

UNITED STATES PATENT AND TRADEMARK OFFICE

BEFORE THE PATENT TRIAL AND APPEAL BOARD

NVIDIA CORPORATION,
Petitioner,

v.

NEURAL AI, LLC,
Patent Owner.

Case No. IPR2025-00606
U.S. Patent No. 8,648,867

Petition for *Inter Partes* Review of U.S. Patent No. 8,648,867

Mail Stop **PATENT BOARD**
Patent Trial and Appeal Board
U.S. Patent and Trademark Office
P.O. Box 1450
Alexandria, VA 22313-1450

TABLE OF CONTENTS

	<u>Page</u>
TABLE OF AUTHORITIES	xviii
PETITIONER’S EXHIBIT LIST	xix
TABLE OF ABBREVIATIONS	xxi
MANDATORY NOTICES.....	xxii
A. Real Parties-in-Interest.....	xxii
B. Related Matters [37 C.F.R. § 42.8(b)(2)].....	xxii
C. Lead and Back-Up Counsel [37 C.F.R. § 42.8(b)(3)].....	xxii
D. Service Information [37 C.F.R. § 42.8(b)(4)]	xxiii
I. INTRODUCTION	1
II. REQUIREMENTS OF <i>INTER PARTES</i> REVIEW	2
A. Standing.....	2
B. Identification of Challenge and Relief Requested	2
1. U.S. Patent No. 7,861,060 (“Nickolls”) (Ex1004)	3
2. U.S. Patent No. 7,139,003 (“Kirk”) (Ex1005).....	3
3. GPU Gems 2: Programming Techniques for High- Performance Graphics and General-Purpose Computation (“GPU Gems”) (Ex1006).....	4
C. How the Challenged Claims Are To Be Construed Under 37 C.F.R. § 42.104(b)(3)	4
D. How the Challenged Claims Are Unpatentable Under 37 C.F.R. § 42.104(b)(4).....	5
E. Supporting Evidence Under 37 C.F.R. § 42.104(b)(5)	5
F. Payment of Fees	5

TABLE OF CONTENTS *(continued)*

	<u>Page</u>
III. TECHNICAL BACKGROUND	6
A. State of the Art Prior to the '867 Patent	6
1. Background on GPU Acceleration of Non-Graphics Computations	6
2. “Controllers” for Parallel Processing in the Prior Art	7
3. Managing Memory Through Swapping Address Pointers	8
B. Alleged Invention of the '867 Patent	9
C. Prosecution History of the '867 Patent	11
D. Person of Ordinary Skill in the Art	12
IV. GROUNDS OF UNPATENTABILITY	12
A. Ground 1: Nickolls and GPU Gems Render Obvious Claims 1–19	12
1. Overview of Ground and Prior Art	13
a. Summary of Nickolls	13
b. Summary of GPU Gems	16
2. Combination of Nickolls and GPU Gems	18
3. Detailed Analysis of Ground 1	21
a. Claim 16	21
i. Element 16[pre]: “A method for performing a numerical simulation on input data in a computer system including a central processing unit and an accelerator, the method comprising”	21

TABLE OF CONTENTS *(continued)*

	<u>Page</u>
ii. Element 16[a]: “receiving, by an accelerator, first input data from the central processing unit”	23
iii. Element 16[b]: “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”	23
iv. Element 16[c]: “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”	29
v. Element 16[d]: “storing, by the accelerator controller, the first output data into a second partition, referenced by a second pointer, of the accelerator memory”	31
vi. Element 16[e]: “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”	33
b. Claim 17: “The method as claimed in claim 16, further comprising: sending, by the accelerator controller, instructions for performing the at least one calculation to the at least one graphics processing unit.”	35

TABLE OF CONTENTS *(continued)*

	<u>Page</u>
c. Claim 18: “The method as claimed in claim 16, further comprising: 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.”	36
d. Claim 19: “The method of claim 16, further comprising: transferring, by the accelerator controller, the first output data to the main memory during the second computational cycle.”	38
e. Claim 1	39
i. Element 1[pre]: “A computer system for performing a numerical simulation over a plurality of computational cycles including at least a first computational cycle and a second computational cycle, the computer system comprising”	39
ii. Element 1[a]: “a central processing unit”	40
iii. Element 1[b]: “a main memory, operably coupled to the central processing unit, to store input data to be accessed by the central processing unit in performing the numerical simulation”	40
iv. Element 1[c]: “a video system, operably coupled to the central processing unit, to drive a video monitor to display an indication of the numerical simulation in response to the computer system performing the numerical simulation”	41

TABLE OF CONTENTS *(continued)*

	<u>Page</u>
v. Element 1[d]: “an accelerator, operably coupled to the central processing unit, to receive at least a portion of the input data from the central processing unit and to provide first output data generated during the first computational cycle to the central processing unit after a conclusion of the first computational cycle, the accelerator comprising”	42
vi. Element 1[d][i]: “at least one graphics processing unit to generate second output data, during the second computational cycle, by performing at least one calculation on the first output data”	42
vii. Element 1[d][ii]: “an accelerator memory, operably coupled to the at least one graphic processing unit, the accelerator memory comprising”	42
viii. Element 1[d][ii][1]: “a first partition, referenced by a first pointer, to store the first output data during the second computational cycle”	43
ix. Element 1[d][ii][2]: “a second partition, referenced by a second pointer, to store the second output data generated during the second computational cycle”	43
x. Element 1[e]: “an accelerator controller, operably coupled to the accelerator memory and the central processing unit”	43
xi. Element 1[e][i]: “to transfer the at least the portion of the input data into the accelerator memory before the first computational cycle”	44

TABLE OF CONTENTS *(continued)*

	<u>Page</u>
xii. Element 1[e][ii]: “to transfer the first output data from the accelerator memory to the main memory during the second computational cycle”	44
xiii. Element 1[e][iii]: “to direct the second output data into the second partition during the second computational cycle”	44
xiv. Element 1[e][iv]: “to swap the first pointer and the second pointer at the conclusion of the second computational cycle such that the second output data becomes an input for a third computational cycle of the plurality of computational cycles”	45
f. Claim 2: “The computer system as claimed in claim 1, wherein the accelerator controller is configured to dictate an order of execution of instructions to the at least one graphics processing unit.”	45
g. Claim 3: “The computer system as claimed in claim 1, wherein the accelerator controller comprises an interface controller to communicate with the central processing unit over a bus of the computer system.”	46
h. Claim 4: “The computer system as claimed in claim 1, wherein the accelerator memory comprises a texture memory bank to store the at least the portion of the input data and the first output data and a shader memory bank to store instructions for performing a set of operations to be performed on the at least the portion of the input data by the at least one graphic processing unit.”	46

TABLE OF CONTENTS *(continued)*

	<u>Page</u>
i. Claim 5: “The computer system as claimed in claim 4, wherein the texture memory is partitioned into the first partition, the second partition, a third partition to store internal variables, a fourth partition to store data textures used as input at a particular computation cycle of the plurality of computational cycles.”	47
j. Claim 6: “The computer system as claimed in claim 1, wherein the accelerator controller inputs the at least the portion of the input data and a series of instructions into the at least one graphic processing unit, wherein the at least one graphics processing unit then executes the instructions on the at least the portion of the input data.”	48
k. Claim 7: “The computer system as claimed in claim 1, wherein the accelerator controller comprises a set of instructions stored on a memory.”	48
l. Claim 8: “The computer system as claimed in claim 1, wherein the at least the portion of the input data represents an initial condition of the numerical simulation.”	48
m. Claim 9.....	49
i. Element 9[pre]: “An accelerator system for a computer system performing a numerical simulation, the accelerator system comprising”	49
ii. Element 9[a]: “at least one graphics processing unit to generate output data by performing at least one computation during a first computational cycle of the numerical simulation”	49

TABLE OF CONTENTS *(continued)*

	<u>Page</u>
iii. Element 9[b]: “an accelerator memory, operably coupled to the at least one graphics processing unit, to store data used to perform the at least one computation”	49
iv. Element 9[c]: “an accelerator controller, operably coupled to the accelerator memory and the at least one graphics processing unit, to execute”	49
v. Element 9[c][i]: “a computational stream controlling performance of the at least one computation by the at least one graphics processing unit”	49
vi. Element 9[c][ii]: “an output stream controlling transfer of the output data from the at least one graphics processing unit to the accelerator memory during the first computational cycle”	50
vii. Element 9[c][iii]: “an input stream controlling transfer of input data to the accelerator memory for use by the at least one graphics processing unit during a second computational cycle of the numerical simulation”	50
n. Claim 10: “The accelerator system as claimed in claim 9, wherein the accelerator controller is configured to dictate an order of execution of instructions to the at least one graphics processing unit.”	51

TABLE OF CONTENTS *(continued)*

	<u>Page</u>
o. Claim 11: “The accelerator system as claimed in claim 9, wherein the accelerator controller is configured to send instructions to the at least one graphics processing unit, wherein the at least one graphics processing unit is configured to execute the instructions on the input data, and wherein the accelerator controller is configured to transfer the output data from the accelerator memory to a main memory during the execution of the instructions by the at least one graphics processing unit.”.....	51
p. Claim 12: “The accelerator system as claimed in claim 9, wherein the accelerator memory comprises a texture memory bank to store the input data and the output data and a shader memory bank to store instructions for processing operations to be performed on the input data by the at least one graphic processing unit.”	51
q. Claim 13: “The accelerator system as claimed in claim 12, wherein the texture memory is partitioned into a first partition to store the input data, a second partition to store internal variables, a third partition to store data textures used as input at a particular computation cycle of the numerical simulation, and a fourth partition to store the output data.”	51
r. Claim 14: “The accelerator system as claimed in claim 9, wherein the accelerator controller is configured to perform successive computational cycles of the numerical simulation by feeding the output data generated by the at least one graphic processing unit from a previous computational cycle and the input data for a next computational cycle into the at least one graphic processing unit.”	51

TABLE OF CONTENTS *(continued)*

	<u>Page</u>
s. Claim 15: “The accelerator system as claimed in claim 9, wherein the accelerator controller comprises a set of instructions stored in a memory.”	51
B. Ground 2: Kirk and GPU Gems Render Obvious Claims 1–19	52
1. Overview of Ground and Prior Art	52
a. Summary of Kirk	52
b. Summary of GPU Gems	53
2. Combination of Kirk and GPU Gems	53
3. Detailed Analysis of Ground 2	58
a. Claim 16	58
i. Element 16[pre]: “A method for performing a numerical simulation on input data in a computer system including a central processing unit and an accelerator, the method comprising”	58
ii. Element 16[a]: “receiving, by an accelerator, first input data from the central processing unit”	59
iii. Element 16[b]: “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”	60
iv. Element 16[c]: “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”	63

TABLE OF CONTENTS *(continued)*

	<u>Page</u>
v. Element 16[d]: “storing, by the accelerator controller, the first output data into a second partition, referenced by a second pointer, of the accelerator memory”	65
vi. Element 16[e]: “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”	67
b. Claim 17: “The method as claimed in claim 16, further comprising: sending, by the accelerator controller, instructions for performing the at least one calculation to the at least one graphics processing unit.”	67
c. Claim 18: “The method as claimed in claim 16, further comprising: 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.”	68
d. Claim 19: “The method of claim 16, further comprising: transferring, by the accelerator controller, the first output data to the main memory during the second computational cycle.”	69
e. Claim 1	71
i. Element 1[pre]: “A computer system for performing a numerical simulation over a plurality of computational cycles including at least a first computational cycle and a second computational cycle, the computer system comprising”	71

TABLE OF CONTENTS *(continued)*

	<u>Page</u>
ii. Element 1[a]: “a central processing unit”	71
iii. Element 1[b]: “a main memory, operably coupled to the central processing unit, to store input data to be accessed by the central processing unit in performing the numerical simulation”	71
iv. Element 1[c]: “a video system, operably coupled to the central processing unit, to drive a video monitor to display an indication of the numerical simulation in response to the computer system performing the numerical simulation”	72
v. Element 1[d]: “an accelerator, operably coupled to the central processing unit, to receive at least a portion of the input data from the central processing unit and to provide first output data generated during the first computational cycle to the central processing unit after a conclusion of the first computational cycle, the accelerator comprising”	73
vi. Element 1[d][i]: “at least one graphics processing unit to generate second output data, during the second computational cycle, by performing at least one calculation on the first output data”	74
vii. Element 1[d][ii]: “an accelerator memory, operably coupled to the at least one graphic processing unit, the accelerator memory comprising”	74

TABLE OF CONTENTS *(continued)*

	<u>Page</u>
viii. Element 1[d][ii][1]: “a first partition, referenced by a first pointer, to store the first output data during the second computational cycle”	74
ix. Element 1[d][ii][2]: “a second partition, referenced by a second pointer, to store the second output data generated during the second computational cycle”	74
x. Element 1[e]: “an accelerator controller, operably coupled to the accelerator memory and the central processing unit”	75
xi. Element 1[e][i]: “to transfer the at least the portion of the input data into the accelerator memory before the first computational cycle”	76
xii. Element 1[e][ii]: “to transfer the first output data from the accelerator memory to the main memory during the second computational cycle”	76
xiii. Element 1[e][iii]: “to direct the second output data into the second partition during the second computational cycle”	76
xiv. Element 1[e][iv]: “to swap the first pointer and the second pointer at the conclusion of the second computational cycle such that the second output data becomes an input for a third computational cycle of the plurality of computational cycles”	77
f. Claim 2: “The computer system as claimed in claim 1, wherein the accelerator controller is configured to dictate an order of execution of instructions to the at least one graphics processing unit.”	77

TABLE OF CONTENTS *(continued)*

	<u>Page</u>
g. Claim 3: “The computer system as claimed in claim 1, wherein the accelerator controller comprises an interface controller to communicate with the central processing unit over a bus of the computer system.”78	78
h. Claim 4: “The computer system as claimed in claim 1, wherein the accelerator memory comprises a texture memory bank to store the at least the portion of the input data and the first output data and a shader memory bank to store instructions for performing a set of operations to be performed on the at least the portion of the input data by the at least one graphic processing unit.”78	78
i. Claim 5: “The computer system as claimed in claim 4, wherein the texture memory is partitioned into the first partition, the second partition, a third partition to store internal variables, a fourth partition to store data textures used as input at a particular computation cycle of the plurality of computational cycles.”79	79
j. Claim 6: “The computer system as claimed in claim 1, wherein the accelerator controller inputs the at least the portion of the input data and a series of instructions into the at least one graphic processing unit, wherein the at least one graphics processing unit then executes the instructions on the at least the portion of the input data.”79	79
k. Claim 7: “The computer system as claimed in claim 1, wherein the accelerator controller comprises a set of instructions stored on a memory.”80	80
l. Claim 8: “The computer system as claimed in claim 1, wherein the at least the portion of the input data represents an initial condition of the numerical simulation.”80	80
m. Claim 9.....81	81

TABLE OF CONTENTS *(continued)*

	<u>Page</u>
i. Element 9[pre]: “An accelerator system for a computer system performing a numerical simulation, the accelerator system comprising” ..81	
ii. Element 9[a]: “at least one graphics processing unit to generate output data by performing at least one computation during a first computational cycle of the numerical simulation”81	
iii. Element 9[b]: “an accelerator memory, operably coupled to the at least one graphics processing unit, to store data used to perform the at least one computation”81	
iv. Element 9[c]: “an accelerator controller, operably coupled to the accelerator memory and the at least one graphics processing unit, to execute”81	
v. Element 9[c][i]: “a computational stream controlling performance of the at least one computation by the at least one graphics processing unit”81	
vi. Element 9[c][ii]: “an output stream controlling transfer of the output data from the at least one graphics processing unit to the accelerator memory during the first computational cycle”81	
vii. Element 9[c][iii]: “an input stream controlling transfer of input data to the accelerator memory for use by the at least one graphics processing unit during a second computational cycle of the numerical simulation”82	

TABLE OF CONTENTS *(continued)*

	<u>Page</u>
n. Claim 10: “The accelerator system as claimed in claim 9, wherein the accelerator controller is configured to dictate an order of execution of instructions to the at least one graphics processing unit.”	82
o. Claim 11: “The accelerator system as claimed in claim 9, wherein the accelerator controller is configured to send instructions to the at least one graphics processing unit, wherein the at least one graphics processing unit is configured to execute the instructions on the input data, and wherein the accelerator controller is configured to transfer the output data from the accelerator memory to a main memory during the execution of the instructions by the at least one graphics processing unit.”	82
p. Claim 12: “The accelerator system as claimed in claim 9, wherein the accelerator memory comprises a texture memory bank to store the input data and the output data and a shader memory bank to store instructions for processing operations to be performed on the input data by the at least one graphic processing unit.”	82
q. Claim 13: “The accelerator system as claimed in claim 12, wherein the texture memory is partitioned into a first partition to store the input data, a second partition to store internal variables, a third partition to store data textures used as input at a particular computation cycle of the numerical simulation, and a fourth partition to store the output data.”	83

TABLE OF CONTENTS *(continued)*

	<u>Page</u>
r. Claim 14: “The accelerator system as claimed in claim 9, wherein the accelerator controller is configured to perform successive computational cycles of the numerical simulation by feeding the output data generated by the at least one graphic processing unit from a previous computational cycle and the input data for a next computational cycle into the at least one graphic processing unit.”	83
s. Claim 15: “The accelerator system as claimed in claim 9, wherein the accelerator controller comprises a set of instructions stored in a memory.”	83
V. ENABLEMENT AND SECONDARY CONSIDERATIONS	83
VI. PTAB DISCRETION SHOULD NOT PRECLUDE INSTITUTION.....	84
A. Discretionary Denial Under Section 314(a) Is Not Warranted.....	84
B. Discretionary Denial Under Section 325(d) Is Not Warranted.....	84
VII. CONCLUSION.....	85
APPENDIX: CHALLENGED CLAIM LISTING	88

TABLE OF AUTHORITIES

	<u>Page(s)</u>
Cases	
<i>Ohio Willow Wood Co. v. Alps South, LLC</i> , 735 F.3d 1333 (Fed. Cir. 2013)	84
<i>Sotera Wireless, Inc. v. Masimo Corp.</i> , IPR2020-01019, Paper 12 (P.T.A.B. Dec. 1, 2020)	84

PETITIONER’S EXHIBIT LIST

Exhibit	Description
1001	U.S. Patent No. 8,648,867
1002	File History for U.S. Patent No. 8,648,867 (Appl. No. 11/860,254)
1003	Declaration of Prof. Tajana Rosing, Ph.D.
1004	U.S. Patent No. 7,861,060 (“Nickolls”)
1005	U.S. Patent No. 7,139,003 (“Kirk”)
1006	GPU Gems 2: Programming Techniques for High-Performance Graphics and General-Purpose Computation (2005)
1007	Plaintiff’s Disclosure of Asserted Claims and Preliminary Infringement Contentions Pursuant to the Court’s Standing Order Governing Patent Proceedings (Jan. 14, 2025)
1008	Declaration of Dr. Mary Bolin
1009	<i>The C programming Language</i> (1988)
1010	Excerpts of Patent Owner’s Infringement Contentions
1011	Numerical Recipes in C (2d ed. 2002)
1012	Jeanne Martin, <i>Fortran 90 Pointers vs. “Cray” Pointers</i> , 11 ACM SIGPLAN Fortran Forum (1992)
1013	Arthur Veen, <i>Dataflow Machine Architecture</i> (1986)
1014	Michael Flynn, <i>Some Computer Organizations and Their Effectiveness</i> (1972)
1015	Press Release – NVIDIA Launches the World’s First Graphics Processing Unit; GeForce 256 (Aug. 31, 1999)
1016	Excerpts of OpenGL Shading Language (2004)
1017	Excerpts of The Cg Tutorial: The Definitive Guide to Programmable Real-Time Graphics (2003)
1018	OpenGL 2.1 Reference Pages
1019	Advanced Image Processing with DirectX 9 Pixel Shaders (2004)
1020	GPGPU: Basic Math Tutorial
1021	Ian Buck, <i>Data Parallel Computation on Graphics Hardware</i> (2003)

Exhibit	Description
1022	Youquan Liu, <i>Real-Time 3D Fluid Simulation on GPU with Complex Obstacles</i> (2004)
1023	Kyoung-Su Oh, <i>GPU implementation of neural networks</i> (2004)
1024	Zhongwen Luo, <i>Artificial Neural Network Computation on Graphic Process Unit</i> (2004)
1025	Thomas Rolfes, <i>Artificial Neural Networks on Programmable Graphics Hardware</i> in Game Programming Gems 4 (2004)
1026	P.J.G. Lisboa, <i>A review of evidence of health benefits from artificial neural networks in medical intervention</i> (2002)
1027	U.S. Patent Publ. No. 2003/0140179 (“Wilt”)
1028	Bertil Svensson, <i>SIMD Processor Array Architectures</i> in PARALLEL PROCESSING IN INDUSTRIAL REAL-TIME APPLICATIONS (1992)
1029	Michael Glover, <i>A Massively-Parallel SIMD Processor for Neural Network and Machine Vision Applications</i> (1993)
1030	Francisco Mesa-Martinez, <i>The UCSC Kestrel High Performance SIMD Processor: Present and Future</i> (2003)
1031	<i>Sotera Stipulation</i> (Feb. 21, 2025)

TABLE OF ABBREVIATIONS

Abbreviation	Term
'254 application	U.S. Patent Appl. No. 11/860,254
'867 patent	U.S. Patent No. 8,648,867
Board	Patent Trial and Appeal Board
Challenged Claims	Claims 1–19 of U.S. Patent No. 8,648,867
Ex.	Exhibit
Fig.	Figure
GPU	graphics processing unit
GPU Gems	GPU Gems 2: Programming Techniques for High-Performance Graphics and General-Purpose Computation (2005) (Ex1006)
Interim Guidance	K. Vidal, Interim Procedure for Discretionary Denials in AIA Post-Grant Proceedings with Parallel District Court Litigation (June 21, 2022)
IPR	<i>inter partes</i> review
Kirk	U.S. Patent No. 7,139,003 (Ex1005)
Litigation	<i>Neural AI, LLC v. NVIDIA Corporation</i> , No. 7:24-cv-00221 (W.D. Tex.)
Nickolls	U.S. Patent No. 7,861,060 (Ex1004)
Patent Owner	Neural AI, LLC
Petitioner	NVIDIA Corporation
POSITA[s]	person[s] of ordinary skill in the art
USPTO	United States Patent and Trademark Office

MANDATORY NOTICES

A. Real Parties-in-Interest

NVIDIA Corporation is the real party-in-interest.

B. Related Matters [37 C.F.R. § 42.8(b)(2)]

The '867 patent is asserted in *Neural AI, LLC v. NVIDIA Corporation*, No. 7:24-cv-00221 (W.D. Tex.) (“the Litigation”). Petitioner is concurrently filing petitions in the following proceedings as to patents related to the '867 patent and asserted in the Litigation: IPR2025-00608 (U.S. Patent No. RE49,461), IPR2025-00609 (U.S. Patent No. RE48,438), and IPR2025-00610 (U.S. Patent No. RE48,438).

C. Lead and Back-Up Counsel [37 C.F.R. § 42.8(b)(3)]

Pursuant to 37 C.F.R. §§ 42.8(b)(3), 42.8(b)(4), and 42.10(a), Petitioner provides the following designation of counsel:

Lead Counsel for Petitioner	Back-Up Counsel for Petitioner
Brian M. Buroker (Reg. No. 39,125) Gibson, Dunn & Crutcher LLP 1050 Connecticut Ave. NW Washington, DC 20036 Phone: (202) 955-8500 Fax: (202) 467-0539 Email: bburoker@gibsondunn.com	L. Kieran Kieckhefer (<i>pro hac vice</i> forthcoming) One Embarcadero Center Suite 2600 San Francisco, CA 94111 Phone: (415) 393-8200 Email: kkieckhefer@gibsondunn.com

Lead Counsel for Petitioner	Back-Up Counsel for Petitioner
	<p>Nathan Curtis (Reg. No. 70,471) Gibson, Dunn & Crutcher LLP 2001 Ross Avenue Suite 2100 Dallas, TX 75201 Phone: (214) 698-3423 Fax: (214) 571-2961 Email: ncurtis@gibsondunn.com</p> <p>Vivian Lu (Reg. No. 74,443) Gibson, Dunn & Crutcher LLP 200 Park Avenue New York, NY 10166 Phone: (212) 351-3827 Email: vlu@gibsondunn.com</p>

A Power of Attorney accompanies this Petition in accordance with 37 C.F.R. § 42.10(b).

D. Service Information [37 C.F.R. § 42.8(b)(4)]

Service via hand delivery or postal mail may be made at the addresses of the lead and back-up counsel above. Petitioner hereby consents to electronic service, and service by email may be made at:

- GDC-NVIDIA-IPR@gibsondunn.com

I. INTRODUCTION

Petitioner hereby requests IPR of U.S. Patent No. 8,648,867 (Ex1001) and cancellation of claims 1–19 as unpatentable under 35 U.S.C. § 103.

The '867 patent is the first in a family of patents directed to a system for performing *general-purpose* computing (such as numerical simulations) on traditionally *special-purpose* graphics processing units (“GPUs”). The patent does not claim to have invented performing general-purpose computing on GPUs, *id.*, 1:21–30, or any specialized hardware for doing so, *id.*, 14:13:63–67. 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. *Id.*, 2:9–20. The USPTO issued the '867 patent solely on the basis that the use of the claimed “accelerator controller” was not disclosed in the scant four references considered during prosecution.

This Petition shows the use of an “accelerator controller” to perform numerical simulations was disclosed before the '867 patent in NVIDIA’s own patents, Nickolls and Kirk, that were not considered during prosecution. Indeed, NVIDIA is a recognized pioneer in the technology of the '867 patent.

Aside from the alleged point of novelty, the claims merely recite well-known techniques, such as swapping pointers to output and input data addresses so the output of one computational cycle becomes the input of a subsequent cycle. Swapping pointers was already a common memory management technique used

throughout computer programming, including GPUs, before the '867 patent. The third reference in this Petition—GPU Gems—is a book published by NVIDIA that describes and provides the code for using such a technique in a numerical simulation.

The combination of either NVIDIA patent with the NVIDIA book renders each of the claims unpatentable. Petitioner therefore respectfully requests the Board institute IPR and invalidate all claims of this patent.

II. REQUIREMENTS OF *INTER PARTES* REVIEW

A. Standing

Pursuant to 37 C.F.R. § 42.104(a), Petitioner certifies the '867 patent is available for IPR and that Petitioner is not barred or estopped from requesting an IPR on the grounds identified herein.

B. Identification of Challenge and Relief Requested

Pursuant to 37 C.F.R. § 42.104(b), Petitioner requests the Board institute IPR of claims 1–19 of the '867 patent under pre-AIA 35 U.S.C. § 103.

The precise relief requested by Petitioner is that claims 1–19 be canceled based on the grounds below:

Ground	Claims	Basis for Rejection¹
1	1–19	Obviousness over Nickolls in view of GPU Gems
2	1–19	Obviousness over Kirk in view of GPU Gems

¹ All obviousness grounds include the knowledge of a POSITA.

These challenges rely on the following references:

1. U.S. Patent No. 7,861,060 (“Nickolls”) (Ex1004)

Nickolls issued on December 28, 2010, from a non-provisional application filed on December 15, 2005. Ex1004, 1. Nickolls is assigned to Petitioner NVIDIA. Nickolls qualifies as prior art to the '867 patent under pre-AIA 35 U.S.C. § 102(e).² Nickolls was not of record during prosecution.

2. U.S. Patent No. 7,139,003 (“Kirk”) (Ex1005)

Kirk issued on November 21, 2006, from a non-provisional application filed on December 15, 2003. Ex1005, 1. Kirk is assigned to Petitioner NVIDIA. Kirk qualifies as prior art to the '867 patent under pre-AIA 35 U.S.C. § 102(e). Kirk was not of record during prosecution.

² In the Litigation, Patent Owner argues that conception occurred by February 25, 2005. Ex1007, 8. Petitioner disagrees that Patent Owner can show conception before Nickolls' filing on December 15, 2005. In any event, the other primary reference (Kirk) is prior art to the '867 patent under pre-AIA 35 U.S.C. § 102(e) even in view of Patent Owner's asserted conception.

3. GPU Gems 2: Programming Techniques for High-Performance Graphics and General-Purpose Computation (“GPU Gems”) (Ex1006)

GPU Gems was published in March 2005 and was publicly available no later than July 2005. Ex1008, ¶¶19–25. GPU Gems’ “[f]irst printing” was March 2005. Ex1006, 6. It was cataloged and indexed at the University of Texas–Austin and Brigham Young University libraries by July 28, 2005. Ex1008, ¶¶23–28. A POSITA, exercising reasonable diligence and searching for references regarding programming GPUs, would have located GPU Gems by around July 2005. *Id.*, ¶¶26–28. For example, a skilled researcher would have located GPU Gems by searching for keywords like “GPU” and “programming,” which are contained in the searchable title, or “[c]omputer graphics” or “[r]eal-time programming,” which are contained in the searchable subject headings for GPU Gems. *Id.* GPU Gems therefore qualifies as prior art to the ’867 patent under pre-AIA 35 U.S.C. § 102(b). GPU Gems was not of record during prosecution.

C. How the Challenged Claims Are To Be Construed Under 37 C.F.R. § 42.104(b)(3)

Petitioner submits that no construction of any claim term is necessary for the Board to resolve, and the Challenged Claims would have been obvious under any reasonable construction.

D. How the Challenged Claims Are Unpatentable Under 37 C.F.R. § 42.104(b)(4)

An explanation of how the Challenged Claims are unpatentable under the statutory grounds identified above, including the identification of where each element of the claim is found in the prior art, is provided below.

E. Supporting Evidence Under 37 C.F.R. § 42.104(b)(5)

The exhibit numbers of the supporting evidence relied upon and the relevance of the evidence to the Challenged Claims, including an identification of specific portions of the evidence that support the challenge, are provided below. A list of exhibits with the exhibit number and a brief description of each exhibit is also included in this Petition pursuant to 37 C.F.R. § 42.63(e). The technical information and grounds for rejection explained in this Petition are further supported by the Declaration of Dr. Tajana Rosing (Ex1003).

F. Payment of Fees

Pursuant to 37 C.F.R. §§ 42.103 and 42.15(a), the required fee is being submitted herewith. The Office is authorized to charge any fee deficiency, or credit overpayment, to deposit account no. 50-1408.

III. TECHNICAL BACKGROUND

A. State of the Art Prior to the '867 Patent

1. Background on GPU Acceleration of Non-Graphics Computations

Parallel processing, for both graphics and general-purpose computing, was known for decades before the '867 patent. *See* Ex1003, ¶¶31–86. In 1999, NVIDIA developed the first consumer-level graphics card that could offload essentially the entire graphics pipeline from the CPU, dubbing it a “Graphics Processing Unit,” or “GPU.” *Id.*, ¶¶35. 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. *Id.*, ¶¶35, 37.

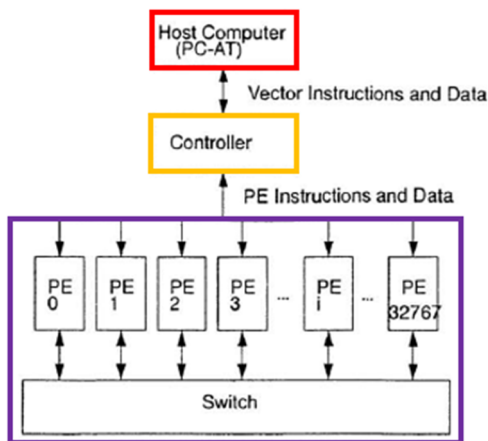
GPUs were used almost immediately to perform parallelized computations for non-graphics applications, such as numerical simulations. *Id.*, ¶¶40. This type of computing has been referred to as “general-purpose computing on GPUs,” or “GPGPU.” *Id.*, ¶¶41. NVIDIA pioneered both GPUs and GPGPU, including developing GPUs used throughout the industry for GPGPU. *Id.*, ¶¶42.

Traditionally, GPUs processed “textures” (stored as data arrays) by using “shaders” (*i.e.*, programs) to modify pixel data and create graphics for display. *Id.*, ¶¶38. In the early 2000s, researchers (including at and supported by NVIDIA) began repurposing shaders in programmable graphics pipelines for non-graphics purposes, such as scientific computing and mathematical simulations. *Id.*, ¶¶41. For these non-

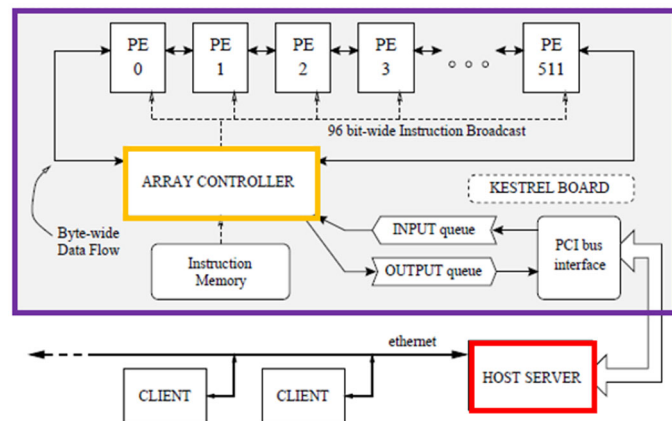
graphics applications, the programs are written as shaders and the data stored in textures. *Id.*, ¶42. NVIDIA actively promoted GPGPU research, including developing hardware and software for general-purpose computations on GPUs, such as the GPUs described in Nickolls and Kirk. *Id.* NVIDIA also published multiple books, including GPU Gems, on programming GPUs. Ex1006, 451.

2. “Controllers” for Parallel Processing in the Prior Art

Prior to the '867 patent, POSITAs had created effective designs for parallel processing. Ex1003, ¶¶44–53. One common feature was a “controller” to coordinate the function of the parallel processors. *Id.*, ¶49. 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. *Id.*, ¶¶49–50. It was 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. Ex1003, ¶¶49–53; Ex1001, 2:19–20, 13:32–39.



Ex1029, 845.



Ex1030, 2.

3. Managing Memory Through Swapping Address Pointers

Swapping address pointers—where the output of one computational step becomes the input for the next—was a basic technique in computer programming long before the '867 patent. Ex1003, ¶¶64–69. This technique had been widely employed in high-performance computing, numerical simulations, and graphics processing. *Id.* Instead of physically moving large amounts of data between memory locations, which incurs significant time and resource costs, pointer swapping allows programs to simply update the memory reference so the output data already in memory is immediately treated as the new input data. *Id.*, ¶¶64–65.

The programming technique for swapping pointers existed in programming before the development of GPUs. *Id.*, ¶¶66. For example, the 1988 textbook *The C programming Language* teaches that when two pieces of data need to be “swapped,” it is more efficient to swap *pointers* to the data, rather than copying and swapping all the data itself.

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;
}
```

Ex1009, 88.

These techniques (also known as “double buffering” or “ping-pong rendering”) were also used in GPUs prior to the ’867 patent. Ex1003, ¶¶67–69. For example, the “double buffering” functionality in the OpenGL programming language used two framebuffers (dedicated areas of memory) and swaps pointers between them at the end of each frame. *Id.*, ¶69. These concepts had been extended to performing numerical simulations on GPUs prior to the ’867 patent, as taught in GPU Gems, which includes code for implementing pointer swapping. *Id.*, ¶68.

B. Alleged Invention of the ’867 Patent

In the Background, the ’867 patent inventors acknowledged that “manufacturers of GPUs have included general purpose programmability into the GPU architecture leading to the increased popularity of using GPUs for highly parallelizable and computationally expensive algorithms outside of the computer graphics domain.” Ex1001, 1:21–25. However, the inventors claimed that “general purpose GPU (GPGPU) applications are not able to achieve optimal performance” because “[t]here is overhead for graphics-related features and algorithms that are not necessary for these non-video applications.” *Id.*, 1:25–30.

To overcome these alleged drawbacks, the patent describes an “accelerator” (Figure 2, below) having at least “one or more graphics processing units” (GPU 240), “two or more associated memory banks that are logically or physically partitioned” (shader memory bank 210 and texture memory bank 250), and “a

specialized controller” (controller 220). *Id.*, 2:6–17. This arrangement allegedly improves performance because, rather than the system CPU, “[t]he controller handles most of the primitive operations needed to set up and control GPU computation.” *Id.*, 2:17–19. As a result, “the CPU is freed from this function and is dedicated to other tasks.” *Id.*, 2:19–20, 13:32–39.

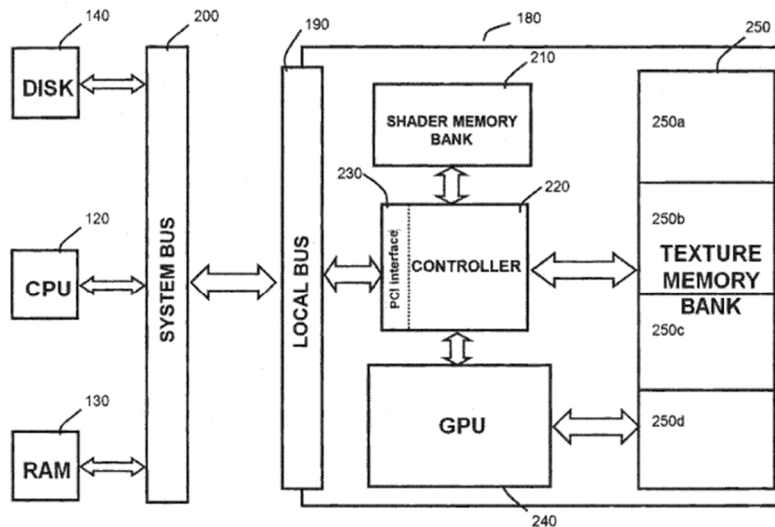


FIG. 2

The '867 patent also references pointer swapping for numerical simulations on GPUs. *Id.*, 2:24–27. The patent explains the technique of “dedicating two textures to output variables—one holds the value computed during the previous time step and is accessible to all computational elements during the current time step, another is not accessible to other elements and is used to accumulate new values for the variable computed during the current time step.” *Id.*, 6:7–12. In between computational steps, these textures (data arrays) are “switched” by “swapping the address pointers to respective textures.” *Id.*, 6:16–18.

The '867 patent alleges several “potential advantages” to its architecture, including “[l]imited computations on the CPU,” “[m]inimizing data transfer” between the accelerator and host system, and “[r]euse of existing and affordable technology” because “[a]ll hardware used in the invention and mentioned here-in are based on currently available and reliable components.” *Id.*, 13:30–67.

C. Prosecution History of the '867 Patent

The application that issued as the '867 patent originated from Appl. No. 11/860,254, filed on September 24, 2007. Ex1002, 1–4. The '867 patent claims priority to U.S. Prov. Appl. No. 60/826,892, filed on September 25, 2006. *Id.*, 6.

In the first Office Action, the Examiner rejected all pending claims as unpatentable as obvious. *Id.*, 105–15. The USPTO issued a Notice of Abandonment after the applicant did not respond to the rejection for seven months. *Id.*, 154. Ten months later, the applicant filed a Petition for Revival and responded to the rejection. *Id.*, 155–56. The applicant substantively amended several claims, canceled others, and added four new claims. *Id.*, 159–65. In its Remarks, the applicant argued no cited reference taught the claimed “accelerator controller.” *Id.*, 168. According to the applicant, “at best the disclosure of [the prior art] suggests an element that simply moves data into and out of a video card. There is no disclosure whatsoever in [the cited prior art] of an accelerator controller” that performs the claim requirements. *Id.*, 169–70.

The Examiner “agree[d] with the applicant that the prior art doesn’t teach” the “accelerator controller” limitations. *Id.*, 190. The ’867 patent issued on February 11, 2014, with 19 claims.

D. Person of Ordinary Skill in the Art

The ’867 patent relates to the field of computer systems and architecture, including GPUs. Ex1001, 1:13–2:5. A POSITA in the field around 2005–2006, would have had a Bachelor’s degree in Electrical or Computer Engineering (or equivalent) and at least two years’ experience in research, design and/or development of computer systems, including experience with GPUs. Ex1003, ¶17. Additional education could substitute for experience, and vice versa. *Id.*

IV. GROUNDS OF UNPATENTABILITY

A. Ground 1: Nickolls and GPU Gems Render Obvious Claims 1–19

Nickolls describes an architecture and methods for accelerating parallel data processing on a GPU (*i.e.*, an “accelerator”), including an “accelerator controller,” as claimed in the ’867 patent. Nickolls further teaches using output data from one step of a computational process as input to a next step. While Nickolls does not explicitly provide the well-known technique of pointer swapping as an example of how to do this, GPU Gems does. GPU Gems describes and provides code for a numerical simulation of a chemical reaction-diffusion on a GPU (such as that in

Nickolls) that swaps pointers to input and output data as claimed. Thus, the combination of Nickolls and GPU Gems renders obvious all Challenged Claims.

1. Overview of Ground and Prior Art

a. Summary of Nickolls

Nickolls discloses a computer system for “parallel data processing” that provides “flexible, general-purpose computational capacity in a GPU that may be used for computations in any field,” such as “modeling and simulation, matrix algebra, physics, chemistry, image processing, supercomputing, and so on.” Ex1004, 1:21–26, 2:21–23, 30:25–31. The computer system of Nickolls includes CPU 102 (red), system memory 104 (blue), and GPU 122 (purple), all of which communicate over bus 113. *Id.*, 6:9–34.

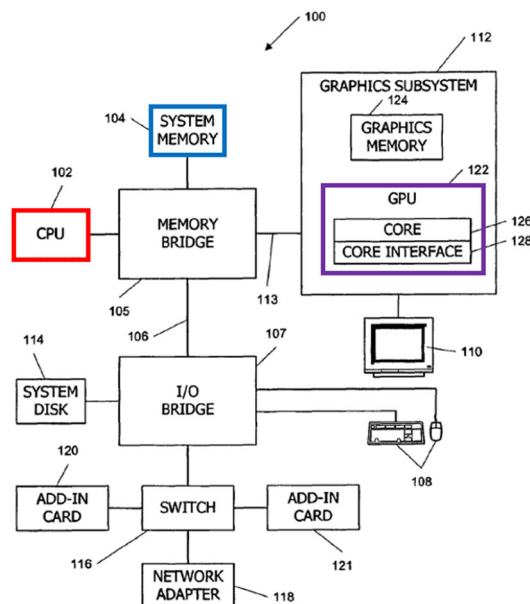


FIG. 1

Id., Fig. 1 (annotated).

GPU 122, shown in Figure 3, includes at least one processing core 126 (green) and core interface 128 (orange). *Id.*, 6:47–55. Each core 126 “includes multiple parallel processing engines that can be used to execute various shader programs” and “can also be leveraged to perform general-purpose computations.” *Id.*, 6:49–55. Each processing engine is further configured to execute multiple “threads” in parallel, where “[d]uring general-purpose computing, a thread can be an instance of a CTA³ program executing on a portion of an input data set.” *Id.*, 10:24–36.

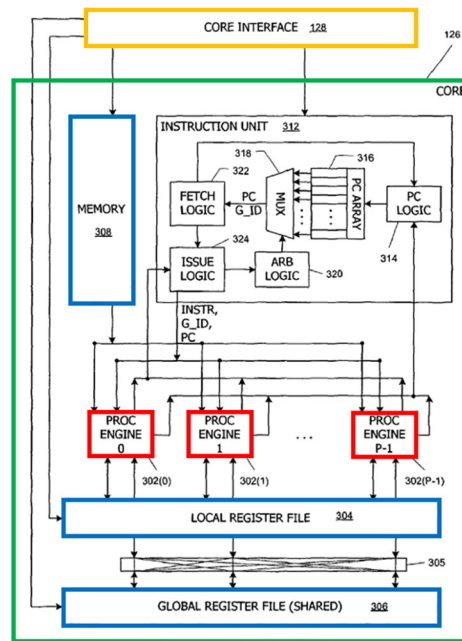


FIG. 3

Id., Fig. 3 (annotated).

³ “CTA” is a “cooperative thread array,” “a group of multiple threads that concurrently execute the same program on an input data set to produce an output data set.” Ex1003, 2:22–27.

Core 126 includes multiple memory modules (all in blue): local register file 304, global register file 306, and memory 308. *Id.*, 10:57–11:18. The processing engines can read from and write to memory, which may include shader programs or other instructions, input data, intermediate results, and output data. *Id.*, 7:27–30, 10:57–61, 20:35–45. The processing engines (and their respective threads) are configured to share data, including results of computations, with each other. *Id.*, 1:22–26, 2:33–35, 8:10–14, 25:8–10. Nickolls teaches the output of one stage of a computation (which it refers to as an “intermediate result”) may be used as an input for another stage of the computation. *Id.*

Core interface 128 “controls operation of core 126.” *Id.*, 6:47–49, 14:20–22. In operation, input unit 402 of core interface 128 receives “state information, data, and commands” from CPU 102. *Id.*, 14:29–31. The operation of core interface then proceeds in three phases. *Id.*, 19:20–25. First, in the “state” phase, state module 404 receives and loads “state information” such as the size of the input data set, the amount of local register space needed, and a memory address for a program to run on a processing engine. *Id.*, 14:41–51, 19:26–39. Next, in the “load” phase, “core interface 128 receives input data ... and loads the input data into (shared) global register file 306 of core 126.” *Id.*, 19:44–50. Finally, once all input data is received, core interface 128 enters the “launch” phase, in which it loads the data and

instructions for each processing core into local register file 304. *Id.*, 19:51–61. Core interface 128 then instructs core 126 to launch the processing. *Id.*

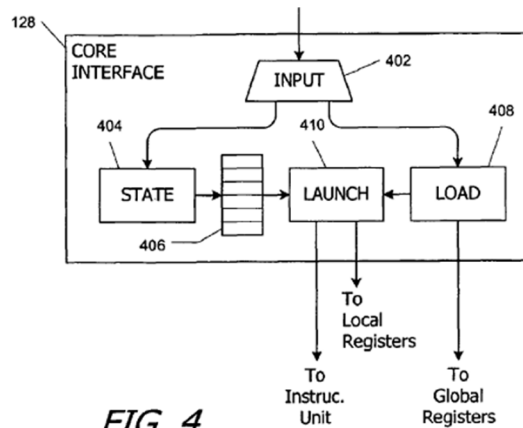


FIG. 4

Id., Fig. 4.

Core interface 128 also uses instruction unit 312 to distribute and execute instructions. *Id.*, 10:37–56. For example, instruction unit 312 distributes instructions to the processing engines and helps manage synchronization of the engines as they execute instructions. *Id.*, 11:27–29, 12:19–33, 21:37–55.

b. Summary of GPU Gems

GPU Gems, published by NVIDIA in 2005, contains 48 chapters describing the state of the art in GPU programming. Ex1006, xxxi–xxxii. Relevant here, Part IV is a “Primer” on “General-Purpose Computation on GPUs,” which “aims to provide a gentle introduction to the world of general-purpose computation on graphics processing units, or ‘GPGPU.’” *Id.*, xx, 453. Part IV includes “details of how to solve general-purpose problems using the GPU.” *Id.*, 454.

GPU Gems describes the concept of swapping pointers of input and output data, as claimed in the '867 patent. Chapter 31, written by Mark Harris of NVIDIA, explains that “most computations are broken into steps,” where “[e]ach step depends on the output of previous steps.” *Id.*, 501. To implement “direct feedback of GPU output to input,” Mr. Harris explains that one can use GPUs’ “render-to-texture” feature. *Id.*, 499, 501. With this method, results of one step of a computation are written to a “texture,” *i.e.*, a data array or “buffer” in memory, which is identified by a pointer. *Id.*, 501, 504. The buffer pointer can then be passed as the input to a later computation. *Id.*

Mr. Harris applies these concepts, including passing outputs to inputs, to “a simulation of a phenomenon known as chemical reaction-diffusion,” and provides code to implement the simulation. *Id.*, 505–08. The simulation code “[c]reates a ‘double buffered’ PUGBuffer,” which “allow[s] the simulation to alternate using one as input, one as output.” *Id.*, 507.

Listing 31-2. C++ Code to Set Up and Run a Reaction-Diffusion Simulation on the GPU
See the source code on the accompanying CD for more details.

```
PUGBuffer *rdBuffer; PUGProgram *rdProgram;

void init_rd(){ // Call this once.
    pugInit(); // Start up the GPU framework

    // Create a "double-buffered" PUGBuffer. Two buffers allow the
    // simulation to alternate using one as input, one as output
    PUGBuffer *rdBuffer = pugAllocateBuffer(width, height,
                                             PUG_READWRITE, 4, true);

    PUGProgram *rdProgram = pugLoadProgram("rd.cg", "rd");

    pugBindFloat(rdProgram, "DuDv", du, dv); // bind parameters
    pugBindFloat(rdProgram, "F", F);
    pugBindFloat(rdProgram, "k", k);
```

Id., 507. At the end of each iteration, the code calls the “swap ()” function, which swaps the pointer for the `currentSource` buffer with that of `currentTarget`, allowing the output of one computational cycle to become the input for the next.

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);
}
```

Id., 508 (annotated).

Other chapters of GPU Gems also include examples of swapping pointers between outputs and inputs. *E.g.*, *id.*, 44, 713 (“swap rho and newrho pointers”), 743 (“To sort the entire field, we just call the next list and swap the buffers until we’re done.”).

2. Combination of Nickolls and GPU Gems

Before the ’867 patent, it was well known that the textures and shaders of a GPU could be repurposed to perform numerical simulations. Section III.A.1. NVIDIA was a leader in these efforts, providing both the GPUs to be used (like Nickolls) and instructions and guidance on how to use those GPUs (like GPU Gems). A POSITA would have been motivated to combine the techniques in GPU Gems

(including the reaction-diffusion example in Chapter 31) with the GPU hardware in Nickolls. Ex1003, ¶¶129–39.

GPU Gems teaches the benefits and discloses examples of using GPUs to perform mathematical simulations, such as the “simulation of ... a chemical reaction-diffusion.” Ex1006, 505–08. GPU Gems emphasizes that “significant speedups can be achieved by applying GPUs to data-parallel computational problems.” *Id.*, 493. While GPU Gems is agnostic as to the specific GPU architecture for implementing the techniques disclosed, a POSITA would have been motivated to implement them on the most advanced GPUs at the time, which a POSITA would have known at the time were provided by NVIDIA. Ex1003, ¶132.

One such GPU is disclosed in Nickolls. A POSITA would have understood that the system disclosed in Nickolls, with the parallel processing engines of GPU 122, would be an ideal system for the numerical simulations of GPU Gems. *Id.*, ¶133. Nickolls provides “flexible, general-purpose computational capacity in a GPU” for fields such as “modeling and simulation” and “chemistry”—which a POSITA would have understood includes the chemical-diffusion reaction of GPU Gems. *Id.*; Ex1004, 28:66–29:5, 30:25–31.

A POSITA would have understood numerous advantages to applying the numerical simulations in GPU Gems on the hardware of Nickolls. Ex1003, ¶¶134–35. Nickolls provides advanced parallel processing capabilities and efficient data-

sharing mechanisms. Section IV.A.1.a. A POSITA would have understood that, through its multiple processing engines that can execute the same program on different portions of an input data set at the same time, the system of Nickolls would significantly reduce execution time compared to other processing architectures. Ex1003, ¶135. Further, mathematical and scientific simulations often involve iterative calculations where results from one computation stage influence subsequent stages. *Id.* With its shared memory, the system of Nickolls would have allowed the iterative computations of GPU Gems in which the outputs of one stage of computations become inputs to subsequent stages. *Id.*

A POSITA would have had a reasonable expectation of success in implementing the numerical simulations of GPU Gems on the system of Nickolls. *Id.*, ¶137. The system of Nickolls was designed for running many tasks (threads) at the same time. Since complex mathematical simulations also require dividing the computation into smaller, discrete tasks, a POSITA would have expected the system of Nickolls to be able to use parallel processing to execute the simulations in GPU Gems. *Id.* A POSITA would have further expected success in the combination because the system of Nickolls has the correct shared memory architecture for swapping pointers of outputs and inputs, as taught in GPU Gems. *Id.*

Likewise, a POSITA would have understood the benefits of swapping pointers while performing numerical computations (as taught in GPU Gems) and been

motivated to apply those techniques when writing programs to perform numerical computations on the accelerator of Nickolls. *Id.*, ¶138. A POSITA would have understood the combination of Nickolls and GPU Gems simply applied a known technique (*e.g.*, the numerical simulations and pointer-swapping of GPU Gems) to a known device (*e.g.*, the GPU-accelerated parallel processing system of Nickolls) ready for improvement to yield predictable results (*e.g.*, fast and efficient computation of complex numerical simulations). *Id.*, ¶139.

3. Detailed Analysis of Ground 1

Nickolls and GPU Gems teach all elements of claims 1–19 of the ’867 patent, rendering those claims obvious. The analysis begins with claims 16–19—the broadest and only asserted claims in the Litigation—then addresses claims 1–15, which add trivial limitations disclosed in the prior art.

a. Claim 16

i. 16[pre]

To the extent the preamble is limiting, Nickolls/GPU Gems discloses it. Ex1003, ¶141–46.

Nickolls discloses “systems and methods” for “parallel data processing.” Ex1004, 1:21–26, 2:21–23. The data processing includes “perform[ing] computations” on input data, which includes numerical simulations. *Id.*, 5:33. For example, Nickolls “provides a flexible, general-purpose computational capacity in a

GPU that may be used for computations in any field, including but not limited to bioinformatics, seismic signal processing, modeling and simulation, matrix algebra, physics, chemistry, image processing, supercomputing, and so on.” *Id.*, 30:26–31, 3:2–4, 7:66–8:9, 29:6–30:25. As an example, in the combination of Nickolls and GPU Gems, the “numerical simulation” is the chemical-diffusion reaction and the “input data” include the initial conditions for the simulation. Ex1006, 505–08; Ex1003, ¶¶143, 146.

Nickolls includes a central processing unit (*e.g.*, CPU 102) and an accelerator (*e.g.*, GPU 122). Ex1004, 6:10–18, 6:35–61. GPU 122 is an “accelerator” because it provides accelerated, parallel data processing, separately from the CPU. Ex1003, ¶145; Ex1001, 3:59 (referring to a “GPU accelerator”).

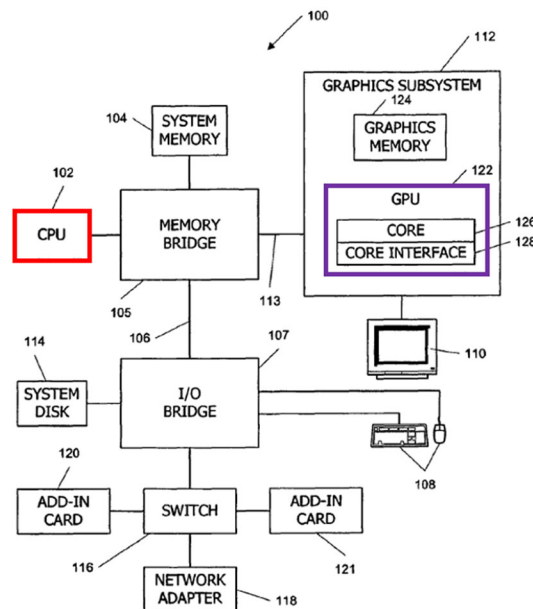


FIG. 1

Ex1004, Fig. 1 (annotated).

ii. 16[a]

Nickolls/GPU Gems discloses this limitation. Ex1003, ¶¶147–49.

The accelerator of Nickolls (*e.g.*, GPU 122) receives input data from CPU 102 at input 402 of core interface 128. Ex1004, 6:40–46 (“data supplied by CPU”), 14:29–31, 19:45–50, Figs. 3, 4, 10. In the combination of Nickolls and GPU Gems, the “input data” may include the initial conditions for the chemical-diffusion reaction, such as the initial U and V chemical concentrations for the reaction, diffusion rates (D_u and D_v), reaction parameters (*e.g.*, k and F), and the initial state of the array. Ex1006, 505–08; Ex1003, ¶149.

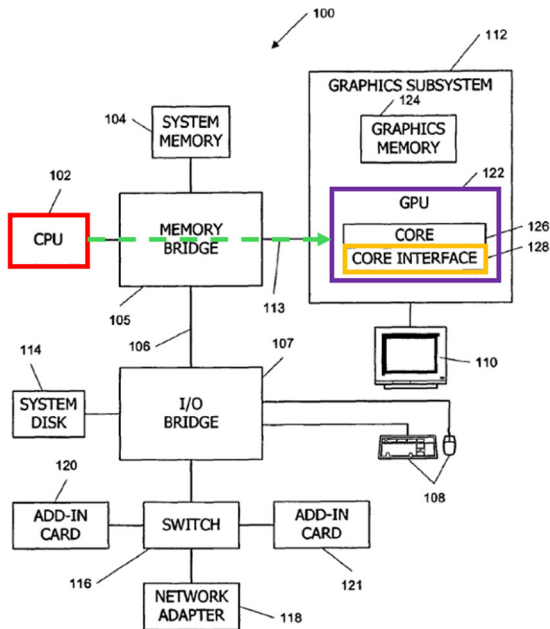


FIG. 1

Ex1003, Figs. 1, 4 (annotated).

iii. 16[b]

Nickolls/GPU Gems discloses this limitation. Ex1003, ¶¶150–60.

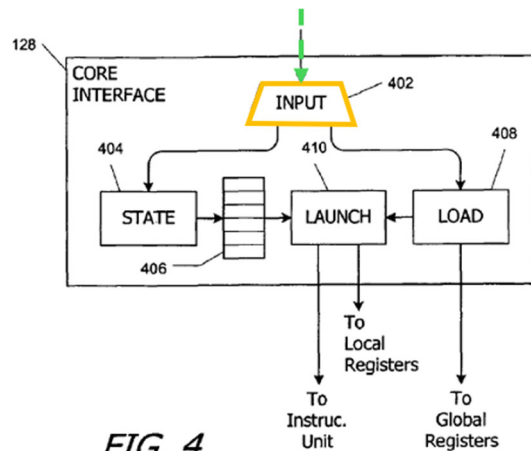


FIG. 4

As illustrated below, Nickolls includes an “accelerator controller,” e.g., core interface 128 (orange), which “controls operation of core 126.” Ex1004, 6:47–55, 13:12–67, 14:17–28, Figs. 3, 4; Ex1003, ¶151. Additionally and alternatively, a POSITA would have understood the “accelerator controller” of Nickolls to also include instruction unit 312 (brown), which works in conjunction with core interface 128 to supply instructions to the processing engines of Nickolls and manage their execution. Ex1004, 12:19–33, 13:12–45, 21:33–63; Ex1003, ¶152. Like the “accelerator controller” in the ’867 patent, the accelerator controller of Nickolls handles the set up and control of the accelerator’s computations, so that “the CPU is freed from this function and is dedicated to other tasks.” Ex1001, 2:17–20.

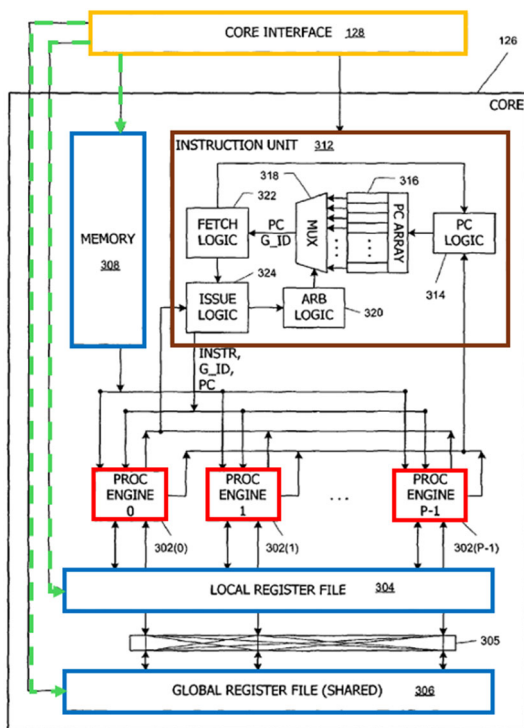


FIG. 3

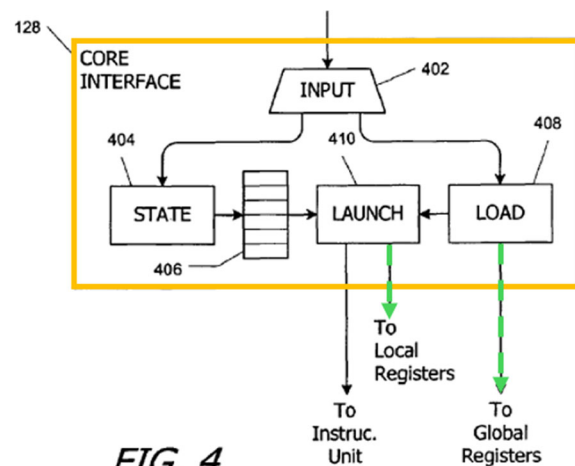


FIG. 4

Id., Figs. 3, 4 (annotated).

GPU 122 of Nickolls includes local register file 304, shared global register file 306, and memory 308 for storing input and output data, *i.e.*, “accelerator memory” (blue). Ex1004, 4:29–31, 10:57–11:18, 13:12–25.⁴

In Nickolls, input and output data is stored in either local register file 304 or global register file 306 by the accelerator controller. Each processing engine of GPU 122 “is allocated space in a local register file 304 for storing its local input data, intermedia results, and the like,” referred to as a “lane.” Ex1004, 10:57–11:2. In Nickolls, after input data is received in core interface 128 and before any computational cycles occur, each processing engine is supplied with “various input parameters required to execute the program.” *Id.*, 13:16–25. These “input parameters” “include the input data to be processed by the program.” *Id.* The input data for a processing engine may be transferred to a “lane” (*i.e.*, partition) of local register file 304 allocated to that engine. *Id.*, 10:57–61; Ex1003, ¶¶153–54. Nickolls also discloses that input data may be stored in “shared memory,” such as global register file 306 or memory 308. Ex1004, 8:14–30, 13:19–25. For example, Nickolls describes examples in which “core interface 128 receives input data (step

⁴ Although Figure 4 illustrates information transmitted to local and global registers, Nickolls teaches that input data can be sent to any memory in GPU 122. Ex1003, 4:29–31, 10:57–11:18, 20:35–45.

1012) and loads the input data into (shared) global register file 306 of core 126.” *Id.*, 19:44–50, Figs. 3, 4, 10. In these teachings, the transfer of data to the memory of core 126 is controlled by and involves core interface 128 and/or instruction unit 312. Ex1004, 13:16–25, 16:63–17:22, 19:44–50.

In Nickolls, the memory block of the local or shared register file, in which the input data for a computation is stored memory before a first computational cycle, is a “first partition.” Ex1003, ¶¶153–54. Nickolls teaches that “[e]ach processing engine 302 is allocated space in a local register file 304 for storing its local input data, intermediate results, and the like.” Ex1004, 10:57–65 (also referring to the allocated space as a “lane,” *i.e.*, a “partition”). Nickolls further teaches that input data is stored in a specific “local register file space” of shared memory, with a pre-determined, addressable portion for each thread of a processing engine. *Id.*, 8:10–30, 14:41–57. Nickolls thus teaches logically allocating memory partitions to hold specific data for specific processing engines or threads, separate from other data. Ex1003, ¶153; Ex1001, 5:33–48 (“partitioning” can “be done logically,” and that different portions of a single memory bank can be used to store different types of data, such as “input data, output data, intermediate results, and parameters”). A POSITA would have understood the portion of local register file 304 or global register file 306 allocated for input data to be a “first partition.” Ex1003, ¶¶153–

54.⁵ Moreover, in order to be used for the computations, the input data to start the computation is transferred to the first partition *before* the first computational cycle. *Id.* In any event, it would have been obvious to store the input data for the computations in a “first partition” in one of the memories of Nickolls because it would enable efficient parallel processing, with each processing engine and thread able to quickly and independently access the input data for its computations. *Id.*, ¶¶154–55.

The input data in Nickolls is “referenced by [a] first pointer.” For example, Nickolls teaches computing an address location in the shared memory from which data will be read. Ex1004, 8:22–25, 20:46–60. The variable holding the computed address is a “pointer” to the first partition. Ex1003, ¶157; Ex1001, 6:16–18 (referring to “address pointers”), 11:38–40 (referring to an “ID” of a texture (array) as a “pointer”). Similarly, where input data is stored in local register file 304, the system would maintain a “memory address” to identify the location of data, which

⁵ Petitioner notes that Patent Owner’s Infringement Contentions in litigation apply a broad construction of “partition,” contending that any data stored in an addressable memory space is in its own “partition.” Ex1010, 27–38. Under Patent Owner’s apparent construction, Nickolls discloses the claimed “partitions.” Ex1003, ¶156.

is implemented as a “pointer” to a first partition. Ex1004, 14:45–48, 3:54–62 (describing a “thread identifier” to identify a “subset of an input data set to process”); Ex1003, ¶157.

GPU Gems also teaches transferring first input data into a first partition, referenced by a first pointer, before a first computational cycle of a numerical simulation. Ex1003, ¶¶157–58. GPU Gems explains that “[a]rrays of data in the framework are called buffers,” and that buffers are identified by a “pointer.” Ex1006, 504. A POSITA would have understood that, in the context of the ’867 patent, a “buffer” is a “partition” because it a logical allocation of memory to hold specific data separate from other data. Ex1003, ¶158. In the example of the chemical reaction-diffusion simulation, before any computational cycles begin, the function `pgInitBuffer()` transfers the input data in `array` to a buffer (partition) that is referenced by a pointer (*i.e.*, `rdBuffer`):

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
pgInitBuffer(rdBuffer, array);
}

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

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

Ex1006, 504, 507.

It would also have been obvious to reference memory partitions in Nickolls using pointers, which was a well-known technique prior to the ’867 patent. Section

III.A.3; Ex1003, ¶160. Prior to the '867 patent, pointers were a fundamental construct of computer programming to efficiently manage memory and data access. *Id.* By using pointers, developers can directly access and manipulate specific memory locations, allowing for efficient data retrieval and modification without the need for copying large data sets. *Id.* This practice was well-established in computer science, and it would have been obvious to apply it to GPGPU frameworks. *Id.* Given the widespread use of pointers in managing data, their application to the parallel GPU processing in Nickolls would have been a routine choice for skilled practitioners in the field. *Id.*, ¶¶61–63, 160.

iv. 16[c]

Nickolls/GPU Gems discloses this limitation. Ex1003, ¶¶161–64.

Core 126 of GPU 122 in Nickolls “includes multiple processing engines” that can “be leveraged to perform general-purpose computations.” Ex1004, 6:35–55, 7:51–65, 30:52–60. Each processing engine of GPU 122 can execute instructions in multiple, parallel threads, including mathematical operations, on input data provided to generate output data. *Id.*, 10:24–56. Each processing core 126 of GPU 122 as a whole (green) and the processing engines (red) within core 126 (because they are the processors the perform the graphics and other computational operations) are a “graphics processing unit.” Ex1003, ¶162; Ex1001, 1:13–17 (“Graphics Processing

Units (GPUs) ... are considered highly parallel processors dedicated to fast computation of graphical content.”).

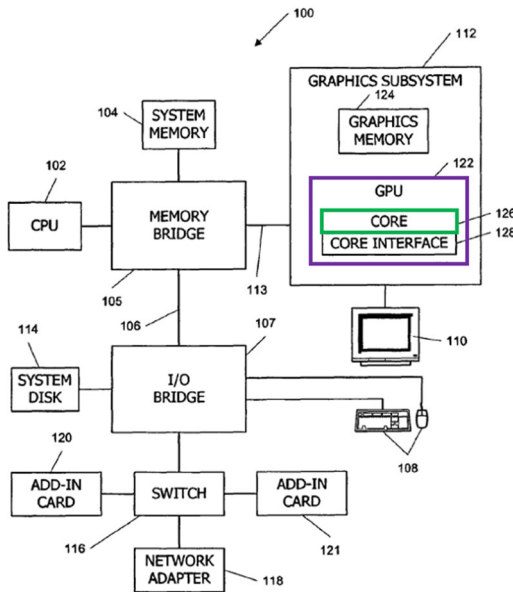


FIG. 1

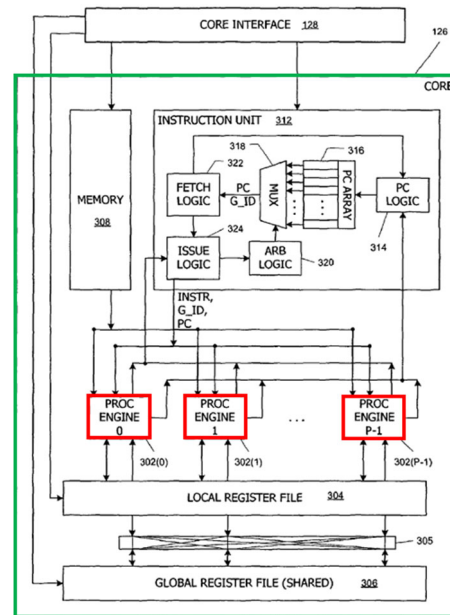


FIG. 3

Ex1004, Figs. 1, 3 (annotated).

In Nickolls, input data is distributed across multiple threads in the processing engines by dividing it into smaller portions that can be processed in parallel. *Id.*, 7:51–57, 7:66–8:3. Each thread of a processing engine is assigned to operate on a segment of the data and accesses it through shared or dedicated memory partitions. *Id.* Once the input data is received, the thread associated with each portion of the input data executes its assigned workload in parallel. *Id.*, 7:66–8:3, 19:51–61 (describing launching threads for execution), 20:20–34. The threads in the processing engine generate and store first output data, which includes what Nickolls refers to as “intermediate results.” *Id.*, 3:25–29, 10:57–59. Thus, in a first

computational cycle, a processing engine performs at least one calculation on a first portion of the input data as to generate first output data. Ex1003, ¶163.

GPU Gems also discloses this limitation. *Id.*, ¶164. For example, in the chemical-diffusion reaction simulation, the “Reaction-Diffusion Kernel Program” is run iteratively on a GPU, beginning with a first computation on the input data array. Ex1006, 506–08. Each step of computation simulates one step of the diffusion reaction and generates a result (first output data) for that step. *Id.*; Ex1003, ¶164.

v. 16[d]

Nickolls/GPU Gems discloses this limitation. Ex1003, ¶¶165–69.

In Nickolls, first output data (*e.g.*, “intermediate results” from a first computational cycle) is stored in a partition of the accelerator memory that is in communication with the processing engine. For example, “[e]ach processing engine 302 is allocated space in a local register file 304 for storing its local input data, intermediate results, and the like.” Ex1004, 10:57–65. Results from computational cycles can also be stored in a dedicated portion (*i.e.*, partition) of shared global register file 306. *Id.*, 8:10–30, 11:3–18, 14:41–57, 20:35–45. In either case, for the same reasons the location of the input data in memory is considered a “first partition” and is referenced by a “first pointer,” so too is the output data of each step considered to be in a “second partition” that is referenced by a “second pointer.” *See* 16[b]; Ex1004, 3:54–62, 8:22–25, 20:46–60, 11:38–40, 14:45–48; Ex1003, ¶166.

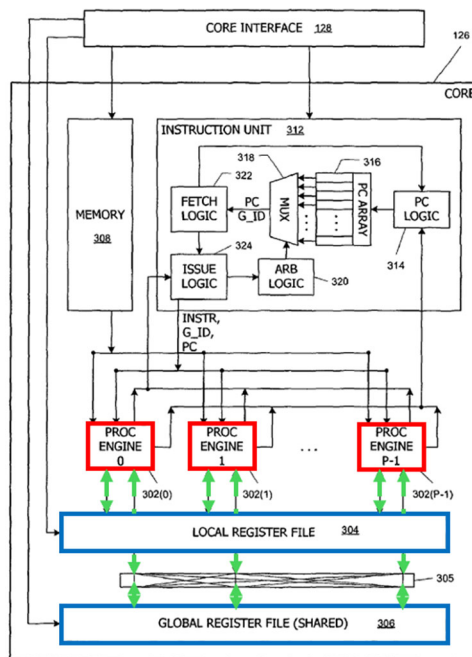


FIG. 3

Ex1004, Fig. 3 (annotated).

The storing of first output data occurs “by the accelerator controller”—*e.g.*, at the direction and control of the accelerator controller (core interface 128). For example, Nickolls explains the “CTA program” that is received from core interface 128 “may also include instructions to read from and/or write to the shared global register file 306 ... and/or other memory.” *Id.*, 20:35–45, 10:37–40, 20:46–67. Because the instructions for reading from and writing to memory are provided by core interface 128 (through instruction unit 312), a POSITA would have understood the storing of first output data is “by the accelerator controller.” Ex1003, ¶167.

GPU Gems also discloses this limitation. *Id.*, ¶168. For example, in the chemical-diffusion reaction simulation, the results of a first iteration of the simulation are written to `currentTarget`. Ex1006, 508. `currentTarget` is a

buffer (partition) in the GPU's memory dedicated to storing the output of the computation.

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);
}
```

Id. (annotated).

For the same reasons it would also have been obvious to reference a “first partition” in Nickolls with a “first pointer,” it would have been obvious to reference the “second partition” with a “second pointer.” *See* 16[b]; Section III.A.3; Ex1003, ¶169.

vi. 16[e]

Nickolls/GPU Gems discloses this limitation. Ex1003, ¶¶170–74.

Nickolls describes using the output of one computational cycle (an “intermediate result”) as an input for another cycle of the simulation. Ex1004, 2:33–35, 25:8–10. While Nickolls does not explicitly disclose doing so by using the well-known and established process of swapping pointers to input and output data, this technique is disclosed in GPU Gems. Ex1003, ¶¶171–72. GPU Gems describes a numerical simulation of a chemical reaction-diffusion on a GPU that swaps pointers

to input and output data as claimed. The simulation code in GPU Gems “[c]reates a ‘double buffered’ PUGBuffer,” which “allow[s] the simulation to alternate using one as input, one as output.” Ex1006, 507. At the end of each iteration, the code calls the “swap ()” function, which swaps the pointers for the `currentSource` buffer with that of the `currentTarget` buffer, thus allowing the output of one computational cycle to become the input for the next computational 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);
}
```

Id., 508 (annotated). A POSITA implementing GPU Gems on the system of Nickolls would therefore include this claim step. Ex1003, ¶¶172–73.

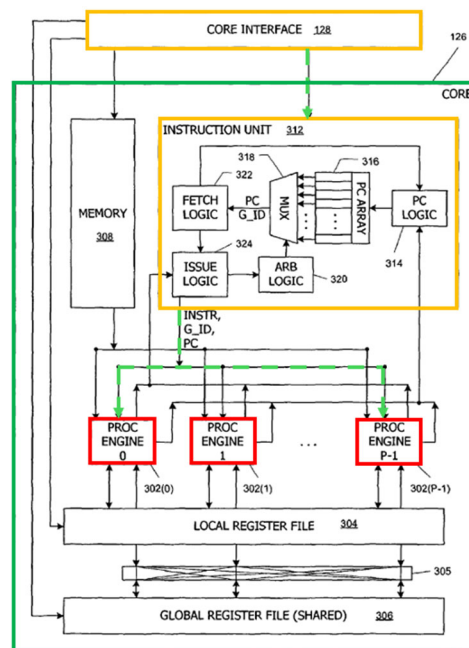
It would also have been obvious to a POSITA to implement an algorithm on the hardware in Nickolls in which the first and second pointers are swapped at the end of a computational cycle. Because Nickolls describes using intermediate results as inputs to a new computational cycle (*see supra*), a POSITA would have been motivated to use known techniques to effectuate this efficiently. Ex1003, ¶174; Section III.A.3. One such known technique was pointer swapping, which avoided copying large chunks of memory. *Id.* GPU Gems includes several examples of

swapping pointers between outputs and inputs at the end of a computational stage, confirming this technique was well known and would have been an obvious implementation of Nickolls' use of the output of one computational cycle (an "intermediate result") as an input for another cycle of the simulation. Ex1006, 44, 713, 743.

b. Claim 17

Nickolls/GPU Gems discloses this limitation. Ex1003, ¶¶175–78.

In Nickolls, instructions for performing computations are sent from core interface 128, through instruction unit 312, to core 126 and the processing engines (*i.e.*, graphics processing units). Ex1004, 10:37–40, 3:48–52, 13:41–45, 14:41–48, 17:20–36.



Id., Fig. 3 (annotated).

c. Claim 18

Nickolls/GPU Gems discloses this limitation. Ex1003, ¶¶179–86.

A POSITA would have understood the memory in GPU 122 of Nickolls is divided into partitions to store specific data. Petitioner incorporates by references its analysis above of the “first partition” (16[b]) and “second partition” (16[d]) of accelerator memory.

Nickolls discloses a third partition for storing “internal variables.” For example, the data stored in shared memory 306 for the processing cores includes “input parameters,” such as information specific to that thread or the total number of threads in a CTA. Ex 1004, 13:16–45, 16:63–17:3, 24:39–47. A POSITA would have understood these parameters to be “internal variables” stored in a “third partition”. Ex1003, ¶181; 16[b].

Nickolls further discloses a fourth partition for storing “data used as input at a particular computation cycle of the numerical simulation.” Global register file 306 stores “the input data to be processed by the program.” Ex1004, 13:16–25. The input data is divided into multiple partitions, each of which is assigned to a thread. Thus, data to be used at a particular computation cycle (*e.g.*, the first cycle) is stored in a “fourth partition.” Ex1003, ¶182. Additionally and alternatively, a partition of shared memory 308 may be “used to store data that is expected to be used in multiple threads, such as coefficients of attribute equations.” Ex1004, 8–14. Because they

are in a separate memory bank (memory 308), a POSITA would have understood this data, which is to be used at a particular computational cycle (and potentially every cycle), to be stored in a “fourth partition.” Ex1003, ¶182.

GPU Gems also discloses this limitation. *Id.*, ¶¶183–84. The memory in the chemical reaction-diffusion simulation is “double buffered,” which “allow[s] the simulation to alternate using one as input, one as output.” Ex1006, 507. One buffer (partition) is `currentSource` and the other is `currentTarget`. *Id.*, 508. These partitions correspond to the claimed first and second partitions. Next, the system stores “internal variables” for the chemical-diffusion reaction, such as the physical locations of each node, passed as “`coords`,” and reaction parameters (*e.g.*, k and F) to calculate the Laplacian of concentration (and thus subsequently the rate of change from diffusion). *Id.*, 507. The buffer in which these internal variables are stored is a “third partition.” Ex1003, ¶184.

With respect to the “fourth partition,” GPU Gems discloses that it “bind[s] parameters” to the GPU program, “`du`” and “`dv`,” which correspond to diffusivity parameters for each of the concentrations, which is input to be used at “a particular computation cycle of the numerical simulation.” Ex1006, 507. The buffer in which this input data is stored is a “fourth partition.” Ex1003, ¶184. Additionally, the buffer (partition) `array` “stores data used as input at a particular computation cycle of the numerical simulation”—*i.e.*, the initial array at the first computational cycle

to begin the analysis. Ex1006, 507–08. This buffer is also a “fourth partition.” Ex1003, ¶185.

d. Claim 19

Nickolls/GPU Gems discloses this limitation. Ex1003, ¶¶187–90.

Nickolls teaches transferring outputs from computations to system memory 104 based on instructions from the accelerator controller. Ex1004, 20:35–45, 10:37–40, 13:41–45. According to Nickolls, “[i]ntermediate results may be written to global register file 306 and/or other memory such as graphics memory 124 or system memory 104 of FIG. 1.” *Id.* Nickolls discloses that “[a]fter execution of the CTA is finished, GPU 122 may transfer the data [generated] (e.g., using a conventional DMA operation) to system memory 104.” *Id.*, 20:67–21:2. Nickolls further discloses that “[c]ore 126 advantageously signals core interface 128 upon completion of a CTA, so that core interface 128 can initiate execution of a next CTA.” *Id.*, 21:4–8. DMA (Direct Memory Access) operations permit a memory bus to access memory at the instruction of a processor, without requiring the processor to be involved after giving an initial instruction. Ex1003, ¶189. Because data can be transferred at the end of a computation cycle using an asynchronous data transfer method, and because Nickolls discloses that it would be “advantageous” for the core interface to “initiate a next CTA” “upon completion of a CTA,” Nickolls discloses a system wherein a controller initiates a parallel program, gets results,

directs a memory bus to transfer those results to system memory, and initiates a new simulation while the transfer occurs. *Id.*

It would also have been obvious to a POSITA to transfer, by the accelerator controller, the first output data to the main memory during the second computational cycle. *Id.*, ¶190. For the reasons explained above, including because a POSITA would have been motivated to *not* have a processor sitting idle to wait for additional GPU processes to complete and to keep a record of how a physical process proceeds over time, a POSITA would have been motivated to copy output data immediately (during the next computational cycle) to system memory. *Id.* Nickolls provides for significant flexibility in when, where, and how data is transferred and stored. Section IV.A.3.a.iii; Ex. 1004, 21:3–4, 6:27–33 (*e.g.*, data transfers over PCI-E). Thus, a POSITA would have been motivated to choose an asynchronous method of data transfer and would have known methods to do so such that it required only the routine application of prior art elements according to known methods to yield predictable results. Ex1003, ¶190.

e. Claim 1

i. 1[pre]

Nickolls/GPU Gems discloses this limitation. Ex1003, ¶¶191–92.

See 16[pre]. Nickolls discloses systems for “parallel data processing” and “perform[ing] computations,” including numerical simulations with multiple

computational cycles. Ex1004, 1:21–26, 2:21–23, 5:33, 30:26–31. In the combination of Nickolls and GPU Gems, the “numerical simulation” is the multi-cycle chemical-diffusion reaction of Chapter 31. Ex1006, 505–08.

ii. 1[a]

Nickolls/GPU Gems discloses this limitation. Ex1003, ¶¶193–94.

See 16[pre].

iii. 1[b]

Nickolls/GPU Gems discloses this limitation. Ex1003, ¶¶195–96.

Nickolls discloses system memory 104, which is operably coupled to CPU 102 through memory bridge 105 (or directly). Ex1004, 6:9–18, 6:35–44, 7:8–12. System memory 104 stores input data for numerical simulations, which the CPU can access and send to GPU 122. *Id.*, 6:35–44, 11:14–18, 13:16–29.

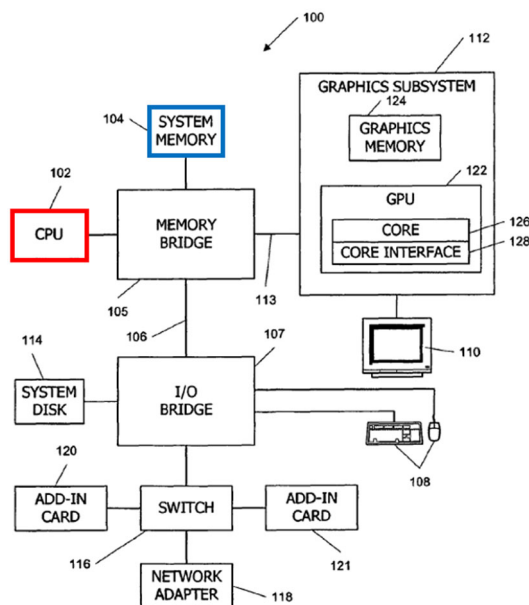


FIG. 1

Ex1004, Fig. 1 (annotated).

iv. 1[c]

Nickolls/GPU Gems discloses this limitation. Ex1003, ¶¶197–99.

Nickolls includes graphics subsystem 112, which connects to display device 110 (a video monitor) to provide “[v]isual output.” Ex1004, 6:18–22, 6:56–61. When running numerical simulations, the system of Nickolls would be used to display results of the simulation (*i.e.*, an “indication of the numerical simulation”). Ex1003, ¶198. At a minimum, in order to be able to view the results of the simulation, it would have been obvious to a POSITA to configure display device 110 to display an indication (such as a result) of the numerical simulation. *Id.*, ¶199.

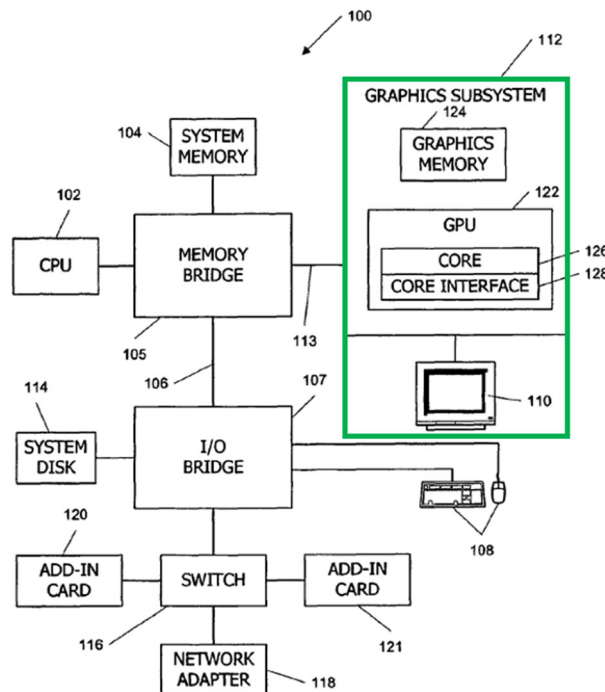


FIG. 1

Id., Fig. 1 (annotated).

v. 1[d]

Nickolls/GPU Gems discloses this limitation. Ex1003, ¶200–02.

See 16[a], 16[b], 16[c]; Claim 19. As explained for Claim 19, Nickolls discloses sending first output data to main memory after the conclusion of a first computational cycle, including using DMA. Once the output data is stored in system memory 104, it is available to CPU 102. Ex1003, ¶201. Thus, the transfer described in Claim 19 “provide[s] first output data ... to the central processing unit.”

vi. 1[d][i]

Nickolls/GPU Gems discloses this limitation. Ex1003, ¶¶203–04.

See 16[c], 16[e]. Nickolls discloses a graphics processing unit (*e.g.*, core 126 and/or processing engine) that generates output data during computational cycles based on input data. Ex1004, 6:35–55, 7:51–65, 10:24–56, 30:52–60. Nickolls further describes using the output of one computational cycle (*e.g.*, first output data) as input for another cycle of a simulation to generate a second output (*e.g.*, second output data). *Id.*, 2:33–35, 25:8–10. For example, in the reaction-diffusion example of GPU Gems, the output of one step of the computation is used as an input to the next step, which generates a further output. Ex1006, 506–08.

vii. 1[d][ii]

Nickolls/GPU Gems discloses this limitation. Ex1003, ¶¶205–06.

See 16[b], 16[d].

viii. 1[d][ii][1]

Nickolls/GPU Gems discloses this limitation. Ex1003, ¶¶207–08.

See 16[b], 16[d]. This limitation to be satisfied if the first output data remains stored in the first partition during the second computational cycle (*i.e.*, the act of storing need not occur during the second computational cycle). Ex1003, ¶208. As explained for 16[d], after a first computation on input data is complete, first output data is stored in a partition (a “second partition” in Claim 16 but a “first partition” in Claim 1) that is referenced by a pointer (a “second pointer” in Claim 16 but a “first pointer” in Claim 1).

ix. 1[d][ii][2]

Nickolls/GPU Gems discloses this limitation. Ex1003, ¶¶209–10.

See 16[b], 16[d]. For similar reasons as above, second output data from a computational cycle of Nickolls is stored in a partition (a “second partition”) that is referenced by a pointer (a “second pointer”).

x. 1[e]

Nickolls/GPU Gems discloses this limitation. Ex1003, ¶¶211–12.

See 16[d]. Core interface 128 (an “accelerator controller”) is operably coupled to CPU 102 and “accelerator memory” within GPU 122 (*e.g.*, local register file 304, global register file 306, and memory 308). Ex1004, 6:40–46, 10:57–11:18, 13:12–25, 14:29–31, 19:45–50.

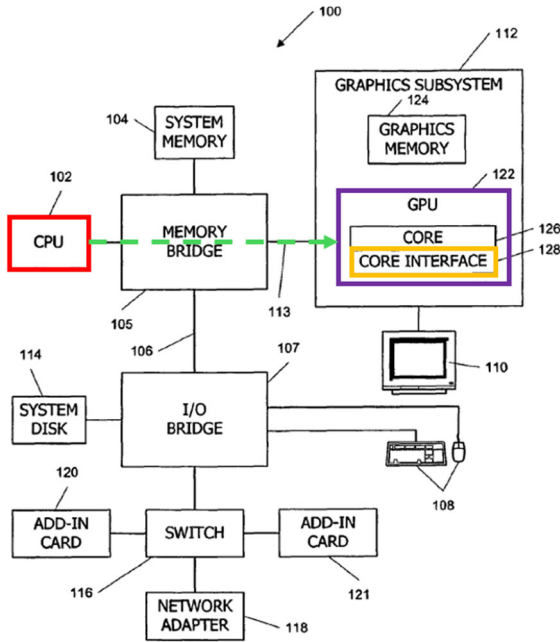


FIG. 1

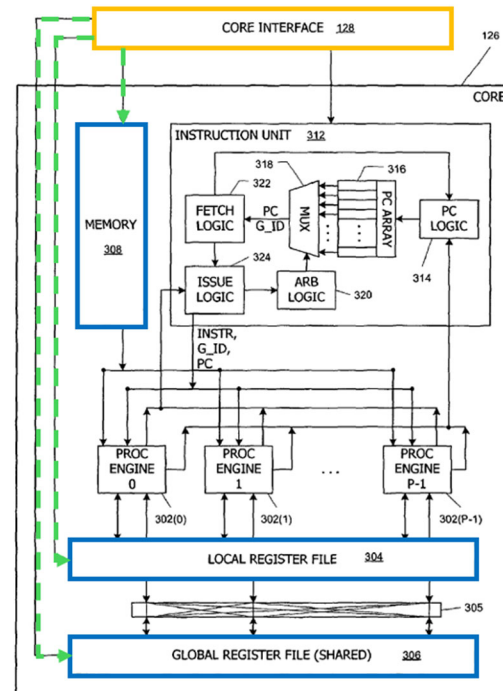


FIG. 3

Ex1003, Figs. 1, 3 (annotated).

xi. 1[e][i]

Nickolls/GPU Gems discloses this limitation. Ex1003, ¶¶213–14.

See 16[b].

xii. 1[e][ii]

Nickolls/GPU Gems discloses this limitation. Ex1003, ¶¶215–16.

See Claim 19.

xiii. 1[e][iii]

Nickolls/GPU Gems discloses this limitation. Ex1003, ¶¶217–18.

See 16[d]. As explained for 16[d], in Nickolls, output data is stored, by an accelerator controller, in a partition during a computational cycle. For similar

reasons, for this element, the accelerator controller directs second output data into a second partition during a second computational cycle. Ex1004, 8:14–17, 10:57–65, 11:3–18, 20:35–45.

xiv. 1[e][iv]

Nickolls/GPU Gems discloses this limitation. Ex1003, ¶¶219–20.

See 16[e]. Just as pointers are swapped at the end of a first computational cycle, such that the output of the first computational cycle becomes the input to a second, pointers are swapped at the end of the second computational cycle, such that the output of the second computational cycle becomes the input to a third. Ex1004, 2:33–35, 25:8–10; Ex1006, 507–08.

f. Claim 2

Nickolls/GPU Gems discloses this limitation. Ex1003, ¶¶221–22.

The accelerator controller of Nickolls (*e.g.*, core interface 128, alone and with instruction unit 312) dictates an order of instructions to the graphics processing unit. For example, core interface 128 includes a “launch” module that “selects” a “first SIMD group to be launched,” and then loads the instructions to locations corresponding to those threads. Ex1004, 19:51–61, 17:37–41, 17:63–18:5 (describing “selection unit 808” in launch module). Additionally, instruction unit 312 includes “[a]rbitration logic 320 and multiplexer 318 [to] determine the order in

which instructions are fetched,” as well as “selection logic 1110 that selects a next instruction to issue.” *Id.*, 12:43–55, 21:37–52, 23:61–67, 3:48–52, 5:57–64.

g. Claim 3

Nickolls/GPU Gems discloses this limitation. Ex1003, ¶¶223–24.

See 16[a]. Core interface 128 receives input data from CPU 102 over bus 113 at input 402 (an “interface controller”). Ex1004, 6:40–46, 14:29–31, 19:45–50, Figs. 3, 4, 10.

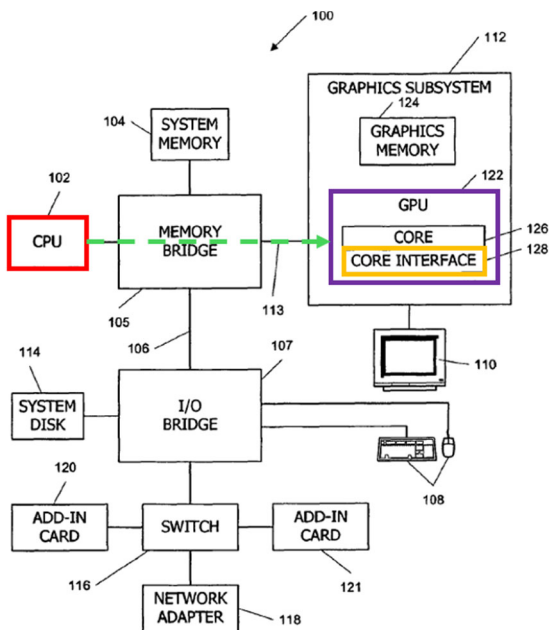


FIG. 1

Ex1003, Figs. 1, 4 (annotated).

h. Claim 4

Nickolls/GPU Gems discloses this limitation. Ex1003, ¶¶225–28.

See 16[b], 16[d]. In the '867 patent, a “texture” refers to a data array and a “shader” refers to instructions to run on a GPU. Ex1001, 5:1–12. Thus, a POSITA

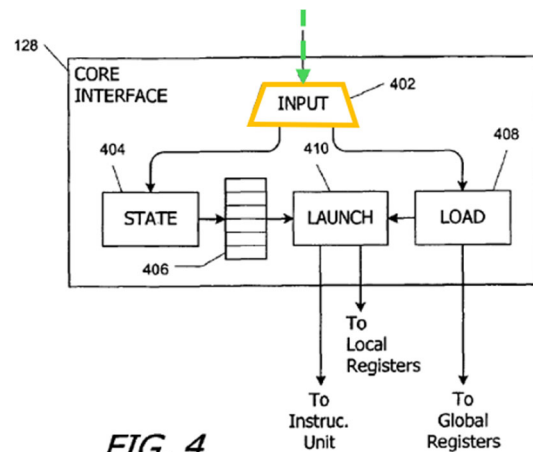


FIG. 4

would understand “texture memory bank” to refer to memory in which textures (or data arrays) are stored and “shader memory bank” to refer to memory in which shaders (or instructions) are stored. A POSITA would have understood the memory partition into which input data is transferred (16[b]) and in which output data is stored (16[b]) to be part of a “texture memory bank.” Furthermore, in Nickolls, program instructions for the processing cores of the GPU are stored in portions of memory in GPU 122, which a POSITA would have understood to be a “shader memory bank.” Ex1004, 6:65–7:4, 11:8–18, 14:41–48; Claim 7. The processing instructions stored in the shader memory bank are executed by the processing cores of the GPU to perform operations on the input data. *Id.*

GPU Gems also teaches using “textures” (stored in a “texture memory bank”) and “shaders” (stored in a “shader memory bank”) to perform a set of operations on input data in a GPU. Ex1006, 500 (“GPU Textures = CPU Arrays”), 503–08. In the reaction-diffusion example, input data in `array` (a “texture”) is stored in a buffer (a partition in a “texture memory bank”) and is operated on by a kernel program (a “shader”) stored in a “shader memory bank.” *Id.*; Ex1003, ¶228.

i. Claim 5

Nickolls/GPU Gems discloses this limitation. Ex1003, ¶¶229–30.

This limitation was addressed with respect to Claim 18, except the “fourth partition” in Claim 18 stores “data used as input at a particular computation cycle of

the numerical simulation” and for this claim the “fourth partition” stores “data textures used as input at a particular computation cycle of the plurality of computational cycles.” Because the data stored in the fourth partition in Claim 18 is an array of data (*i.e.*, a “texture”), Claim 5 is disclosed for the same reasons as Claim 18.

j. Claim 6

Nickolls/GPU Gems discloses this limitation. Ex1003, ¶¶231–32.

See 16[b], 16[c]. Core interface 128 alone and with instruction unit 312 (the “accelerator controller”) supplies input data and instructions to core 126, including the processing engines of Nickolls, and manages their execution by the processing engines. Ex1004, 12:19–33, 13:12–45, 21:33–63.

k. Claim 7

Nickolls/GPU Gems discloses this limitation. Ex1003, ¶¶233–34.

Input unit 402 of core interface 128 receives incoming data and commands (*i.e.*, instructions) and directs them to state module 404 and load module 406, where they are stored in temporary memory, such as registers. Ex1004, 14:16–43.

l. Claim 8

Nickolls/GPU Gems discloses this limitation. Ex1003, ¶¶235–39.

See 16[pre], 16[a]. In the combination of Nickolls and GPU Gems, the “input data” may include the initial conditions for the chemical-diffusion reaction, such as

the initial U and V chemical concentrations for the reaction, diffusion rates (D_u and D_v), reaction parameters (*e.g.*, k and F), and the initial state of the array. Ex1006, 505–08; Ex1003, ¶238.

m. Claim 9

i. 9[pre]

Nickolls/GPU Gems discloses this limitation. Ex1003, ¶¶240–41.

See 1[pre], 16[pre].

ii. 9[a]

Nickolls/GPU Gems discloses this limitation. Ex1003, ¶¶242–43.

See 1[d][i], 16[c].

iii. 9[b]

Nickolls/GPU Gems discloses this limitation. Ex1003, ¶¶244–45.

See 1[d][ii], 16[b].

iv. 9[c]

Nickolls/GPU Gems discloses this limitation. Ex1003, ¶¶246–47.

See 1[d][i], 1[e], 16[c], 16[d].

v. 9[c][i]

Nickolls/GPU Gems discloses this limitation. Ex1003, ¶¶248–51.

See 1[d][i], 16[c]. The computations performed by the processing engines are controlled by a “computational stream” because the processing occurs as part of a sequential execution of a “stream of commands.” Ex1004, 6:65–7:4, 14:17–57,

17:4–36, 20:61–21:8. The “accelerator controller” (*i.e.*, core interface 128 alone and with instruction unit 312) passes a stream of instructions to the graphics processing unit (*e.g.*, core 126 and/or processing engine) to perform computations, which then executes the instructions. *Id.*; Ex1003, ¶249–51.

vi. 9[c][ii]

Nickolls/GPU Gems discloses this limitation. Ex1003, ¶¶252–53.

See 16[d]. The transfer of output data from the GPU to accelerator memory is controlled by an “output stream” because the transfer occurs as part of a stream of commands directing output data into accelerator memory. Ex1004, 6:65–7:4, 14:17–57, 17:4–36, 20:61–21:8. The “accelerator controller” (*i.e.*, core interface 128 alone and with instruction unit 312) passes a stream of instructions to the graphics processing unit (*e.g.*, core 126 and/or processing engine), which includes a direction to return an output to a location specified by the controller. *Id.*; Ex1003, ¶253.

vii. 9[c][iii]

Nickolls/GPU Gems discloses this limitation. Ex1003, ¶¶254–55.

See 1[d], 16[a], 16[d]. The transfer of input data to accelerator memory is controlled by an “input stream” because the transfer occurs as part of a stream of commands directing input data into accelerator memory. Ex1004, 13:16–25, 14:16–26, 17:4–18, 19:26–61.

n. Claim 10

Nickolls/GPU Gems discloses this limitation. Ex1003, ¶¶256–57.

See Claim 2.

o. Claim 11

Nickolls/GPU Gems discloses this limitation. Ex1003, ¶¶258–59.

See Claims 6, 19; 1[e][ii].

p. Claim 12

Nickolls/GPU Gems discloses this limitation. Ex1003, ¶¶260–61.

See Claim 4.

q. Claim 13

Nickolls/GPU Gems discloses this limitation. Ex1003, ¶¶262–63.

See Claims 5, 18. Although the order of what is on each partition differs in Claims 5 and 18 from Claim 13, Claims 5 and 8 collectively describe the required storage in the four partitions required in Claim 13.

r. Claim 14

Nickolls/GPU Gems discloses this limitation. Ex1003, ¶¶264–65.

See 1[e][iv], 16[e].

s. Claim 15

Nickolls/GPU Gems discloses this limitation. Ex1003, ¶¶266–67.

See Claim 7.

B. Ground 2: Kirk and GPU Gems Render Obvious Claims 1–19

Kirk describes an architecture and methods for accelerating parallel data processing on a GPU (*i.e.*, an “accelerator”), including an “accelerator controller,” as claimed in the ’867 patent. Kirk further teaches using output data from one step of a computational process as input to a next step. While Kirk does not explicitly state that it uses the well-known technique of pointer swapping, GPU Gems does. It describes and provides code for a numerical simulation of a chemical reaction-diffusion on a GPU (such as that in Kirk) that swaps pointers to input and output data as claimed. Thus, the combination of Kirk and GPU Gems renders obvious all Challenged Claims.

1. Overview of Ground and Prior Art

a. Summary of Kirk

Kirk discloses a “Computing System” with a “programmable graphics processor” for “processing of data.” Ex1005, 1:40–47, 26:24–39, 26:56–61, Figs. 12A, 12B.⁶ Kirk explains the “graphics data” it can process “is any data that is input to or output from computation units within Programmable Graphics

⁶ Many components in Figures 12A and 12B have the same reference numerals as in earlier embodiments. Thus, this Petition also relies on disclosure provided in the context of those other embodiments, which also apply to Figure 12.

Processor 105.” *Id.*, 3:35–41. Kirk includes Host Processor 114 (red), Host Memory 112 (blue), and Graphics Subsystem 107 (purple), all of which communicate through System Interface 115. *Id.*, 3:17–33.

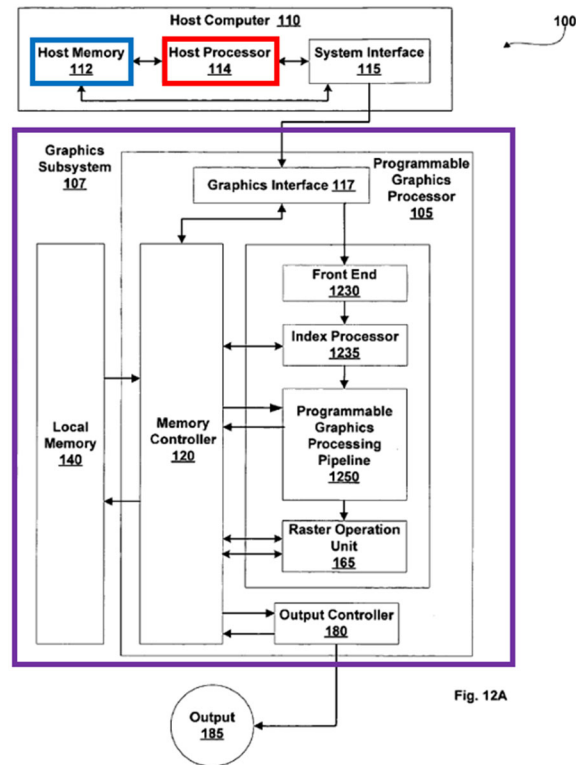


Fig. 12A

Id., Fig. 12A (annotated).

Graphics Subsystem 107 includes Local Memory 140 (blue) and Programmable Graphics Processor 105 (green). *Id.*, 3:32–46, 26:24–39. Programmable Graphics Processor 105 interfaces with the host system through Graphics Interface 117 (orange) and with Local Memory 140 (blue) through Memory Controller 120 (brown). *Id.*, 3:32–67, 26:24–39. Local Memory 140 stores input data, output data, and program instructions for graphics/data processing. *Id.*, 3:34–46. Programmable Graphics Processor 105 also includes Programmable

Graphics Processing Pipeline 1250 (red) in which data is processed according to instructions stored in memory. *Id.*, 26:32–61.

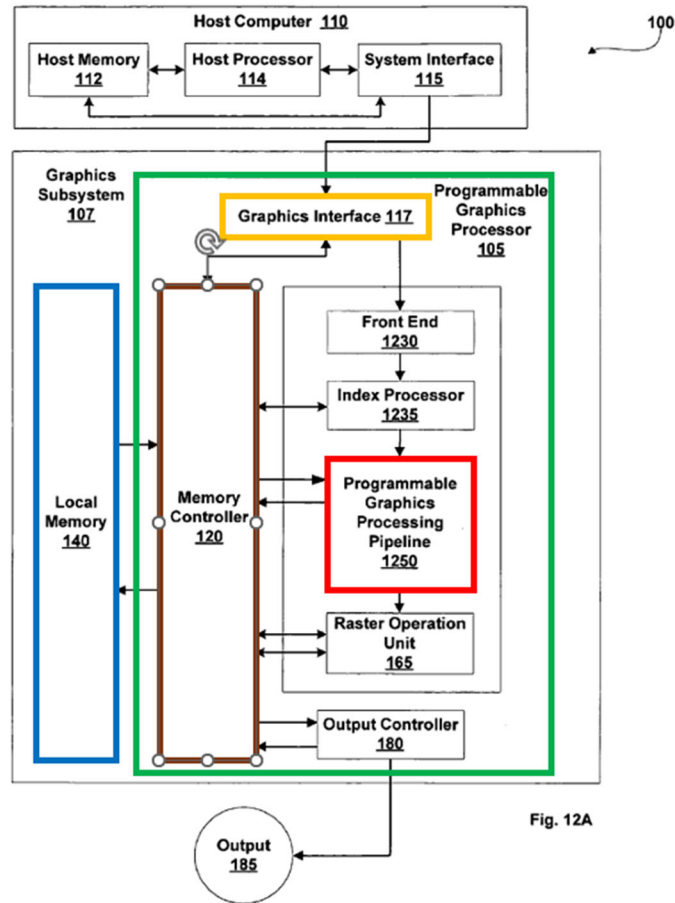


Fig. 12A

Id., Fig. 12A (annotated).

Programmable Graphics Processing Pipeline 1250 includes multiple Execution Pipelines 1240, each of which “includes at least one multithreaded processing unit.” *Id.*, 27:2–11. Each execution pipeline communicates with graphics memory (including Local Memory 140) “to read program instructions and graphics data stored in buffers in graphics memory.” *Id.*, 27:12–28. Programmable

Graphics Processing Pipeline 1250 can also contain additional on-board memory to store data and instructions. *Id.*

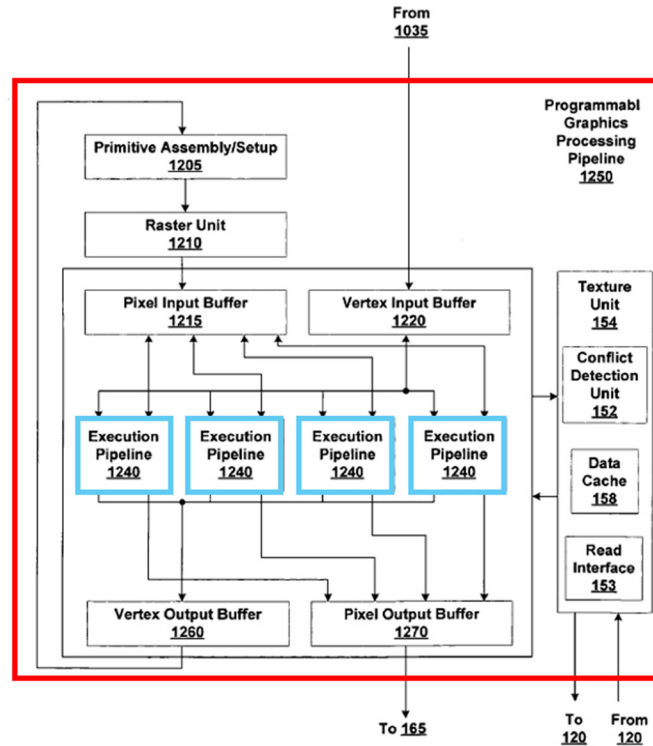


Fig. 12B

Id., Fig. 12B (annotated).

In operation, “[a] graphics program ... is executed within one or more Execution Pipelines 1240 as a plurality of threads where each vertex or fragment to be processed by the program is assigned to a thread.” *Id.*, 27:40–43. Results from the computations are sent to memory, such as Local Memory 140, Host Memory 112, or memory within Programmable Graphics Processing Pipeline. *Id.*, 3:35–46. Data produced by the processing pipeline “may be written to a buffer in graphics memory to be read from during a subsequent pass.” *Id.*, 4:16–20.

b. Summary of GPU Gems

Petition incorporates by reference its summary of GPU Gems with respect to Ground 1. Section IV.A.1.b.

2. Combination of Kirk and GPU Gems

A POSITA would have been motivated to combine Kirk (an NVIDIA patent) and GPU Gems (a contemporaneous NVIDIA book) for many of the same reasons a POSITA would have combined Nickolls and GPU Gems. Section IV.A.2; Ex1003, ¶¶269–79.

GPU Gems teaches the benefits and discloses examples of using GPUs to perform mathematical simulations, such as the “simulation of ... a chemical reaction-diffusion,” because “significant speedups can be achieved by applying GPUs to data-parallel computational problems.” Ex1006, 483, 505–08. A POSITA would have been motivated to implement the techniques in GPU Gems on the most advanced GPUs at the time, which were provided by NVIDIA. Ex1003, ¶276.

One such GPU is disclosed in Kirk, which has multiple Execution Pipelines to process data in parallel. *Id.*, ¶273. There are numerous advantages to applying the numerical simulations in GPU Gems on the hardware of Kirk. *Id.*, ¶¶272–74. A POSITA would have understood that, through its multiple processing engines, the system of Kirk would significantly reduce execution time compared to other processing architectures. *Id.* The system of Kirk would also be advantageous due

to its capabilities for data sharing among threads. *Id.*, ¶¶274–75 With the connections between the programmable graphics processor and Local Memory 140, Kirk would have allowed the iterative computations of GPU Gems in which the outputs of one stage of computations become inputs to subsequent stages. *Id.*

A POSITA would have also had a reasonable expectation of success in implementing the numerical simulations of GPU Gems on the system of Kirk. *Id.*, ¶¶277–78. The system of Kirk was designed for running many tasks (threads) at the same time. Since complex mathematical simulations also require splitting dividing the computation into smaller, discrete tasks, a POSITA would have expected the system of Kirk to be able to use parallel processing to execute the simulations in GPU Gems. *Id.* Indeed, GPU Gems teaches exactly how to repurpose a GPU for performing numerical simulations. A POSITA would have further expected success in the combination because the system of Kirk has the correct memory architecture (*e.g.*, shared memory) for swapping pointers of outputs and inputs, as taught in GPU Gems. *Id.*

Likewise, a POSITA understanding the benefits of swapping pointers while performing numerical computations (as taught in GPU Gems) would have been motivated to apply those techniques when writing programs to perform numerical computations on the accelerator of Kirk. *Id.*, ¶277X. A POSITA would have understood the combination of Kirk and GPU Gems applied a known technique (*e.g.*,

the numerical simulations and pointer-swapping of GPU Gems) to a known device (*e.g.*, the GPU-accelerated parallel processing system of Kirk) ready for improvement to yield predictable results (*e.g.*, fast and efficient computation of complex numerical simulations). *Id.*, ¶279.

3. Detailed Analysis of Ground 2

a. Claim 16

i. 16[pre]

To the extent the preamble is limiting, Kirk/GPU Gems discloses it. Ex1003, ¶281–84.

Kirk discloses a “Computing System” with a “programmable graphics processor” for “processing of data.” Ex1005, 1:40–47, 3:35–41, 26:24–39, 26:56–61, Figs. 12A, 12B. Kirk includes a central processing unit (*e.g.*, Host Processor 114) and an accelerator (*e.g.*, Graphics Subsystem 107). *Id.*, 3:17–46, Figs. 1A, 12A.

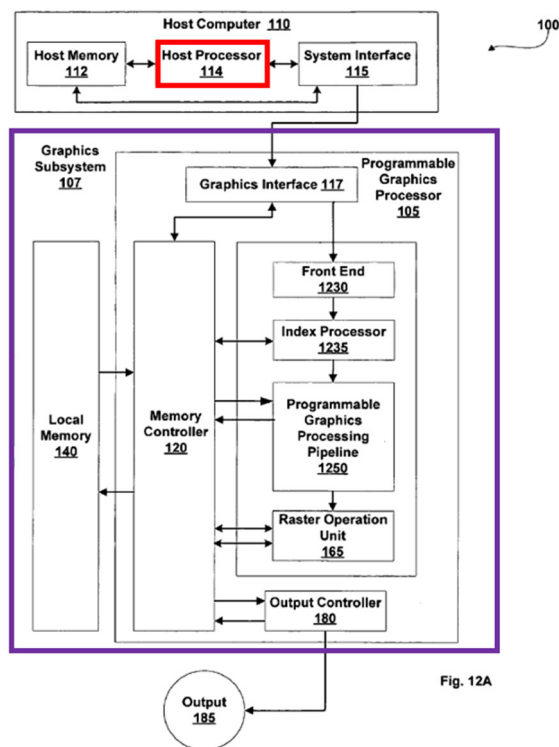


Fig. 12A

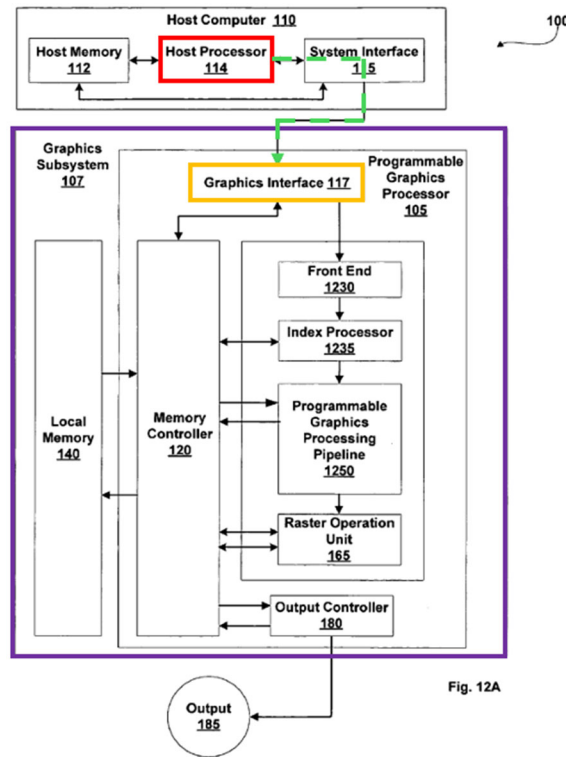
Id., Fig. 12A (annotated).

In the combination of Kirk and GPU Gems, when the chemical reaction-diffusion simulation is applied to the system of Kirk, the “input data” includes the initial conditions for the simulation. Ex1006, 505–08; Ex1003, ¶284.

ii. 16[a]

Kirk/GPU Gems discloses this limitation. Ex1003, ¶¶285–87.

The accelerator of Kirk (*e.g.*, Graphics Subsystem 107) receives input data from Host Processor 114 at Graphics Interface 117. Ex1005, 3:32–34, 3:50–67, 4:26–31; Ex1003, ¶286.



Id., Fig. 12A (annotated).

iii. 16[b]

Kirk/GPU Gems discloses this limitation. Ex1003, ¶¶288–97.

Kirk includes an “accelerator controller,” *e.g.*, Graphics Interface 117 along with Front End 1230 and Index Processor 1235. Ex1005, 3:32–67, 26:24–44. These components to constitute an “accelerator controller” because they receive, interpret, format, and send data and commands to the Programmable Graphics Processing Pipeline 1250 for execution. *Id.*; Ex1003, ¶289. Graphics Interface 117 and Index Processor 1235 are coupled to Local Memory 140 (*i.e.*, “accelerator memory”) through Memory Controller 120. Ex1005, 3:47–67, 26:24–39, Fig. 12A.

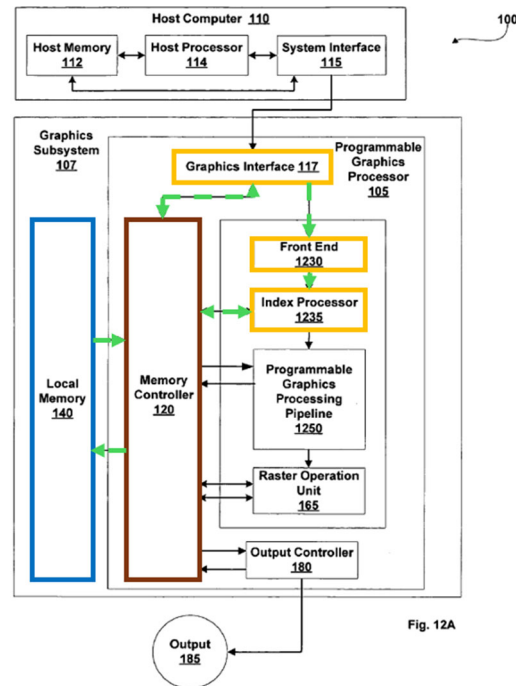


Fig. 12A

Id., Fig. 12A (annotated).

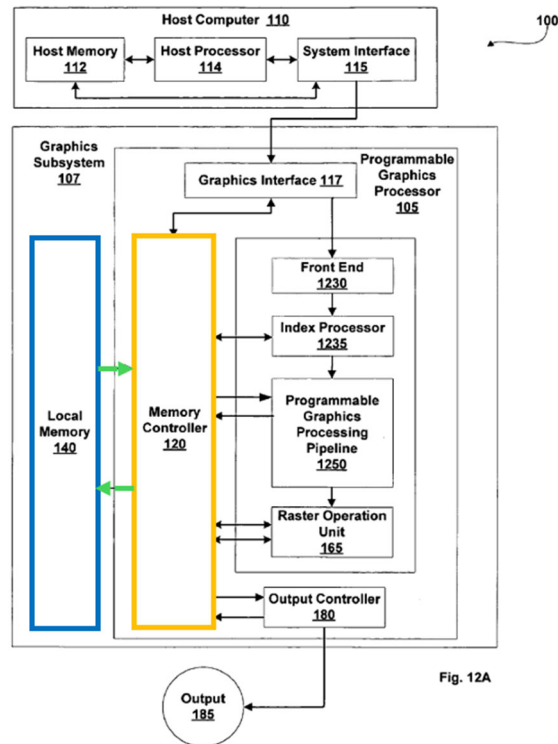
In Kirk, input and output data, as well as commands, are stored in Local Memory 140. *Id.*, 3:34–46. When input data is received in Graphics Subsystem 107 from the host computer, prior to any computations occurring on the data, Graphics Interface 117 transfers the data to Local Memory 140. *Id.*, 3:50–58. Kirk teaches identifying data by an “address for a location in graphics memory.” Ex1005, 8:3–7, 9:41–55. Figures 3A and 3B, for example, illustrate storing data in buffers, such as “Buffer 320,” which are accesses using an “address” such as by a pointer, as was known in the art. *Id.*, 9:32–55, Figs. 3A, 3B. Thus, for the same reasons explained for Ground 1, a POSITA would have understood the dedicated memory block of Local Memory 140 in which the input data for a computation is stored is a “first partition” of memory that is “referenced by [a] first pointer.” Ex1003, ¶¶289–91,

294; Section IV.A.3.a.iii. Moreover, in order to be used for the computations, the input data to start the computation is transferred to the first partition *before* the first computational cycle. Ex1003, ¶289. In any event, it would have been obvious to store the input data for the computations in a “first partition” in Local Memory 140, referenced by a first pointer, because it would enable efficient parallel processing, with each Execution Pipeline able to quickly and independently access the input data for its computations. *Id.*, ¶¶294, 297.

Kirk includes Memory Controller 120, which transfers data into and out of Local Memory 140 and communicates with the Programmable Graphics Processing Pipeline 1250. Ex1004, 3:47–67, 4:26–31, 26:24–39. In the Litigation, Patent Owner asserts in its Infringement Contentions that a memory controller in the accused products (like Memory Controller 120 in Kirk) is the claimed “accelerator controller.” Ex1010, 27 (“[The] Maxwell-architecture GPUs ... include ‘memory controllers’ (e.g., *accelerator controller*)...”).⁷ Thus, additionally and alternatively,

⁷ Petitioner disagrees with Patent Owner that a memory controller is an “accelerator controller” within the meaning of the ’867 patent. Nevertheless, for purpose of this alternative argument in this Petition only, Petitioner accepts Patent Owner’s allegation that “accelerator controller” in the claims of the ’867 patent is broad enough to encompass a memory controller.

under Patent Owner’s view of the claims, Kirk’s memory controller (either alone or in combination with Graphics Interface 117) is the claimed “accelerator controller,” which transfer data into Local Memory 140 prior to the start of computations.



Id., Fig. 12A.

GPU Gems teaches this limitation, as explained in Ground 1. Section IV.A.3.a.iii; Ex1003, ¶¶295–96.

iv. 16[c]

Kirk/GPU Gems discloses this limitation. Ex1003, ¶¶298–301.

Kirk includes Programmable Graphics Processing Pipeline 1250, which has multiple Execution Pipelines 1240, each of which “includes at least one multithreaded processing unit.” *Id.*, 27:2–11. Each execution pipeline

communicates with graphics memory (e.g., Local Memory 140) “to read program instructions and graphics data stored in buffers in graphics memory.” *Id.*, 27:12–28. In Kirk, “graphics data” refers to “any data that is input to or output from computation units within Programmable Graphics Processor 105.” *Id.*, 3:35–41, 26:44–61. Programmable Graphics Processing Pipeline 1250 as a whole (red) and the Execution Pipelines 1240 (blue) within the pipeline (because they are the processors the perform the graphics and other computational operations) are a “graphics processing unit.” *Id.*, 26:44–51; Ex1003, ¶299; Ex1001, 2:17–21. The graphics processing unit thus receives input data and performs, in a first computational cycle, at least one calculation on the input data, resulting in first output data. Ex1003, ¶300.

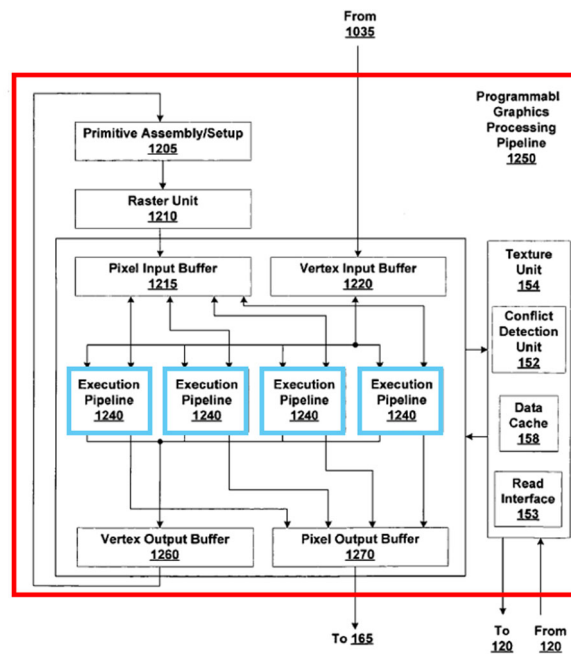


Fig. 12B

Id., Fig. 12B (annotated).

GPU Gems teaches this limitation, as explained in Ground 1. Section IV.A.3.a.iv; Ex1003, ¶301.

v. 16[d]

Kirk/GPU Gems discloses this limitation. Ex1003, ¶¶302–06.

In Kirk, the first output data from the execution pipelines of programmable graphics processing pipeline is stored in a partition of accelerator memory (*e.g.*, Local Memory 140). For example, as shown in Figure 12A, Programmable Graphics Processing Pipeline 1250 is in communication with Local Memory 140 through Memory Controller 120. Ex1005, 26:24–39, 27:19–28. Output data can be “written back to Local Memory 140.” *Id.*, 5:8–11, 5:28–32, 7:17–25. “Execution Pipelines 1240 are optionally configured ... such that data processing operations are performed in multiple passes through at least one multithreaded processing unit within Execution Pipelines 1240.” *Id.*, 28:33–37. In this disclosure, first output data is stored in a memory partition within Programmable Graphics Processing Pipeline 1250, such as Vertex Output Buffer 1260 or Texture Unit 154. *Id.*, 27:29–31, 27:24–28. Because these memory locations are within Graphics Subsystem 107, they are also “accelerator memory.” Ex1003, ¶304. In either case, for the same reasons the location of the input data in memory is considered a “first partition” and is referenced by a “first pointer,” so too is the location of the output data of each step considered a “second partition” and is referenced by a “second pointer.” *See* 16[b].

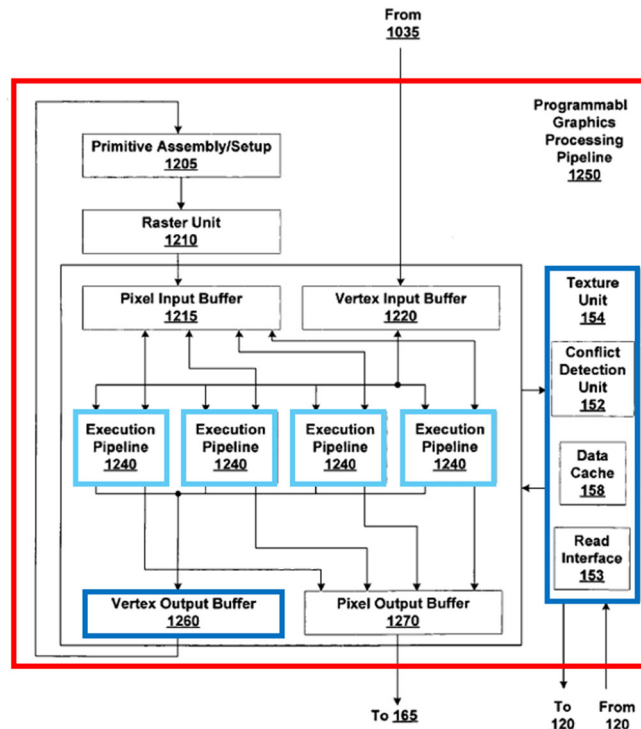


Fig. 12B

Id., Fig. 12B (annotated).

The storing of first output data occurs “by the accelerator controller”—*e.g.*, at the direction and control of the accelerator controller (Graphics Interface 117, along with Front End 1230 and Index Processor 1235). For example, Kirk explains that Graphics Interface 117 passes program instructions to other components of Programmable Graphics Processor 105 or to memory. *Id.*, 3:50–54, 4:21–42. Additionally, Front End 1230 receives, interprets, formats, and sends commands to the programmable graphics processing pipeline, and it is these instructions that cause output data to be stored in memory. *Id.*, 26:24–39. In the alternative mapping in which Memory Controller 120 is part of the “accelerator controller,” storing of first output data occurs “by the accelerator controller” because access to Local Memory

140 occurs “via Memory Controller 120.” Ex1004, 6:25–30, 3:47–67, 4:26–31, 6:34–55, 26:24–39; Ex1003, ¶305.

GPU Gems teaches this limitation, as explained in Ground 1. Section IV.A.3.a.v; Ex1003, ¶306.

vi. 16[e]

Kirk/GPU Gems discloses this limitation. Ex1003, ¶¶307–11.

Kirk discloses using the output of one computational cycle as an input for another computational cycle. Ex1005, 4:16–20. GPU Gems teaches that this would be accomplished, in a system such as Kirk’s, by swapping the first pointer with the second pointer. Section IV.A.3.a.vi; Ex1006, 508; Ex1003, ¶308.

It would also have been obvious to a POSITA to implement an algorithm on the hardware in Kirk in which the first and second pointers are swapped at the end of a computational cycle. Because Kirk describes using outputs as inputs to a new computational cycle (*see supra*), a POSITA would have been motivated to use well-known techniques (such as pointer swapping) to effectuate this efficiently. Ex1003, ¶¶309–11. One such known technique was pointer swapping, which avoided copying large chunks of memory. *Id.*; Ex1006, 44, 713, 743.

b. Claim 17

Kirk/GPU Gems discloses this limitation. Ex1003, ¶¶312–15.

Graphics Interface 117 receives and then passes program instructions to other components of Programmable Graphics Processor 105. *Id.*, 3:50–54, 4:21–42. For example, Front End 1230 receives, interprets, formats, and sends commands to the programmable graphics processing pipeline (a graphics processing unit). *Id.*, 26:24–39. When Programmable Graphics Processing Pipeline 1250 executes these instructions, it will perform the at least one calculation. *See* 16[c]. At a minimum, it would have been obvious to a POSITA to pass commands through an accelerator controller, which were well known in the art and would free up the CPU. Ex1003, ¶¶314–15.

c. Claim 18

Kirk/GPU Gems discloses this limitation. Ex1003, ¶¶316–21.

A POSITA would have understood that Local Memory 140 of Kirk is divided into partitions to store specific data. 16[b]. Petitioner incorporates by references its analysis above of the “first partition” (16[b]) and “second partition” (16[d]) of accelerator memory.

GPU Gems, as implemented on Kirk’s system, discloses this limitation. The memory in the reaction-diffusion simulation of GPU Gems is “double buffered.” Ex1006, 507. One buffer (partition) is `currentSource` and the other is `currentTarget`. *Id.*, 508. These partitions correspond to the claimed first and second partitions.

The system stores “internal variables” for the chemical-diffusion reaction, such as the physical locations of each node, passed as “coords,” and reaction parameters (*e.g.*, *k* and *F*). *Id.*, 507. The buffer in which these internal variables are stored is a “third partition.” Ex1003, ¶318.

GPU Gems discloses that it “bind[s] parameters” to the GPU program, “du” and “dv,” which correspond to diffusivity parameters, which is input to be used at “a particular computation cycle of the numerical simulation.” Ex1006, 507. The buffer in which this input data is stored is a “fourth partition.” Ex1003, ¶319. Additionally, the buffer (partition) array “stores data used as input at a particular computation cycle of the numerical simulation”—*i.e.*, the initial array at the first computational cycle—which is also in a “fourth partition.” Ex1006, 507–08.

d. Claim 19

Kirk/GPU Gems discloses this limitation. Ex1003, ¶¶322–25.

Kirk teaches transferring output data to Host Memory 112 (system memory). Ex1005, 5:8–11. This transfer occurs based on instructions received from the accelerator controller, as defined above. *Id.*, 3:50–54, 4:21–42, 26:24–39. In the alternative mapping in which Memory Controller 120 is part of the “accelerator controller,” storing of data in Host Memory 112 occurs “by the accelerator controller” when the transfer to host memory occurs “via Memory Controller 120.” *Id.*, 6:25–30, 6:49–55.

A POSITA would have understood that Kirk discloses that this transfer occurs during the second computational cycle. Kirk allows for data to be read from local memory “after the write to the source address is complete.” *Id.*, 1:48–54. Thus, once all writing to a source address is complete, the data could be read and transferred immediately to Host Memory 112. Ex1003, ¶324. In an implementation where output results are used as an input to a later computational cycle, the output data would be available to transfer to host memory as soon as a computational cycle completes, so the output data can be used as an input for the next data cycle *and* be transferred to Host Memory 112 simultaneously. *Id.*

It would also have been obvious to a POSITA to transfer, by the accelerator controller, the first output data to the main memory during the second computational cycle, as explained for Ground 1. *Id.*, ¶325; Section IV.A.3.d. For example, where there are many threads running and limited GPU memory, for efficient data and memory management and to not leave a processor sitting idle while waiting for the GPU to completely finish computations, a POSITA would have been motivated to immediately transfer intermediate results to system memory and free up GPU memory for further computational cycles. Ex1003. ¶325. Additionally, to more permanently store some intermediate results to keep a record of how a physical process proceeds over time, a POSITA would have been motivated to copy output data during the next computational cycle to system memory. *Id.*

e. Claim 1

i. 1[pre]

Kirk/GPU Gems discloses this limitation. Ex1003, ¶¶326–27.

See 16[pre]. Kirk discloses a “Computing System” with a “programmable graphics processor” for “processing of data.” Ex1005, 1:40–47, 3:35–41, 26:24–39, 26:56–61, Figs. 12A, 12B. The data processing of Kirk includes numerical simulations with multiple computational cycles. *Id.* As an example, in the combination of Kirk and GPU Gems, the “numerical simulation” is the multi-cycle chemical-diffusion reaction of Chapter 31. Ex1006, 505–08.

ii. 1[a]

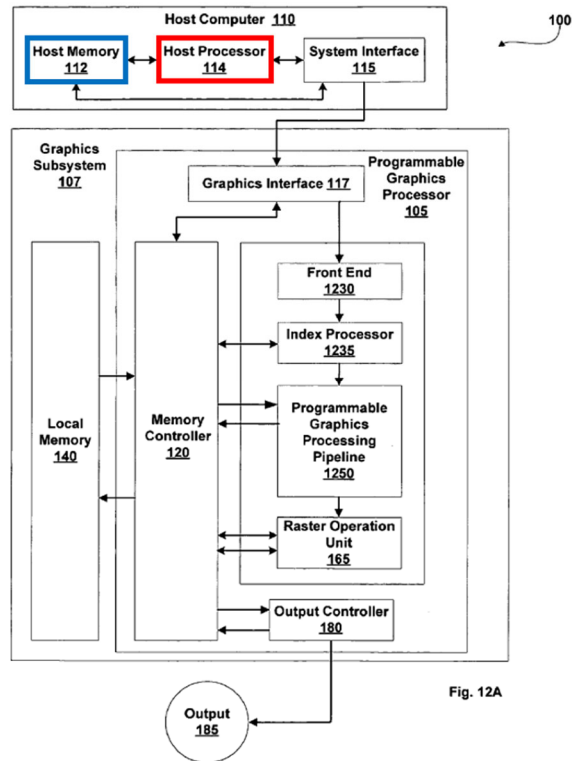
Kirk/GPU Gems discloses this limitation. Ex1003, ¶¶328–29.

See 16[pre].

iii. 1[b]

Kirk/GPU Gems discloses this limitation. Ex1003, ¶¶330–31.

Kirk discloses Host Memory 112, which is coupled to CPU 114. Ex1004, 3:17–31. Host Memory 112 stores input data (such as for numerical simulations), which the CPU can access and send to Graphics Subsystem 107 for processing. *Id.*, 3:32–34, 3:50–67, 4:26–31.

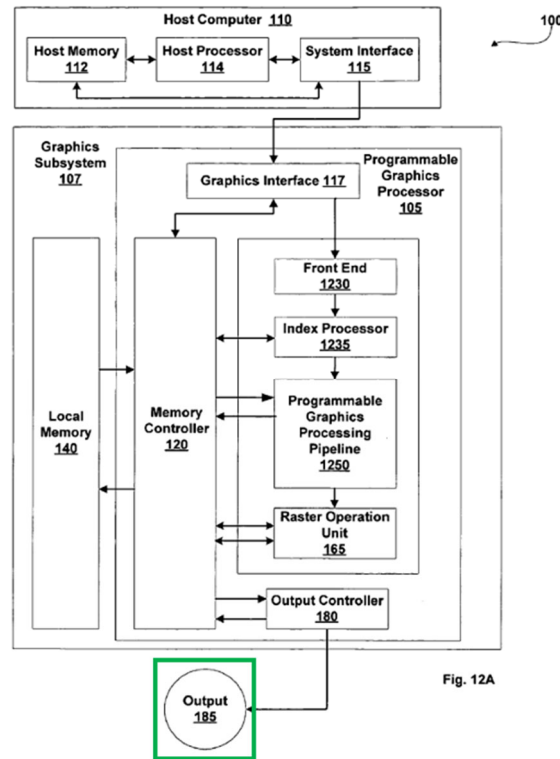


Ex1005, Fig. 12A (annotated).

iv. 1[c]

Kirk/GPU Gems discloses this limitation. Ex1003, ¶¶332–34.

Kirk includes output 185, coupled to Graphics Subsystem 107, which can be “a display or the like.” Ex1005, 5:60–6:3, 21:61–66, 26:10–18. When running numerical simulations, the system of Kirk would be used to display results of the simulation (*i.e.*, an “indication of the numerical simulation”). Ex1003, ¶333. At a minimum, in order to be able to view the results of the simulation, it would have been obvious to a POSITA to configure output 185 to display an indication (such as a result) of the numerical simulation. *Id.*, ¶334.



Ex1005, Fig. 12A (annotated).

v. 1[d]

Kirk/GPU Gems discloses this limitation. Ex1003, ¶¶335–37.

See 16[a], 16[b], 16[c]; Claim 19. As explained for Claim 19, Kirk discloses sending first output data to main memory after the conclusion of a first computational cycle. Once the output data is stored in Host Memory 112, it is available to Host Processor 114. Ex1003, ¶336. It would also have been obvious to provide first output data generated during the first computational cycle to the CPU after a conclusion of the first computational cycle so the data can be used by other programs in the system. *Id.*, ¶337.

vi. 1[d][i]

Kirk/GPU Gems discloses this limitation. Ex1003, ¶¶338–39.

See 16[c], 16[e]. Kirk discloses a graphics processing unit that generates output data during computational cycles based on input data. Ex1005, 27:2–28, 3:35–41, 26:44–61. Kirk further describes using the output of one computational cycle (*e.g.*, first output data) as input for another cycle of a simulation to generate a second output (*e.g.*, second output data). *Id.*, 4:16–20. For example, in the reaction-diffusion example of GPU Gems, the output of one step of the computation is used as an input to the next step, which generates a further output. Ex1006, 506–08.

vii. 1[d][ii]

Kirk/GPU Gems discloses this limitation. Ex1003, ¶¶340–41.

See 16[b], 16[d].

viii. 1[d][ii][1]

Kirk/GPU Gems discloses this limitation. Ex1003, ¶¶342–43.

See 16[b], 16[d]. As explained for 16[d], after a first computation on input data is complete, first output data is stored in a partition (a “second partition” in Claim 16 but a “first partition” in Claim 1) that is referenced by a pointer (a “second pointer” in Claim 16 but a “first pointer” in Claim 1).

ix. 1[d][ii][2]

Kirk/GPU Gems discloses this limitation. Ex1003, ¶¶344–45.

See 16[b], 16[d]. For similar reasons as above, second output data from a computational cycle of the system of Kirk is stored in a partition (a “second partition”) that is referenced by a pointer (a “second pointer”).

x. 1[e]

Kirk/GPU Gems discloses this limitation. Ex1003, ¶¶346–47.

See 16[d]. Kirk includes an “accelerator controller,” *e.g.*, Graphics Interface 117 along with Front End 1230 and Index Processor 1235, which are operably coupled to both Local Memory 140 (through Memory Controller 120) and Host Processor 114 (through System Interface 115). Ex1005, 3:17–67, 26:24–44. Furthermore, as shown below, if Memory Controller 120 is considered to be part of the claimed “accelerator controller,” it too is operably coupled to Local Memory 140 and Host Processor 114.

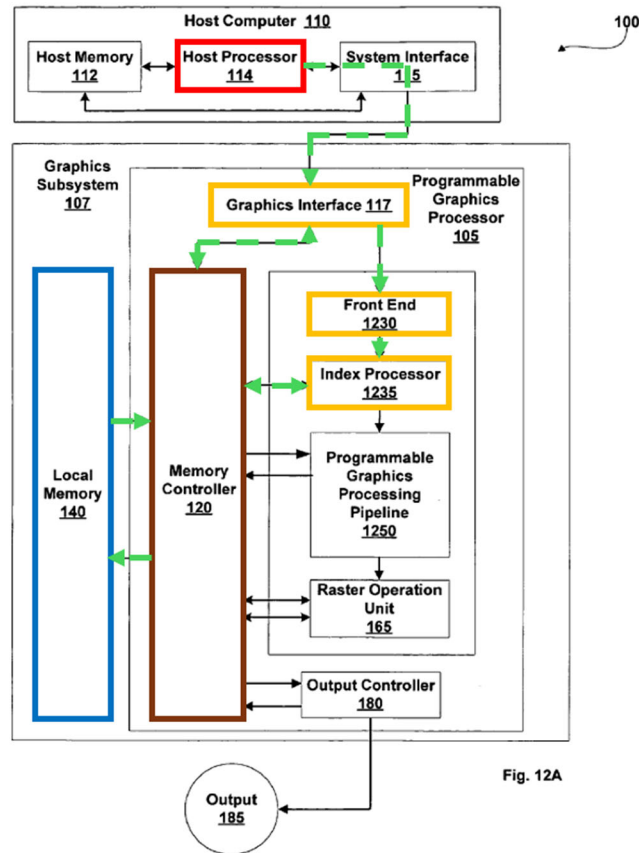


Fig. 12A

Ex1005, Fig. 12 (annotated).

xi. 1[e][i]

Kirk/GPU Gems discloses this limitation. Ex1003, ¶¶348–49.

See 16[b].

xii. 1[e][ii]

Kirk/GPU Gems discloses this limitation. Ex1003, ¶¶350–51.

See Claim 19.

xiii. 1[e][iii]

Kirk/GPU Gems discloses this limitation. Ex1003, ¶¶352–53.

See 16[d]. As explained for 16[d], in Kirk, output data is stored, by an accelerator controller, in a partition during a computational cycle. For similar reasons, as required for this element, the accelerator controller directs second output data into a second partition during a second computational cycle. Ex1005, 5:8–11, 5:28–32, 7:17–25.

xiv. 1[e][iv]

Kirk/GPU Gems discloses this limitation. Ex1003, ¶¶354–55.

See 16[e]. Just as pointers are swapped at the end of a first computational cycle, such that the output of the first computational cycle becomes the input to a second, pointers are swapped at the end of the second computational cycle, such that the output of the second computational cycle becomes the input to a third. Ex1005, 4:16–20; Ex1006, 507–08.

f. Claim 2

Kirk/GPU Gems discloses this limitation. Ex1003, ¶¶356–57.

The accelerator controller of Kirk (*e.g.*, Graphics Interface 117 along with Front End 1230 and Index Processor 1235) dictates an order of execution of instructions to Programmable Graphics Processing Pipeline 1250. Ex1005, 26:24–39; Ex1003, ¶357. For example, “Front End 120 interprets and formats the commands and outputs the formatted commands and data to an Index Processor 1235,” which then get passed to Programmable Graphics Processing Pipeline 1250.

Id., 26:29–31. Alternatively, to the extent Memory Controller 120 is considered the “accelerator controller,” a POSITA would have understood the memory controller to dictate the order of execution of instructions by passing instructions to Programmable Graphics Processing Pipeline 1250 in a specific order. Ex1005, 26:35–39; Ex1003, ¶357.

g. Claim 3

Kirk/GPU Gems discloses this limitation. Ex1003, ¶¶358–59.

See 16[a]. A POSITA would have understood that Graphics Interface 117 uses an interface controller (not depicted in the figures) in order to communicate with System Interface 115, which is in further communication with Host Processor 114 over a bus. Ex1005, 3:23–34, 3:47–54; Ex1003, ¶359. The interface controller would include the circuitry and/or logic that interfaces through the bus. *Id.* Similarly, to the extent Memory Controller 120 is considered an “accelerator controller,” a POSITA would understand that it has an interface to communicate over a bus (through Graphics Interface 117 and System Interface 115) with Host Processor 114. *Id.*, 3:23–34, 3:47–67; Ex1003, ¶359.

h. Claim 4

Kirk/GPU Gems discloses this limitation. Ex1003, ¶¶360–61.

See 16[b], 16[d]. A POSITA would understand “texture memory bank” to refer to memory in which textures (or data arrays) are stored and “shader memory

bank” to refer to memory in which shaders (or instructions) are stored. Ex1001, 5:1–12; Ex1003, ¶361. A POSITA would have understood the memory partition into which input data is transferred (16[b]) and in which output data is stored (16[b]) to be part of a “texture memory bank.” Furthermore, in Kirk, “program instructions” for the programmable graphics processing pipeline are stored in local memory in Graphics Subsystem 107, which a POSITA would have understood to include a “shader memory bank.” Ex1004, 3:32–39, 3:50–57; Claim 7. The processing instructions stored in the shader memory bank are executed by the processing cores of the GPU to perform operations on the input data. *Id.*

GPU Gems also discloses this limitation, as explained for Ground 1.

i. Claim 5

Kirk/GPU Gems discloses this limitation. Ex1003, ¶¶362–63.

This limitation was addressed with respect to Claim 18, with the same caveats as explained for Ground 1. Because the data stored in the fourth partition in Claim 18 is an array of data (i.e., a “texture”), Claim 5 is disclosed for the same reasons as Claim 18.

j. Claim 6

Kirk/GPU Gems discloses this limitation. Ex1003, ¶¶364–65.

See 16[b], 16[c]. Graphics Interface 117, along with Front End 1230 and Index Processor 1235, supply input data and instructions to the Programmable

Graphics Processing Pipeline 1250, which then manages their execution. Ex1005, 27:12–28. Alternatively, Memory Controller 120 (by virtue of its connection to Local Memory 140) supplies input data and instructions to Programmable Graphics Processing Pipeline. *Id.*, 3:32–67, 26:24–39.

k. Claim 7

Kirk/GPU Gems discloses this limitation. Ex1003, ¶¶366–67.

Because Front End 1230 “interprets and formats the commands and outputs the formatted commands a data,” it stores instructions on a memory during the process. Ex1005, 26:24–35; Ex1003, ¶367. Alternatively, a POSITA would have understood that Memory Controller 120 temporarily stores instructions on a memory when communicating the instructions from Local Memory 140 to Programmable Graphics Processing Pipeline 1250. Ex1004, 26:35–39; Ex1003, ¶367.

l. Claim 8

Kirk/GPU Gems discloses this limitation. Ex1003, ¶¶368–71.

See 16[pre], 16[a]. In the combination of Kirk and GPU Gems, the “input data” may include the initial conditions for the chemical-diffusion reaction, such as the initial U and V chemical concentrations for the reaction, diffusion rates (D_u and D_v), reaction parameters (*e.g.*, k and F), and the initial state of the array. Ex1006, 505–08; Ex1003, ¶370.

m. Claim 9

i. 9[pre]

Kirk/GPU Gems discloses this limitation. Ex1003, ¶¶372–73.

See 1[pre], 16[pre].

ii. 9[a]

Kirk/GPU Gems discloses this limitation. Ex1003, ¶¶374–75.

See 1[d][i], 16[c].

iii. 9[b]

Kirk/GPU Gems discloses this limitation. Ex1003, ¶¶376–77.

See 1[d][ii], 16[b].

iv. 9[c]

Kirk/GPU Gems discloses this limitation. Ex1003, ¶¶378–79.

See 1[d][i], 1[e], 16[c], 16[d].

v. 9[c][i]

Kirk/GPU Gems discloses this limitation. Ex1003, ¶¶380–81.

See 1[d][i], 16[c]. The computations performed by the processing engines are controlled by a “computational stream” because the “accelerator controller” passes a stream of commands or instructions to the graphics processing unit to perform computations. Ex1005, 3:32–67, 26:24–39, 27:12–28; Ex1003, ¶¶381–83.

vi. 9[c][ii]

Kirk/GPU Gems discloses this limitation. Ex1003, ¶¶384–85.

See 16[d]. The transfer of output data from the GPU to accelerator memory is controlled by an “output stream” because the “accelerator controller” passes a stream of instructions to the graphics processing unit, which includes a direction to return an output to a location specified by the controller. Ex1005, 5:8–11, 5:28–32, 7:17–25, 26:24–44, 27:12–31.

vii. 9[c][iii]

Kirk/GPU Gems discloses this limitation. Ex1003, ¶¶386–87.

See 1[d], 16[a], 16[d]. The transfer of input data to accelerator memory is controlled by an “input stream” because the “accelerator controller” receives input data from Host Processor 114, and the input data is directed into accelerator memory. Ex1005, 3:32–67, 26:24–44, 27:12–31.

n. Claim 10

Kirk/GPU Gems discloses this limitation. Ex1003, ¶¶388–89.

See Claim 2.

o. Claim 11

Kirk/GPU Gems discloses this limitation. Ex1003, ¶¶390–91.

See Claims 6, 19; 1[e][ii].

p. Claim 12

Kirk/GPU Gems discloses this limitation. Ex1003, ¶¶392–93.

See Claim 4.

q. Claim 13

Kirk/GPU Gems discloses this limitation. Ex1003, ¶¶394–95.

See Claims 5, 18. Although the order of what is on each partition differs in Claims 5 and 18 from Claim 13, they collectively disclose the required storage in the four partitions required in Claim 13.

r. Claim 14

Kirk/GPU Gems discloses this limitation. Ex1003, ¶¶396–97.

See 1[e][iv], 16[e].

s. Claim 15

Kirk/GPU Gems discloses this limitation. Ex1003, ¶¶398–99.

See Claim 7.

V. ENABLEMENT AND SECONDARY CONSIDERATIONS

The prior art in this Petition is enabled. Ex1003, ¶¶112, 119, 125.

Petitioner is not aware of any secondary considerations of non-obviousness. *Id.*, ¶400. To the extent Patent Owner asserts any secondary considerations, Petitioner may request leave to file a reply. In any event, any alleged secondary considerations of non-obviousness cannot overcome the strong showing of obviousness detailed above. Where, as here, there is a strong showing of obviousness, secondary considerations cannot overcome a *prima facie* case of

obviousness. *Ohio Willow Wood Co. v. Alps South, LLC*, 735 F.3d 1333, 1344 (Fed. Cir. 2013).

VI. PTAB DISCRETION SHOULD NOT PRECLUDE INSTITUTION

A. Discretionary Denial Under Section 314(a) Is Not Warranted

Under the Director's Interim Guidance, the Board is instructed that "[c]onsistent with *Sotera Wireless, Inc.*, the PTAB will not discretionarily deny institution in view of parallel district court litigation where a petitioner presents a stipulation not to pursue in a parallel proceeding the same grounds or any grounds that could have reasonably been raised before the PTAB." Interim Guidance, 3; *Sotera Wireless, Inc. v. Masimo Corp.*, IPR2020-01019, Paper 12 (P.T.A.B. Dec. 1, 2020) (precedential). Petitioner stipulates that if IPR is instituted, Petitioner will not pursue in the Litigation the same grounds as in this Petition or any grounds that could have reasonably been raised in this Petition. Ex1031. Thus, the Board should not exercise its discretion to deny institution.

B. Discretionary Denial Under Section 325(d) Is Not Warranted

No reference relied on in this Petition was considered during prosecution of the '867 patent. Each Ground in this Petition therefore presents new prior art and arguments not previously before the Office.

VII. CONCLUSION

Petitioner has established a reasonable likelihood the Challenged Claims are unpatentable. Petitioner therefore respectfully requests that IPR be instituted.

DATED: February 24, 2025

Respectfully Submitted,

/Brian M. Buroker/

Brian M. Buroker (Reg. No. 39,125)
GIBSON, DUNN & CRUTCHER LLP
1700 M Street, NW
Washington, DC 20036
Phone: (202) 955-8500
Fax: (202) 467-0539
Email: bburoker@gibsondunn.com

Attorney for Petitioner NVIDIA Corporation

CERTIFICATION UNDER 37 C.F.R. § 42.24(d)

Under the provisions of 37 C.F.R. § 42.24(d), the undersigned attorney hereby certifies that the word count for Sections I–VII of the foregoing Petition for *Inter Partes* Review is 13,987 according to the word count tool in Microsoft Word.

DATED: February 24, 2025

Respectfully Submitted,

/Brian M. Buroker/

Brian M. Buroker (Reg. No. 39,125)
GIBSON, DUNN & CRUTCHER LLP
1700 M Street, NW
Washington, DC 20036
Phone: (202) 955-8500
Fax: (202) 467-0539
Email: bburoker@gibsondunn.com

Attorney for Petitioner NVIDIA Corporation

CERTIFICATE OF SERVICE

The undersigned certifies service pursuant to 37 C.F.R. §§ 42.6(e) and 42.105(a), (b) on the Patent Owner via FedEx overnight mail of a copy of this Petition for *Inter Partes* Review, all exhibits and supporting materials, and Power of Attorney at the correspondence address of record for the '867 patent:

COOLEY LLP
ATTN: IP Docketing Department
1299 Pennsylvania Avenue, NW
Suite 700
Washington, DC

A courtesy copy has also been mailed to and served on litigation counsel at:

Christopher C. Campbell
KING & SPALDING LLP
1700 Pennsylvania Avenue, NW
Suite 900
Washington, DC 20006
ccampbell@kslaw.com

DATED: February 24, 2025

Respectfully Submitted,

/Brian M. Buroker/

Brian M. Buroker (Reg. No. 39,125)
GIBSON, DUNN & CRUTCHER LLP
1700 M Street NW
Washington, DC 20036
Phone: (202) 955-8500
Fax: (202) 467-0539
Email: bburoker@gibsondunn.com
Attorney for Petitioner NVIDIA Corporation

APPENDIX: CHALLENGED CLAIM LISTING

No.	Limitation
1[pre]	A computer system for performing a numerical simulation over a plurality of computational cycles including at least a first computational cycle and a second computational cycle, the computer system comprising:
1[a]	a central processing unit;
1[b]	a main memory, operably coupled to the central processing unit, to store input data to be accessed by the central processing unit in performing the numerical simulation;
1[c]	a video system, operably coupled to the central processing unit, to drive a video monitor to display an indication of the numerical simulation in response to the computer system performing the numerical simulation;
1[d]	an accelerator, operably coupled to the central processing unit, to receive at least a portion of the input data from the central processing unit and to provide first output data generated during the first computational cycle to the central processing unit after a conclusion of the first computational cycle, the accelerator comprising:
1[d][i]	at least one graphics processing unit to generate second output data, during the second computational cycle, by performing at least one calculation on the first output data; and
1[d][ii]	an accelerator memory, operably coupled to the at least one graphic processing unit, the accelerator memory comprising:
1[d][ii][1]	a first partition, referenced by a first pointer, to store the first output data during the second computational cycle; and
1[d][ii][2]	a second partition, referenced by a second pointer, to store the second output data generated during the second computational cycle; and
1[e]	an accelerator controller, operably coupled to the accelerator memory and the central processing unit,
1[e][i]	to transfer the at least the portion of the input data into the accelerator memory before the first computational cycle,

No.	Limitation
1[e][ii]	to transfer the first output data from the accelerator memory to the main memory during the second computational cycle,
1[e][iii]	to direct the second output data into the second partition during the second computational cycle, and
1[e][iv]	to swap the first pointer and the second pointer at the conclusion of the second computational cycle such that the second output data becomes an input for a third computational cycle of the plurality of computational cycles.
2	The computer system as claimed in claim 1, wherein the accelerator controller is configured to dictate an order of execution of instructions to the at least one graphics processing unit.
3	The computer system as claimed in claim 1, wherein the accelerator controller comprises an interface controller to communicate with the central processing unit over a bus of the computer system.
4	The computer system as claimed in claim 1, wherein the accelerator memory comprises a texture memory bank to store the at least the portion of the input data and the first output data and a shader memory bank to store instructions for performing a set of operations to be performed on the at least the portion of the input data by the at least one graphic processing unit.
5	The computer system as claimed in claim 4, wherein the texture memory is partitioned into the first partition, the second partition, a third partition to store internal variables, a fourth partition to store data textures used as input at a particular computation cycle of the plurality of computational cycles.
6	The computer system as claimed in claim 1, wherein the accelerator controller inputs the at least the portion of the input data and a series of instructions into the at least one graphic processing unit, wherein the at least one graphics processing unit then executes the instructions on the at least the portion of the input data.
7	The computer system as claimed in claim 1, wherein the accelerator controller comprises a set of instructions stored on a memory.

No.	Limitation
8	The computer system as claimed in claim 1, wherein the at least the portion of the input data represents an initial condition of the numerical simulation.
9[pre]	An accelerator system for a computer system performing a numerical simulation, the accelerator system comprising:
9[a]	at least one graphics processing unit to generate output data by performing at least one computation during a first computational cycle of the numerical simulation;
9[b]	an accelerator memory, operably coupled to the at least one graphics processing unit, to store data used to perform the at least one computation; and
9[c]	an accelerator controller, operably coupled to the accelerator memory and the at least one graphics processing unit, to execute:
9[c][i]	a computational stream controlling performance of the at least one computation by the at least one graphics processing unit;
9[c][ii]	an output stream controlling transfer of the output data from the at least one graphics processing unit to the accelerator memory during the first computational cycle; and
9[c][iii]	an input stream controlling transfer of input data to the accelerator memory for use by the at least one graphics processing unit during a second computational cycle of the numerical simulation.
10	The accelerator system as claimed in claim 9, wherein the accelerator controller is configured to dictate an order of execution of instructions to the at least one graphics processing unit.
11	The accelerator system as claimed in claim 9, wherein the accelerator controller is configured to send instructions to the at least one graphics processing unit, wherein the at least one graphics processing unit is configured to execute the instructions on the input data, and wherein the accelerator controller is configured to transfer the output data from the accelerator memory to a main memory during the execution of the instructions by the at least one graphics processing unit.

No.	Limitation
12	The accelerator system as claimed in claim 9, wherein the accelerator memory comprises a texture memory bank to store the input data and the output data and a shader memory bank to store instructions for processing operations to be performed on the input data by the at least one graphic processing unit.
13	The accelerator system as claimed in claim 12, wherein the texture memory is partitioned into a first partition to store the input data, a second partition to store internal variables, a third partition to store data textures used as input at a particular computation cycle of the numerical simulation, and a fourth partition to store the output data.
14	The accelerator system as claimed in claim 9, wherein the accelerator controller is configured to perform successive computational cycles of the numerical simulation by feeding the output data generated by the at least one graphic processing unit from a previous computational cycle and the input data for a next computational cycle into the at least one graphic processing unit.
15	The accelerator system as claimed in claim 9, wherein the accelerator controller comprises a set of instructions stored in a memory.
16[pre]	A method for performing a numerical simulation on input data in a computer system including a central processing unit and an accelerator, the method comprising:
16[a]	receiving, by an accelerator, first input data from the central processing unit;
16[b]	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;
16[c]	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;
16[d]	storing, by the accelerator controller, the first output data into a second partition, referenced by a second pointer, of the accelerator memory; and

No.	Limitation
16[e]	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.
17	The method as claimed in claim 16, further comprising: sending, by the accelerator controller, instructions for performing the at least one calculation to the at least one graphics processing unit.
18	The method as claimed in claim 16, further comprising: 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.
19	The method of claim 16, further comprising: transferring, by the accelerator controller, the first output data to the main memory during the second computational cycle.