiff's theories of infringement.

While NVIDIA does not agree that the functionalities accused of infringement in Plaintiff's Infringement Contentions meet the claim limitations, the functionalities that Plaintiff apparently accuses of infringement were in development prior to the '438 Patent. Thus, to the extent those functionalities infringe the '438 Patent, they also invalidate it. *Peters v. Active Mfg. Co.*, 129 U.S. 530, 547 (1889) ("That which infringes, if later, would anticipate, if earlier."). For example, as explained in Section II.D.6, CUDA development at NVIDIA had been going on long before the priority date of the Asserted Patents, including functionalities from prior art systems like Merrimac/Brook and Cg, as discussed above. CUDA was built on the foundations laid by earlier programming languages such as Brook and Cg, and CUDA-related development documents show functionalities that Plaintiff may accuse of infringement as having existed prior to any asserted or actual priority date for the '438 Patent. For example, libraries such as cuFFT and cuBLAS were developed prior to the priority date of the '438 Patent. The following exemplary categories of documents evidence prior invention and development by NVIDIA:

- CUDA Internal Programming Guides
- CUDA Language Specifications
- CUDA API Specifications and Documentation
- CUDA Driver Compiler Internal Documentation
- cuFFT Internal Documentation
- cuBLAS Internal Documentation

NVIDIA reserves the right to rely on these and any other documents showing NVIDIA's prior invention. NVIDIA further incorporates by reference its charts on Brook/Merrimac and Cg, which (as explained above) were precursors to the accused CUDA functionality developed at NVIDIA.

Thus, to the extent Plaintiff clarifies its infringement allegations by identifying and explaining how specific CUDA or other functionality allegedly meets the claim limitations, NVIDIA reserves the right to demonstrate that the same functionality is prior art under 35 U.S.C. § 102(g)(2).

2. Anticipation

NVIDIA has identified the following prior art known to NVIDIA as anticipating the Asserted Claims of the '438 Patent. Each of the references cited below anticipates the Asserted Claims, either expressly or inherently as understood by a person of ordinary skill in the art, depending on the Court's interpretation of the claims. In some instances, NVIDIA has treated certain prior art as anticipatory, where certain elements are present based on at least Plaintiff's apparent claim construction in its Infringement Contentions. Exhibits B1–B12 are charts setting forth where in the prior art references each element of the Asserted Claims of the '438 Patent are found. Where NVIDIA charts a prior art product or system based on patents and/or printed publications related to that product/system, NVIDIA reserves the right to rely on such patents and/or printed publications alone as anticipatory references to the Asserted Claims of the '438 Patent.

Exhibit	Reference
B1	NV40
B2	G70
B3	NV50/G80
B4	Cg
B5	Brook
B6	Imagine Stream Processor
B7	SANNDRA / KInNeSS
B8	U.S. Patent No. 7,681,060 (Nickolls)
B9	U.S. Patent No. 7,139,003 (Kirk)

Exhibit	Reference
B10	U.S. Patent No. 7,219,085 (Buck '085)
B11	U.S. Patent No. 7,747,070 (Puri)
B12	U.S. Patent Publ. No. 2005/0197977A1 (Buck '977)

3. Obviousness

To the extent that Plaintiff argues that any of the above references do not anticipate the Asserted Claims of the '438 Patent, the claims are invalid as obvious under 35 U.S.C. § 103 in view of these references alone, in combination with the knowledge of a POSITA, and in combination with other references as described below. Exhibits B1–B12 and Appendix B further identify specific examples of art that alone or in combination render obvious the limitations of the Asserted Claims of the '438 Patent. Additional evidence of obviousness can be found in the reference identified in the section identifying prior art patents, publications, and systems.

While some of the above identified prior art may anticipate the asserted claims, this presents no obstacle to also relying on this reference as a basis for invalidity based on obviousness. It is established law that "a rejection for obviousness under § 103 can be based on a reference which happens to anticipate the claimed subject matter." *In re Meyer*, 599 F.2d 1026, 1031 (C.C.P.A. 1979). And to the extent any prior art item cited herein may not fully disclose a limitation of an asserted claim or is alleged by Plaintiff to lack disclosure of the limitation, such limitation is present and identified in another prior art item as shown in the attached claim charts. It would have been obvious to combine the disclosed claim elements based on the teachings of other art in existence at that time, the general knowledge of a person of ordinary skill in the art, and the recognition of similarities between the art and the claimed features. On such basis, on an element-by-element basis, NVIDIA expressly intends to combine one or more prior art items identified herein with each other to address any further contentions from Plaintiff that a particular

prior art item supposedly lacks one or more elements of an asserted claim. In other words, NVIDIA contends that each charted prior art item can be combined with other prior art items if a particular prior art item lacks or does not explicitly disclose an element or feature of an asserted claim.

a. Level of Ordinary Skill in the Art

A person of ordinary skill in the art at the time of the earliest effective filing date of the '438 Patent would have had a Bachelor's degree in electrical or computer engineering (or equivalent discipline) and at least two years' experience in research, design and/or development of computer systems, including experience with GPUs. Additional education could substitute for experience, and vice versa.

b. Scope and Content of the Prior Art

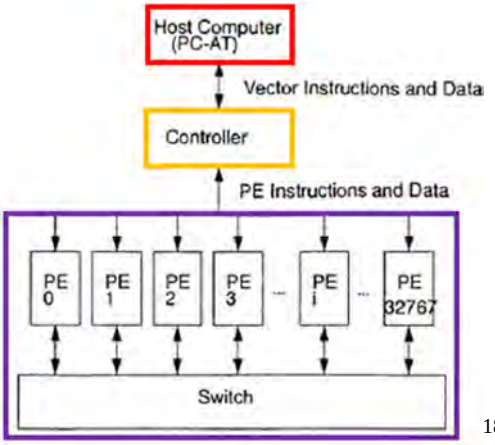
The '438 Patent is directed to a system for performing general-purpose computing (such as simulating an artificial neural network) on traditionally special-purpose GPUs. The '438 Patent does not claim to have invented performing general-purpose computing on GPUs or any specialized hardware for doing so.[179] Instead, the patent claims to have invented a novel architecture that offloads the setup and control of GPU computations from the host CPU to a "controller" in an "accelerator," freeing the CPU for other tasks.[180] The claims also focus on well-known applications and features of using GPUs for general computations, including (1) performing computations "representing an artificial neural network," and (2) transferring input and output data between the accelerator and the host system while the GPU performs computations in layers of the artificial neural network. As explained throughout these Invalidity Contentions, including in

[179] '438 Patent at 1:30–47, 14:38–42.

[180] *Id.* at 2:36–41.

Exhibits B1–B12 and Appendix B, all of these concepts were well known by those skilled in the art prior to the '438 Patent.

As explained in Section II.B.1 (incorporated herein by reference), prior to the '438 Patent, those skilled in the art had created effective designs for parallel processing. One common feature was a "controller" to coordinate the function of the parallel processors. These systems included a host computer or processor (red), an accelerator (*e.g.*, a series of processors) (purple), and a controller (orange) separate from the CPU. It was well known that having the controller, rather than the CPU, manage the processors frees up the CPU to perform other functions—one of the benefits the inventors allege for the "controller" in the '438 Patent.[181] Indeed, it was understood that one benefit of GPGPU applications was that, with the architecture described above and with calculations occurring in parallel on the GPU, they "free up the CPU for other tasks."[182]

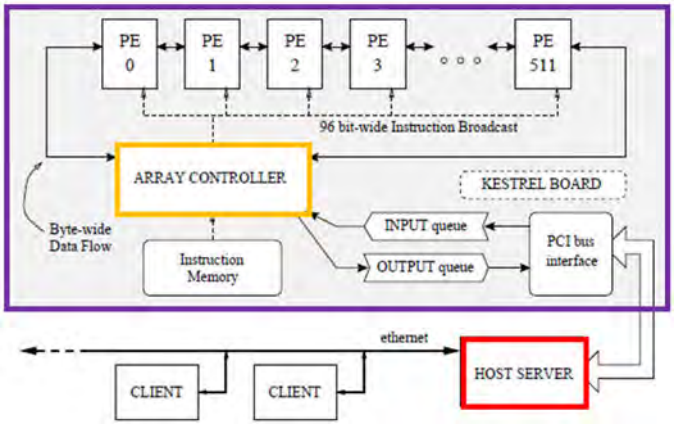


[183]

[181] '438 Patent at 2:36–40, 14:5–13.

[182] NVIDIA-NAI-0002604 (GPU Gems 2: Programming Techniques for High-Performance Graphics and General-Purpose Computation (2005)) at 650.

[183] NVIDIA-NAI-0003517 (Massively-Parallel SIMD Processor) at 845 (SIMD processor for neural network computation) (annotated).



[184]

The idea of leveraging the parallel processing abilities of GPUs to perform "computationally expensive algorithms,"[185] such as artificial neural networks, was well known prior to the '438 Patent. For example, prior art article *Artificial Neural Network Computation on Graphic Process Unit* describes using a multi-layer artificial neural network on an NVIDIA GPU to track a ball in real time for robot soccer.[186] Similarly, prior art article *GPU implementation of neural networks* describes using a multi-layer artificial neural network on a GPU to identify objects in video data.[187] As explained in Section II.AB.5, these are only some of the many applications of artificial neural networks on GPUs known prior to the '438 Patent.

As explained in Section II.B.4.b, it was also well known in the prior art to manage data transfer between a host system and an accelerator. One example of this is Tamura, which teaches a system for simulating a neural network on a dedicated process. Tamura recognized that "neural network calculations require a lot of processing time because they involve a very large amount of computation," and the neural network processors of the prior art were inefficient because they would sequentially input data, perform calculations, input further data, and so forth. To solve this

[184] NVIDIA-NAI-0004674 (High-Performance SIMD) at 2 (annotated).

[185] '438 Patent at 1:38–42.

[186] NVIDIA-NAI-0000919 (Artificial Neural Network Computation).

[187] NVIDIA-NAI-0003913 (GPU Implementation of Neural Networks).

problem, Tamura proposes inputting data and writing it to memory while calculations are being performed on previously inputted data. By carrying out the "data input operation in parallel with calculations," "the total processing time is reduced."[188]

c. Motivations to Combine Prior Art

A person of ordinary skill in the art would have been motivated to combine the above identified prior art items. Here, the claimed elements of the '438 Patent are disclosed in the prior art, as established above in Exhibits B1–B12 and Appendix B. Further, any elements claimed in the Asserted Claims would have been obvious to a person of ordinary skill in the art and predictable solutions resulting from routine optimization. Certain of these rationales are discussed more specifically below. The fact that others are not discussed more specifically should not be interpreted as an admission or concession that it does not apply. To the contrary, the discussion below simply provides more explanation of these specific rationales.

As set forth below, it would have been obvious to one of person of ordinary skill in the art to combine the teachings of similar prior art references to render the Asserted Claims obvious. The references identified in the exemplary combinations set forth below are analogous prior art and for at least the reasons set forth below, a person of ordinary skill in the art would have had reasons for combining them. The various teachings, suggestions, and/or reasons to modify any of the references and/or to combine any two or more references come from various sources, including the prior art (specific and as a whole), common knowledge, common sense, predictability, expectations, industry trends, design incentives or need, market demand or pressure, market forces, the nature of the problem faced, and/or the knowledge possessed by one of ordinary skill in the art.

---

[188] NVIDIA-NAI-0003465 (Tamura) ¶¶ 26–27, 30, 35, Figs. 3, 4.

Furthermore, to the extent that Plaintiff argues any modification of a base reference is required to meet the claim elements, such modification was routine and predictable, within the skill of a person of ordinary skill in the art, and a person of ordinary skill in the art would have had a reasonable expectation of success in making such a combination.

The following are exemplary motivations to combine:

(1) Performing Operations by a "Controller"

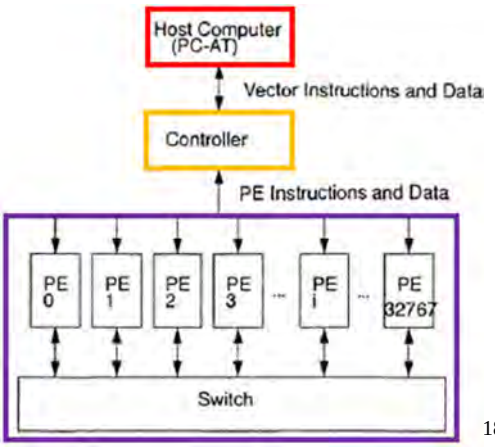
Some limitations of the Asserted Claims of the '438 Patent relate to operations performed by a "controller," including:

- "a controller, operably coupled to the at least one graphics processing unit and the accelerator memory, to initialize textures and shaders in the accelerator memory for performing the sequence of computations . . ." (cl. 1);
- "the controller is configured to transfer the output data from the accelerator memory to the main memory without transferring any of the intermediate results from the accelerator memory to the main memory so as to reduce data transfer via the bus" (cls. 7, 50);
- "the controller is configured to transfer at least a portion of the output data from the accelerator memory to the main memory after the at least one graphics processing unit has begun to perform another sequence of computations" (cls. 8, 51);
- "the controller is configured to initiate transfer of the at least a portion of the input data and to transfer the at least a portion of the output data in parallel with performance of at least one computation in the other sequence of computations by the at least one graphics processing unit" (cls. 9, 52);
- "the controller is configured to control execution of the sequence of computations by the at least one graphics processing unit" (cls. 10, 53);

- "initializing, by a controller operably coupled to a graphics processing unit (GPU), textures and shaders in a memory operably coupled to the GPU" (cl. 21);
- "a controller, operably coupled to the processing unit and the memory, to . . ." (cl. 40);
- "the controller is configured to send instructions for performing the at least one calculation to the processing unit" (cl. 42); and
- "a controller, operably coupled to the at least one processing unit and the accelerator memory, to control transfer . . ." (cl. 44).

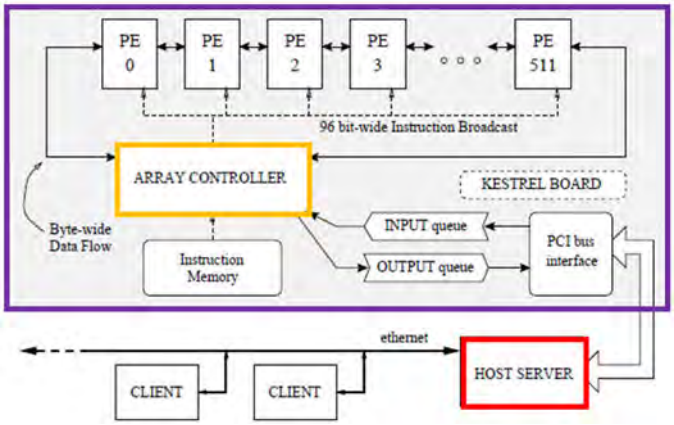
To the extent that any of the primary references do not expressly disclose these or any related limitations, a POSITA would have been motivated to modify any such references based on any of the references in Exhibits B1–B12 and Appendix B disclosing those limitations.

Prior to GPUs being used as "accelerators," it was well known that parallel processing systems (which a GPU is) were most effective with a "controller" that coordinated the function of the parallel processors, including transferring data and providing instructions for computations. As shown in the examples below, these "controllers" could be part of or separate from the accelerators themselves.



[189]

[189] NVIDIA-NAI-0003517 (Massively-Parallel SIMD Processor) at 845 (SIMD processor for neural network computation) (annotated).



190

These same principles had been applied where GPUs were used as accelerators, such as in the example below from Nickolls, where core interface 128 (yellow) (a "controller") controls operation of core 126 (an "accelerator"), including communicating with accelerator memory (blue) and providing instructions to the processing engines (red).[191]

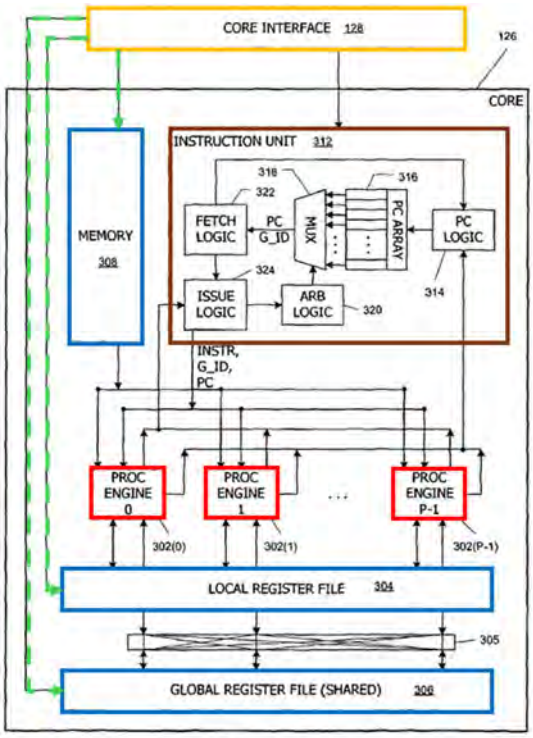


FIG. 3

[190] NVIDIA-NAI-0004674 (High-Performance SIMD) at 2 (annotated).

[191] NVIDIA-NAI-0004975 (Nickolls) at 6:47–55, Fig. 3 (annotated).

A POSITA would have been motivated to apply these principles to any of the primary references in Exhibits B1–B12. One of the primary known advantages of an accelerator controller was the offloading of tasks from the CPU to the GPU. GPUs are designed to handle parallel processing tasks more efficiently than CPUs, making them ideal for intensive computations such as graphics rendering, scientific simulations, and machine learning. By delegating the administration of these tasks to a controller separate from the CPU, the CPU is freed up to handle other critical system functions, leading to improved overall system performance and responsiveness.

Another motivation to using a "controller" is the enhanced efficiency in managing memory and data transfers. A controller dedicated to the accelerator can better optimize the flow of data between the CPU, GPU, and memory, reducing bottlenecks and ensuring that the GPU operates at peak efficiency. This is particularly important in high-performance computing applications where large datasets need to be processed quickly. A POSITA would have understood that with an accelerator controller separate from the CPU, there is better synchronization between the CPU and GPU, ensuring that tasks are executed in the correct order and that data integrity is maintained. This coordination is crucial for applications that require precise timing and accuracy, such as real-time simulations and complex data analysis.

In view of the knowledge demonstrated in the art in Exhibits B1–B12 and Appendix B, it would have been obvious for a POSITA to implement a "controller" as claimed, and he or she would have been motivated to do so to for the advantages described above.

(2) Transferring Data to Main Memory During a Computational Cycle

Some limitations of the Asserted Claims of the '438 Patent relate to the timing of data transfer, including between the CPU/main memory and accelerator/accelerator memory:

- "to transfer the at least a portion of the input data into the accelerator memory during performance of the intermediate computations in the sequence of computations by the at least one graphics processing unit" (cl. 1);
- "to transfer at least a portion of the output data from the accelerator memory to the main memory during performance of the intermediate computations in the sequence of computations by the at least one graphics processing unit" (cl. 1);
- "the controller is configured to transfer the output data from the accelerator memory to the main memory without transferring any of the intermediate results from the accelerator memory to the main memory so as to reduce data transfer via the bus" (cls. 7, 50);
- "the controller is configured to transfer at least a portion of the output data from the accelerator memory to the main memory after the at least one graphics processing unit has begun to perform another sequence of computations" (cls. 8, 51);
- "the controller is configured to initiate transfer of the at least a portion of the input data and to transfer the at least a portion of the output data in parallel with performance of at least one computation in the other sequence of computations by the at least one graphics processing unit" (cls. 9, 52);
- "(B) in parallel with performing the sequence of computations by the GPU in (A), transferring a second portion of the input data from the main memory to the accelerator via the bus" (cl. 12);
- "(C) in parallel with performing the sequence of computations by the GPU in (A), transferring a second portion of the output data from the accelerator memory to the main memory via the bus" (cl. 12);

- "transferring the second portion of the output data from the accelerator memory to the main memory after the GPU has begun to perform another sequence of computations" (cl. 17);
- "initiating transfer of the second portion of the output data in parallel with performance of at least one computation in the other sequence of computations" (cl. 18);
- "transferring second input data acquired from the external system in real time into the memory operably coupled to the GPU after the GPU starts the first computation and before the GPU starts a second computation of the sequence of computations" (cl. 21);
- "transferring the first output data from the memory to another memory during the second computation in the sequence of computations" (cl. 23);
- "transfer the first output data to another memory during computation of the second layer of neurons" (cl. 40);
- "to control transfer of the at least a portion of the input data into the accelerator memory during performance of the intermediate computations in the sequence of computations by the at least one processing unit" (cl. 44); and
- "to control transfer at least a portion of the output data from the accelerator memory to the main memory during performance of the intermediate computations in the sequence of computations by the at least one processing unit" (cl. 44).

To the extent that any of the primary references do not expressly disclose these or any related limitations, a POSITA would have been motivated to modify any such references based on any of the references in Exhibits B1–B12 and Appendix B disclosing those limitations.

With respect to transfer of input data (*e.g.*, from CPU/main memory to the accelerator/accelerator memory), prior to the '438 Patent, it was known that it was that additional input data should be transferred while computations occur on prior received data. This is especially

true with real-time input, such as from video. For example, Tamura relates to simulation of artificial neural networks and teaches inputting data and writing it to memory while calculations are being performed on previously inputted data. By carrying out the "data input operation in parallel with calculations," "the total processing time is reduced."[192] For real-time applications or other applications with significant data input, it would have been important for a POSITA to implement an accelerator (*e.g.*, GPU) to be able to simultaneously and continuously receive and process data. A POSITA would have been motivated to apply these principles to any of the primary references in Exhibits B1–B12. A POSITA would have looked to the art for solutions to ensure the ability to keep up with data input, with one such technique disclosed in Tamura. A POSITA would have understood that Tamura's teachings would be applicable to any GPU-accelerated system processing or simulating neural networks.

With respect to transfer of output data, prior to the '438 Patent, it was known that in numerical simulations and other computer applications, a user did not need to preserve all information from all time steps. Rather, a user could elect when and how to store output data. A user running a numerical simulation would understand that runtime data is primarily stored in the processor's cache as long as it fits within the available cache size. If the cache capacity is exceeded, data is transferred to other levels of memory, such as RAM, which offers larger storage but with increased access latency. Since transferring data between different memory hierarchies incurs computational overhead in terms of time and resource management, the user must balance the need for storing output data with the cost of moving it. For example, the user might only require a steady-state solution at the final step of the simulation, minimizing storage needs. Alternatively, he or she might need to track the evolution of a concentration profile over time,

[192] NVIDIA-NAI-0003465 (Tamura) ¶¶ 26–27, 30, 35, Figs. 3, 4.

choosing an appropriate step size to achieve a desired accuracy. In some cases, a variable step-size method might be employed to maintain numerical stability, while still ensuring output is recorded at regular intervals. In each scenario, the user configures the simulation to store output selectively, discarding or overwriting intermediate results as necessary. Ultimately, the user must strike an appropriate balance between data retention, computational efficiency, and storage constraints, based on the specific objectives of the simulation.[193]

These considerations would have made it obvious to a POSITA to transfer output data to main memory during continued computational cycles. A POSITA would have been motivated to apply these principles to any of the primary references in Exhibits B1–B12. A POSITA would have understood that forcing processors to sit idle as they wait for another process to complete increases overall runtime, and that there were already well-known techniques for asynchronous data transfer. A POSITA would have further understood that users sometimes prefer to permanently store some intermediate as well as final results, for example to visualize how a physical process proceeds over time.

In view of the knowledge demonstrated in the art in Exhibits B1–B12 and Appendix B, it would have been obvious for a POSITA to transfer output data to main memory during a later computational cycle as claimed, and he or she would have been motivated to do so to for the advantages described above.

(3) Simulating Artificial Neural Networks on GPU Hardware

Some limitations of the Asserted Claims of the '438 Patent relate to simulating an artificial neural network, including:

[193] NVIDIA-NAI-0003632 (Numerical Recipes in C) at 139–43 (showing a selection of step size according to tolerance for error and demand for speed).

- "at least one graphics processing unit to perform a sequence of computations on the at least a portion of the input data so as to generate output data, the sequence of computations representing an artificial neural network, intermediate computations in the sequence of computations representing respective layers of the artificial neural network and yielding intermediate results" (cl. 1);
- "an output of at least one computation in the sequence of computations represents an output of at least one neuron in an artificial neural network" (cl. 5);
- "[a] method of performing a sequence of computations representing an artificial neural network on a computer system" (cl. 12);
- "(A) performing, by the GPU, the sequence of computations on a first portion of input data so as to generate a first portion of output data, the first portion of the output data representing an output of a neuron in a first layer of the artificial neural network, intermediate computations in the sequence of computations yielding intermediate results" (cl. 12);
- "(C) in parallel with performing the sequence of computations by the GPU in (A), transferring a second portion of the output data from the accelerator memory to the main memory via the bus, the second portion of the output data representing an output of a neuron in a second layer in the artificial neural network" (cl. 12);
- "performing, by the graphics processing unit (GPU), a first computation in the sequence of computations on the first input data based on the textures and shaders to generate first output data, computations in the sequence of computations representing respective layers of neurons in the artificial neural network, an output of the first computation in the

sequence of computations representing an output of a first neuron in a first layer in the artificial neural network" (cl. 21);

- "transferring second input data acquired from the external system in real time into the memory operably coupled to the GPU after the GPU starts the first computation and before the GPU starts a second computation of the sequence of computations, an output of the second computation in the sequence of computations representing an output of a second neuron in a second layer in the artificial neural network" (cl. 21);
- "performing the first computation comprises executing a plurality of computational elements representing a layer of neurons in an artificial neural network" (cl. 31);
- "all neurons in the layer of neurons are described by the same equation" (cl. 32);
- "a memory, operably coupled to the CPU, to store the first input data in a first partition, referenced by a first pointer, before computing a first layer of neurons of the artificial neural network" (cl. 40);
- "store the first output data in a second partition of the memory, the second partition referenced by a second pointer, and to swap the first pointer with the second pointer at the end of the computation of the first layer of neurons, such that the first output data becomes an input for a second layer of neurons of the artificial neural network" (cl. 40);
- "a fourth partition to store data used as input at a particular layer of neurons of the artificial neural network" (cl. 43);
- "at least one processing unit to perform a sequence of computations representing an artificial neural network on the at least a portion of the input data so as to generate output data, intermediate computations in the sequence of computations representing layers of the neural network and yielding intermediate results" (cl. 44); and

- "an output of at least one computation in the sequence of computations represents an output of at least one neuron in an artificial neural network" (cl. 48).

To the extent that any of the primary references do not expressly disclose these or any related limitations, a POSITA would have been motivated to modify any such references based on any of the references in Exhibits B1–B12 and Appendix B disclosing those limitations.

As explained in Section II.B.5, one significant application that arose from early GPGPU advancements was the ability to simulate artificial neural networks on GPUs.[194] Prior to the Asserted Patents, it was known that the parallel processing power of GPUs was crucial for efficiently training these networks, as GPUs are particularly well-suited for the matrix operations and numerical calculations required for neural network training.[195] For example, before the Asserted Patents, GPUs were already being utilized to power artificial neural networks in a variety of applications, including tasks such as image and pattern recognition,[196] object detection,[197] medical diagnostics,[198] and checkers.[199] For example, prior art article *Artificial Neural Network Computation on Graphic Process Unit* describes using a multi-layer artificial neural network on an NVIDIA GPU to track a ball in real time for robot soccer.[200] Similarly, prior art article *GPU*

[194] NVIDIA-NAI-0003913 (GPU Implementation of Neural Networks) at 322–24; NVIDIA-NAI-0000919 (Artificial Neural Network Computation) at 622.

[195] NVIDIA-NAI-0003913 (GPU Implementation of Neural Networks) at 322–23 ("[M]any inner product operations can be replaced with matrix multiplication, which is more appropriate for GPU implementation.").

[196] NVIDIA-NAI-0003913 (GPU Implementation of Neural Networks).

[197] NVIDIA-NAI-0000919 (Artificial Neural Network Computation) at 622.

[198] NVIDIA-NAI-0000859 (ANN in Medical Intervention).

[199] NVIDIA-NAI-0003443 (Game Programming Gems) at 6.

[200] NVIDIA-NAI-0000919 (Artificial Neural Network Computation).

implementation of neural networks describes using a multi-layer artificial neural network on a GPU to identify objects in video data.[201]

It would have been obvious, and a POSITA would have been motivated, to implement and simulate an artificial neural network on the GPU-accelerator architectures described in Exhibits B1–B12. Some of the references described in these Invalidity Contentions provide programs and algorithms for simulating artificial neural networks, which are agnostic as to the GPU hardware on which they are executed. A POSITA would have been motivated to implement these neural networks on the GPU accelerator hardware disclosed in Exhibits B1–B12, many of which relate to NVIDIA GPUs. Indeed, a POSITA would have been motivated to implement GPU programming techniques on the most advanced GPUs at the time, which a POSITA would have known at the time were provided by NVIDIA.

Examples of the accelerator architectures on which these techniques could be applied are Nickolls,[202] Kirk,[203] NV40,[204] G70,[205] and NV50/G80[206]—all of which are from NVIDIA. These and the other GPU architectures charted provided advanced parallel processing capabilities and efficient data-sharing mechanisms, which are beneficial for artificial neural networks. A POSITA would have understood that these systems would significantly reduce execution time compared to other processing architectures. Mathematical and scientific simulations (including artificial neural networks) often involve iterative calculations where results from one computation stage influence

[201] NVIDIA-NAI-0003913 (GPU Implementation of Neural Networks).

[202] *See* Exhibit B8.

[203] *See* Exhibit B9.

[204] *See* Exhibit B1.

[205] *See* Exhibit B2.

[206] *See* Exhibit B3.

subsequent stages, and these systems disclose shared memory that would allow for the outputs of one stage of computations become inputs to subsequent stages.

A POSITA would have understood that these combinations simply applied a known technique (*e.g.*, simulation of artificial neural networks) to a known device (*e.g.*, GPU-accelerated parallel processing systems, such as that in Nickolls, Kirk, NV40, G70, NV50/G80, and the other primary references) ready for improvement to yield predictable results (*e.g.*, fast and efficient computation). Furthermore, to the extent any of the primary references do not discuss the structure of an artificial neural network (such as its "layers" and "neurons"), those fundamental features of artificial neural networks were well known and understood prior to the '438 Patent, as explained in Section II.B.5.

(4) Storing Data in a Partition

Some limitations of the Asserted Claims of the '438 Patent relate to storing specific data (such as input or output data) in a "partition" or "memory bank," including:

- "a first memory bank to store parameters common to all of the computations in the sequence of computations" (cl. 6);
- "a second memory bank to store data specific to at least one computation in the sequence of computations" (cl. 6);
- "storing parameters common to all of the computations in the sequence of computations in a first memory bank in the accelerator memory" (cl. 20);
- "storing data specific to at least one computation in the sequence of computations in a second memory bank in the accelerator memory" (cl. 20);
- "storing, in a first memory partition of the memory, parameters common to all of the computations in the sequence of computations" (cl. 26);

- "storing, in a second memory partition of the memory, data specific to the first computation in the sequence of computations" (cl. 27);
- "storing, in the second memory partition, external input data patterns, representations of internal variables, an input of the computation in the sequence of computations, and the output of the computation in the sequence of computations" (cl. 28);
- "a memory, operably coupled to the CPU, to store the first input data in a first partition, referenced by a first pointer, before computing a first layer of neurons of the artificial neural network" (cl. 40);
- "store the first output data in a second partition of the memory, the second partition referenced by a second pointer, and to swap the first pointer with the second pointer at the end of the computation of the first layer of neurons, such that the first output data becomes an input for a second layer of neurons of the artificial neural network" (cl. 40);
- "a third partition to store internal variables" (cl. 43);
- "a fourth partition to store data used as input at a particular layer of neurons of the artificial neural network" (cl. 43);
- "a first memory partition to store parameters common to all of the computations in the sequence of computations" (cl. 49); and
- "a first memory partition to store parameters common to all of the computations in the sequence of computations" (cl. 49).

To the extent that any of the primary references do not expressly disclose these or any related limitations, a POSITA would have been motivated to modify any such references based on any of the references in Exhibits B1–B12 and Appendix B disclosing those limitations.[207]

Prior to the '438 Patent, it was well known that memory could be sub-divided and partitioned, such that specific data is stored in a specific "partition" of memory (which may also be referred to as a "memory bank"). For example, references disclose storing data in "buffers" (which can be "partitions").[208] As another example, one reference describes "partition[ing] (physically or logically) [a frame buffer] into a number B of partitions 602, where B might be equal to or different from the number N of processing clusters 302. . . . Each cluster 302 can write pixel data to any one (or more) of the B partitions 602."[209]

A POSITA would have been motivated to apply these principles to any of the primary references in Exhibits B1–B12. One of the primary benefits of a "partition" is the organization and isolation of data. By partitioning data, it becomes easier to manage and access specific datasets without interference from other data. This is particularly useful in complex systems where different types of data need to be processed simultaneously. Another advantage of using storing specific data in a partition is the improvement in performance and scalability. When data is partitioned, each partition can be processed independently, which allows for parallel processing.

---

[207] Plaintiff's Infringement Contentions apply a broad interpretation of "partition," apparently contending that any data stored in an addressable memory space is in its own "partition." *E.g.*, '867 Inf. Cont. Chart for Maxwell Architecture at 27–38. Under this interpretation of "partition," even primary reference would disclose the claimed "partitions."

[208] *E.g.*, NVIDIA-NAI-0002604 (GPU Gems 2: Programming Techniques for High-Performance Graphics and General-Purpose Computation (2005)) at 474 ("First, commands, textures, and vertex data are received from the host CPU through shared buffers in system memory or local frame-buffer memory."), 504–08, 511, 525–26.

[209] NVIDIA-NAI-0005033 (U.S. Patent No. 8,730,249) at 2:45–56, 11:19–12:37, Figs. 5, 6.

This means that multiple processors or threads can work on different partitions simultaneously, significantly speeding up data processing tasks.

In view of the knowledge demonstrated in the art in Exhibits B1–B12 and Appendix B, it would have been obvious for a POSITA to use the "partitions" or "memory banks" as claimed, and he or she would have been motivated to do so to for the advantages described above.

(5) Referencing Data by a Pointer

Some limitations of the Asserted Claims of the '438 Patent relate to referencing data by a "pointer," including:

- "a memory, operably coupled to the CPU, to store the first input data in a first partition, referenced by a first pointer, before computing a first layer of neurons of the artificial neural network" (cl. 40); and
- "store the first output data in a second partition of the memory, the second partition referenced by a second pointer, and to swap the first pointer with the second pointer at the end of the computation of the first layer of neurons, such that the first output data becomes an input for a second layer of neurons of the artificial neural network" (cl. 40).

To the extent that any of the primary references do not expressly disclose these or any related limitations, a POSITA would have been motivated to modify any such references based on any of the references in Exhibits B1–B12 and Appendix B disclosing those limitations.

As explained in Section II.A.4, it was well known prior to the '438 Patent (and, indeed, decade before in some programming languages) to use "pointers" for memory addresses, such that the pointers can be shared without copying the entirety of a memory location. As an example, the C programming language (released in the 1970s) has tools to handle memory use and access, including by using "pointers." In the C programming language, to pass an array (*i.e.*, data) to a

function, programmers create a pointer variable which stores the address of the array in the memory, and then pass that address to the function.[210] This approach allows the function to read data from the space in memory directly and to write data back to that same space directly. Without using pointers, a system would have to copy into a memory location accessed by the function and then copy the resulting data back at the end of the function, which would require a lot of time and system resources. Fortran, another programming language, provided for pointers starting in 1990.[211]

A POSITA would have been motivated to apply these principles to any of the primary references in Exhibits B1–B12. As explained above and throughout these Invalidity Contentions, using pointers to share memory addresses provides significant efficiency and performance benefits. Pointers allow direct access to memory locations, enabling functions to manipulate data stored in memory without the need for copying it. This direct access reduces the number of read and write operations, which are time consuming and resource intensive. By using pointers, programmers can pass the address of a variable or an array to a function, allowing the function to operate on the original data rather than a copy. This approach not only saves time but also conserves memory, as it eliminates the need to allocate additional space for duplicate data. Because pointers provide the address of data, rather than the value of the data, the use of pointers additionally ensures that the most up-to-date data is read and written. This approach maintains data coherency across the system by allowing access to the latest version of the data.

---

[210] NVIDIA-NAI-0001112 (C Programming Language) at 86–116.

[211] NVIDIA-NAI-0002039 (Fortran Pointers) at 17–20 ("There are at least two reasons why pointers are needed. The most frequently mentioned reason is so that Fortran programmers can manage the memory required by their applications.").

Pointers facilitate dynamic memory management, which is crucial for optimizing resource usage in complex applications. With pointers, programmers can allocate and deallocate memory as needed, ensuring that memory is used efficiently and that there is no wastage. This is particularly important in systems with limited memory resources. By using pointers for memory addresses in any of the primary references in Exhibits B1–B12, a POSITA would have achieved these known benefits, providing for efficiency in computations.

In view of the knowledge demonstrated in the art in Exhibits B1–B12 and Appendix B, it would have been obvious for a POSITA to use pointers to identify data in memory as claimed, and he or she would have been motivated to do so to for the advantages described above.

(6) Swapping Pointers and Using Output Data as Inputs

Some limitations of the Asserted Claims of the '438 Patent relates to swapping pointers or otherwise making output data from one computational cycle available to be used as input data for a next cycle:

- "(ii) making the first value of the output variable in the first texture accessible to other computational elements in the plurality of computational elements during the second time step" (cl. 12);
- "accessing, by the GPU, the output of the previous computation during performance of the computation in the sequence of computations" (cl. 30); and
- "store the first output data in a second partition of the memory, the second partition referenced by a second pointer, and to swap the first pointer with the second pointer at the end of the computation of the first layer of neurons, such that the first output data becomes an input for a second layer of neurons of the artificial neural network" (cl. 40).

To the extent that any of the primary references do not expressly disclose this or any related limitations, a POSITA would have been motivated to modify any such references based on any of the references in Exhibits B1–B12 and Appendix B disclosing those limitations.

Prior to the '438 Patent, it was well known that in many numerical simulations and computations, the output of one computational step is used as the input for the next step. For example, in a simulation of a physical system over time, the result (output) of one time step is fed into the simulation as an input for the next time step, creating a feedback loop that allows the model to evolve accurately over time. This is implemented commonly with artificial neural networks, where the computational results at one layer (outputs) are used as inputs for the next layer of neurons. Copying output data into a new array for the next step can be slow and uses a lot of memory. A better option, known before the '438 Patent, is to avoid copying by swapping the pointers of the input and output arrays. Redirecting the pointer allows the simulation to continue without duplicating data, saving time and memory—essential in high-performance computing where resources are limited.

Programming techniques for swapping pointers existed in computer programming long before the '438 Patent. For example, one textbook relating to the C programming language teaches a "swap" function, which swaps *pointers* to values, rather than swapping the values themselves, for efficiency.

The way to obtain the desired effect is for the calling program to pass *pointers* to the values to be changed:

```
swap(
```

Since the operator & produces the address of a variable, &a is a pointer to a. In swap itself, the parameters are declared as pointers, and the operands are accessed indirectly through them.

```
void swap(int *px, int *py)  /* interchange *px and *py */
{
    int temp;

    temp = *px;
    *px = *py;
    *py = temp;
}
```

[212]

Prior to the '438 Patent, GPUs had been designed in ways to easily allow the above principles to be applied. For example, GPUs (such as NVIDIA's) allowed shaders to render results to a buffer (memory) rather than a display. It was known to use these buffers to perform iterative computations, using the output of one rendering step as the input to the next, until a final frame was ready to render to the display (in the case of graphics) or final result was passed back to a CPU and/or system memory (in the case of GPGPU). It was well understood that programmers could use these buffers to solve iterative numerical simulations with limited read-write cycles between CPU and GPU. One known way to accomplish this was through "render-to-texture," through which the results of a program (shader) are written to local texture memory "so they can be used as input to future programs."[213]

The concept of "pointer swapping" was well known in the graphics and GPGPU space prior to the '438 Patent as a way to leverage the output of one computation as the input to the next. For example, in the book GPU Gems 2, NVIDIA employee and author Mark Harris provides C++ code for simulating a reaction-diffusion system on a GPU that includes a "`swap()`" function at

[212] NVIDIA-NAI-0001112 (C Programming Language) at 88.

[213] NVIDIA-NAI-0002604 (GPU Gems 2: Programming Techniques for High-Performance Graphics and General-Purpose Computation (2005)) at 501–08.

the end of each iteration that allows the output of one step of the simulation to become the input of the next step. The code defines one buffer as the input (`currentSource`) and another as the output (`currentTarget`). That is, the input data for a step is stored in a memory location pointed to by the variable `currentSource` and the output data is saved in a memory location pointed to by the variable `currentTarget`. At the end of computational cycle, the function `swap()` swaps the pointers for the `currentSource` buffer with that of the `currentTarget` buffer, so that the data pointed to by `currentTarget` becomes the input pointed to by `currentSource` for the next cycle.

Listing 31-2 *(continued)*. C++ Code to Set Up and Run a Reaction-Diffusion Simulation on the GPU

```
  // Initialize the state of the simulation with values in array
  pugInitBuffer(rdBuffer, array);
}

void update_rd(){ // Call this every iteration
  pugBindStream(rdProgram, "concentration", rdBuffer, currentSource);
  PUGRect range(0, width, 0, height);
  pugRunProgram(rdProgram, rdBuffer, range, currentTarget);

  std::swap(currentSource, currentTarget);
}
```

[214]

GPU Gems 2 contains several other examples of pointer swapping during numerical computations.[215] In this art, these techniques were sometimes known as "double buffering" or "ping-pong rendering," and had been applied to GPUs extensively.[216]

---

[214] NVIDIA-NAI-0002604 (GPU Gems 2: Programming Techniques for High-Performance Graphics and General-Purpose Computation (2005)) at 501–08 (annotated).

[215] NVIDIA-NAI-0002604 (GPU Gems 2: Programming Techniques for High-Performance Graphics and General-Purpose Computation (2005)) at 44 ("Then, we swap the roles of the two textures and iterate the process . . . ."), 713 ("swap rho and newrho pointers" at the conclusion of each step of a "Conjugate Gradient Solver"), 743 ("To sort the entire field, we just call the next list and swap the buffers until we're done.").

[216] NVIDIA-NAI-0003831 (OpenGL 2.1 Reference) at 1; NVIDIA-NAI-0004483 (Advanced Image Processing) at 1, 18, 22–23; NVIDIA-NAI-0001953 (GPGPU Tutorials) at 17 (describing the "ping pong technique," in which the user "alternately use[s] the output of a given rendering pass as an input in the next one"); NVIDIA-NAI-0003913 (GPU Implementation of Neural Networks) at 323–24.

A POSITA would have been motivated to apply these principles to any of the primary references in Exhibits B1–B12. A POSITA would have understood the benefits of swapping pointers while performing numerical computations and would have been motivated to apply those techniques when writing programs to perform numerical computations on GPUs. As explained above, pointers allow processes to access data in the most efficient manner possible—using pointers eliminates the computational and communication overhead associated with copying, transmitting, and tracking the location of the data. By swapping pointers during computations, programmers can avoid the costly process of copying large datasets between memory locations. Instead, they simply redirect the pointer to the new data, allowing the computation to proceed without duplicating data. This approach saves time and minimizes memory usage, which is crucial in high-performance computing applications where processing power and memory are critical resources.

Moreover, pointer swapping enhances the overall system performance by reducing the overhead associated with data transfers. When data is copied, it incurs a performance cost due to the time and memory bandwidth required to duplicate potentially large datasets. By swapping pointers, this overhead is eliminated, allowing the system to operate more efficiently. This technique is particularly valuable in iterative computations, such as simulations of physical systems over time, where the output of one time step is fed into the next. Pointer swapping ensures that these computations can be performed quickly and accurately, without unnecessary delays or resource consumption.

Additionally, a POSITA would have been motivated to use pointer swapping because it was a well-established technique in the graphics and GPGPU space, where it is used to leverage the output of one computation as the input to the next. GPUs, such as those from NVIDIA, were

designed to be able to apply these principles, using buffers to perform iterative computations with limited read-write cycles. Techniques like "render-to-texture" and "double buffering" allow efficient use of memory and processing power, making pointer swapping an attractive solution for a wide range of computing applications. Overall, the motivation to use pointer swapping in numerical computations (such as simulations of artificial neural networks) is driven by the need for improved performance, efficiency, and resource management.

In view of the knowledge demonstrated in the art in Exhibits B1–B12 and Appendix B, it would have been obvious for a POSITA to swap pointers at the end of a computational cycle, such that the output data becomes an input for another computational cycle as claimed, and he or she would have been motivated to do so to for the advantages described above.

(7) Receiving Input in Response to User Interaction

Some limitations of the Asserted Claims of the '438 Patent relate to having a "user interaction" to cause data to be received, including:

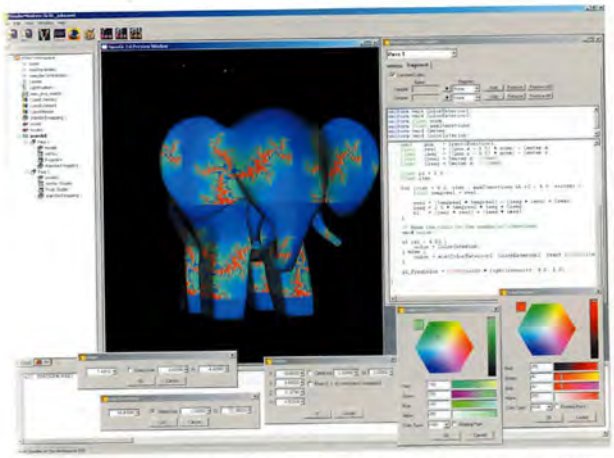
- "the central processing unit is configured to receive the input data in response to a user interaction" (cls. 2, 45); and
- "storing the input data in the main memory in response to a user interaction" (cl. 13).

To the extent that any of the primary references do not expressly disclose these or any related limitations, a POSITA would have been motivated to modify any such references based on any of the references in Exhibits B1–B12 and Appendix B disclosing those limitations.

As explained in Section II.B.3, prior to the '438 Patent, graphics hardware manufacturers such as NVIDIA developed improved software tools for programmable GPUs, such as advanced application programming interfaces (APIs) and programming languages. These tools facilitated interactions with GPU accelerators, allowing users to trigger when tasks are executed. For

example, programmers made use of NVIDIA's GPUs through graphics Application Programing Interfaces (APIs), such as Direct3D and OpenGL, as well as GPU languages such as HLSL, GLSL, and Cg.[217]

User interfaces, such as GUIs, were widely used before the '438 Patent, including for designing the simulation of artificial neural networks. As an example, nnet describes a "simple and user friendly" user interface through which users can create, edit, train, and simulate artificial neural networks.[218] Visual tools facilitated GPU programming and interacting with programs such as modeling and simulation programs. For example, the book *OpenGL Shading Language* shows a graphical user interface (GUI) that allowed a user to interactively adjust parameters used by the GPU for shading an object.



olor Plate 1 Screen shot of the RenderMonkey IDE user interface. The shader that is procedurally nerating a Julia set on the surface of the elephant is shown in the source code window. Color selection ols and user interface elements for manipulating user-defined uniform variables are also shown. [219]

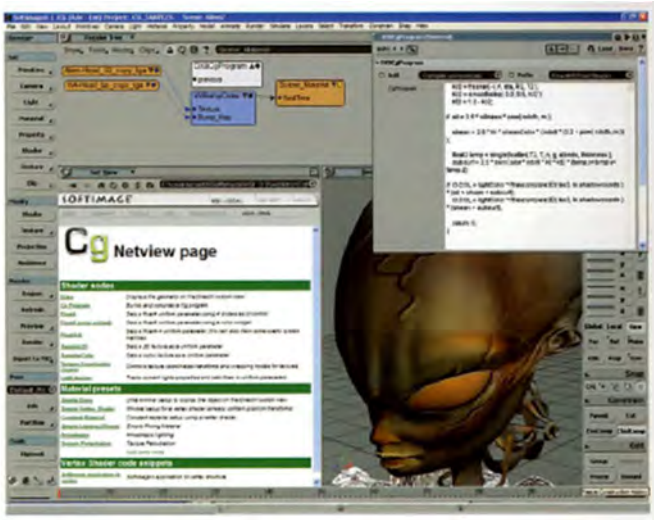
Similarly, the book *Cg: The Cg Tutorial* describes and shows various Cg viewers that can be used to facilitate user interaction.

---

[217] NVIDIA-NAI-0002604 (GPU Gems 2: Programming Techniques for High-Performance Graphics and General-Purpose Computation (2005)) at xxxii; '461 Patent at 2:20–22.

[218] NVIDIA-NAI-0006426 (Neural Network Toolbox) at 3-23.

[219] NVIDIA-NAI-0003832 (OpenGL Shading) at 27.



[220]

A POSITA would have been motivated to implement these kinds of user interactivity (including GUIs) for simulating artificial neural networks in the accelerator systems. GUIs, such as those described above, would allow users to interact with an artificial neural network in an accessible way and input information and commands that are processed by the CPU, managed and scheduled by the scheduler, and executed by the GPU while providing visual feedback in response to user-indicated changes and display the results of the computations so that users can review, analyze, and use the results of the computations. Command line interfaces and GUIs had been ubiquitous in the field for decades by the time of the '438 Patent. The user interfaces identified above are similar to those used with graphics APIs, such as OpenGL and DirectX, that the '438 Patent explicitly characterizes as prior art. Given the common nature of such user interfaces and functions, and the benefits of them, a POSITA would have had a reasonable expectation of success of implementing a known feature (a GUI or other user interaction) into a known system (a GPU-accelerated system simulating an artificial neural network) to yield predictable results (*e.g.*, allowing a user to easily provide input and control).

[220] NVIDIA-NAI-0005887 (Cg Tutorial) at 39–41, 44.

In view of the knowledge demonstrated in the art in Exhibits B1–B12 and Appendix B, it would have been obvious for a POSITA to receive user input as claimed, and he or she would have been motivated to do so to for the advantages described above.

(8) Transferring Output Data Without Transferring Intermediate Results

Some limitations of the Asserted Claims of the '438 Patent relate to transferring computational results and outputs to memory without transferring "intermediate results," including:

- "the controller is configured to transfer the output data from the accelerator memory to the main memory without transferring any of the intermediate results from the accelerator memory to the main memory so as to reduce data transfer via the bus" (cls. 7, 50);
- "transferring the second portion of the output data from the accelerator memory to the main memory without transferring any of the intermediate results of the plurality of sequential computations from the accelerator memory to the main memory so as to reduce data transfer via the bus" (cl. 16); and
- "wherein transferring the first output data from the memory to the other memory occurs without transferring the intermediate results of the sequence of computations" (cl. 25).

To the extent that any of the primary references do not expressly disclose these or any related limitations, a POSITA would have been motivated to modify any such references based on any of the references in Exhibits B1–B12 and Appendix B disclosing those limitations.

Prior to the '438 Patent, it was a known objective in accelerator systems to "decrease data exchange between CPU and GPU."[221] This was because, as explained in Section II.B, there is a

[221] NVIDIA-NAI-0000919 (Artificial Neural Network Computation) at 625.

cost in transferring data between different memories and systems, including increased latency and reduced overall system efficiency. It was also known that in some computations on GPUs (such as simulations of artificial neural networks), certain intermediate computations (such as computations in a middle layer of a multi-layer neural network) are not important once the final output is determined, making their transfer unnecessary. Additionally, minimizing data movement reduces power consumption, which is particularly critical in high-performance computing environments where energy efficiency is a key design consideration. A POSITA would have been motivated to not transfer unnecessarily between CPU and GPU, as excessive data transfers could create bottlenecks that degrade performance and limit scalability. Thus, a POSITA would have been motivated to not transfer intermediate results from computations where not necessary, optimizing both processing speed and resource utilization.

In view of the knowledge demonstrated in the art in Exhibits B1–B12 and Appendix B, it would have been obvious for a POSITA to transfer data as claimed, and he or she would have been motivated to do so to for the advantages described above.

d. Secondary Considerations of Non-Obviousness

A patentee bears the burden of production with respect to evidence of secondary considerations of non-obviousness. *ZUP, LLC v. Nash Mfg., Inc.*, 896 F.3d 1365, 1373 (Fed. Cir. 2018). As of the date of these Invalidity Contentions, Plaintiff has not identified any evidence of secondary considerations. NVIDIA reserves the right to present additional evidence and argument regarding secondary considerations of non-obviousness to the extent Plaintiff purports to identify any such evidence. In any event, as shown in these Invalidity Contentions, other companies and individuals described, built, and/or patented the exact same concepts in the '438 Patent before Plaintiff ever did—often many years before. Even where others' invention(s) occurred around the

same time as the '438 Patent, such simultaneous invention demonstrates the obviousness of the '438 Patent.

4. Priority Date

U.S. Application No. 15/808,201 (hereinafter, "the '201 application"), which led to the '438 Patent, was filed on November 9, 2017. The '201 application was a reissue application for U.S. Patent No. 9,189,828 ("the '828 Patent"), which was filed on January 3, 2014, and issued on November 17, 2015. The '201 application purports to claim priority to provisional appl. no. 60/826,892 (hereinafter, "the '892 provisional"), which was filed on September 25, 2006. To the extent that Plaintiff asserts that the '438 Patent is entitled to an earlier priority date based on any provisional application or documentation, Plaintiff has the burden of proof and has not demonstrated an earlier priority date.

a. Provisional Application

To the extent that Plaintiff asserts that the Asserted Claims of the '438 Patent are entitled to an earlier priority date based on the disclosure of the '892 provisional, Plaintiff has the burden of proof and has not demonstrated support for all claim limitations in the '892 provisional.

b. Asserted Conception, Reduction to Practice, and Diligence

In its Infringement Contentions, Plaintiff contends that the Asserted Claims of the '438 Patent "are entitled to a priority date corresponding to the conception of the claimed inventions, which occurred no later than February 25, 2005," and that "[t]he conception of the inventions . . . was followed by continuous diligence and actual reduction to practice of the claimed inventions no later than December 14, 2005." The documents that Plaintiff identifies in support of this contentions fail to demonstrate earlier conception, reduction to practice, or diligence.

The documents produced to date by Plaintiff fail to support earlier conception, let alone conception as early as February 25, 2005. Plaintiff's documents do not disclose several limitations

of the Asserted Claims of the '438 Patent, including, without limitation, the "controller," referencing input data and output data by a "pointer," simulating an artificial neural network in a system of a specific architecture, and transferring data between CPU/accelerator during continued computations. For example, [redacted] There is nothing disclosed relating to the claim limitations of the '438 Patent.

The documents produced to date also do not evidence an actual reduction to practice, let alone diligence from the alleged conception date until either an actual or constructive reduction to practice. The document produced by Plaintiff dated December 14, 2005 (the alleged date of actual reduction to practice) is a presentation titled "First Look Venture Assessment." *See* NAI_0007861. The presentation describes the Synapcard at a high-level, but it does not corroborate any of the claim limitations of the Asserted Claims of the '438 Patent. *See id.* For example, there is no disclosure at all of the limitations relating to data transfer. Indeed, as late as October 12, 2005, Synapcard was described as still "a pen and paper idea," and the document states that "[a] prototype . . . could potentially be developed *within the next year*." NAI_0007840 at 845 (emphasis added). There is no indication in any documents that the card was actually reduced to practice only two months later.

As such, the Asserted Claims of the '438 Patent are not entitled to an earlier priority date.

D. Prior Art Invalidity of U.S. Patent No. RE49,461

1. Identification of Prior Art

NVIDIA identifies the following prior art now known to NVIDIA that anticipate or render obvious one or more of the Asserted Claims of the '461 Patent. Each of the following patents, publications, systems, and products is prior art under at least one or more of 35 U.S.C. § 102(a),

(b), (e), (f), or (g) (pre-AIA) or § 102(a) (post-AIA), as applicable. NVIDIA's reliance on each prior art reference below includes the reference itself, anything incorporated by the reference, and any testimony by those with knowledge of the reference, such as named authors and inventors.

a. Patents and Patent Publications

Patent / Publication No.	First Named Inventor	Date of Issue / Publication
Japanese Patent Appl. H11-232247	Takahashi Nakamoto	Aug. 27, 1999
Japanese Patent Publ. H04-237388	Yoichi Tamura	Aug. 25, 1992
U.K. Patent Appl. No. 2291553	David Wilson	Jan. 24, 1996
U.S. Patent No. 6,853,382	James Van Dyke	Feb. 8, 2005
U.S. Patent No. 7,139,003	David Kirk	Nov. 21, 2006
U.S. Patent No. 7,219,085	Ian Buck	May 15, 2007
U.S. Patent No. 7,256,788	Viet-Tam Luu	August 14, 2007
U.S. Patent No. 7,339,592	John Lindholm	Mar. 4, 2008
U.S. Patent No. 7,418,576	John Lindholm	Aug. 26, 2008
U.S. Patent No. 7,673,304	Anuj Gosalia	Mar. 2, 2010
U.S. Patent No. 7,681,060	John Nickolls	Dec. 28, 2010
U.S. Patent No. 7,694,304	Kevin Kissell	Apr. 6, 2010
U.S. Patent No. 7,747,070	Siddhartha Puri	Jun. 29, 2010
U.S. Patent No. 7,830,392	John Danskin	Nov. 9, 2010
U.S. Patent No. 7,899,995	Radoslav Danilak	Mar. 1, 2011
U.S. Patent No. 8,059,131	Donald Bittel	Nov. 15, 2011
U.S. Patent No. 8,730,249	John Danskin	May 20, 2014
U.S. Patent No. 9,262,129	Michael Santori	Feb. 16, 2016
U.S. Patent Publ. No. 2003/0140179	Nicholas Wilt	Jul. 24, 2003
U.S. Patent Publ. No. 2003/0195938	Kevin Howard	Oct. 16, 2003
U.S. Patent Publ. No. 2005/0197977	Ian Buck	Sept. 8, 2005
U.S. Patent Publ. No. 2006/0098019	David Tarditi	May 11, 2006

b. Printed Publications

Citation	Publication Date
A. Gorchetchnikov, *KInNeSS: A new software environment for simulations of neuronal activity*	2004

Citation	Publication Date
A. Lefohn, *GPU Memory Model Overview*	2005
A. Veen, *Dataflow Machine Architecture*	1986
B. Kernigham et al., The C Programming Language	1988
B. Khailany, *Exploring the VLSI Scalability of Stream Processors*	2003
B. Khailany, *The VLSI Implementation and Evaluation of Area- and Energy-Efficient Streaming Media Processors*	2003
B. Svensson, *SIMD Architectures*	1992
C. Davis, *Graphics Processing Unit Computation of Neural Networks*	2005
Cg Toolkit: User's Manual	2005
D. Steinkraus, *Using GPUs for Machine Learning Algorithms*	2005
Dominik Göddeke -- GPGPU::Basic Math Tutorial	2005
E. Chu, *FFT algorithms and their adaptation to parallel processing*	1998
E. Kilgariff, *The GeForce 6 Series GPU Architecture*	2006
Game Programming Gems 4	2004
General Purpose Computation on Graphics Hardware	2005
GPU Gems 2: Programming Techniques for High-Performance Graphics and General-Purpose Computation	2005
H. Demuth, *Neural Network Toolbox for Use with MATLAB*	2004
H. Esmaeilzadeh, *Neural Network Stream Processing Core (NnSP) for Embedded Systems*	2006
H. Esmaeilzadeh, *NnSP: Embedded Neural Network Stream Processor*	2005
I. Buck et al., *Brook for GPUs: Stream Computing on Graphics Hardware* (presentation)	2004
I. Buck et al., *Brook for GPUs: Stream Computing on Graphics Hardware*	2004
I. Buck et al., *Data Parallel Computation on Graphics Hardware*	2003
I. Buck, *Brook for GPUs*	2003
I. Buck, *Stream Computing on Graphics Hardware*	2005
J. Ahn, *Scatter-Add in Data Parallel Architecture*	2005
J. Chen, *Fast Fourier Transform with BrookGPU*	2006
J. Fung, *Computer Vision Signal Processing on Graphics Processing Units*	2004
J. Fung, *OpenVIDIA: Parallel GPU Computer Vision*	2005
J. Georgii, *Interactive Simulation of Deformable Bodies on GPUs*	2005
J. Leiterman, *Learn Vertex and Pixel Shader Programming with DirectX 9*	2004

Citation	Publication Date
J. Martin, *Fortran 90 Pointers vs. "Cray" Pointers*	1991
J. Montrym, *The GeForce 6800*	2005
J. Ohmer, *Implementation of Kernel Methods on the GPU*	2005
J. Owens, *A Survey of General-Purpose Computation on Graphics Hardware*	2005
K. Jung, *Neural network-based text location in color images*	2001
K. Moreland, *The FFT on a GPU*	2003
K. Oh, *GPU implementation of neural networks*	2004
M. Botnen et al., *Using Commodity Graphics Hardware for Medical Image Segmentation*	2005
M. Erez, *Analysis and Performance Results of a Molecular Modeling Application on Merrimac*	2004
M. Erez, *Fault Tolerance Techniques for the Merrimac Streaming Supercomputer*	2005
M. Flynn, *Some Computer Organizations and Their Effectiveness*	1972
M. Glover et al., *A Massively-Parallel SIMD Processor for Neural Network and Machine Vision Applications*	1993
M. Harris, *GPGPU: General-Purpose Computation on GPUs*	2004
M. Harris, *Mapping Computational Concepts to GPUs*	2005
M. Harris, *Real-Time Cloud Simulation and Rendering*	2003
M. Kilgard, *A Follow-up Cg Runtime Tutorial for Readers of The Cg Tutorial*	2005
M. Kilgard, *Cg – A System for Programming Graphics Hardware in a C-like Language*	2006
M. Kilgard, *Cg and NVIDIA*	2006
M. Kilgard, *Cg in Two Pages*	2003
M. Kilgard, *NVIDIA OpenGL 2.0 Support*	2005
M. Kilgard, *Re-implementing the Follow-up Cg Runtime Tutorial with CgFX*	2006
M. Segal, *The OpenGL Graphics System: A Specification*	2004
MATLAB: The Language of Technical Computing	2005
Merrimac: The Streaming Supercomputer	2003
NVIDIA GPU Programming Guide (Version 2.4.0)	2005
OpenGL Super Bible	1996
P. Lisboa, *A review of evidence of health benefit from artificial neural networks in medical intervention*	2002

Citation	Publication Date
R. Agarwal, *A High Performance Parallel Algorithm for 1-D FFT*	1994
R. Hughey, *The UCSC Kestrel high performance SIMD processor: present and future*	2001
R. Rost, *OpenGL Shading Language*	2004
S. Rixner, *Stream Processor Architecture*	2001
S. Vitabile, *Real-Time Road Signs Recognition on a SIMD Architecture*	2004
The Cg Tutorial: The Definitive Guide to Programmable Real-Time Graphics	2003
U. Kapasi, *Programmable Stream Processors*	2003
U. Kapasi, *The Imagine Stream Processor*	2002
W. Dally, *Merrimac: Supercomputing with Streams*	2003
W. Press, *Numerical Recipes in C* (2d ed.)	2002
W. Thies, *StreamIt: A Language for Streaming Applications*	2002
Y Liu et al., *Real-Time 3D Fluid Simulation on GPU with Complex Obstacles*	2004
Z. Fan, *GPU Cluster for High Performance Computing*	2004
Z. Luo, *Artificial Neural Network Computation on Graphic Process Unit*	2005

c. Prior Art Products and Systems

Product / System
NVIDIA – NV40 (*e.g.*, GeForce 6800)
NVIDIA – G70 (*e.g.*, GeForce 7900)
NVIDIA – NV50/G80
BrookGPU-Merrimac ("Brook")
NVIDIA – Cg
The Imagine Stream Processor
SANNDRA / KInNeSS

d. Prior Invention Under § 102(g)

Prior Invention	Inventor / Assignee
NV50/G80	NVIDIA

Additionally, as explained in Section I.C, Plaintiff's Infringement Contentions are deficient and fail to provide sufficient allegations as to how the accused products infringe each and every

claim limitation of the '461 Patent. Indeed, for most limitations, Plaintiff's Infringement Contentions are vague and unclear as to which particular functionality Plaintiff contends satisfies a claim limitation, prejudicing NVIDIA's ability to fully investigate invalidity under Plaintiff's theories of infringement.

While NVIDIA does not agree that the functionalities accused of infringement in Plaintiff's Infringement Contentions meet the claim limitations, the functionalities that Plaintiff apparently accuses of infringement were in development prior to the '461 Patent. Thus, to the extent those functionalities infringe the '461 Patent, they also invalidate it. *Peters v. Active Mfg. Co.*, 129 U.S. 530, 547 (1889) ("That which infringes, if later, would anticipate, if earlier."). For example, as explained in Section II.D.6, CUDA development at NVIDIA had been going on long before the priority date of the Asserted Patents, including functionalities from prior art systems like Merrimac/Brook and Cg, as discussed above. CUDA was built on the foundations laid by earlier programming languages such as Brook and Cg, and CUDA-related development documents show functionalities that Plaintiff may accuse of infringement as having existed prior to any asserted or actual priority date for the '461 Patent. For example, libraries such as cuFFT and cuBLAS were developed prior to the priority date of the '461 Patent. The following exemplary categories of documents evidence prior invention and development by NVIDIA:

- CUDA Internal Programming Guides
- CUDA Language Specifications
- CUDA API Specifications and Documentation
- CUDA Driver Compiler Internal Documentation
- cuFFT Internal Documentation
- cuBLAS Internal Documentation

NVIDIA reserves the right to rely on these and any other documents showing NVIDIA's prior invention. NVIDIA further incorporates by reference its charts on Brook/Merrimac and Cg, which (as explained above) were precursors to the accused CUDA functionality developed at NVIDIA.

Thus, to the extent Plaintiff clarifies its infringement allegations by identifying and explaining how specific CUDA or other functionality allegedly meets the claim limitations, NVIDIA reserves the right to demonstrate that the same functionality is prior art under 35 U.S.C. § 102(g)(2).

2. Anticipation

NVIDIA has identified the following prior art known to NVIDIA as anticipating the Asserted Claims of the '461 Patent. Each of the references cited below anticipates the Asserted Claims, either expressly or inherently as understood by a person of ordinary skill in the art, depending on the Court's interpretation of the claims. In some instances, NVIDIA has treated certain prior art as anticipatory, where certain elements are present based on at least Plaintiff's apparent claim construction in its Infringement Contentions. Exhibits C1–C15 are charts setting forth where in the prior art references each element of the Asserted Claims of the '461 Patent are found. Where NVIDIA charts a prior art product or system based on patents and/or printed publications related to that product/system, NVIDIA reserves the right to rely on such patents and/or printed publications alone as anticipatory references to the Asserted Claims of the '461 Patent.

Exhibit	Reference
C1	NV40
C2	G70
C3	NV50/G80
C4	Cg
C5	Brook

Exhibit	Reference
C6	Imagine Stream Processor
C7	SANNDRA / KInNeSS
C8	U.S. Patent No. 7,747,070 (Puri)
C9	U.S. Patent Publ. No. 2003/0140179 (Wilt)
C10	U.S. Patent No. 7,219,085 (Buck '085)
C11	GPU Gems 2: Programming Techniques for High-Performance Graphics and General-Purpose Computation
C12	U.S. Patent Publ. No. 2005/0197977A1 (Buck '977)
C13	Matlab Neural Network Toolbox User's Guide Version 4 by Howard Demuth and Mark Beale (nnet)
C14	Artificial Neural Network Computation on Graphic Process Unit (ANN)
C15	Using GPUs for Machine Learning Algorithms (GPUs for MLA)

3. Obviousness

To the extent that Plaintiff argues that any of the above references do not anticipate the Asserted Claims of the '461 Patent, the claims are invalid as obvious under 35 U.S.C. § 103 in view of these references alone, in combination with the knowledge of a POSITA, and in combination with other references as described below. Exhibits C1–C15 and Appendix C further identify specific examples of art that alone or in combination render obvious the limitations of the Asserted Claims of the '461 Patent. Additional evidence of obviousness can be found in the reference identified in the section identifying prior art patents, publications, and systems.

While some of the above identified prior art may anticipate the asserted claims, this presents no obstacle to also relying on this reference as a basis for invalidity based on obviousness. It is established law that "a rejection for obviousness under § 103 can be based on a reference which happens to anticipate the claimed subject matter." *In re Meyer*, 599 F.2d 1026, 1031 (C.C.P.A. 1979). And to the extent any prior art item cited herein may not fully disclose a limitation of an asserted claim or is alleged by Plaintiff to lack disclosure of the limitation, such

limitation is present and identified in another prior art item as shown in the attached claim charts. It would have been obvious to combine the disclosed claim elements based on the teachings of other art in existence at that time, the general knowledge of a person of ordinary skill in the art, and the recognition of similarities between the art and the claimed features. On such basis, on an element-by-element basis, NVIDIA expressly intends to combine one or more prior art items identified herein with each other to address any further contentions from Plaintiff that a particular prior art item supposedly lacks one or more elements of an asserted claim. In other words, NVIDIA contends that each charted prior art item can be combined with other prior art items if a particular prior art item lacks or does not explicitly disclose an element or feature of an asserted claim.

a. Level of Ordinary Skill in the Art

A person of ordinary skill in the art at the time of the earliest effective filing date of the '461 Patent would have had a Bachelor's degree in electrical or computer engineering (or equivalent discipline) and at least two years' experience in research, design and/or development of computer systems, including experience with GPUs. Additional education could substitute for experience, and vice versa.

b. Scope and Content of the Prior Art

The '461 Patent is directed to a system for performing *general-purpose* computations (such as simulating an artificial neural network) on traditionally *special-purpose* graphics GPUs. The '461 Patent's previous claims (in the '828 Patent from which it was reissued) were each discarded and replaced by reissue claims drafted with nearly 14 years of hindsight. But the new claims cover only well-known techniques for systems that involve both a CPU and a GPU (called the "processing unit" in the claims), set in the context of artificial neural networks. For example, the claims recite maintaining separate instruction "streams" (or threads) for execution on the different

processors for separate data types, which the patent refers to as "user interaction" (for the CPU) and "computational" streams (for the GPU), and the implementation of artificial neural network computations on a GPU.[222] The claims also recite other basic techniques used in systems with multiple processors. For example, the claims ensure that the CPU cannot corrupt the data being used by the GPU for its calculations by "shifting control of a data exchange" to the GPU while it is executing a calculation and then back to the CPU after the calculation is done or interrupted, as well as "queueing" commands intended for the GPU while the GPU is performing other calculations so that they are ready when the GPU is next available.

As explained further in Section II.B (incorporated herein by reference), prior to the '461 Patent, those skilled in the art had already taught these well-known concepts. Stream processing, for example, was a basic, fundamental concept in CPU and GPU programming in the prior art. It was known in the art that GPUs operated based on the "stream programming model," where data is represented as a stream—an ordered set of elements of the same data type—and processed by kernels, which take one or more input streams and produce one or more output streams.[223] A key feature of this model is that kernels process entire streams rather than individual elements, ensuring that computations on each element remain independent. This independence allows what appears to be a serial computation to be executed efficiently on data-parallel hardware, optimizing storage, access, and execution. The architecture of GPUs, including NVIDIA's, was

[222] '461 Patent at 1:59–63, 2:11–19, 7:60–8:6.

[223] NVIDIA-NAI-0002604 (GPU Gems 2: Programming Techniques for High-Performance Graphics and General-Purpose Computation (2005)) at 464 ("In the stream programming model, all data is represented as a stream—which we define as an ordered set of data of the same data type. That data type can be simple (a stream of integers or floating-point numbers) or complex (a stream of points or triangles or transformation matrices).").

well-suited to this model, as it aligned with the graphics pipeline's reliance on data parallelism.[224] By the time of the '461 Patent, GPUs had evolved into high-performance data-parallel processors capable of implementing essential kernels like vertex and fragment programs, which sustained high computation rates on user-defined tasks. This transformation effectively turned GPUs into programmable stream processors, making them ideal for computationally demanding applications beyond graphics rendering, such as solving systems of linear equations and other tasks requiring high arithmetic intensity.[225]

The task and resource management elements of the '461 Patent were also well known and established in the art prior to the patent. Because GPGPU relies on running many tasks concurrently within a single stage of processing while managing dependencies between stages, it was known that efficient task scheduling was critical for optimizing GPU performance. Thus, those skilled in the art had already developed "schedulers," which play a key role in allocating system resources, determining which tasks—whether individual threads or complete processes—are executed at any given time based on factors like priority, resource demands, and system load. Scheduling policies, such as priority scheduling, assign priority values to tasks, allowing higher-priority tasks to preempt lower-priority ones in systems that support preemption, ensuring that critical operations run promptly.[226] Additionally, task queuing methods were known to help manage system resources by placing tasks in a queue when necessary resources are unavailable, optimizing execution order based on scheduling algorithms.[227] By dynamically balancing

---

[224] NVIDIA-NAI-0002604 (GPU Gems 2: Programming Techniques for High-Performance Graphics and General-Purpose Computation (2005)) at 464.

[225] NVIDIA-NAI-0002604 (GPU Gems 2: Programming Techniques for High-Performance Graphics and General-Purpose Computation (2005)) at 464–67, 495.

[226] NVIDIA-NAI-0004859 (Lindholm) at 4:11–5:62.

[227] *See, e.g.*, NVIDIA-NAI-0004733 (Buck '977).

computational loads, managing data dependencies, and improving data coherency, schedulers enabled efficient multitasking and maximize GPU utilization. Thus, there was nothing novel about the task scheduling and management elements of the '461 Patent.

c. Motivations to Combine Prior Art

A person of ordinary skill in the art would have been motivated to combine the above identified prior art items. Here, the claimed elements of the '461 Patent are disclosed in the prior art, as established above in Exhibits C1–C15 and Appendix C. Further, any elements claimed in the Asserted Claims would have been obvious to a person of ordinary skill in the art and predictable solutions resulting from routine optimization. Certain of these rationales are discussed more specifically below. The fact that others are not discussed more specifically should not be interpreted as an admission or concession that it does not apply. To the contrary, the discussion below simply provides more explanation of these specific rationales.

As set forth below, it would have been obvious to one of person of ordinary skill in the art to combine the teachings of similar prior art references to render the Asserted Claims obvious. The references identified in the exemplary combinations set forth below are analogous prior art and for at least the reasons set forth below, a person of ordinary skill in the art would have had reasons for combining them. The various teachings, suggestions, and/or reasons to modify any of the references and/or to combine any two or more references come from various sources, including the prior art (specific and as a whole), common knowledge, common sense, predictability, expectations, industry trends, design incentives or need, market demand or pressure, market forces, the nature of the problem faced, and/or the knowledge possessed by one of ordinary skill in the art.

Furthermore, to the extent that Plaintiff argues any modification of a base reference is required to meet the claim elements, such modification was routine and predictable, within the

skill of a person of ordinary skill in the art, and a person of ordinary skill in the art would have had a reasonable expectation of success in making such a combination.

The following are exemplary motivations to combine:

(1) Executing Computations Representing an Artificial Neural Network

The asserted independent claim of the '461 Patent relates to executing computations representing an artificial neural network and includes several steps relating to the computations. Examples of these steps include:

- "[a] method of executing computations representing an artificial neural network on a computer system" (cl. 21)
- "executing, by the at least one CPU, a user interaction stream, the user interaction stream controlling transfer of inputs to the artificial neural network to the first memory partition and the second memory partition" (cl. 21);
- "executing the user command during execution of the computations representing the artificial neural network at times determined by the computational stream" (cl. 21).

To the extent that any of the primary references do not expressly disclose these or any related limitations, a POSITA would have been motivated to modify any such references based on any of the references in Exhibits C1–C15 and Appendix C disclosing those limitations.

As explained in Section II.B.5, one significant application that arose from early GPGPU advancements was the ability to simulate artificial neural networks on GPUs.[228] Prior to the Asserted Patents, it was known that the parallel processing power of GPUs was crucial for efficiently training these networks, as GPUs are particularly well-suited for the matrix operations

[228] NVIDIA-NAI-0003913 (GPU Implementation of Neural Networks) at 322–24; NVIDIA-NAI-0000919 (Artificial Neural Network Computation) at 622.

and numerical calculations required for neural network training.[229] For example, before the Asserted Patents, GPUs were already being utilized to power artificial neural networks in a variety of applications, including tasks such as image and pattern recognition,[230] object detection,[231] medical diagnostics,[232] and checkers.[233] Relevant to the '461 Patent, Buck discloses a computer system that uses a GPU to process and execute "operations traditionally handled by the CPU," such as simulating "layers in a neural network." Buck's optimized GPU-accelerated system "take[s] full advantage of the new flexibility introduced by programmable pixel shaders of GPUs" and "parallel architecture of the GPU" to train, test, and use neural networks.[234]

It would have been obvious, and a POSITA would have been motivated, to implement and simulate an artificial neural network on the GPU-accelerator architectures described in Exhibits C1–C15. Some of the references described in these Invalidity Contentions provide programs and algorithms for simulating artificial neural networks, which are agnostic as to the GPU hardware on which they are executed. A POSITA would have been motivated to implement these neural networks on the GPU accelerator hardware disclosed in Exhibits C1–C15, many of which relate to NVIDIA GPUs. Indeed, a POSITA would have been motivated to implement GPU programming techniques on the most advanced GPUs at the time, which a POSITA would have known at the time were provided by NVIDIA.

---

[229] NVIDIA-NAI-0003913 (GPU Implementation of Neural Networks) at 322–23 ("[M]any inner product operations can be replaced with matrix multiplication, which is more appropriate for GPU implementation.").

[230] NVIDIA-NAI-0003913 (GPU Implementation of Neural Networks).

[231] NVIDIA-NAI-0000919 (Artificial Neural Network Computation) at 622.

[232] NVIDIA-NAI-0000859 (ANN in Medical Intervention).

[233] NVIDIA-NAI-0003443 (Game Programming Gems) at 6.

[234] NVIDIA-NAI-0004733 (Buck '977) ¶¶ 11, 29–31, 64.

Examples of the accelerator architectures on which these techniques could be applied are NV40,[235] G70,[236] and NV50/G80[237]—all of which are from NVIDIA. These and the other GPU architectures charted provided advanced parallel processing capabilities and efficient data-sharing mechanisms, which are beneficial for artificial neural networks. A POSITA would have understood that these systems would significantly reduce execution time compared to other processing architectures. Mathematical and scientific simulations (including artificial neural networks) often involve iterative calculations where results from one computation stage influence subsequent stages, and these systems disclose shared memory that would allow for the outputs of one stage of computations become inputs to subsequent stages.

A POSITA would have understood that these combinations simply applied a known technique (*e.g.*, simulation of artificial neural networks) to a known device (*e.g.*, GPU-accelerated parallel processing systems, such as that in NV40, G70, NV50/G80, and the other primary references) ready for improvement to yield predictable results (*e.g.*, fast and efficient computation). Furthermore, to the extent any of the primary references do not discuss the structure of an artificial neural network (such as its "layers" and "neurons"), those fundamental features of artificial neural networks were well known and understood prior to the '461 Patent, as explained in Section II.B.5.

(2) Streams

Some limitations of the Asserted Claims of the '461 Patent relate to executing various "streams" for different purposes and shifting control between the streams, including:

---

[235] *See* Exhibit C1.

[236] *See* Exhibit C2.

[237] *See* Exhibit C3.

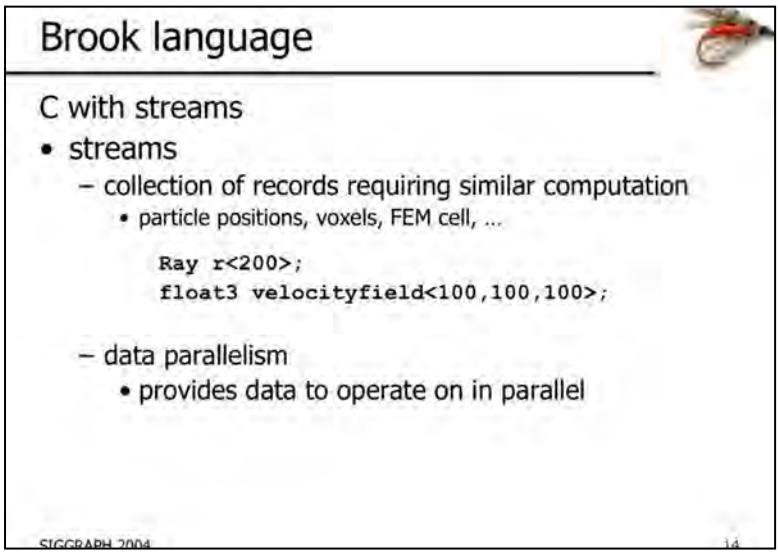
- "executing, by the at least one CPU, a user interaction stream, the user interaction stream controlling transfer of inputs to the artificial neural network to the first memory partition and the second memory partition" (cl. 21);
- "executing, by the processing unit, a computational stream, the computational stream controlling data exchange between the user interaction stream and the computational stream during execution of the computations representing the artificial neural network" (cl. 21);
- "shifting control of a data exchange between the user interaction stream and the computational stream to the computational stream in response to starting execution of the computations representing the artificial neural network" (cl. 21);
- "shifting control of the data exchange between the user interaction stream and the computational stream to the user interaction stream in response to completion or interruption of the computations representing the artificial neural network" (cl. 21); and
- "executing, by the at least one CPU, a data output stream, the data output stream controlling transfer of outputs of the computations representing the artificial neural network to disk" (cl. 31); *see also* cls. 22–30.

To the extent that any of the primary references do not expressly disclose these or any related limitations, a POSITA would have been motivated to modify any such references based on any of the references in Exhibits C1–C15 and Appendix C disclosing those limitations.

The stream programming model, including shifting control between streams during setting and execution of computations, was well known prior to the '461 Patent. As explained in Section II.B.2, parallel processing, for both graphics and general-purpose computing, was known for decades before the '461 Patent. Unlike CPUs, which excel at handling a few tasks sequentially,

GPUs are optimized for parallel processing, making them ideal for the thousands of simultaneous calculations required to render graphics. Prior art GPUs operated according to the "stream programming model," which represents data as a "stream," or "an ordered set of data of the same data type."[238] Computations on individual stream elements can be performed independently, allowing serial operations to be mapped onto data-parallel hardware.[239] The prior art "streams" include both user interaction and computational streams. For example, prior art patent Buck incudes a CPU that processes user interactions and commands by executing a user interaction stream, and it maintains its own "stream[s]" of instructions and data.[240]

Stream computing concepts were disclosed in numerous other prior art references. An exemplary prior art stream computing architecture could be a Merrimac multiprocessor architecture, or a GPU-based architecture.



[241]

[238] NVIDIA-NAI-0002604 (GPU Gems 2: Programming Techniques for High-Performance Graphics and General-Purpose Computation (2005)) at 464.

[239] NVIDIA-NAI-0002604 (GPU Gems 2: Programming Techniques for High-Performance Graphics and General-Purpose Computation (2005)) at 465.

[240] *See, e.g.*, NVIDIA-NAI-0004733 (Buck '977) ¶¶ 8, 35, 46, 179.

[241] NVIDIA-NAI-0000925 (SIGGRAPH 2004 Presentation) at 14.

kernels

- kernels
  - functions applied to streams
    - similar to for_all construct
    - no dependencies between stream elements

```
kernel void foo (float a<>, float b<>,
                 out float result<>) {
  result = a + b;
}

float a<100>;
float b<100>;
float c<100>;

foo(a,b,c);  ←  for (i=0; i<100; i++)
                      c[i] = a[i]+b[i];
```

SIGGRAPH 2004 15

242

3.2 Kernels

Brook kernels are special functions, specified by the `kernel` keyword, which operate on streams. Calling a kernel on a stream performs an implicit loop over the elements of the stream, invoking the body of the kernel for each element. An example kernel is shown below.

```
kernel void saxpy (float a, float4 x<>, float4 y<>,
                   out float4 result<>) {
  result = a*x + y;
}

void main (void) {
  float a;
  float4  X[100], Y[100], Result[100];
  float4  x<100>, y<100>, result<100>;
  ... initialize a, X, Y ...
  streamRead(x, X);             // copy data from mem to stream
  streamRead(y, Y);
  saxpy(a, x, y, result);       // execute kernel on all elements
  streamWrite(result, Result);  // copy data from stream to mem
}
```

243

[242] NVIDIA-NAI-0000925 (SIGGRAPH 2004 Presentation) at 15.

[243] NVIDIA-NAI-0000961 (Brook for GPU 2004) at 779.

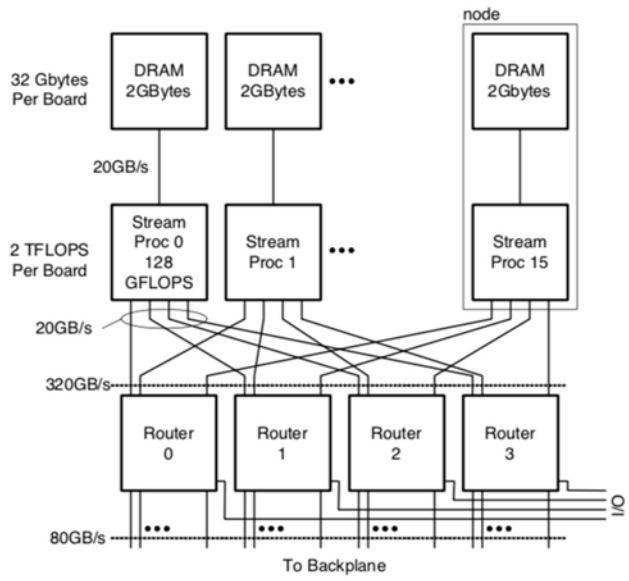


Figure 6: Sixteen 128 GFLOPS stream processors each with 2 GBytes of DRAM memory can be packaged on a single board. The board has a total of 2 TFLOPS of arithmetic and 32 GBytes of memory. Such a board is useful as a stand-alone scientific computer and as a building-block for larger systems.

[244]

Stream computing concepts were also taught by prior art NVIDIA patents, describing NVIDIA's inventions. For example, the stream computing kernel concept was also referred to as a CTA, which stands for "cooperative thread array:"

[244] NVIDIA-NAI-0090973 (Merrimac: Supercomputing with Streams) at 5.

BRIEF SUMMARY OF THE INVENTION

Embodiments of the present invention provide data processing systems and methods that use cooperative thread arrays (CTAs) to perform computations. As used herein, a "cooperative thread array," or "CTA," is a group of multiple threads that concurrently execute the same program on an input data set to produce an output data set. Each thread in a CTA has a unique identifier (thread ID) assigned at thread launch time that controls various aspects of the thread's processing behavior. For instance, a thread ID may be used to determine which portion of the input data set a thread is to process, to identify one or more other threads with which a given thread is to share an intermediate result, and/or to determine which portion of the output data set the thread is to compute or write.

CTAs can be executed in a variety of processing architectures, including sequential single processors, parallel processors, multithreaded processors, and any architecture that can support multiple concurrent threads and that provides at least some shared memory, interconnection network, or other technology that allows threads to communicate with each other. In some embodiments, CTAs are executed using a processing core of a graphics processor that has multiple parallel processing engines, each capable of supporting multiple concurrent threads. The threads are advantageously executed in SIMD (single instruction, multiple data) groups, with one thread of the group being associated with each processing engine. A single instruction unit issues an instruction to an entire SIMD group in parallel, and each processing engine executes the instruction in the context of its thread of the current SIMD group; instructions for different SIMD groups can be issued in any order. By executing each instruction in the appropriate context, each processing engine executes one thread in each of multiple concurrent SIMD groups. [245]

A POSITA would have been motivated to apply these principles to any of the primary references in Exhibits C1–C15. A POSITA would have been motivated to implement stream programming in a GPU for accelerating artificial neural networks (ANNs) due to the inherent parallelism and efficiency of this model. Stream programming, which processes data as ordered sequences through kernels, aligns well with the layered structure of ANNs, where computations in each layer can be executed in parallel. By incorporating a user interaction stream and a

[245] NVIDIA-NAI-0004975 (Nickolls) at 2:20–54.

computational stream, a POSITA could efficiently manage real-time data input while ensuring high-throughput execution of matrix operations—critical for ANN training and inference. The computational stream would handle intensive tasks such as weight updates and activations, leveraging the GPU's ability to process thousands of parallel computations, while the user interaction stream would enable dynamic adjustments based on input data without stalling execution. Given that GPUs were already optimized for data-parallel workloads, stream programming would allow for seamless scheduling and efficient memory access, reducing latency and maximizing throughput. This approach would have been particularly attractive for ANNs, which require high arithmetic intensity and continuous data flow, making GPUs the ideal hardware for accelerating neural network computations.

In view of the knowledge demonstrated in the art in Exhibits C1–C15 and Appendix C, it would have been obvious for a POSITA to implement stream processing as claimed, and he or she would have been motivated to do so to for the advantages described above.

(3) Resource Management and Task Scheduling

Some limitations of the Asserted Claims of the '461 Patent relate to controlling the setting and editing of computational elements by a user, including:

- "shifting control of a data exchange between the user interaction stream and the computational stream to the computational stream in response to starting execution of the computations representing the artificial neural network" (cl. 21);
- "shifting control of the data exchange between the user interaction stream and the computational stream to the user interaction stream in response to completion or interruption of the computations representing the artificial neural network" (cl. 21);

- "queueing a user command received by the user interaction stream during execution of the computations representing the artificial neural network" (cl. 21);
- "executing the user command during execution of the computations representing the artificial neural network at times determined by the computational stream" (cl. 21);
- "wherein the user interaction stream controls the data exchange between the user interaction stream and the computational stream outside of execution of the computations representing the artificial neural network" (cl. 22);

To the extent that any of the primary references do not expressly disclose these or any related limitations, a POSITA would have been motivated to modify any such references based on any of the references in Exhibits C1–C15 and Appendix C disclosing those limitations.

As explained in Section II.B.7, prior to the '461 Patent, graphics hardware manufacturers such as NVIDIA developed improved software tools for programmable GPUs, such as advanced application programming interfaces (APIs) and programming languages. These tools facilitated interactions with GPU accelerators, allowing users to trigger when tasks are executed. For example, programmers made use of NVIDIA's GPUs through graphics Application Programing Interfaces (APIs), such as Direct3D and OpenGL, as well as GPU languages such as HLSL, GLSL, and Cg.[246]

GPGPU involves running many tasks concurrently within a single stage of processing and coordinating dependencies between stages of processing. It was therefore known that efficient management of tasks assigned to the GPU and of system resources was needed for optimal utilization of a GPU's parallel processing capabilities. Well before the '461 patent, the operating

[246] NVIDIA-NAI-0002604 (GPU Gems 2: Programming Techniques for High-Performance Graphics and General-Purpose Computation (2005)) at xxxii; '461 Patent at 2:20–22.

system of a computer was typically responsible for managing resources (*e.g.*, GPUs, CPUs, memory, and software) within the system. As an example, Wilt describes "methods and [a] system for managing computational resources of a coprocessor in a computing system.[247] More specifically, an operating system kernel component called a "scheduler" assigns resources to perform tasks. Schedulers determine when tasks should be executed and make necessary computational resources available to the executing tasks according to scheduling policies that account for requirements such as task priority, timing, and resource needs. As tasks come in, they can be processed and queued for later execution.

Similarly, Gosalia describes systems and methods "for scheduling the processing of a coprocessor whereby applications can submit tasks to a scheduler, and the scheduler can determine how much processing each application is entitled to as well as an order for processing."[248] As a further example, Lindholm discloses a multithreaded processor instruction having an instruction scheduler that "prioritizes the set of current instructions and issues on each issue cycle a set of instructions from the instruction buffer to the plurality of different types of execution units."[249]

A POSITA would have been motivated to apply these principles to any of the primary references in Exhibits C1–C15. For example, a POSITA would have been motivated to implement these techniques for managing tasks in accelerator systems. By managing tasks that require access to the same data, computational resources currently in-use, or have data dependencies, schedulers optimize efficiency, balance computational loads, and improve data coherency in a system. Given the common nature of such mechanisms in computing, and the benefits of them, a POSITA would

[247] NVIDIA-NAI-0004704 (Wilt).

[248] NVIDIA-NAI-0004886 (Gosalia) at Abstract.

[249] NVIDIA-NAI-0004877 (Lindholm) at Abstract, 1:31–43.

have had a reasonable expectation of success of implementing known features (resource management and task scheduling) into a known system (a GPU-accelerated system simulating an artificial neural network) to yield predictable results (e.g., enabling GPUs to handle large, complex workloads).

In view of the knowledge demonstrated in the art in Exhibits C1–C15 and Appendix C, it would have been obvious for a POSITA to implement resource scheduling and management as claimed, and he or she would have been motivated to do so to for the advantages described above.

(4) Controlling Setting and Editing of Computational Elements

Some limitations of the Asserted Claims of the '461 Patent relate to controlling the setting and editing of computational elements by a user, including:

- "executing, by the at least one CPU, a user interaction stream, the user interaction stream controlling transfer of inputs to the artificial neural network to the first memory partition and the second memory partition" (cl. 21);
- "executing the user interaction stream comprises: controlling setting and editing of computational elements of the computations representing the artificial neural network" (cl. 23);
- "executing the user interaction stream comprises: controlling setting and editing of parameters of the computations representing the artificial neural network" (cl. 24);
- "executing the user interaction stream comprises: controlling setting and editing of parameters of the inputs to the artificial neural network" (cl. 25);
- "executing the user interaction stream comprises: specifying an output to be saved to disk and/or displayed on a screen" (cl. 26); and

- "executing the user interaction stream comprises: parsing elements to be used in the computations representing the artificial neural network" (cl. 27).

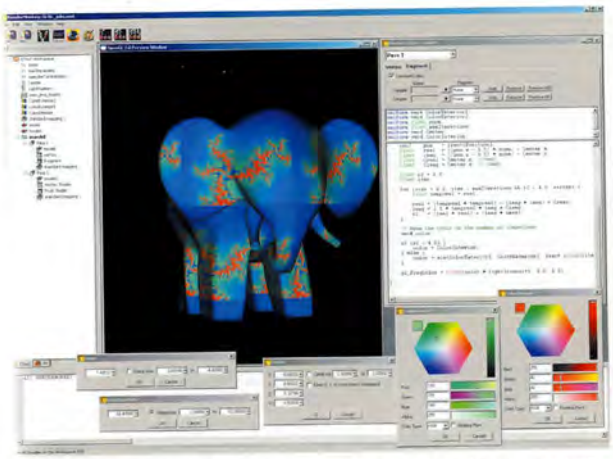
To the extent that any of the primary references do not expressly disclose these or any related limitations, a POSITA would have been motivated to modify any such references based on any of the references in Exhibits C1–C15 and Appendix C disclosing those limitations.

As explained in Section II.B.3, prior to the '461 Patent, graphics hardware manufacturers such as NVIDIA developed improved software tools for programmable GPUs, such as advanced application programming interfaces (APIs) and programming languages. These tools facilitated interactions with GPU accelerators, allowing users to trigger when tasks are executed. For example, programmers made use of NVIDIA's GPUs through graphics Application Programing Interfaces (APIs), such as Direct3D and OpenGL, as well as GPU languages such as HLSL, GLSL, and Cg.[250]

User interfaces, such as GUIs, were widely used before the '461 Patent, including for designing the simulation of artificial neural networks. As an example, nnet describes a "simple and user friendly" user interface through which users can create, edit, train, and simulate artificial neural networks.[251] Visual tools facilitated GPU programming and interacting with programs such as modeling and simulation programs. For example, the book *OpenGL Shading Language* shows a graphical user interface (GUI) that allowed a user to interactively adjust parameters used by the GPU for shading an object.

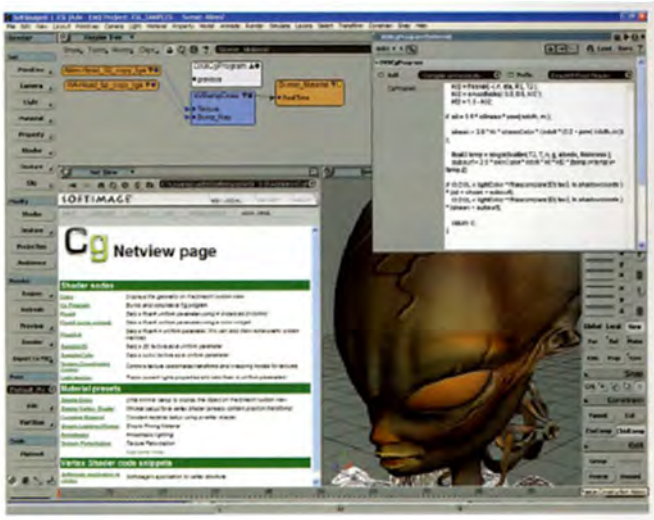
[250] NVIDIA-NAI-0002604 (GPU Gems 2: Programming Techniques for High-Performance Graphics and General-Purpose Computation (2005)) at xxxii; '461 Patent at 2:20–22.

[251] NVIDIA-NAI-0006426 (Neural Network Toolbox) at 3-23.



olor Plate 1 Screen shot of the RenderMonkey IDE user interface. The shader that is procedurally nerating a Julia set on the surface of the elephant is shown in the source code window. Color selection ols and user interface elements for manipulating user-defined uniform variables are also shown. [252]

Similarly, the book *Cg: The Cg Tutorial* describes and shows various Cg viewers that can be used to facilitate user interaction.



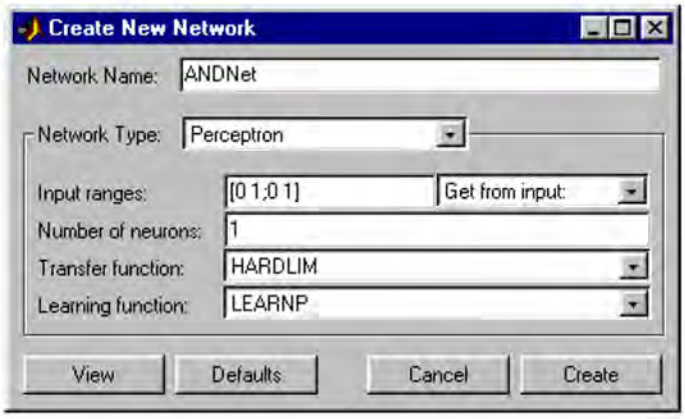
[253]

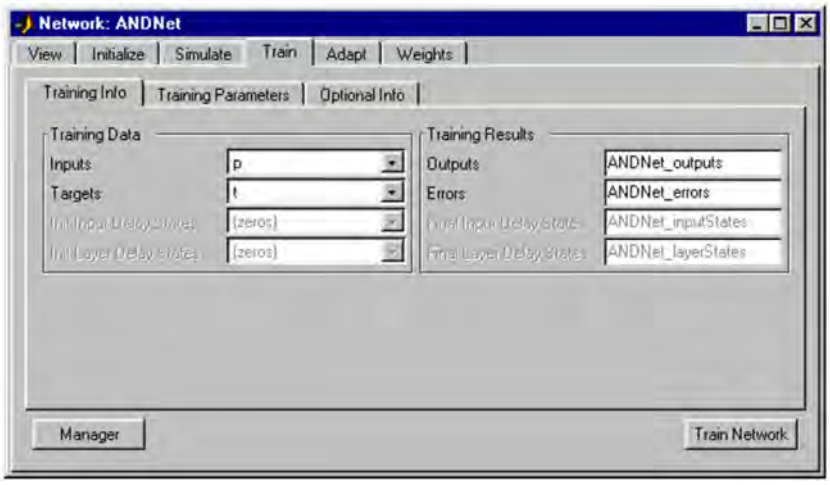
As a further example, nnet discloses an interactive GUI through which a user can define, set, or edit functions, parameters, variables, and sources of inputs to an artificial neural network,

[252] NVIDIA-NAI-0003832 (OpenGL Shading) at 27.

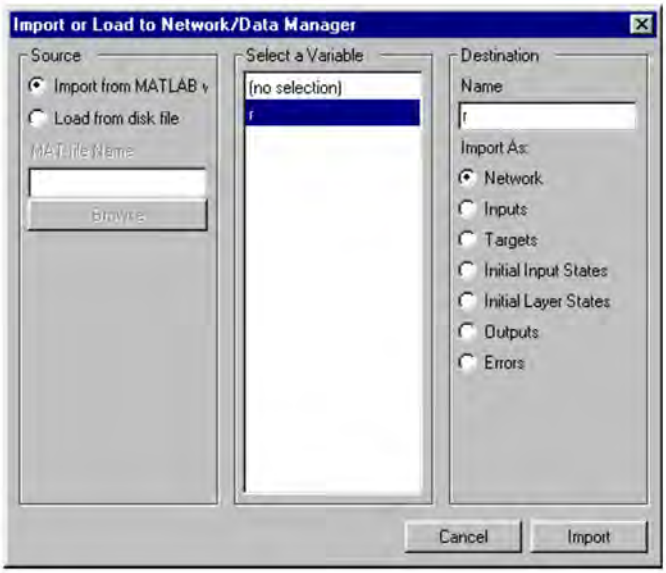
[253] NVIDIA-NAI-0005887 (Cg Tutorial) at 39–41, 44.

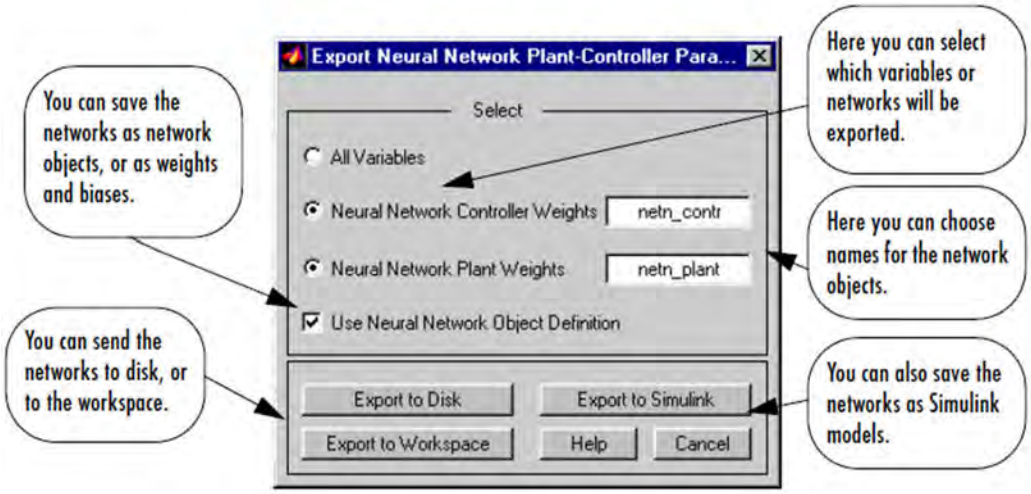
such as commands and data. The GUI also allows users to export, save, display, or otherwise manipulate outputs of the artificial neural network as well.[254]

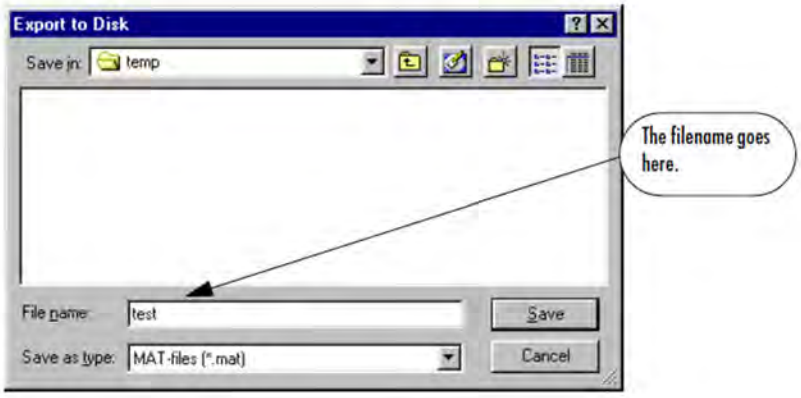


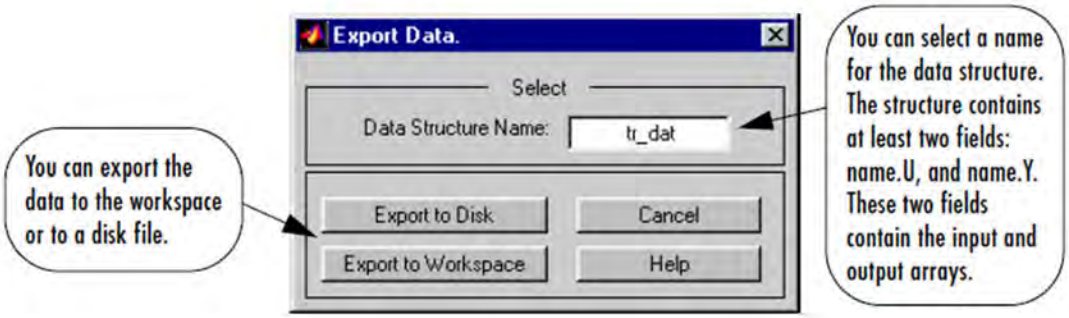


[254] NVIDIA-NAI-0006426 (Neural Network Toolbox) at 92–100, 237–243, 392–413.


Import or Load to Network/Data Manager
Source
Import from MATLAB v
Load from disk file
MAT-file Name
Browse
Select a Variable
(no selection)
r
Destination
Name
r
Import As:
Network
Inputs
Targets
Initial Input States
Initial Layer States
Outputs
Errors
Cancel
Import


Export Neural Network Plant-Controller Para...
Select
All Variables
Neural Network Controller Weights
netn_contr
Neural Network Plant Weights
netn_plant
Use Neural Network Object Definition
Export to Disk
Export to Simulink
Export to Workspace
Help
Cancel
You can save the networks as network objects, or as weights and biases.
Here you can select which variables or networks will be exported.
Here you can choose names for the network objects.
You can send the networks to disk, or to the workspace.
You can also save the networks as Simulink models.


Export to Disk
Save in:
temp
The filename goes here.
File name:
test
Save
Save as type:
MAT-files (*.mat)
Cancel



[255]

A POSITA would have been motivated to apply these principles to any of the primary references in Exhibits C1–C15. For example, a POSITA would have been motivated to implement these kinds of user interactivity (including GUIs) for simulating artificial neural networks in the accelerator systems. GUIs, such as those described above, would allow users to interact with an artificial neural network in an accessible way and input information and commands that are processed by the CPU, managed and scheduled by the scheduler, and executed by the GPU while providing visual feedback in response to user-indicated changes and display the results of the computations so that users can review, analyze, and use the results of the computations. Command line interfaces and GUIs had been ubiquitous in the field for decades by the time of the '461 Patent. The user interfaces identified above are similar to those used with graphics APIs, such as OpenGL and DirectX, that the '461 Patent explicitly characterizes as prior art. Given the common nature of such user interfaces and functions, and the benefits of them, a POSITA would have had a reasonable expectation of success of implementing a known feature (a GUI or other user interaction) into a known system (a GPU-accelerated system simulating an artificial neural network) to yield predictable results (e.g., allowing a user to easily provide input and control).

[255] NVIDIA-NAI-0006426 (Neural Network Toolbox) at 94, 95, 99, 237, 238, 241.

In view of the knowledge demonstrated in the art in Exhibits C1–C15 and Appendix C, it would have been obvious for a POSITA to implement a GUI or other user interaction as claimed, and he or she would have been motivated to do so to for the advantages described above.

d. Secondary Considerations of Non-Obviousness

A patentee bears the burden of production with respect to evidence of secondary considerations of non-obviousness. *ZUP, LLC v. Nash Mfg., Inc.*, 896 F.3d 1365, 1373 (Fed. Cir. 2018). As of the date of these Invalidity Contentions, Plaintiff has not identified any evidence of secondary considerations. NVIDIA reserves the right to present additional evidence and argument regarding secondary considerations of non-obviousness to the extent Plaintiff purports to identify any such evidence. In any event, as shown in these Invalidity Contentions, other companies and individuals described, built, and/or patented the exact same concepts in the '461 Patent before Plaintiff ever did—often many years before. Even where others' invention(s) occurred around the same time as the '867 Patent, such simultaneous invention demonstrates the obviousness of the '461 Patent.

4. Priority Date

U.S. Application No. 17/136,343 (hereinafter, "the '343 application"), which led to the '461 Patent, was filed on December 29, 2020. The '343 application was a continuation of a reissue application for U.S. Patent No. 9,189,828 ("the '828 Patent"), which was filed on January 3, 2014, and issued on November 17, 2015. The '343 application purports to claim priority to provisional appl. no. 60/826,892 (hereinafter, "the '892 provisional"), which was filed on September 25, 2006. To the extent that Plaintiff asserts that the '461 Patent is entitled to an earlier priority date based on any provisional application or documentation, Plaintiff has the burden of proof and has not demonstrated an earlier priority date.

a. Provisional Application

To the extent that Plaintiff asserts that the Asserted Claims of the '461 Patent are entitled to an earlier priority date based on the disclosure of the '892 provisional, Plaintiff has the burden of proof and has not demonstrated support for all claim limitations in the '892 provisional.

b. Asserted Conception, Reduction to Practice, and Diligence

In its Infringement Contentions, Plaintiff contends that the Asserted Claims of the '461 Patent "are entitled to a priority date corresponding to the conception of the claimed inventions, which occurred no later than February 25, 2005," and that "[t]he conception of the inventions . . . was followed by continuous diligence and actual reduction to practice of the claimed inventions no later than December 14, 2005." The documents that Plaintiff identifies in support of this contentions fail to demonstrate earlier conception, reduction to practice, or diligence.

The documents produced to date by Plaintiff fail to support earlier conception, let alone conception as early as February 25, 2005. Plaintiff's documents do not disclose several limitations of the Asserted Claims of the '461 Patent, including, without limitation, the limitations relating to an artificial neural network and the different claimed "streams." For example, the only document produced by Plaintiff [redacted] There is nothing disclosed relating to the claim limitations of the '461 Patent.

The documents produced to date also do not evidence an actual reduction to practice, let alone diligence from the alleged conception date until either an actual or constructive reduction to practice. The document produced by Plaintiff [redacted]

[redacted]

As such, the Asserted Claims of the '461 Patent are not entitled to an earlier priority date.

IV. INVALIDITY UNDER 35 U.S.C. § 112

NVIDIA alleges that the Asserted Claims are invalid under 35 U.S.C. § 112 for failing to satisfy the written description and enablement requirements (Section 112, ¶ 1) and/or as indefinite (Section 112, ¶ 2). NVIDIA reserves all rights to amend their Invalidity Contentions under 35 U.S.C. § 112, including after the Asserted Claims are ultimately construed by the Court, in response to any interpretation of the Asserted Claims embodied in Plaintiff's infringement positions, and/or to account for any changes in the law concerning invalidity under 35 U.S.C. § 112. NVIDIA also reserves the right to provide additional explanation and/or argument for their contentions under Section 112, including, for example, based on expert testimony. NVIDIA's contentions that the Asserted Claims are invalid under Section 112 are not admissions regarding the construction or scope of the claims of the Asserted Patents, or that any of the claims of the Asserted Patents are not anticipated or rendered obvious by any prior art. All of the below statements are by way of example only. Any item disclosed below with respect to any patent or claim may, as appropriate, be asserted as a defense against any other patent or claim that has a similar claim limitation and patent disclosure.

A. Legal Standards

1. Invalidity Under Pre-AIA 35 U.S.C. § 112 ¶ 1 (Written Description / Non-Enablement)

Pre-AIA 35 U.S.C. § 112 ¶ 1 requires that the specification contain a written description of the invention. "[T]he hallmark of written description is disclosure." *Boston Sci. Corp. v. Johnson & Johnson*, 647 F.3d 1353, 1361–62 (Fed. Cir. 2011) (citation omitted). The test for whether a specification adequately describes an invention is "whether the disclosure of the application relied upon reasonably conveys to those skilled in the art that the inventor had possession of the claimed subject matter as of the filing date. . . . [T]he test requires an objective inquiry into the four corners of the specification from the perspective of a person of ordinary skill in the art . . . . [It] is a question of fact." *Ariad Pharms., Inc. v. Eli Lilly and Co.*, 598 F.3d 1336, 1351 (Fed. Cir. 2010) (en banc); *Boston Sci.*, 647 F.3d at 1362.

The enablement requirement of Section 112 demands that the patent specification enable "those skilled in the art to make and use the full scope of the claimed invention without 'undue experimentation.'" *Genentech, Inc. v. Novo Nordisk A/S*, 108 F.3d 1361, 1365 (Fed. Cir. 1997) (quoting *In re Wright*, 999 F.2d 1557, 1561 (Fed. Cir. 1993)). "[T]he scope of the claims must be less than or equal to the scope of the enablement." *Nat'l Recovery Tech., Inc. v. Magnetic Separation Sys., Inc.*, 166 F.3d 1190, 1196 (Fed. Cir. 1999). In other words, the claims must be enabled over their entire claimed range. *See id.* (finding lack of enablement because the "claim [] [was] broader than the enablement as taught in the specification").

2. Invalidity Under Pre-AIA 35 U.S.C. § 112 ¶ 2 (Indefiniteness)

Claims are indefinite under pre-AIA 35 U.S.C. § 112 ¶ 2 when they "fail to inform, with reasonable certainty, those skilled in the art about the scope of the invention." *Nautilus, Inc. v. Biosig Instruments, Inc.*, 572 U.S. 898, 901 (2014). For example, to the extent an asserted

apparatus claim includes both apparatus and method limitations, that claim is invalid for indefiniteness under pre-AIA Section 112 ¶ 2 because it fails to identify or notify the public of what constitutes direct infringement. *See IPXL Holdings, L.L.C. v. Amazon.com, Inc.*, 430 F.3d 1377 (Fed. Cir. 2005). Similarly, a claim that uses means-plus function claiming is indefinite "[i]f the patentee fails to disclose adequate corresponding structure." *Williamson v. Citrix Online, LLC*, 792 F. 3d 1339, 1352 (Fed. Cir. 2015). A claim can also be found indefinite if it uses terms of degree. *See Interval Licensing LLC v. AOL, Inc.*, 766 F.3d 1364, 1371 (Fed. Cir. 2014). As the Federal Circuit has "explained, a term of degree fails to provide sufficient notice of its scope if it depends 'on the unpredictable vagaries of any one person's opinion.'" *Id.* (citation omitted). In addition, "a claim term that lacks an antecedent basis may . . . render a claim indefinite." *In re Downing*, 754 F. App'x 988, 996 (Fed. Cir. 2018).

3. Invalidity for Improper Recapture

The recapture rule prevents a patentee from reclaiming subject matter surrendered during the original prosecution of a patent when seeking a reissue patent under 35 U.S.C. § 251. The rule serves to uphold the public notice function of prosecution history estoppel and to prevent patentees from unfairly broadening their claims after issuance. Under *MBO Labs, Inc. v. Becton, Dickinson & Co.*, 602 F.3d 1306 (Fed. Cir. 2010), improper recapture occurs if a reissue application attempts to enlarge the scope of a claim to cover subject matter deliberately given up to secure the original patent.

4. Invalidity Under the Original Patent Doctrine

The original patent doctrine limits a patentee's ability to obtain a reissue patent that is not directed to the same invention as originally disclosed in the issued patent. *See Antares Pharma, Inc. v. Medac Pharma Inc.*, 771 F.3d 1354, 1359–60 (Fed. Cir. 2014). This doctrine ensures that

reissue patents correct only errors in the original patent without introducing claims to subject matter that was neither contemplated nor supported by the original disclosure.

B. Section 112 Invalidity of U.S. Patent No. 8,648,867

1. Lack of Written Description / Enablement

NVIDIA contends that the following Asserted Claims of the '867 Patent are invalid under pre-AIA 35 U.S.C. § 112 ¶ 1. Each Asserted Claim identified below (and each Asserted Claim that depends therefrom) is invalid under Section 112, ¶ 1 because the specification of the Asserted Patent fails to provide a sufficient written description and/or enabling disclosure. For each listed term or phrase, NVIDIA believes the term or phrase is invalid under pre-AIA 35 U.S.C. § 112 ¶ 1, as is any limitation including such terms or phrases, for the same reason(s).

- "transferring, by an accelerator controller, the first input data into a first partition, referenced by first pointer, of an accelerator memory before a first computational cycle of the numerical simulation" (cl. 16)
- "storing, by the accelerator controller, the first output data into a second partition, referenced by a second pointer, of the accelerator memory" (cl. 16)
- "partitioning the accelerator memory into the first partition, the second partition, a third partition to store internal variables, and a fourth partition to store data used as input at a particular computation cycle of the numerical simulation" (cl. 18)
- "transferring, by the accelerator controller, the first output data to the main memory during the second computational cycle" (cl. 19)

2. Indefiniteness

NVIDIA contends that the following Asserted Claims of the '867 Patent are invalid under pre-AIA 35 U.S.C. Section 112 ¶ 2. Each Asserted Claim identified below (and each Asserted Claim that depends therefrom) is invalid under pre-AIA Section 112 ¶ 2 because they fail to

inform, with reasonable certainty, those skilled in the art about the scope of the claimed invention. For each listed term or phrase, NVIDIA believes the term or phrase is invalid under pre-AIA Section 112 ¶ 2, and any limitation including such term or phrase is also indefinite. Therefore, identification of a term or phrase below necessarily identifies any broad term/phrase or claim limitation in which the identified term or phrase appears.

- "transferring, by an accelerator controller, the first input data into a first partition, referenced by first pointer, of an accelerator memory before a first computational cycle of the numerical simulation" (cl. 16)
- "performing, by at least one graphics processing unit during the first computational cycle, at least one calculation on the first portion of the input data as to generate first output data" (cl. 16)
- "storing, by the accelerator controller, the first output data into a second partition, referenced by a second pointer, of the accelerator memory" (cl. 16)
- "sending, by the accelerator controller, instructions for performing the at least one calculation to the at least one graphics processing unit" (cl. 17)
- "partitioning the accelerator memory into the first partition, the second partition, a third partition to store internal variables, and a fourth partition to store data used as input at a particular computation cycle of the numerical simulation" (cl. 18)
- "transferring, by the accelerator controller, the first output data to the main memory during the second computational cycle" (cl. 19)

C. Section 112 Invalidity of U.S. Patent No. RE48,438

1. Lack of Written Description / Enablement

NVIDIA contends that the following Asserted Claims of the '438 Patent are invalid under pre-AIA 35 U.S.C. § 112 ¶ 1. Each Asserted Claim identified below (and each Asserted Claim that depends therefrom) is invalid under Section 112, ¶ 1 because the specification of the Asserted Patent fails to provide a sufficient written description and/or enabling disclosure. For each listed term or phrase, NVIDIA believes the term or phrase is invalid under pre-AIA 35 U.S.C. § 112 ¶ 1, as is any limitation including such terms or phrases, for the same reason(s).

- "at least one graphics processing unit to perform a sequence of computations on the at least a portion of the input data so as to generate output data, the sequence of computations representing an artificial neural network, intermediate computations in the sequence of computations representing respective layers of the artificial neural network and yielding intermediate results" / "A method of performing a sequence of computations representing an artificial neural network on a computer system…performing, by the GPU, the sequence of computations on a first portion of the input data so as to generate a first portion of the output data" / "computations in the sequence of computations representing respective layers of neurons in the artificial neural network" / "performing, by the GPU, the sequence of computations on a first portion of the input data so as to generate a first portion of the output data…intermediate computations in the sequence of computations yielding intermediate results" / "a processing unit, operably coupled to the memory, to perform, during computation of the first layer of neurons, at least one calculation on the first input data so as to generate first output data, the first output data representing an output of at least one neuron in the first layer of neurons" / "at least one processing unit to perform a sequence of computations representing an artificial neural network on the at least a portion

of the input data so as to generate output data, intermediate computations in the sequence of computations representing layers of the neural network and yielding intermediate results" (cls. 1, 12, 21, 40, 44)

- "accelerator memory, operably coupled to the at least one [graphics] processing unit, to store the results of the sequence of computations" (cls. 1, 44)
- "the central processing unit is configured to receive the input data at a first rate; and the at least one graphics processing unit is configured to perform the sequence of computations at a second rate different than the first rate" / "receiving the input data at a first rate; and wherein (A) comprises performing the sequence of computations at a second rate different than the first rate" / "the central processing unit is configured to receive the input data at a first rate; and the at least one processing unit is configured to perform the sequence of computations at a second rate different than the first rate" (cls. 3, 14, 46)
- "wherein the controller is configured to transfer the output data from the accelerator memory to the main memory without transferring any of the intermediate results from the accelerator memory to the main memory so as to reduce data transfer via the bus" / "transferring the second portion of the output data from the accelerator memory to the main memory without transferring any of the intermediate results of the plurality of sequential computations from the accelerator memory to the main memory so as to reduce data transfer via the bus" / "wherein transferring the first output data from the memory to the other memory occurs without transferring the intermediate results of the sequence of computations" (cls. 7, 16, 25)
- "accessing, by the GPU, the output of the previous computation during performance of the computation in the sequence of computations" (cl. 30)

- "a memory, operably coupled to the CPU, to store the first input data in a first partition, referenced by a first pointer, before computing a first layer of neurons of the artificial neural network" (cl. 40)
- "store the first output data in a second partition of the memory, the second partition referenced by a second pointer, and to swap the first pointer with the second pointer at the end of the computation of the first layer of neurons, such that the first output data becomes an input for a second layer of neurons of the artificial neural network" (cl. 40)

2. Indefiniteness

NVIDIA contends that the following Asserted Claims of the '438 Patent are invalid under pre-AIA 35 U.S.C. Section 112 ¶ 2. Each Asserted Claim identified below (and each Asserted Claim that depends therefrom) is invalid under pre-AIA Section 112 ¶ 2 because they fail to inform, with reasonable certainty, those skilled in the art about the scope of the claimed invention. For each listed term or phrase, NVIDIA believes the term or phrase is invalid under pre-AIA Section 112 ¶ 2, and any limitation including such term or phrase is also indefinite. Therefore, identification of a term or phrase below necessarily identifies any broad term/phrase or claim limitation in which the identified term or phrase appears.

- "at least one graphics processing unit to perform a sequence of computations on the at least a portion of the input data so as to generate output data . . . and a controller . . . to transfer at least a portion of the output data from the accelerator memory to the main memory during performance of the intermediate computations in the sequence of computations by the at least one graphics processing unit" (cl. 1)
- "accelerator memory, operably coupled to the at least one [graphics] processing unit, to store the results of the sequence of computations" (cls. 1, 44)

- "to transfer at least a portion of the output data [from the accelerator memory to the main memory]" (cls. 1, 8, 9, 44, 51, 52)
- "the central processing unit is configured to receive the input data at a first rate; and the at least one graphics processing unit is configured to perform the sequence of computations at a second rate different than the first rate" (and related terms) (cls. 3, 14, 46)
- "The computer system of claim 1, wherein the controller is configured to transfer at least a portion of the output data from the accelerator memory to the main memory after the at least one graphics processing unit has begun to perform another sequence of computations." (cls. 8, 51)
- "The computer system of claim 8, wherein the controller is configured to initiate transfer of the at least a portion of the input data and to transfer the at least a portion of the output data in parallel with performance of at least one computation in the other sequence of computations by the at least one graphics processing unit." (cls. 9, 52)
- "A method of performing a sequence of computations representing an artificial neural network on a computer system . . . the method comprising: (A) performing, by the GPU, the sequence of computations on a first portion of the input data so as to generate a first portion of the output data, the first portion of the output data representing an output of a neuron in a first layer of the artificial neural network, intermediate computations in the sequence of computations yielding intermediate results, . . . (C) in parallel with performing the sequence of computations by the GPU in (A), transferring a second portion of the output data from the accelerator memory to the main memory via the bus, the second portion of the output data representing an output of a neuron in a second layer in the artificial neural network" (cl. 12)

- "receiving the input data at a first rate; and wherein (A) comprising performing the sequence of computations at a second rate different than the first rate" (cl. 14)
- "receiving, at a central processing unit (CPU), first input data acquired from an external system in real time;" (cl. 21)
- "external input data patterns" (cl. 28)
- "accessing, by the GPU, the output of the previous computation during performance of the computation in the sequence of computations" (cl. 30)
- "The method of claim 21, wherein performing the first computation comprises executing a plurality of computational elements representing a layer of neurons in an artificial neural network." (cl. 31)
- "the central processing unit is configured to receive the input data at a first rate; and the at least one processing unit is configured to perform the sequence of computations at a second rate different than the first rate" (cl. 46)
- "a memory, operably coupled to the CPU, to store the first input data in a first partition, referenced by a first pointer, before computing a first layer of neurons of the artificial neural network" (cl. 40)

3. Improper Recapture

NVIDIA contends that the following Asserted Claims of the '438 Patent are invalid under 35 U.S.C. § 251 for attempting to reclaim subject matter surrendered during the original prosecution.

By way of example, during prosecution of the '828 Patent (from which the '438 Patent reissued), the applicant distinguished the claims based on recitation of "a controller . . . to transfer . . . the output data from the accelerator memory to [a] main memory during performance of [a]

sequence of computations." Not all asserted claims of the '438 Patent, however, recite a "controller" with the specific operations claimed in the '828 Patent. For example, asserted claim 12 does not recite a "controller." The '438 Patent therefore improperly attempted to recapture claim scope surrendered during prosecution of the '828 Patent.

4. Invalidity Under the Original Patent Rule

NVIDIA contends that the following Asserted Claims of the '438 Patent are invalid under the original patent doctrine because they are not directed to the same invention as the original surrendered patent.

The claims of the '828 Patent, at a high level, related to having a GPU perform a sequence of operations on a portion of input data, with requirements for transferring input and output data. During reissue, the applicant amended the claims to recite, *inter alia*, (1) use of an accelerator, such as a GPU, to perform computations "representing an artificial neural network," with each computation representing the output of a neuron in a layer of the artificial neural network, and (2) transferring input and output data between the accelerator and the host system *while* the GPU computes the layers of the artificial neural network. But this collection of concepts was not disclosed as an invention in the original '828 Patent. Indeed, the phrase "neural network" appears only four times in the specification—and never with any details about how the computations occur.[256] The '438 Patent is therefore invalid under the original patent doctrine.

D. Section 112 Invalidity of U.S. Patent No. RE49,461

1. Lack of Written Description / Enablement

NVIDIA contends that the following Asserted Claims of the '461 Patent are invalid under pre-AIA 35 U.S.C. § 112 ¶ 1. Each Asserted Claim identified below (and each Asserted Claim

[256] '438 Patent at 1:51–55, 5:42–44, 6:14–15, 9:28–33.

that depends therefrom) is invalid under Section 112, ¶ 1 because the specification of the Asserted Patent fails to provide a sufficient written description and/or enabling disclosure. For each listed term or phrase, NVIDIA believes the term or phrase is invalid under pre-AIA 35 U.S.C. § 112 ¶ 1, as is any limitation including such terms or phrases, for the same reason(s).

- "executing, by the at least one CPU, a user interaction stream, the user interaction stream controlling transfer of inputs to the artificial neural network to the first memory partition and the second memory partition" (cl. 21)
- "executing, by the processing unit, a computational stream, the computational stream controlling data exchange between the user interaction stream and the computational stream during execution of the computations representing the artificial neural network" (cl. 21)
- "shifting control of a data exchange between the user interaction stream and the computational stream" (cl. 21)

2. Indefiniteness

NVIDIA contends that the following Asserted Claims of the '461 Patent are invalid under pre-AIA 35 U.S.C. Section 112 ¶ 2. Each Asserted Claim identified below (and each Asserted Claim that depends therefrom) is invalid under pre-AIA Section 112 ¶ 2 because they fail to inform, with reasonable certainty, those skilled in the art about the scope of the claimed invention. For each listed term or phrase, NVIDIA believes the term or phrase is invalid under pre-AIA Section 112 ¶ 2, and any limitation including such term or phrase is also indefinite. Therefore, identification of a term or phrase below necessarily identifies any broad term/phrase or claim limitation in which the identified term or phrase appears.

- "executing, by the at least one CPU, a user interaction stream, the user interaction stream controlling transfer of inputs to the artificial neural network to the first memory partition and the second memory partition" (cl. 21)
- "executing, by the processing unit, a computational stream, the computational stream controlling data exchange between the user interaction stream and the computational stream during execution of the computations representing the artificial neural network" (cl. 21)
- "controlling data exchange between the user interaction stream and the computational stream" / "controls the data exchange between the user interaction stream and the computational stream" (cls. 21, 22)
- "shifting control of a data exchange between the user interaction stream and the computational stream" (cl. 21)
- "executing the user command during execution of the computations representing the artificial neural network at times determined by the computational stream" (cl. 21)
- "a data output stream" (cl. 31)
- "controlling transfer of outputs of the computations representing the artificial neural network to disk" (cl. 31)

3. Improper Recapture

NVIDIA contends that the following Asserted Claims of the '461 Patent are invalid under 35 U.S.C. § 251 for attempting to reclaim subject matter surrendered during the original prosecution.

By way of example, during prosecution of the '828 Patent (from which the '461 Patent reissued), the applicant distinguished the claims based on recitation of "a controller . . . to transfer

. . . the output data from the accelerator memory to [a] main memory during performance of [a] sequence of computations." Asserted claim 21 of the '461 Patent, however, does not recite a "controller," let alone the specific operations claimed in the '828 Patent. The Asserted Claims of the '461 Patent are therefore improperly attempting to recapture claim scope surrendered during prosecution of the '828 Patent.

4. Invalidity Under the Original Patent Rule

NVIDIA contends that the following Asserted Claims of the '461 Patent are invalid under the original patent doctrine because they are not directed to the same invention as the original surrendered patent.

The claims of the '828 Patent, at a high level, related to having a GPU perform a sequence of operations on a portion of input data, with requirements for transferring input and output data. During reissue, the applicant removed features and added others, such as the hardware implementation of the computational stream and the elimination of thread and context initialization in the form of the accelerator controller. Furthermore, the claims were also amended to recite implementing artificial neural networks. But this collection of concepts was not disclosed as an invention in the original '828 Patent. Indeed, the phrase "neural network" appears only four times in the specification—and never with any details about how the computations occur.[257] The '461 Patent is therefore invalid under the original patent doctrine.

V. INEQUITABLE CONDUCT

One or more of the named inventors or prosecuting attorney committed inequitable conduct by (1) intentionally withholding material prior art, including GPU Gems 2 and

[257] '461 Patent at 1:59–62, 5:49–51, 6:21–22, 9:35–40.

SANNDRA/KInNeSS, and (2) filing a petition to revive an abandoned patent application, falsely claiming that the abandonment was "unintentional."

A. Legal Standards

1. Inequitable Conduct

Inequitable conduct is a judicially-created equitable defense to patent infringement that derives from the equitable doctrine of unclean hands.[258] When a court determines that inequitable conduct has occurred as to one or more claims of a patent, the entire patent is rendered unenforceable.[259] Inequitable conduct generally requires a showing of (1) "materiality" and (2) a specific intent to deceive the Patent Office.[260]

a. Materiality

In *Therasense*, the Federal Circuit set forth two standards for satisfying the materiality prong of inequitable conduct.

First, in instances where a patentee deliberately withholds a prior art reference, a court uses a "but for" analysis, analyzing whether the USPTO would not have issued the patent but for the patentee withholding the prior art reference.[261] "In making this patentability determination, the court should apply the preponderance of the evidence standard and give claims their broadest reasonable construction."[262] Thus, "if a claim is properly invalidated in district court based on the deliberately withheld reference, then that reference is necessarily material because a finding of

---

[258] *Therasense, Inc. v. Becton, Dickinson & Co.*, 649 F.3d 1276, 1285 (Fed. Cir. 2011) (en banc).

[259] *Therasense*, 649 F.3d at 1287; *Kingsdown Med. Consultants, Ltd. v. Hollister Inc.*, 863 F.2d 867, 877 (Fed. Cir. 1988).

[260] Therasense, 649 F.3d at 1287.

[261] *Therasense*, 649 F.3d at 1290–91.

[262] *Therasense*, 649 F.3d at 1291–92.

invalidity in a district court requires clear and convincing evidence, a higher evidentiary burden than that used in prosecution at the PTO."[263]

Second, the material prong is met "[w]hen the patentee has engaged in affirmative acts of egregious misconduct, such as the filing of an unmistakably false affidavit."[264] An affirmative act of egregious misconduct is inherently material.[265] The filing of a false oath of inventorship,[266] the filing a false revival petition under 37 CFR § 1.137(a),[267] and the filing of a false certification of small entity status[268] are all affirmative acts of egregious misconduct. And affirmative acts of egregious misconduct are not limited to the filing of unmistakably false affidavits; courts can consider varied facts and equitable considerations on a case-by-case basis.[269]

b. Intent

Under the intent prong of the standard two-part inequitable conduct test, a party can prove intent to deceive the Patent Office based on direct evidence or on circumstantial evidence "with

---

[263] *Therasense*, 649 F.3d at 1292; *see also MONKEYmedia, Inc. v. Twentieth Century Fox Home Ent., LLC*, 242 F. Supp. 3d 551, 554 (W.D. Tex. 2017) ("Information is 'material' for purposes of inequitable conduct 'if there is a substantial likelihood that a reasonable examiner would have considered the information important in deciding whether to allow the application to issue as a patent.'") (quoting *Honeywell Int'l Inc. v. Universal Avionics Sys. Corp.*, 488 F.3d 982, 1000 (Fed. Cir. 2007).

[264] *Therasense*, 649 F.3d at 1292; *see also Rohm & Haas Co. v. Crystal Chem. Co.*, 722 F.2d 1556, 1571 (Fed. Cir. 1983); *Intellect Wireless, Inc. v. HTC Corp.*, 732 F.3d 1339, 1342 (Fed. Cir. 2013); *Apotex, Inc. v. UCB, Inc.*, 763 F.3d 1354 (Fed. Cir. 2014).

[265] *Therasense*, 649 F.3d at 1292.

[266] Advanced Magnetic Closures, Inc. v. Rome Fastener Corp., 607 F.3d 817, 828 (Fed. Cir. 2010).

[267] *In re Rembrandt Techs. LP Patent Litig.*, 899 F.3d 1254, 1272–74 (Fed. Cir. 2018).

[268] *Nilssen v. Osram Sylvania, Inc.*, 504 F.3d 1223 (Fed. Cir. 2007) ("[I]t is not beyond the authority of a district court to hold a patent unenforceable for inequitable conduct in misrepresenting one's status as justifying small entity maintenance payments.").

[269] *See, e.g.*, *Ohio Willow Wood Co. v. Alps S., LLC*, 735 F.3d 1333 (Fed. Cir. 2013) (misrepresenting opposing party's declarant's level of interest to undermine his credibility before the U.S. Patent Office is tantamount to filing an unmistakably false affidavit).

the collection of inferences permitting a confident judgment that deceit has occurred."[270] Intent to deceive "cannot be inferred solely from the fact that information was not disclosed; there must be a factual basis for a finding of deceptive intent."[271] Intent "may be proven by a showing of acts the natural consequences of which are presumably intended by the actor."[272] Intent can also be proven based on a pattern of misconduct by those owing a duty of candor to the USPTO[273] and based on whether there was one or multiple misrepresentations and/or omissions.[274]

In evaluating intent, the factfinder must consider evidence that a patent applicant withheld information from the Patent Office in good faith.[275] In short, a factfinder should only infer intent to deceive when the evidence is clear and convincing. "[T]he inference [of intent to deceive] must not only be based on sufficient evidence and be reasonable in light of that evidence, but it must also be the single most reasonable inference able to be drawn from the evidence to meet the clear and convincing standard."[276]

If a defendant demonstrates both materiality and intent by clear and convincing evidence, the Court may then determine whether the equities warrant a conclusion that the applicant's conduct warrants holding the patent unenforceable.[277]

---

[270] *McKesson Info. Sols., Inc. v. Bridge Med., Inc.*, 487 F.3d 897, 913 (Fed. Cir. 2007) (internal quotation marks omitted); *see also Merck & Co., Inc. v. Danbury Pharmacal, Inc.*, 873 F.2d 1418, 1422 (Fed. Cir. 1989) ("Intent need not, and rarely can, be proven by direct evidence.").

[271] *Purdue Pharma L.P. v. Endo Pharms., Inc.*, 438 F.3d 1123, 1134 (Fed. Cir. 2006) (citing *Hebert v. Lisle Corp.*, 49 F.3d 1109, 1116 (Fed. Cir. 1996)).

[272] *Rohm & Haas Co. v. Crystal Chem. Co.*, 722 F.2d 1556, 1571 (Fed. Cir. 1983).

[273] *Apotex, Inc. v. UCB, Inc.*, 763 F.3d 135 (Fed. Cir. 2014).

[274] *See Ohio Willow Wood Co. v. Alps S., LLC*, 735 F.3d 1333 (Fed. Cir. 2013).

[275] *See Purdue Pharma L.P. v. Endo Pharm. Inc.*, 438 F.3d 1123, 1134 (Fed. Cir. 2006).

[276] *Star Sci., Inc. v. R.J. Reynolds Tobacco Co.*, 537 F.3d 1357, 1366 (Fed. Cir. 2008).

[277] *See Leviton Mfg. Co. v. Universal Security Instruments, Inc.*, 606 F. 3d 1353, 1368 (Fed. Cir. 2010).

c. Infectious Unenforceability

A finding of inequitable conduct in relation to one patent can render related patents (those in the same family) and others that are not in the same family but are in the same technology area unenforceable where the inequitable conduct has an immediate and necessary relation to the enforcement of the asserted patent.[278] Such a finding can be made where there is a direct effect between the inequitable conduct and the cause of action for patent infringement.[279] Where an applicant's inequitable conduct in pursuit of an earlier patent enables the applicant to obtain the later patent, the later patent may be held unenforceable.[280] For example, where an inventor files a patent application based on fraud, and where that fraud allows a benefit that adheres through the later filing of other patent applications, the later-filed patents can be held unenforceable.[281]

A breach of the duty of candor early in the prosecution may also render unenforceable all claims that eventually issue from the same or related applications.[282] Inequitable conduct in relation to one patent can also result in the unenforceability of related patents where there has been "a broad pattern of inequitable conduct."[283] Accordingly, where two patents are closely related, inequitable conduct as to one can result in the unenforceability of both. Examples of close relation

---

[278] *Therasense, Inc. v. Becton, Dickinson & Co.*, 649 F.3d 1276, 1288-89 (Fed. Cir. 2011); *Consol. Aluminum Corp. v. Foseco Int'l Ltd.*, 910 F.2d 804, 810 (Fed. Cir. 1990) (citing *Keystone Driller v General Excavator Co.*, 290 U.S. 240, 245 (1933)).

[279] *Consol. Aluminum*, 910 F.2d at 810.

[280] *Consol. Aluminum*, 910 F.2d at 811.

[281] *Consol. Aluminum*, 910 F.2d at 811; *Lumenyte Intern. Corp. v. Cable Lite Corp.*, 96-1011, 1996 U.S. App. LEXIS 16400 (Fed. Cir. July 9, 1996) (a false affidavit filed to revive an abandoned patent results in the unenforceability of later-filed, related patents).

[282] *Agfa Corp. v. Creo Prods. Inc.*, 451 F.3d 1366, 1379 (Fed. Cir. 2006) (citing *Fox Indus. v. Structural Preservation Sys.*, 922 F.2d 801, 804 (Fed. Cir. 1990)).

[283] *Consol. Aluminum*, 920 F.2d at 809-812.

include similar disclosures, similar claims, and one of the patents relying on the other for priority date.[284]

2. Reviving Abandoned Patent Applications

During prosecution of a patent application, a patentee is required to respond to the USPTO within a specified amount of time.[285] Failure to respond within the prescribed period will result in the patent application being abandoned.[286] If a patent owner fails to respond within the prescribed period, the USPTO Director is authorized to accept delayed maintenance fees and to revive an expired patent under certain conditions.[287] Specifically:

> Petition requirements. A grantable petition pursuant to this section must be accompanied by:
>
> (1) The reply required to the outstanding Office action or notice, unless previously filed;
>
> (2) The petition fee as set forth in § 1.17(m);
>
> (3) Any terminal disclaimer (and fee as set forth in § 1.20(d)) required pursuant to paragraph (d) of this section; and
>
> (4) ***A statement that the entire delay in filing the required reply from the due date for the reply until the filing of a grantable petition pursuant to this section was unintentional.*** The Director may require additional information where there is a question whether the delay was unintentional.[288]

The USPTO often grants petitions to revive automatically, relying on the patentee's duty of candor when a patentee submits such a sworn statement:

> While the Office reserves the authority to require further information concerning the cause of abandonment and delay in filing a petition to revive, the Office relies

---

[284] *eSpeed v. Brokertec*, 417 F. Supp. 2d 580 (D. Del. 2006); *see also Molins PLC v. Textron*, 821 F. Supp. 1551 (D. Del. 1992) (holding unenforceable a patent due to inequitable conduct and also holding unenforceable a patent based off of a divisional application of the first and issued from the same "series of applications," because it "relies on" the first patent).

[285] *See* 37 C.F.R. § 1.134.

[286] *See* 37 C.F.R. § 1.135.

[287] *See* 37 CFR 1.137(a).

[288] 37 C.F.R. § 1.137(a).

> upon the applicant's duty of candor and good faith and accepts the statement that "the entire delay in filing the required reply from the due date for the reply until the filing of a grantable petition pursuant to 37 CFR 1.137(a) was unintentional" without requiring further information in the vast majority of petitions under 37 CFR 1.137(a). This is because the applicant is obligated under 37 CFR 11.18 to inquire into the underlying facts and circumstances when a practitioner provides this statement to the Office. In addition, providing an inappropriate statement in a petition under 37 CFR 1.137(a) to revive an abandoned application may have an adverse effect when attempting to enforce any patent resulting from the application.[289]

Thus, should a patentee submit a false statement that the entire delay was "unintentional," without which an abandoned patent application will not be revived, the patent can be held unenforceable for inequitable conduct.[290]

B. Inequitable Conduct Based on Withheld References

During prosecution of the Asserted Patents, one or more of the named inventors were aware of the references GPU Gems 2 and SANNDRA/KInNeSS, and they withheld them from the USPTO with an intent to deceive.

Knowledge of and Withholding GPU Gems 2. [redacted]

[289] Manual of Patent Examining Procedure § 711.03(c)(II)(C) (citing *Lumenyte Int'l Corp. v. Cable Lite Corp.*, Nos. 96-1011, 96-1077, 1996 WL 383927 (Fed. Cir. July 9, 1996) (unpublished) (patents held unenforceable due to a finding of inequitable conduct in submitting an inappropriate statement that the abandonment was unintentional)).

[290] *See, e.g.*, *In re Rembrandt Techs. LP Pat. Litig.*, 899 F.3d 1254, 1272–75 (Fed. Cir. 2018) (affirming finding of inequitable conduct based on false statement that failure to pay maintenance fees was "unintentional").

[redacted], GPU Gems 2 was not provided to the USPTO during prosecution of any of the Asserted Patents.

Knowledge of and Withholding SANNDRA/KInNeSS. The specification of the Asserted Patents acknowledges that "SANNDRA (Synchronous Artificial Intelligence Network Distributed Runtime Algorithm); http://www.kinness.net/Docs/SANNDRA.html) was developed to accelerate and optimize processing of numerical integration of large non-homogenous systems of differential equations."[291] The Asserted Patents reference version 2.x.x of SANNDRA as "an example practical software implementation of the method and architecture described above and pictorially represented in FIG. 3."[292] [redacted] no further information on it was provided to the examiner during prosecution of the Asserted Patents.

Materiality. Both GPU Gems 2 and SANNDRA/KInNeSS would have been highly material to the prosecutions of the Asserted Patents, and but-for the references being withheld, the Asserted Patents would not have issued. For example, Exhibits A7, A11, B7, C7, and C11 demonstrate the materiality of these withheld references to the Asserted Patents. The withheld references also are not cumulative of anything before the examiner during prosecution. For the '867 Patent, as explained in further detail below, there were only four references of record. For the reissue patents, although many references were submitted during prosecution, Exhibits A7,

[291] '867 Patent at 9:25–35.

[292] *Id.* at 9:32–35.

A11, B7, C7, and C11 show how the disclosures of GPU Gems 2 and SANNDRA/KInNeSS are unique and would have led to rejections.

Intent to Deceive. The evidence demonstrates that the most reasonable inference to draw from the facts is that the named inventors intended to deceive the USPTO by not providing these highly relevant references during prosecution. For the '867 Patent, only four references are disclosed on the face as having been considered during prosecution, and all four were identified by the *examiner* in a Notice of References Cited. That means that the named inventors did not provide *any* prior art to the USPTO during prosecution of the '867 Patent. It is not credible that the named inventors did not think that *any* prior art was material to prosecution. These facts demonstrate an intent to deceive the USPTO by not providing *any* prior art for its consideration. Thus, for the '867 Patent, by withholding all known references, the inventors may have aimed to create a misleading impression of the uniqueness and inventiveness of their claims. For the two reissue patents, the applicant took the opposite approach, flooding the examiner with *hundreds* of references, providing the appearance of being forthcoming. Yet, still missing from those hundreds of references are two that were extremely pertinent to the claims of those patents—GPU Gems 2 and SANNDRA/KInNeSS.

These facts, and others to be ascertained during discovery, demonstrate that the most reasonable inference to draw is that the named inventors intended to deceive the USPTO by withholding references during prosecution.

C. Inequitable Conduct Based on a False Declaration Regarding Abandonment

The application that issued as the '867 Patent (U.S. Patent Appl. No. 11/860,254) was filed on September 24, 2007. The USPTO issued a non-final Office Action on September 16, 2010, with a three month statutory time period for reply. The Office Action rejected all pending claims of the '254 application. No extension of time was obtained under the provisions of 37 C.F.R.

§ 1.136(a) and a response to the Office Action was due on December 17, 2010. No response was filed by the applicant and, therefore, the application became abandoned on December 17, 2010. A Notice of Abandonment was mailed to the applicant on April 12, 2011.

On July 31, 2013, the applicant filed a petition to revive the application. The petition was signed by Christopher Max Colice of Foley & Lardner LLP and included the statement that "[t]he entire delay in filing the required reply from the due date for the required reply until the filing of a grantable petition under 37 CFR 1.137(b) was unintentional." There is no indication in the statement what investigation Mr. Colice undertook or how he determined that abandonment was "unintentional."

At least Mr. Colice, and any others involved in filing the Petition for Revival, committed inequitable conduct when they caused the applicant to submit a false declaration to the USPTO stating that the delay in responding to an Office Action that resulted in the '254 application being abandoned was "unintentional." Because the '867 Patent (which issued from the '254 application) was the first patent in the family, but for the false statement in the Petition for Revival, all Asserted Patents would no longer be in force. The false statement to the USPTO is, thus, material to patentability.[293] Alternatively, or in addition, each knowingly false statement made to the USPTO constitutes an "affirmative act[] of egregious misconduct," which is inherently material.[294]

The intent of Mr. Colice and/or the named inventors to deceive the USPTO is shown from at least the length of time between the dates of the Office Action (September 24, 2007), when the application became abandoned (December 17, 2010), the Notice of Abandonment (April 12, 2011), and the applicant's Petition for Revival (July 31, 2013). For example, there is no

---

[293] *See, e.g.*, *3D Med. Imaging Sys. LLC v. Visage Imaging Inc.*, 228 F. Supp. 3d 1331, 1338–39 (N.D. Ga. 2017)

[294] *Therasense*, 649 F.3d at 1292; *In re Rembrandt*, 899 F.3d at 1272–74.

explanation why it took almost a full year from when the Office Action was issued for the applicant (Neurala LLC) to respond to the Office Action. Indeed, even after Neurala LLC was notified of the abandonment, it took more than three months to file a Petition for Revival, accompanied by only seven-pages of remarks. These facts demonstrate that there was an intent to allow the application to become abandoned. Even if Neurala LLC later changed its mind and sought to revive the application, it was too late and the Petition for Revival was based on a false statement that the long delay was "unintentional." There is no explanation anywhere in the prosecution history what kind of investigation was done to determine that the delay was "unintentional."

As noted above, the '254 application was the first application in the chain of applications for the Asserted Patents (it issued as the asserted '867 Patent). If the '254 application had been deemed abandoned and no false declaration had been filed, no patents in the asserted patent family would have ever issued. Therefore, all Asserted Patents should be rendered unenforceable due to the filing of a false declaration in the prosecution of the '867 Patent.

VI. ACCOMPANYING DOCUMENT PRODUCTION

Pursuant to the Court's Scheduling Order and the Standing Order Governing Proceedings 4.4 for Patent Cases, NVIDIA produces with these Invalidity Contentions (1) all prior art referenced in the Invalidity Contentions; and (2) technical documents sufficient to show operation of the accused products.

Dated: April 1, 2025

By *\/s/ L. Kieran Kieckhefer*

L. Kieran Kieckhefer (*pro hac vice*)
Jaysen S. Chung (*pro hac vice*)
GIBSON, DUNN & CRUTCHER LLP
One Embarcadero Center
Suite 2600
San Francisco, CA 94111-3715
(415) 393-8200
kkieckhefer@gibsondunn.com
jschung@gibsondunn.com

Brian Rosenthal (*pro hac vice*)
Ahmed ElDessouki (*pro hac vice*)
GIBSON, DUNN & CRUTCHER LLP
200 Park Avenue
New York, NY 10166-0193
(212) 351-4000
brosenthal@gibsondunn.com
aeldessouki@gibsondunn.com

Lillian J. Mao (*pro hac vice*)
GIBSON DUNN & CRUTCHER LLP
310 University Avenue
Palo Alto, CA 94301-1744
(650) 849-5307
lmao@gibsondunn.com

Barry K. Shelton (Texas State Bar No. 24055029)
SHELTON COBURN LLP
311 RR 620 S, Suite 205
Austin, TX 78734-4775
(512) 263 2165
bshelton@sheltoncoburn.com

Attorneys for Defendant
NVIDIA Corporation

CERTIFICATE OF SERVICE

I certify that the foregoing document was served electronically on April 1, 2025, on all counsel who have consented to electronic service.

/s/ L. Kieran Kieckhefer
L. Kieran Kieckhefer