

Ex. C12 – Invalidity of U.S. Patent No. RE49,461 – Buck ’977

Prior Art Reference: U.S. Patent Application Publication No. 2005/0197977A1 (“Buck ’977”)

Buck ’977 was filed on June 25, 2004 and therefore qualifies as prior art to U.S. Patent No. RE49,461 (the “’461 patent”) under, at least, (pre-AIA) 35 U.S.C. § 102(a), (b) and/or (e). Buck ’977 anticipates and/or renders obvious claims 21–31 of the ’461 patent (the “Asserted Claims”).

The chart below provides representative examples of where each element of each claim is found within Buck ’977. Citations are meant to be exemplary, not exhaustive, and NVIDIA reserves the right to identify and discuss additional portions of the reference in support of its contentions and/or to rebut arguments made by Plaintiff. Where NVIDIA states that Buck ’977 “discloses” a limitation, that disclosure may be express, implicit, and/or inherent. Citations to figures, drawings, tables, and the like include reference to any accompanying or related text. All internal cross-references are meant to incorporate the cross-referenced material as if fully set forth therein.

It is NVIDIA’s position that Plaintiff’s Infringement Contentions have not established that any accused product or service infringes any valid claim. Thus, NVIDIA’s statements below should not be treated as an admission, implication, or suggestion that NVIDIA agrees with Plaintiff regarding either the scope, construction, or interpretation of any of the Asserted Claims or the infringement theories advanced by Plaintiff in its Infringement Contentions, including whether any Asserted Claim satisfies 35 U.S.C. §§ 101 or 112. The exemplary disclosures in these invalidity contentions are informed by the apparent interpretation of the asserted claims by Plaintiff. These statements are not intended to suggest that NVIDIA agrees with Plaintiff’s application of any claim term, suggest a proposed construction at this stage of the case, or suggest that construction is needed.

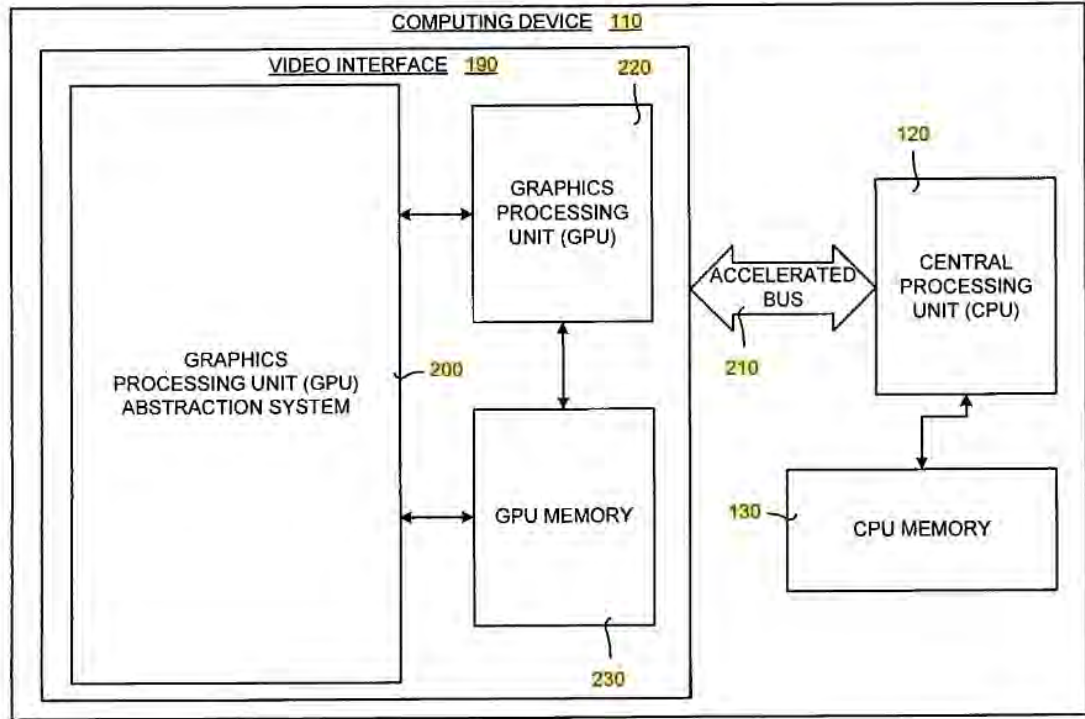
To the extent Plaintiff argues that any element below is not disclosed by Buck ’977, a person of ordinary skill in the art would have found it obvious to combine the teachings of Buck ’977 with the background knowledge of a person of ordinary skill in the art and/or the additional references, and exemplary teachings, set forth in NVIDIA’s Invalidity Contentions, including all exhibits and appendices thereto. Additional exemplary combinations and reasons for obviousness are described in the cover pleading, and are incorporated by reference herein.

Plaintiff has yet to identify any limitation of the Asserted Claims that it contends is not anticipated and/or rendered obvious by Buck ’977. NVIDIA therefore expressly reserves the right to respond to any such contention, including by identifying additional obviousness combinations, if Plaintiff makes any such contention. NVIDIA reserves the right to rely on additional citations or sources of evidence that also may be applicable, or that may become applicable in light of claim construction, changes in Plaintiff’s infringement contentions, and/or information obtained during discovery as the case progresses.

Ex. C12 – Invalidity of U.S. Patent No. RE49,461 – Buck ’977

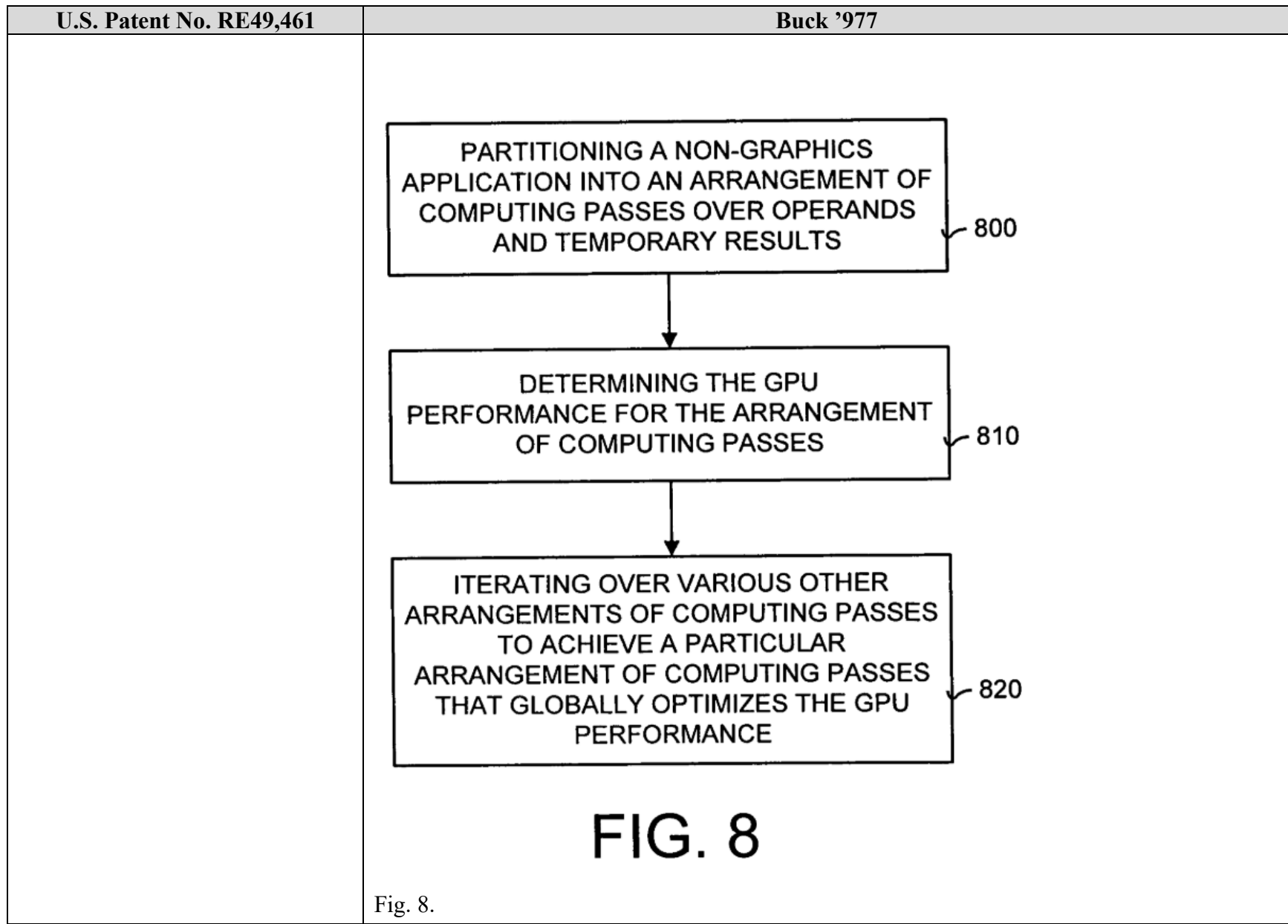
U.S. Patent No. RE49,461	Buck ’977
21[pre]: A method of executing computations representing an artificial neural network on a computer system comprising at least one central processing unit (CPU), a processing unit, a first memory partition, and a second memory partition, the method comprising:	NVIDIA takes no position in these Invalidity Contentions on whether the entirety of the preamble of this claim is a limitation. To the extent it is a limitation, Buck ’977 discloses, expressly and/or inherently a method of executing computations representing an artificial neural network on a computer system comprising at least one central processing unit (CPU), a processing unit, a first memory partition, and a second memory partition. <i>See e.g.:</i> <i>As a non-limiting example, Buck ’977 discloses a central processing unit (e.g., 130 in Fig. 2).</i>

Ex. C12 – Invalidity of U.S. Patent No. RE49,461 – Buck '977

U.S. Patent No. RE49,461	Buck '977
	 FIG. 2 Fig. 2. <i>As a non-limiting example, Buck '977 discloses a first memory partition and second memory partition (e.g., 620, 630, 640 as the first partition, and 610 as the second partition in Fig. 6).</i>

Ex. C12 – Invalidity of U.S. Patent No. RE49,461 – Buck '977

U.S. Patent No. RE49,461	Buck '977
	The diagram shows a system 600 enclosed in a large rectangle. Inside, there are three rectangular blocks at the top: 620 (vertical), 630 (square), and 640 (horizontal). Below them is a dashed oval labeled 650. At the bottom center is a square block labeled 610. Arrows point from each of the top blocks (620, 630, 640) down to block 610. A dashed line with an arrow at its right end (labeled 650) originates from the area of blocks 620 and 630 and points towards block 610. The text 'OPERAND READS' is placed to the left of the dashed line 650, and 'OPERATION DESTINATION' is placed below block 610. The number 600 is at the bottom right corner of the system rectangle. FIG. 6 Fig. 6. <i>As a non-limiting example, Buck '977 discloses a method of executing computations representing an artificial neural network on a computer system (e.g., 800, 810, 820 in Fig. 8)</i>



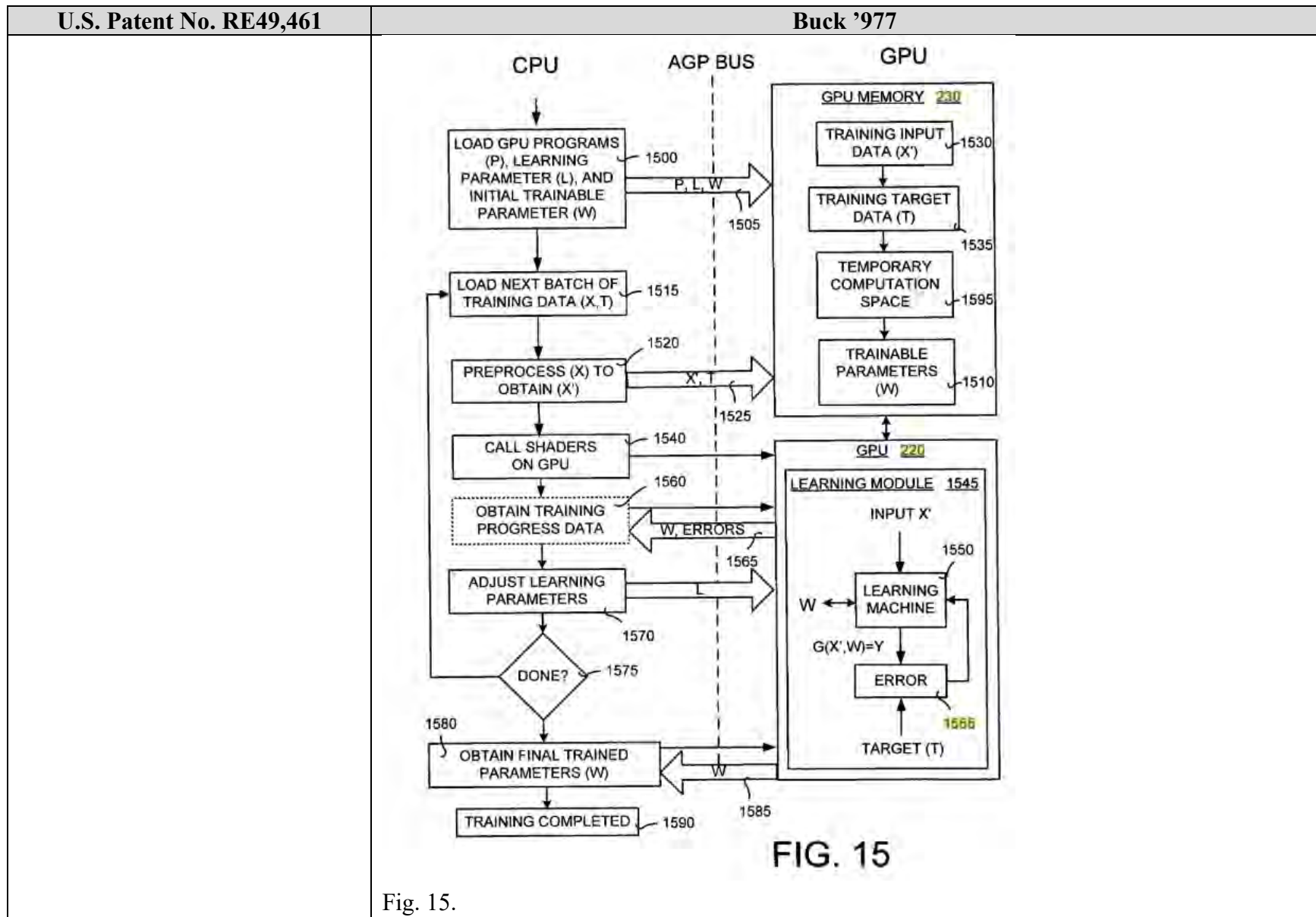
Ex. C12 – Invalidity of U.S. Patent No. RE49,461 – Buck '977

U.S. Patent No. RE49,461	Buck '977
	<i>As a non-limiting example, Buck '977 discloses a first memory partition and second memory partition (e.g., W1, W2 as first and second partitions in Fig. 9).</i> The diagram illustrates a memory layout. A large rectangle is divided into a top section and a bottom section. The top section contains several smaller rectangles: 'W1' on the left, followed by two narrow vertical rectangles labeled 'H' and '?', then 'W2', followed by two more narrow vertical rectangles labeled 'O' and '?'. The bottom section of the large rectangle is labeled 'SCRATCH SPACE' and 'WORKSPACE'. To the right of the large rectangle are two vertical rectangles, each labeled 'INPUT VECTORS'. Below these two rectangles is the label 'READ ONLY BUFFERS'. A bracket at the bottom left of the large rectangle points to the label '900'. FIG. 9 Fig. 9. <i>As a non-limiting example, Buck '977 discloses a method of executing computations representing an artificial neural network on a computer system (e.g., Figs. 15-16).</i>

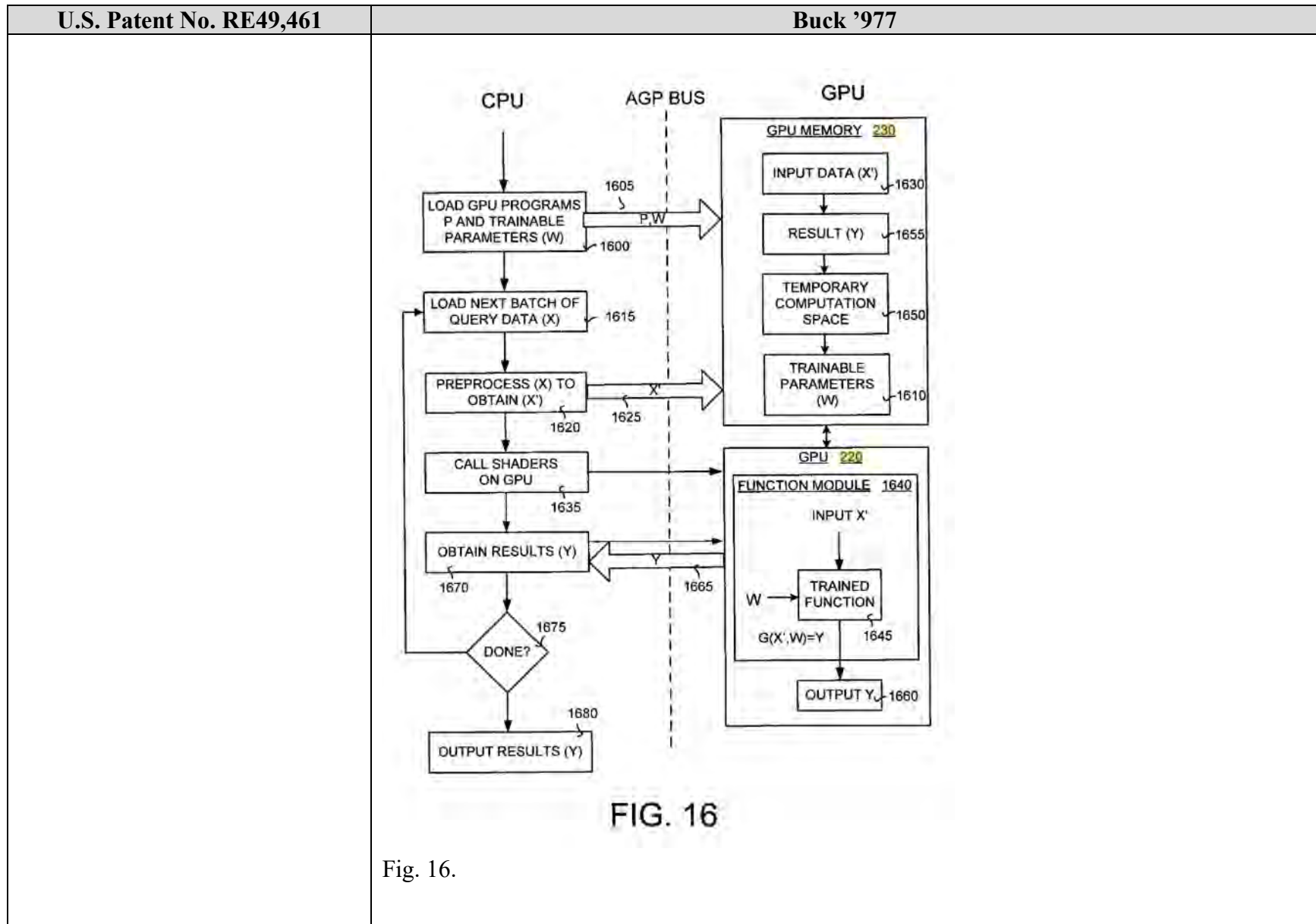
Ex. C12 – Invalidity of U.S. Patent No. RE49,461 – Buck '977

U.S. Patent No. RE49,461	Buck '977

Ex. C12 – Invalidity of U.S. Patent No. RE49,461 – Buck '977



Ex. C12 – Invalidity of U.S. Patent No. RE49,461 – Buck '977



Ex. C12 – Invalidity of U.S. Patent No. RE49,461 – Buck '977

U.S. Patent No. RE49,461	Buck '977
	A system and method for optimizing the performance of a graphics processing unit (GPU) for processing and execution of general matrix operations such that the operations are accelerated and optimized. The system and method describes the layouts of operands and results in graphics memory, as well as partitioning the processes into a sequence of passes through a macro step. Specifically, operands are placed in memory in a pattern, results are written into memory in a pattern appropriate for use as operands in a later pass, data sets are partitioned to insure that each pass fits into fixed sized memory, and the execution model incorporates generally reusable macro steps for use in multiple passes. These features enable greater efficiency and speed in processing and executing general matrix operations. Abstract. The present invention relates in general to computer processing and more particularly to a system and method for optimizing performance of a graphics processing unit (GPU) for processing and execution of general matrix operations such that the operations are accelerated and optimized. A graphics processing unit (GPU) is a standard element of the modern desktop personal computer (PC). Initially a feature of high-end graphics workstations, the GPU has found its way onto the PC bus as an accelerator of graphics functions for which a conventional central processing unit (CPU) was ill suited or simply too slow. Presently, the GPU is a prominent component of the PC with its own dedicated path to main CPU memory as well as its own dedicated graphics memory. Interactive computer graphics began as line drawings on calligraphic displays, which were basically modified oscilloscopes. The computation for these early displays required vector operations including general geometric transformations, clipping to boundaries of the display devices, and perspective transformations for three-dimensional (3D) displays. The advent of inexpensive commodity semiconductor memory prompted the replacement of these line

Ex. C12 – Invalidity of U.S. Patent No. RE49,461 – Buck ’977

U.S. Patent No. RE49,461	Buck ’977
	drawing systems by raster graphics processors, which refreshed television-like displays through a frame buffer memory. [0002]-[0004]. Co-processors are not a new feature of personal computers. The integral floating point co-processor shares the host CPU's instruction stream and has access to CPU memory. Additional integrated co-processors include CPU extensions such as multimedia extensions (MMX) or streaming SIMD extensions (SSE), which have parallel data paths, asynchronous execution, and also access to the CPU memory. GPUs, on the other hand are autonomous special purpose processors with their own instruction streams, datapaths, and dedicated memory. Trends in GPU design and configuration have given them larger dedicated memory, higher, bandwidth to graphics memory, increased internal parallelism, and increased degrees of programmability. [0008]. FIG. 2 is a block diagram illustrating an exemplary implementation of the GPU optimization system and method disclosed herein. [0016]. FIG. 6 illustrates a GPU memory organization for vectors of the GPU optimization system and method. [0020]. FIG. 8 is a flow diagram illustrating a general operation of the GPU optimization method of the GPU optimization system. [0022].

Ex. C12 – Invalidity of U.S. Patent No. RE49,461 – Buck '977

U.S. Patent No. RE49,461	Buck '977
	FIG. 9 is a block diagram illustrating the segmentation of the graphics memory. [0023]. FIG. 15 is a flow diagram illustrating an example of the data flow and control structure of a training session performed on the graphics processing unit in the working example. [0030]. FIG. 16 is a flow diagram illustrating an example of the data flow and control structure of a testing session performed on the graphics processing unit in the working example. [0031]. Components of the computer 110 may include, but are not limited to, a processing unit 120 (such as a central processing unit, CPU), a system memory 130, and a system bus 121 that couples various system components including the system memory to the processing unit 120. The system bus 121 may be any of several types of bus structures including a memory bus or memory controller, a peripheral bus, and a local bus using any of a variety of bus architectures. By way of example, and not limitation, such architectures include Industry Standard Architecture (ISA) bus, Micro Channel Architecture (MCA) bus, Enhanced ISA (EISA) bus, Video Electronics Standards Association (VESA) local bus, and Peripheral Component Interconnect (PCI) bus also known as Mezzanine bus. [0040]. Fortunately, many machine learning techniques, such as neural networks, expectation minimization, and many other gradient descent based algorithms are composed of simple primitives (or building blocks). These primitives include: Inner products (between vectors or matrix and vector)

Ex. C12 – Invalidity of U.S. Patent No. RE49,461 – Buck '977

U.S. Patent No. RE49,461	Buck '977
	Outer products (between vectors) Linear algebra (such as addition, subtraction, multiplication by a scalar on vectors or matrices) Non-linearity (such as tanh, sigmoid, thresholding) applied to a vector or a matrix Matrix transpose [0071]. The GPU optimization system and method implements each of these operations using pixel shaders. In addition, the method allows each of these operations to be used together for both training and use in a real setting. It should be noted that the method can be used with neural networks or any other learning algorithms made out of the same primitives as described below, or simple extensions thereof. [0077]. While either memory layout can be used with the associated code fragment shown below it, it has been seen that for the matrix-vector inner product, the layout shown in FIG. 10A leads to a more efficient algorithm. However, the layout shown in FIG. 10B can be used to implement a transposed matrix-vector multiply. This is necessary for the neural net training algorithm, since in-memory transposition of matrices is too expensive. In other words, it involves a complete copy of the largest amount of array data. [0122]. The characteristics of the class of problems that the invention disclosed herein seeks to solve include 1) large matrices or vectors, 2) operations too complex to fit in a single pass, and 3) iteration. In such applications, there is excessive cost for moving data to or from the host CPU. Thus, it is desirable to do as much computation as possible within the GPU and its memory. Using neural net training as a specific example, it can be seen that a number of matrix operations are required.

Ex. C12 – Invalidity of U.S. Patent No. RE49,461 – Buck ’977

U.S. Patent No. RE49,461	Buck ’977
	[0143]. The above has shown that the machine learning GPU execution method contains all the operations needed for training neural nets (and any other machine learning algorithm made up of the same operations). Further, these operations can be implemented as pixel shaders on a GPU, and in all cases the algorithms lend themselves to a high degree of parallelism to provide maximum processing efficiency. [0193]. This working example illustrates the GPU optimization system and method used to perform handwritten character recognition. The task was to find the class (‘0’ to ‘9’) of a pixel image of a handwritten digit. The training database was made up of 60,000 training images, with their correct labels. This training set is sometimes referred to as MNIST and is available on the web, and is well-known in the machine learning algorithm literature. To solve this task, a neural network approach was used. In particular, the neural network was a 2-layer fully connected neural network. [0196]. A number of machine learning techniques contain, as a kernel, computation that can be cast as very large matrix operations. Of particular interest are large matrix/vector multiplies. By way of example, neural networks used for handwriting recognition typically contain a pair of layers. FIG. 14 is a detailed illustration of a pair of layers in a neural network used in the working example. [0210]. In this section, an embodiment of the invention used to do handwritten character recognition will be discussed. The task is to find the class (‘0’ to ‘9’) of a pixel image of a handwritten

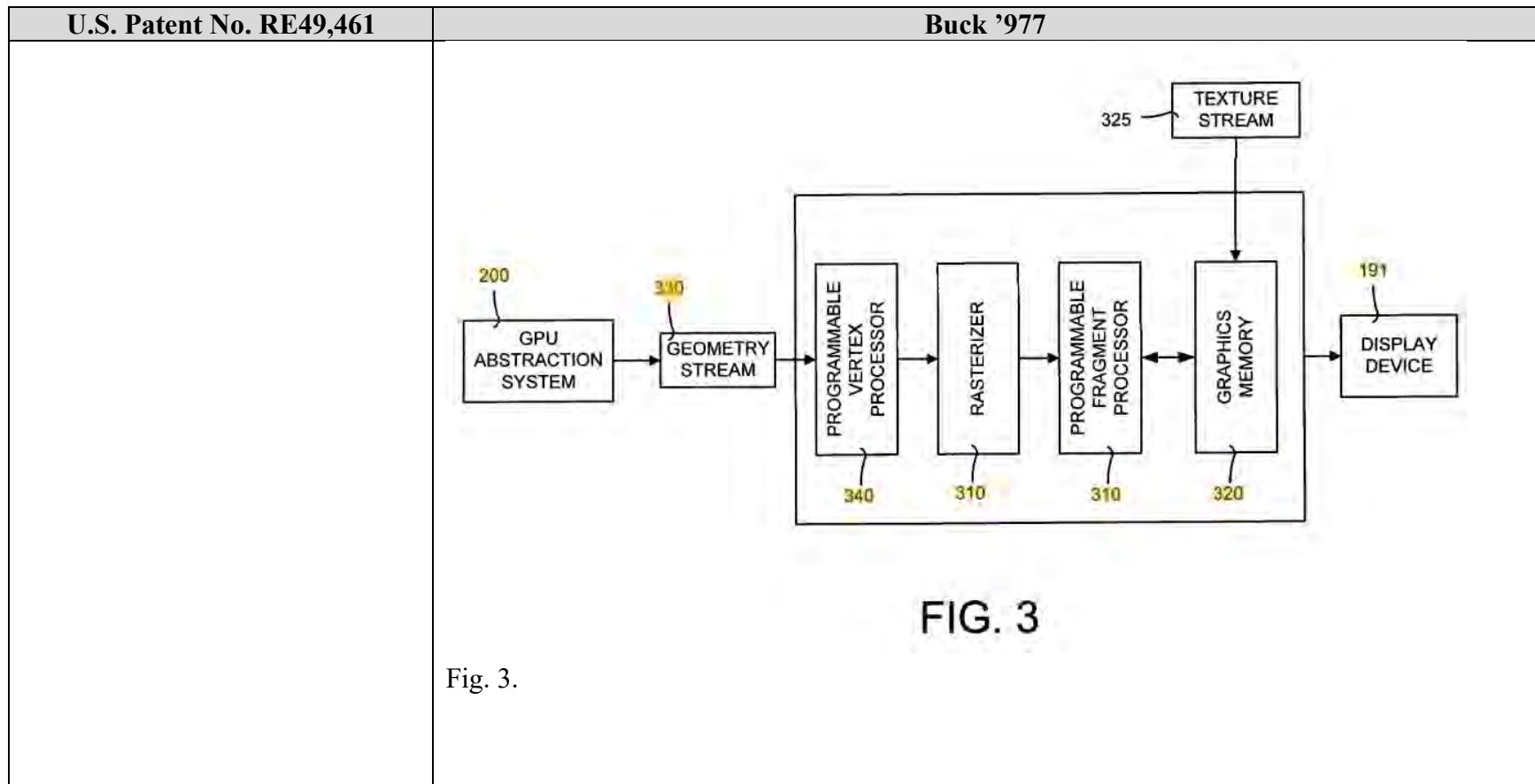
Ex. C12 – Invalidity of U.S. Patent No. RE49,461 – Buck ’977

U.S. Patent No. RE49,461	Buck ’977
	digit. The training database is made up of 60,000 training images together with their correct labels. This training set is sometimes referred to as MNIST and is available on the web, and is well-known in the machine learning algorithm literature. To solve this task, a Neural Network approach is used, and in particular a simple 2-layer fully connected network. [0212]. <i>As a non-limiting example, Buck ’977 discloses a method of executing computations representing an artificial neural network.</i> For training the 2 layer neural networks, the forward and backpropagation correspond to about twenty different shaders (some of which are called multiple times). The number and complexity of shaders can of course vary depending on the algorithm used. The shaders are called for each pattern in a group (500 times in our case). For stochastic gradient descent, the learning parameters are updated after processing each pattern in the group. For batch gradient descent, the learning parameter gradients are accumulated over several patterns before the learning parameters are updated. Whether stochastic or batch gradient descent is used depends heavily on the application and the learning algorithm (and for some learning algorithms, such as SVM, the questions does not arise). For handwriting recognition and neural networks, stochastic gradient descent is preferable, and this is what is implemented in this example. [0219]. To the extent Plaintiff contends that this claim element is not expressly or inherently disclosed, or is not obvious in view of Buck ’977 and the knowledge of a POSITA, it would have been obvious to combine the teachings of Buck ’977 with any of the prior art references in NVIDIA’s Invalidity Contentions and any supplements thereto and the relevant section of charts for other prior art for the ’461 patent in a manner that would result in the subject matter of this limitation given, at the very least, the person's understanding of the state of the art, the problems addressed and solved in the prior art, and the teachings of Buck ’977. Specific

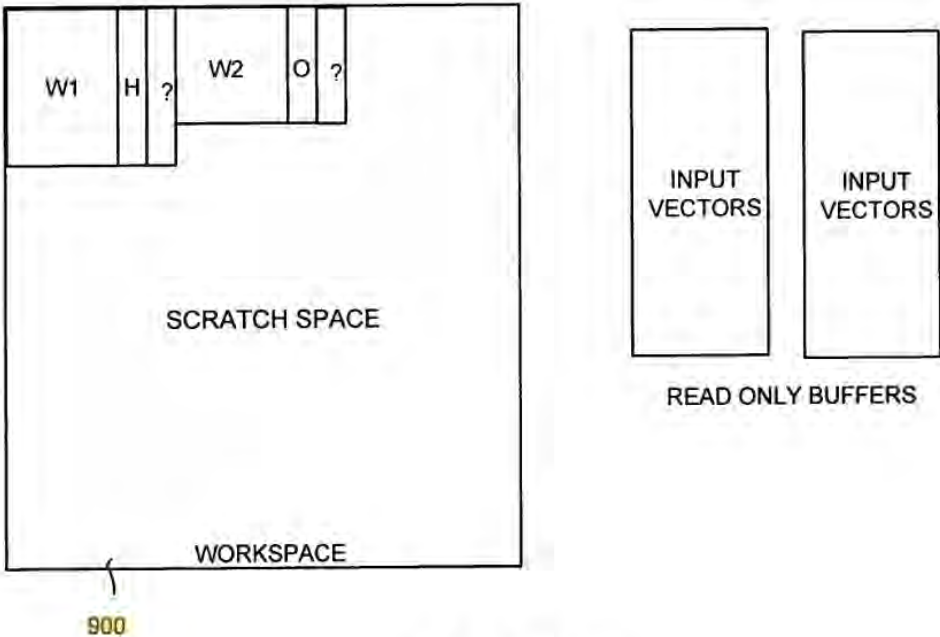
Ex. C12 – Invalidity of U.S. Patent No. RE49,461 – Buck '977

U.S. Patent No. RE49,461	Buck '977
	examples of such references are cited and explained in these invalidity contentions, including the cover pleading and charts C1–C15, and Appendix C.
21[a]: executing, by the at least one CPU, a user interaction stream, the user interaction stream controlling transfer of inputs to the artificial neural network to the first memory partition and the second memory partition;	Buck '977 discloses, expressly and/or inherently, executing, by the at least one CPU, a user interaction stream, the user interaction stream controlling transfer of inputs to the artificial neural network to the first memory partition and the second memory partition. <i>See e.g.:</i> <i>As a non-limiting example, Buck '977 discloses a user interaction stream controlling transfer of inputs to the artificial neural network to the first memory partition and the second memory partition where 200, 330, and 325 are the user interaction streams and 320 is the first and second memory partition.</i>

Ex. C12 – Invalidity of U.S. Patent No. RE49,461 – Buck '977



Ex. C12 – Invalidity of U.S. Patent No. RE49,461 – Buck '977

U.S. Patent No. RE49,461	Buck '977
	 The diagram, labeled FIG. 9, illustrates a memory layout. A large rectangular area is divided into a top section and a bottom section. The top section contains several smaller rectangular blocks labeled 'W1', 'H', '?', 'W2', 'O', and '?'. The bottom section is labeled 'SCRATCH SPACE' and 'WORKSPACE'. To the right of this main area are two vertical rectangular blocks labeled 'INPUT VECTORS', which are collectively labeled 'READ ONLY BUFFERS' below them. A bracket at the bottom left of the main area points to the label '900'. FIG. 9 Fig. 9. Co-processors are not a new feature of personal computers. The integral floating point co-processor shares the host CPU's instruction stream and has access to CPU memory. Additional integrated co-processors include CPU extensions such as multimedia extensions (MMX) or streaming SIMD extensions (SSE), which have parallel data paths, asynchronous execution, and also access to the CPU memory. GPUs, on the other hand are autonomous special purpose processors with their own instruction streams, datapaths, and dedicated memory. Trends in GPU design and configuration have given them larger dedicated memory, higher, bandwidth to graphics memory, increased internal parallelism, and increased degrees of programmability.

Ex. C12 – Invalidity of U.S. Patent No. RE49,461 – Buck '977

U.S. Patent No. RE49,461	Buck '977
	[0008]. FIG. 3 is a block diagram illustrating the interaction between the GPU and GPU optimization system shown in FIG. 2. [0017]. FIG. 9 is a block diagram illustrating the segmentation of the graphics memory. [0023]. In particular, the GPU optimization system and method optimizes the performance of a GPU for matrix operations and associated applications. The GPU optimization system and method achieve this in at least two ways. First, the GPU optimization system and method cascades a series of computing passes over operands and temporary results. Second, the GPU optimization system and method iterates over various combinations of computing passes to achieve a particular combination that globally optimizes GPU performance. [0035]. <i>As a non-limiting example, Buck '977 discloses at least one CPU (e.g., 120).</i> Components of the computer 110 may include, but are not limited to, a processing unit 120 (such as a central processing unit, CPU), a system memory 130, and a system bus 121 that couples various system components including the system memory to the processing unit 120. The system bus 121 may be any of several types of bus structures including a memory bus or memory controller, a peripheral bus, and a local bus using any of a variety of bus architectures. By way of example, and not limitation, such architectures include Industry Standard Architecture (ISA) bus, Micro Channel Architecture (MCA) bus, Enhanced ISA (EISA) bus, Video Electronics Standards Association (VESA) local bus, and Peripheral Component Interconnect (PCI) bus also known as Mezzanine bus.

Ex. C12 – Invalidity of U.S. Patent No. RE49,461 – Buck ’977

U.S. Patent No. RE49,461	Buck ’977
	[0040]. The drives and their associated computer storage media discussed above and illustrated in FIG. 1, provide storage of computer readable instructions, data structures, program modules and other data for the computer 110. In FIG. 1, for example, hard disk drive 141 is illustrated as storing operating system 144, application programs 145, other program modules 146, and program data 147. Note that these components can either be the same as or different from operating system 134, application programs 135, other program modules 136, and program data 137. Operating system 144, application programs 145, other program modules 146, and program data 147 are given different numbers here to illustrate that, at a minimum, they are different copies. A user may enter commands and information into the computer 110 through input devices such as a keyboard 162 and pointing device 161, commonly referred to as a mouse, trackball or touch pad. [0047]. Other input devices (not shown) may include a microphone, joystick, game pad, satellite dish, scanner, radio receiver, or a television or broadcast video receiver, or the like. These and other input devices are often connected to the processing unit 120 through a user input interface 160 that is coupled to the system bus 121, but may be connected by other interface and bus structures, such as, for example, a parallel port, game port or a universal serial bus (USB). A monitor 191 or other type of display device is also connected to the system bus 121 via an interface, such as a video interface 190. In addition to the monitor, computers may also include other peripheral output devices such as speakers 197 and printer 196, which may be connected through an output peripheral interface 195. [0048]. The video interface 190 includes a GPU 220 and a GPU memory 230. The GPU 220 is capable of transferring data to the CPU 120 over the bus 210. The GPU optimization system 200 is also in communication with the GPU 220 and the GPU memory 230. Results of the processing

Ex. C12 – Invalidity of U.S. Patent No. RE49,461 – Buck ’977

U.S. Patent No. RE49,461	Buck ’977
	by the GPU optimization system and method 200 are transferred from the GPU optimization system 200 to the CPU 120 for use in interactive use applications, such as speech recognition. [0054]. The GPU 220 is a complex single-chip processor, containing well over 100 million transistors. Its components are grouped into the functional units outlined in the block diagram of FIG. 3. FIG. 3 is a block diagram illustrating the interaction between the GPU and GPU system shown in FIG. 2. For general purpose computation, a programmable fragment processor (or shader) 310 and a dedicated graphics memory 320 are used. The programmable fragment processor 310 may include a single instruction/multiple data (SIMD) processor array that provides parallel processing such that many processing elements perform the same operations on different data. A texture stream from the GPU optimization system 200 is received by the graphics memory 320. In a preferred embodiment, the graphics memory includes approximately 100 megabytes or more of (DRAM) [0055]. A geometry stream 330 from the GPU optimization system 200 consists of a single polygon per processing pass. A programmable vertex processor 340 is idle except for passing this single polygon. A rasterizer 350 operates on this single polygon to effect a “do loop” for the programmable fragment processor 310. Output from the interaction of the GPU optimization system 200 and the GPU 220 is displayed on the display device 191. [0056]. The texture mapping hardware then serves as the address generator for input operands. A third feature of GPUs, the programmable fragment shader 310 shown in FIG. 3, permits functions of these operands to be written into the target rectangle 400. FIG. 5 illustrates a GPU memory organization for scalars of the GPU optimization system and method. FIG. 6 illustrates a GPU memory organization for vectors of the GPU optimization system and method. FIGS. 5 and 6 are discussed in detail below. [0061].

Ex. C12 – Invalidity of U.S. Patent No. RE49,461 – Buck ’977

U.S. Patent No. RE49,461	Buck ’977
	A pixel shader is expressed as a series of instructions in DirectX shader assembly language, which is a limited, hardware-independent language defined by DirectX. The code in a shader is executed once for each pixel in a triangle being rendered, and its only effect is to set the values of the 4-vector for that pixel. The limitations of the shader language, and the lack of side effects, mean that the GPU is free to render pixels in any order and using as much parallelism as its hardware can support, resulting in very high performance. The fact that a pixel is a 4-vector affords yet another kind of parallelism; each execution of a pixel shader can calculate four elements (e.g. four adjacent elements of a vector) simultaneously. [0080]. The key to optimization of GPU performance is to make maximum use of SIMD parallelism in the GPU pixel shaders. This requires that the operands (the inputs to the overall process) and the intermediate results (the outputs from and inputs to intermediate computing passes) be stored in memory in an arrangement that supports SIMD execution. The 4-component parallelism of GPUs is an additional axis of arrangement that must be considered when optimizing for SIMD execution. In addition, this GPU optimization is subject to constraints imposed by current GPU architectures. [0113]. Program access to graphics memory differs in several ways from memory access from conventional CPUs. FIG. 9 is a block diagram illustrating the segmentation of the graphics memory. As shown in FIG. 9, the graphics memory 900 is segmented (such as being accessed in 2K×2K chunks), accessed with two dimensional addresses, and each reference is to a floating point 4-vector. For general purpose programs, the read/write memory is constrained to a 2K×2K workspace. Additional read only buffers (segments) outside this workspace may be used as input data. [0114].

Ex. C12 – Invalidity of U.S. Patent No. RE49,461 – Buck ’977

U.S. Patent No. RE49,461	Buck ’977
	In FIG. 9, part of the scratch space is temporary buffer and part is a permanent buffer. By placing everything into a single scratch space, the space where the optimization GPU method is reading from and writing to never has to change. However, it can vary within the scratch space. [0115]. The characteristics of the class of problems that the invention disclosed herein seeks to solve include 1) large matrices or vectors, 2) operations too complex to fit in a single pass, and 3) iteration. In such applications, there is excessive cost for moving data to or from the host CPU. Thus, it is desirable to do as much computation as possible within the GPU and its memory. Using neural net training as a specific example, it can be seen that a number of matrix operations are required. [0143]. To the extent Plaintiff contends that this claim element is not expressly or inherently disclosed, or is not obvious in view of Buck ’977 and the knowledge of a POSITA, it would have been obvious to combine the teachings of Buck ’977 with any of the prior art references in NVIDIA’s Invalidity Contentions and any supplements thereto and the relevant section of charts for other prior art for the ’461 patent in a manner that would result in the subject matter of this limitation given, at the very least, the person's understanding of the state of the art, the problems addressed and solved in the prior art, and the teachings of Buck ’977. Specific examples of such references are cited and explained in these invalidity contentions, including the cover pleading and charts C1–C15, and Appendix C.
21[b]: executing, by the processing unit, a computational stream, the computational stream controlling data exchange between the user interaction stream and the computational stream during execution of the computations	Buck ’977 discloses, expressly and/or inherently, executing, by the processing unit, a computational stream, the computational stream controlling data exchange between the user interaction stream and the computational stream during execution of the computations representing the artificial neural network. <i>See e.g.:</i>

Ex. C12 – Invalidity of U.S. Patent No. RE49,461 – Buck ’977

U.S. Patent No. RE49,461	Buck ’977
representing the artificial neural network;	<i>As a non-limiting example, Buck ’977 discloses a computational stream controlling data exchange between a user interaction stream and a computational stream where 200 contains the computational stream.</i> As shown in FIG. 2, the computing device 110 contains a GPU optimization system 200. The computing device further contains the CPU 120 and the CPU memory 130. The CPU is in communication with the video interface 190 through an accelerated bus 210. Preferably, this bus 210 is an Accelerated Graphics Port (AGP), which is designed especially for the throughput demand of 3D graphics. The video interface 190 includes a GPU 220 and a GPU memory 230. The GPU 220 is capable of transferring data to the CPU 120 over the bus 210. The GPU optimization system 200 is also in communication with the GPU 220 and the GPU memory 230. Results of the processing by the GPU optimization system and method 200 are transferred from the GPU optimization system 200 to the CPU 120 for use in interactive use applications, such as speech recognition. The GPU 220 is a complex single-chip processor, containing well over 100 million transistors. Its components are grouped into the functional units outlined in the block diagram of FIG. 3. FIG. 3 is a block diagram illustrating the interaction between the GPU and GPU system shown in FIG. 2. For general purpose computation, a programmable fragment processor (or shader) 310 and a dedicated graphics memory 320 are used. The programmable fragment processor 310 may include a single instruction/multiple data (SIMD) processor array that provides parallel processing such that many processing elements perform the same operations on different data. A texture stream from the GPU optimization system 200 is received by the graphics memory 320. In a preferred embodiment, the graphics memory includes approximately 100 megabytes or more of (DRAM) A geometry stream 330 from the GPU optimization system 200 consists of a single polygon per processing pass. A programmable vertex processor 340 is idle except for passing this single polygon. A rasterizer 350 operates on this single polygon to effect a “do loop” for the

Ex. C12 – Invalidity of U.S. Patent No. RE49,461 – Buck ’977

U.S. Patent No. RE49,461	Buck ’977
	programmable fragment processor 310. Output from the interaction of the GPU optimization system 200 and the GPU 220 is displayed on the display device 191. [0053]-[0056]. The texture mapping hardware then serves as the address generator for input operands. A third feature of GPUs, the programmable fragment shader 310 shown in FIG. 3, permits functions of these operands to be written into the target rectangle 400. FIG. 5 illustrates a GPU memory organization for scalars of the GPU optimization system and method. FIG. 6 illustrates a GPU memory organization for vectors of the GPU optimization system and method. FIGS. 5 and 6 are discussed in detail below. [0061]. A pixel shader is expressed as a series of instructions in DirectX shader assembly language, which is a limited, hardware-independent language defined by DirectX. The code in a shader is executed once for each pixel in a triangle being rendered, and its only effect is to set the values of the 4-vector for that pixel. The limitations of the shader language, and the lack of side effects, mean that the GPU is free to render pixels in any order and using as much parallelism as its hardware can support, resulting in very high performance. The fact that a pixel is a 4-vector affords yet another kind of parallelism; each execution of a pixel shader can calculate four elements (e.g. four adjacent elements of a vector) simultaneously. [0080]. The key to optimization of GPU performance is to make maximum use of SIMD parallelism in the GPU pixel shaders. This requires that the operands (the inputs to the overall process) and the intermediate results (the outputs from and inputs to intermediate computing passes) be stored in memory in an arrangement that supports SIMD execution. The 4-component parallelism of GPUs is an additional axis of arrangement that must be considered when optimizing for SIMD execution. In addition, this GPU optimization is subject to constraints imposed by current GPU architectures.

Ex. C12 – Invalidity of U.S. Patent No. RE49,461 – Buck '977

U.S. Patent No. RE49,461	Buck '977
	[0113]. Program access to graphics memory differs in several ways from memory access from conventional CPUs. FIG. 9 is a block diagram illustrating the segmentation of the graphics memory. As shown in FIG. 9, the graphics memory 900 is segmented (such as being accessed in 2K×2K chunks), accessed with two dimensional addresses, and each reference is to a floating point 4-vector. For general purpose programs, the read/write memory is constrained to a 2K×2K workspace. Additional read only buffers (segments) outside this workspace may be used as input data. [0114]. In FIG. 9, part of the scratch space is temporary buffer and part is a permanent buffer. By placing everything into a single scratch space, the space where the optimization GPU method is reading from and writing to never has to change. However, it can vary within the scratch space. [0115]. The characteristics of the class of problems that the invention disclosed herein seeks to solve include 1) large matrices or vectors, 2) operations too complex to fit in a single pass, and 3) iteration. In such applications, there is excessive cost for moving data to or from the host CPU. Thus, it is desirable to do as much computation as possible within the GPU and its memory. Using neural net training as a specific example, it can be seen that a number of matrix operations are required. [0143]. In this working example, the GPU optimization system and method was run on a single processor 2.8 GHz Pentium 4 with an ATI® Radeon 9800 graphics card. The 9800 graphics card has 94 locations of instruction memory and 256 MB of working space, 16 MB of which

Ex. C12 – Invalidity of U.S. Patent No. RE49,461 – Buck '977

U.S. Patent No. RE49,461	Buck '977
	we use for our read/write workspace, the remaining is used for read only data. The CPU-only reference implementations are already optimized to take advantage of the SSE SIMD extensions of the Pentium 4, so the comparisons are SIMD-to-SIMD. Still the GPU acceleration is nearly an order of magnitude for the examples tested. [0195]. In the next step, the CPU starts to loop on the training data (box 1515), and accumulates groups of training data. The reason for the groups is that there is a cost of initiating a data transfer between CPU and GPU. Transferring the data by groups of several patterns at a time is more efficient. In our training session, the groups consist of 500 patterns. The patterns are pairs of 28 by 28 pixels images (X) and their target labels (T). [0215]. FIG. 15 is a flow diagram illustrating an example of the data flow and control structure of a training session performed on the graphics processing unit in the working example. Note that the FIG. 15 is not specific to neural network training: many other learning algorithms (SVM, K-means, LVQ, EM, etc.) can use the same architecture. In this example, the CPU gives instruction to the GPU and blocks until the GPU complete these instructions. It is also possible to have the CPU and GPU compute at the same time. [0213]. Next, pre-processing is performed whereby X is transformed into X' and X' is sent to the GPU instead of X (box 1520). The pre-processing can have many different functions such as putting the data in better form (normalization), extracting intelligent or complex features, generating distortions (enriching the data set), and so forth. In theory, the pre-processing could be done either on the GPU or the CPU. In practice, it is much easier to program on the CPU than the GPU. This means that if the preprocessing is computationally inexpensive, it is much easier to run it on the CPU. In some cases, certain operations may not be easily, efficiently, or at all be implementable on the GPU, and must therefore run on the CPU. In our case, the preprocessing is used to generate artificial data (translation, rotation, etc) from the original data. This process is known to improve generalization.

Ex. C12 – Invalidity of U.S. Patent No. RE49,461 – Buck ’977

U.S. Patent No. RE49,461	Buck ’977
	[0216]. The next two processes have been added for generality, so that the CPU can get feedback from the GPU inside the training loop. For instance, the CPU can collect training statistics in order to adjust the learning parameters, or even the frequency of presentation of certain kind of patterns. For instance, with neural networks, it is sometimes desirable to decrease the learning rate as the learning progress. In another class of algorithm called “boosting”, the frequency of certain patterns, or their learning influence, can be changed as a function of the errors made by the system. Trainable parameters in the preprocessing can also be modified as a function of error signals from the GPU, as we previously mentioned in the preprocessing section. Finally, the termination criteria can be a function of the performance. The first box, therefore, corresponds to the CPU collecting training data (e.g. errors, gradients, weights, etc) from the GPU, and the following box corresponds to the CPU sending adjustments to the parameters governing the learning on the GPU. In this example, the current weights are periodically transferred from the GPU to run test statistics on the training. Because moving data from GPU to CPU is expensive, this is done as seldom as possible. [0220]. FIG. 16 is a flow diagram illustrating an example of the data flow and control structure of a testing session performed on the graphics processing unit in the working example. FIG. 16 illustrates how to use the GPU to compute a given function G of some parameters W and input X. Note that the parameters may have been computed using a GPU, a CPU, or by any other means. The parameters may or may not be the result of training and in this respect, FIG. 16 is independent of FIG. 15. For the handwriting example, however, architecture very similar to the FIG. 15 is used and shares the data structure and shaders (except that the backpropagation shaders are no longer needed since training is done). The first step is to load the shaders and the “trained” parameters onto the GPU (box 1600). Again, as much downloading and pre-computing as possible should be done outside the recognition/usage loop. [0223].

Ex. C12 – Invalidity of U.S. Patent No. RE49,461 – Buck ’977

U.S. Patent No. RE49,461	Buck ’977
	To the extent Plaintiff contends that this claim element is not expressly or inherently disclosed, or is not obvious in view of Buck ’977 and the knowledge of a POSITA, it would have been obvious to combine the teachings of Buck ’977 with any of the prior art references in NVIDIA’s Invalidity Contentions and any supplements thereto and the relevant section of charts for other prior art for the ’461 patent in a manner that would result in the subject matter of this limitation given, at the very least, the person's understanding of the state of the art, the problems addressed and solved in the prior art, and the teachings of Buck ’977. Specific examples of such references are cited and explained in these invalidity contentions, including the cover pleading and charts C1–C15, and Appendix C.
21[c]: shifting control of a data exchange between the user interaction stream and the computational stream to the computational stream in response to starting execution of the computations representing the artificial neural network;	Buck ’977 discloses, expressly and/or inherently, shifting control of a data exchange between the user interaction stream and the computational stream to the computational stream in response to starting execution of the computations representing the artificial neural network. <i>See e.g.:</i> <i>As a non-limiting example, Buck ’977 discloses shifting control of a data exchange via simultaneous calculation of four adjacent elements.</i> A pixel shader is expressed as a series of instructions in DirectX shader assembly language, which is a limited, hardware-independent language defined by DirectX. The code in a shader is executed once for each pixel in a triangle being rendered, and its only effect is to set the values of the 4-vector for that pixel. The limitations of the shader language, and the lack of side effects, mean that the GPU is free to render pixels in any order and using as much parallelism as its hardware can support, resulting in very high performance. The fact that a pixel is a 4-vector affords yet another kind of parallelism; each execution of a pixel shader can calculate four elements (e.g. four adjacent elements of a vector) simultaneously. [0080]. <i>As a non-limiting example, Buck ’977 discloses shifting control of a data exchange via partitioning of tasks into phases, creating an order of operations within a pass, and by determining layout of inputs and outputs.</i>

Ex. C12 – Invalidity of U.S. Patent No. RE49,461 – Buck '977

U.S. Patent No. RE49,461	Buck '977
	Taking into consideration all of the factors noted above, the matrix multiply example can be used to show how to 1) partition the task into passes, 2) order operations within a pass, and 3) determine layout of inputs and outputs. Then, iterate over possible alternatives to partitioning, ordering, and layout to maintain maximum throughput. Memory Layout/Layout for SIMD [0112]. <i>As a non-limiting example, Buck '977 discloses shifting control of a data exchange via the CPU instructing the GPU and blocking until the GPU completes the instructions.</i> FIG. 15 is a flow diagram illustrating an example of the data flow and control structure of a training session performed on the graphics processing unit in the working example. Note that the FIG. 15 is not specific to neural network training: many other learning algorithms (SVM, K-means, LVQ, EM, etc.) can use the same architecture. In this example, the CPU gives instruction to the GPU and blocks until the GPU complete these instructions. It is also possible to have the CPU and GPU compute at the same time. [0213]. <i>As a non-limiting example, Buck '977 discloses shifting control of a data exchange via the CPU instructing the GPU.</i> Next, the CPU instructs the GPU to use the shaders to compute the function $Y=G(X',W)$ (box 1635). A simple diagram illustrates the trained function in the GPU box. The results for each of the patterns of the group are accumulated in the GPU memory (Result Y box in FIG. 16). [0225]. To the extent Plaintiff contends that this claim element is not expressly or inherently disclosed, or is not obvious in view of Buck '977 and the knowledge of a POSITA, it would have been

Ex. C12 – Invalidity of U.S. Patent No. RE49,461 – Buck ’977

U.S. Patent No. RE49,461	Buck ’977
	obvious to combine the teachings of Buck ’977 with any of the prior art references in NVIDIA’s Invalidity Contentions and any supplements thereto and the relevant section of charts for other prior art for the ’461 patent in a manner that would result in the subject matter of this limitation given, at the very least, the person's understanding of the state of the art, the problems addressed and solved in the prior art, and the teachings of Buck ’977. Specific examples of such references are cited and explained in these invalidity contentions, including the cover pleading and charts C1–C15, and Appendix C.
21[d]: shifting control of the data exchange between the user interaction stream and the computational stream to the user interaction stream in response to completion or interruption of the computations representing the artificial neural network;	Buck ’977 discloses, expressly and/or inherently, shifting control of the data exchange between the user interaction stream and the computational stream to the user interaction stream in response to completion or interruption of the computations representing the artificial neural network. <i>See e.g.:</i> <i>As a non-limiting example, Buck ’977 discloses shifting control of a data exchange via simultaneous calculation of four adjacent elements.</i> A pixel shader is expressed as a series of instructions in DirectX shader assembly language, which is a limited, hardware-independent language defined by DirectX. The code in a shader is executed once for each pixel in a triangle being rendered, and its only effect is to set the values of the 4-vector for that pixel. The limitations of the shader language, and the lack of side effects, mean that the GPU is free to render pixels in any order and using as much parallelism as its hardware can support, resulting in very high performance. The fact that a pixel is a 4-vector affords yet another kind of parallelism; each execution of a pixel shader can calculate four elements (e.g. four adjacent elements of a vector) simultaneously. [0080]. <i>As a non-limiting example, Buck ’977 discloses shifting control of a data exchange via partitioning of tasks into phases, creating an order of operations within a pass, and by determining layout of inputs and outputs.</i>

Ex. C12 – Invalidity of U.S. Patent No. RE49,461 – Buck '977

U.S. Patent No. RE49,461	Buck '977
	Taking into consideration all of the factors noted above, the matrix multiply example can be used to show how to 1) partition the task into passes, 2) order operations within a pass, and 3) determine layout of inputs and outputs. Then, iterate over possible alternatives to partitioning, ordering, and layout to maintain maximum throughput. Memory Layout/Layout for SIMD [0112]. <i>As a non-limiting example, Buck '977 discloses shifting control of a data exchange via the CPU instructing the GPU and blocking until the GPU completes the instructions.</i> FIG. 15 is a flow diagram illustrating an example of the data flow and control structure of a training session performed on the graphics processing unit in the working example. Note that the FIG. 15 is not specific to neural network training: many other learning algorithms (SVM, K-means, LVQ, EM, etc.) can use the same architecture. In this example, the CPU gives instruction to the GPU and blocks until the GPU complete these instructions. It is also possible to have the CPU and GPU compute at the same time. [0213]. <i>As a non-limiting example, Buck '977 discloses shifting control of a data exchange via the GPU instructing the CPU.</i> Next, the results Y are transferred back to the CPU 1665). Again, transfer from GPU to CPU is quite expensive, so there is an advantage to grouping, and to keeping the size of Y to a minimum. In the handwriting classification example, Y is just the class of the image X, and is therefore very small. The results Y are obtained (box 1670) and a determination is made as to whether the iterations are completed (box 1675). If not, then the iteration begins again. Otherwise, the results Y are output (box 1680). [0226].

Ex. C12 – Invalidity of U.S. Patent No. RE49,461 – Buck ’977

U.S. Patent No. RE49,461	Buck ’977
	To the extent Plaintiff contends that this claim element is not expressly or inherently disclosed, or is not obvious in view of Buck ’977 and the knowledge of a POSITA, it would have been obvious to combine the teachings of Buck ’977 with any of the prior art references in NVIDIA’s Invalidity Contentions and any supplements thereto and the relevant section of charts for other prior art for the ’461 patent in a manner that would result in the subject matter of this limitation given, at the very least, the person's understanding of the state of the art, the problems addressed and solved in the prior art, and the teachings of Buck ’977. Specific examples of such references are cited and explained in these invalidity contentions, including the cover pleading and charts C1–C15, and Appendix C.
21[e]: queueing a user command received by the user interaction stream during execution of the computations representing the artificial neural network; and	Buck ’977 discloses, expressly and/or inherently, queueing a user command received by the user interaction stream during execution of the computations representing the artificial neural network. <i>See e.g.:</i> <i>As a non-limiting example, Buck ’977 discloses queueing of user commands via reception of operands.</i> Referring to FIGS. 5 and 6 and using the terminology of graphics applications, three operands, t 1 520, 620, t 2 530, 630, and t 3 540, 640, are texture maps. An operation destination d 510, 610 is a rendered image, and the function that operates on the three operands is called a shader. For general purpose applications, this terminology is shed and different model is adopted. [0066]. <i>As a non-limiting example, Buck ’977 discloses queueing of user commands via use of sequential values in computations performed by the kernel.</i> This model differs from convention stream processing in that the inputs are sequential addresses. In other words, texture map indices, rather than input stream data. Typically, operands are fetched using highly structured reads from locations within the workspace, though the kernel is free to perform its own address calculation or use the sequential values directly in the computation. Output to the destination, on the other hand, fits the stream model

Ex. C12 – Invalidity of U.S. Patent No. RE49,461 – Buck ’977

U.S. Patent No. RE49,461	Buck ’977
	very well. The output of the kernel is sequential data values that are written to the workspace. The writes are only sequential as per traditional stream computing. [0067]. <i>As a non-limiting example, Buck ’977 discloses queueing of user commands via sequences of instructions executed by the SIMD processor.</i> Functions operating on the inputs are expressed as sequences of instructions executed by a SIMD processor (such as the programmable fragment processor 310). A complete program is broken into successive passes of these functions over the data. Each pass represents a single data parallel operation in the program. The algorithm may be divided up into further passes due to the limitations present in existing GPU architectures. These include limited instruction length, temporary storage available, and limited number of outputs permitted by the fragment program. As the program is broken into a sequence of passes, the output of the previous pass becomes the input of following pass. [0068]. <i>As a non-limiting example, Buck ’977 discloses queueing of user commands via acceptance of GPU instructions via DirectX assembly language commands.</i> Note that following sections refer to GPU instructions using the notation of DirectX assembly language and may discuss programming conventions in terms of DirectX features. This is simply a matter of convenience and in no way limits the techniques to DirectX implementations. [0070]. To the extent Plaintiff contends that this claim element is not expressly or inherently disclosed, or is not obvious in view of Buck ’977 and the knowledge of a POSITA, it would have been obvious to combine the teachings of Buck ’977 with any of the prior art references in

Ex. C12 – Invalidity of U.S. Patent No. RE49,461 – Buck ’977

U.S. Patent No. RE49,461	Buck ’977
	NVIDIA’s Invalidity Contentions and any supplements thereto and the relevant section of charts for other prior art for the ’461 patent in a manner that would result in the subject matter of this limitation given, at the very least, the person's understanding of the state of the art, the problems addressed and solved in the prior art, and the teachings of Buck ’977. Specific examples of such references are cited and explained in these invalidity contentions, including the cover pleading and charts C1–C15, and Appendix C.
21[f]: executing the user command during execution of the computations representing the artificial neural network at times determined by the computational stream.	Buck ’977 discloses, expressly and/or inherently, executing the user command during execution of the computations representing the artificial neural network at times determined by the computational stream. <i>See e.g.:</i> <i>As a non-limiting example, Buck ’977 discloses executing a user command during execution of computations via accepting input from input devices configured on the computer system the invention is disclosed to operate within.</i> Other input devices (not shown) may include a microphone, joystick, game pad, satellite dish, scanner, radio receiver, or a television or broadcast video receiver, or the like. These and other input devices are often connected to the processing unit 120 through a user input interface 160 that is coupled to the system bus 121, but may be connected by other interface and bus structures, such as, for example, a parallel port, game port or a universal serial bus (USB). A monitor 191 or other type of display device is also connected to the system bus 121 via an interface, such as a video interface 190. In addition to the monitor, computers may also include other peripheral output devices such as speakers 197 and printer 196, which may be connected through an output peripheral interface 195. [0048]. <i>As a non-limiting example, Buck ’977 discloses executing a user command during execution of computations via functions operating on user input.</i> Functions operating on the inputs are expressed as sequences of instructions executed by a SIMD processor (such as the programmable fragment processor 310). A complete program is

Ex. C12 – Invalidity of U.S. Patent No. RE49,461 – Buck '977

U.S. Patent No. RE49,461	Buck '977
	broken into successive passes of these functions over the data. Each pass represents a single data parallel operation in the program. The algorithm may be divided up into further passes due to the limitations present in existing GPU architectures. These include limited instruction length, temporary storage available, and limited number of outputs permitted by the fragment program. As the program is broken into a sequence of passes, the output of the previous pass becomes the input of following pass. [0068]. <i>As a non-limiting example, Buck '977 discloses executing a user command during execution of computations via DirectX assembly commands.</i> Note that following sections refer to GPU instructions using the notation of DirectX assembly language and may discuss programming conventions in terms of DirectX features. This is simply a matter of convenience and in no way limits the techniques to DirectX implementations. [0070]. <i>As a non-limiting example, Buck '977 discloses executing a user command during execution of computations via simultaneous element calculation during rendering.</i> A pixel shader is expressed as a series of instructions in DirectX shader assembly language, which is a limited, hardware-independent language defined by DirectX. The code in a shader is executed once for each pixel in a triangle being rendered, and its only effect is to set the values of the 4-vector for that pixel. The limitations of the shader language, and the lack of side effects, mean that the GPU is free to render pixels in any order and using as much parallelism as its hardware can support, resulting in very high performance. The fact that a pixel is a 4-vector affords yet another kind of parallelism; each execution of a pixel shader can calculate four elements (e.g. four adjacent elements of a vector) simultaneously. [0080].

Ex. C12 – Invalidity of U.S. Patent No. RE49,461 – Buck ’977

U.S. Patent No. RE49,461	Buck ’977
	To the extent Plaintiff contends that this claim element is not expressly or inherently disclosed, or is not obvious in view of Buck ’977 and the knowledge of a POSITA, it would have been obvious to combine the teachings of Buck ’977 with any of the prior art references in NVIDIA’s Invalidity Contentions and any supplements thereto and the relevant section of charts for other prior art for the ’461 patent in a manner that would result in the subject matter of this limitation given, at the very least, the person's understanding of the state of the art, the problems addressed and solved in the prior art, and the teachings of Buck ’977. Specific examples of such references are cited and explained in these invalidity contentions, including the cover pleading and charts C1–C15, and Appendix C.
22. The method of claim 21, wherein the user interaction stream controls the data exchange between the user interaction stream and the computational stream outside of execution of the computations representing the artificial neural network.	Buck ’977 discloses, expressly and/or inherently, the method of claim 21, wherein the user interaction stream controls the data exchange between the user interaction stream and the computational stream outside of execution of the computations representing the artificial neural network. <i>See claim 21.</i> <i>See e.g.:</i> <i>As a non-limiting example, Buck ’977 discloses the user interaction stream controlling the data exchange via the CPU giving instruction the GPU and blocking until the GPU completes the instructions.</i> FIG. 15 is a flow diagram illustrating an example of the data flow and control structure of a training session performed on the graphics processing unit in the working example. Note that the FIG. 15 is not specific to neural network training: many other learning algorithms (SVM, K-means, LVQ, EM, etc.) can use the same architecture. In this example, the CPU gives instruction to the GPU and blocks until the GPU complete these instructions. It is also possible to have the CPU and GPU compute at the same time.

Ex. C12 – Invalidity of U.S. Patent No. RE49,461 – Buck ’977

U.S. Patent No. RE49,461	Buck ’977
	[0213]. To the extent Plaintiff contends that this claim element is not expressly or inherently disclosed, or is not obvious in view of Buck ’977 and the knowledge of a POSITA, it would have been obvious to combine the teachings of Buck ’977 with any of the prior art references in NVIDIA’s Invalidity Contentions and any supplements thereto and the relevant section of charts for other prior art for the ’461 patent in a manner that would result in the subject matter of this limitation given, at the very least, the person's understanding of the state of the art, the problems addressed and solved in the prior art, and the teachings of Buck ’977. Specific examples of such references are cited and explained in these invalidity contentions, including the cover pleading and charts C1–C15, and Appendix C.
23. The method of claim 21, wherein executing the user interaction stream comprises: controlling setting and editing of computational elements of the computations representing the artificial neural network.	Buck ’977 discloses, expressly and/or inherently, the method of claim 21, wherein executing the user interaction stream comprises controlling setting and editing of computational elements of the computations representing the artificial neural network. <i>See claim 21.</i> <i>See e.g.:</i> <i>As a non-limiting example, Buck ’977 discloses controlling setting and editing of computational elements via loading shaders, initial weights and other learning parameters to the GPU.</i> The first step is to load the shaders, initial weights, and other learning parameters on the GPU (box 1500). Data transfer from CPU to GPU is relatively expensive, so the CPU pre-loads as much as possible onto the GPU before entering the training loop. In this example, the learning parameters are the weights of each of the neural network layers, and the threshold of each unit. The learning parameter is a single scalar called the learning rate. The programs P are for the

Ex. C12 – Invalidity of U.S. Patent No. RE49,461 – Buck '977

U.S. Patent No. RE49,461	Buck '977
	different shaders used for the forward and backward propagation, and for the weight updates (box 1505). [0214]. <i>As a non-limiting example, Buck '977 discloses controlling setting and editing of computational elements via including trainable parameters in the pre-processing.</i> Note that it is possible to include trainable parameters in the pre-processing, even if they are part of the global optimization and trained in conjunction with the learning parameters that reside on the GPU. The difficulty with this is that information (e.g. error gradients, negative feedback, etc) must be communicated from the GPU back to the CPU to update those parameters that reside on the CPU, possibly after each pattern, so that the next pattern can profit from the updated pre-processing. Today, dataflow from GPU to CPU is not efficient, since graphic cards are typically designed to send data to the screen, but not back to the CPU. As a result, with the current architecture, it is preferable to keep all the learning parameters on the GPU. This is the case in our example. This limitation may go away in the future. The current algorithm can be modified accordingly by allowing error signals to flow back to the CPU and update the learning parameters on the CPU. The group size can be modified accordingly. In the extreme, the group size is 1, which means that the training parameters on the CPU are updated after each presentation to the GPU, as soon as the error signal comes back from the GPU. [0217]. <i>As a non-limiting example, Buck '977 discloses controlling setting and editing of computational elements via shaders and learning parameters.</i> For training the 2 layer neural networks, the forward and backpropagation correspond to about twenty different shaders (some of which are called multiple times). The number and complexity of shaders can of course vary depending on the algorithm used. The shaders are called for each pattern in a group (500 times in our case). For stochastic gradient descent, the

Ex. C12 – Invalidity of U.S. Patent No. RE49,461 – Buck ’977

U.S. Patent No. RE49,461	Buck ’977
	learning parameters are updated after processing each pattern in the group. For batch gradient descent, the learning parameter gradients are accumulated over several patterns before the learning parameters are updated. Whether stochastic or batch gradient descent is used depends heavily on the application and the learning algorithm (and for some learning algorithms, such as SVM, the questions does not arise). For handwriting recognition and neural networks, stochastic gradient descent is preferable, and this is what is implemented in this example. [0219]. FIG. 16 is a flow diagram illustrating an example of the data flow and control structure of a testing session performed on the graphics processing unit in the working example. FIG. 16 illustrates how to use the GPU to compute a given function G of some parameters W and input X. Note that the parameters may have been computed using a GPU, a CPU, or by any other means. The parameters may or may not be the result of training and in this respect, FIG. 16 is independent of FIG. 15. For the handwriting example, however, architecture very similar to the FIG. 15 is used and shares the data structure and shaders (except that the backpropagation shaders are no longer needed since training is done). The first step is to load the shaders and the “trained” parameters onto the GPU (box 1600). Again, as much downloading and pre-computing as possible should be done outside the recognition/usage loop. [0223]. To the extent Plaintiff contends that this claim element is not expressly or inherently disclosed, or is not obvious in view of Buck ’977 and the knowledge of a POSITA, it would have been obvious to combine the teachings of Buck ’977 with any of the prior art references in NVIDIA’s Invalidity Contentions and any supplements thereto and the relevant section of charts for other prior art for the ’461 patent in a manner that would result in the subject matter of this limitation given, at the very least, the person's understanding of the state of the art, the problems addressed and solved in the prior art, and the teachings of Buck ’977. Specific examples of such references are cited and explained in these invalidity contentions, including the cover pleading and charts C1–C15, and Appendix C.

Ex. C12 – Invalidity of U.S. Patent No. RE49,461 – Buck ’977

U.S. Patent No. RE49,461	Buck ’977
24. The method of claim 21, wherein executing the user interaction stream comprises: controlling setting and editing of parameters of the computations representing the artificial neural network.	Buck ’977 discloses, expressly and/or inherently, the method of claim 21, wherein executing the user interaction stream comprises controlling setting and editing of parameters of the computations representing the artificial neural network. <i>See claims 21, 23.</i> <i>See e.g.:</i> <i>As a non-limiting example, Buck ’977 discloses controlling setting and editing of parameters of computations via loading of shaders, initial weights, and other learning parameters.</i> The first step is to load the shaders, initial weights, and other learning parameters on the GPU (box 1500). Data transfer from CPU to GPU is relatively expensive, so the CPU pre-loads as much as possible onto the GPU before entering the training loop. In this example, the learning parameters are the weights of each of the neural network layers, and the threshold of each unit. The learning parameter is a single scalar called the learning rate. The programs P are for the different shaders used for the forward and backward propagation, and for the weight updates (box 1505). [0214]. Note that it is possible to include trainable parameters in the pre-processing, even if they are part of the global optimization and trained in conjunction with the learning parameters that reside on the GPU. The difficulty with this is that information (e.g. error gradients, negative feedback, etc) must be communicated from the GPU back to the CPU to update those parameters that reside on the CPU, possibly after each pattern, so that the next pattern can profit from the updated pre-processing. Today, dataflow from GPU to CPU is not efficient, since graphic cards are typically designed to send data to the screen, but not back to the CPU. As a result, with the current architecture, it is preferable to keep all the learning parameters on the GPU. This is the case in our example. This limitation may go away in the future. The current algorithm can be modified accordingly by allowing error signals to flow back to the

Ex. C12 – Invalidity of U.S. Patent No. RE49,461 – Buck ’977

U.S. Patent No. RE49,461	Buck ’977
	CPU and update the learning parameters on the CPU. The group size can be modified accordingly. In the extreme, the group size is 1, which means that the training parameters on the CPU are updated after each presentation to the GPU, as soon as the error signal comes back from the GPU. [0217]. For training the 2 layer neural networks, the forward and backpropagation correspond to about twenty different shaders (some of which are called multiple times). The number and complexity of shaders can of course vary depending on the algorithm used. The shaders are called for each pattern in a group (500 times in our case). For stochastic gradient descent, the learning parameters are updated after processing each pattern in the group. For batch gradient descent, the learning parameter gradients are accumulated over several patterns before the learning parameters are updated. Whether stochastic or batch gradient descent is used depends heavily on the application and the learning algorithm (and for some learning algorithms, such as SVM, the questions does not arise). For handwriting recognition and neural networks, stochastic gradient descent is preferable, and this is what is implemented in this example. [0219]. FIG. 16 is a flow diagram illustrating an example of the data flow and control structure of a testing session performed on the graphics processing unit in the working example. FIG. 16 illustrates how to use the GPU to compute a given function G of some parameters W and input X. Note that the parameters may have been computed using a GPU, a CPU, or by any other means. The parameters may or may not be the result of training and in this respect, FIG. 16 is independent of FIG. 15. For the handwriting example, however, architecture very similar to the FIG. 15 is used and shares the data structure and shaders (except that the backpropagation shaders are no longer needed since training is done). The first step is to load the shaders and the “trained” parameters onto the GPU (box 1600). Again, as much downloading and pre-computing as possible should be done outside the recognition/usage loop. [0223].

Ex. C12 – Invalidity of U.S. Patent No. RE49,461 – Buck ’977

U.S. Patent No. RE49,461	Buck ’977
	To the extent Plaintiff contends that this claim element is not expressly or inherently disclosed, or is not obvious in view of Buck ’977 and the knowledge of a POSITA, it would have been obvious to combine the teachings of Buck ’977 with any of the prior art references in NVIDIA’s Invalidity Contentions and any supplements thereto and the relevant section of charts for other prior art for the ’461 patent in a manner that would result in the subject matter of this limitation given, at the very least, the person's understanding of the state of the art, the problems addressed and solved in the prior art, and the teachings of Buck ’977. Specific examples of such references are cited and explained in these invalidity contentions, including the cover pleading and charts C1–C15, and Appendix C.
25. The method of claim 21, wherein executing the user interaction stream comprises: controlling setting and editing of parameters of the inputs to the artificial neural network.	Buck ’977 discloses, expressly and/or inherently, the method of claim 21, wherein executing the user interaction stream comprises controlling setting and editing of parameters of the inputs to the artificial neural network. <i>See claim 21.</i> <i>See e.g.:</i> <i>As a non-limiting example, Buck ’977 discloses controlling setting and editing of parameters of inputs via loading of shaders, initial weights, and other learning parameters.</i> The first step is to load the shaders, initial weights, and other learning parameters on the GPU (box 1500). Data transfer from CPU to GPU is relatively expensive, so the CPU pre-loads as much as possible onto the GPU before entering the training loop. In this example, the learning parameters are the weights of each of the neural network layers, and the threshold of each unit. The learning parameter is a single scalar called the learning rate. The programs P are for the different shaders used for the forward and backward propagation, and for the weight updates (box 1505). [0214].

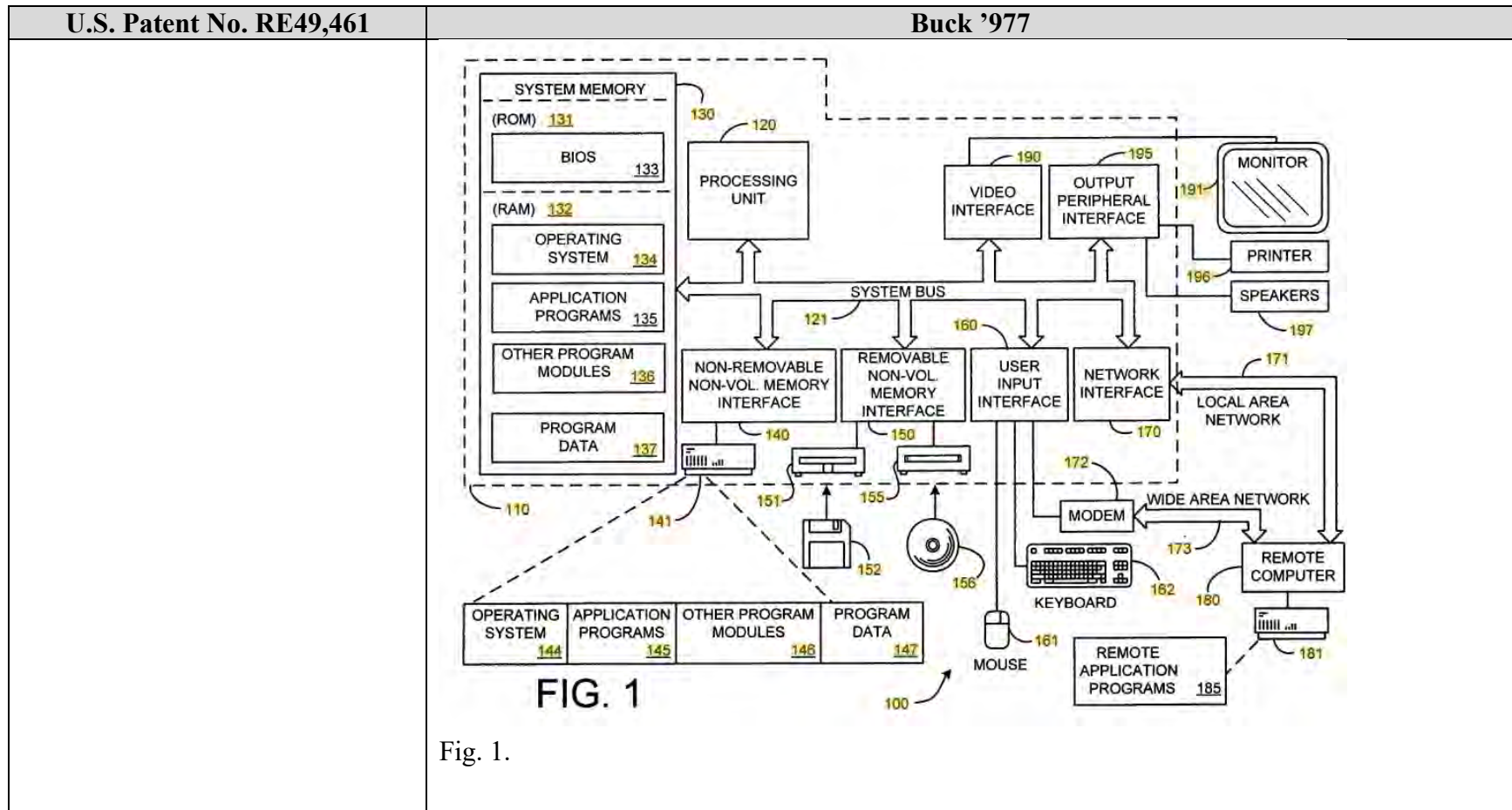
Ex. C12 – Invalidity of U.S. Patent No. RE49,461 – Buck '977

U.S. Patent No. RE49,461	Buck '977
	Note that it is possible to include trainable parameters in the pre-processing, even if they are part of the global optimization and trained in conjunction with the learning parameters that reside on the GPU. The difficulty with this is that information (e.g. error gradients, negative feedback, etc) must be communicated from the GPU back to the CPU to update those parameters that reside on the CPU, possibly after each pattern, so that the next pattern can profit from the updated pre-processing. Today, dataflow from GPU to CPU is not efficient, since graphic cards are typically designed to send data to the screen, but not back to the CPU. As a result, with the current architecture, it is preferable to keep all the learning parameters on the GPU. This is the case in our example. This limitation may go away in the future. The current algorithm can be modified accordingly by allowing error signals to flow back to the CPU and update the learning parameters on the CPU. The group size can be modified accordingly. In the extreme, the group size is 1, which means that the training parameters on the CPU are updated after each presentation to the GPU, as soon as the error signal comes back from the GPU. [0217]. For training the 2 layer neural networks, the forward and backpropagation correspond to about twenty different shaders (some of which are called multiple times). The number and complexity of shaders can of course vary depending on the algorithm used. The shaders are called for each pattern in a group (500 times in our case). For stochastic gradient descent, the learning parameters are updated after processing each pattern in the group. For batch gradient descent, the learning parameter gradients are accumulated over several patterns before the learning parameters are updated. Whether stochastic or batch gradient descent is used depends heavily on the application and the learning algorithm (and for some learning algorithms, such as SVM, the questions does not arise). For handwriting recognition and neural networks, stochastic gradient descent is preferable, and this is what is implemented in this example. [0219]. FIG. 16 is a flow diagram illustrating an example of the data flow and control structure of a testing session performed on the graphics processing unit in the working example. FIG. 16 illustrates how to use the GPU to compute a given function G of some parameters W and input

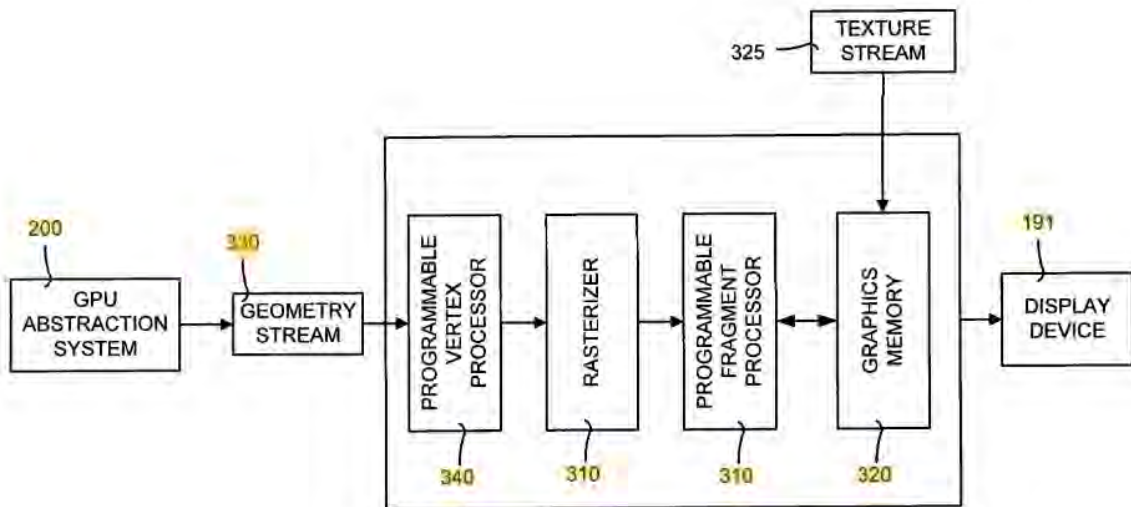
Ex. C12 – Invalidity of U.S. Patent No. RE49,461 – Buck ’977

U.S. Patent No. RE49,461	Buck ’977
	X. Note that the parameters may have been computed using a GPU, a CPU, or by any other means. The parameters may or may not be the result of training and in this respect, FIG. 16 is independent of FIG. 15. For the handwriting example, however, architecture very similar to the FIG. 15 is used and shares the data structure and shaders (except that the backpropagation shaders are no longer needed since training is done). The first step is to load the shaders and the “trained” parameters onto the GPU (box 1600). Again, as much downloading and pre-computing as possible should be done outside the recognition/usage loop. [0223]. To the extent Plaintiff contends that this claim element is not expressly or inherently disclosed, or is not obvious in view of Buck ’977 and the knowledge of a POSITA, it would have been obvious to combine the teachings of Buck ’977 with any of the prior art references in NVIDIA’s Invalidity Contentions and any supplements thereto and the relevant section of charts for other prior art for the ’461 patent in a manner that would result in the subject matter of this limitation given, at the very least, the person's understanding of the state of the art, the problems addressed and solved in the prior art, and the teachings of Buck ’977. Specific examples of such references are cited and explained in these invalidity contentions, including the cover pleading and charts C1–C15, and Appendix C.
26. The method of claim 21, wherein executing the user interaction stream comprises: specifying an output to be saved to disk and/or displayed on a screen.	Buck ’977 discloses, expressly and/or inherently, the method of claim 21, wherein executing the user interaction stream comprises specifying an output to be saved to disk and/or displayed on a screen. <i>See claim 21.</i> <i>See e.g.:</i> <i>As a non-limiting example, Buck ’977 discloses specifying output to be save to a disk and or displayed on a screen via the removable non-vol memory interface and monitor</i>

Ex. C12 – Invalidity of U.S. Patent No. RE49,461 – Buck '977



Ex. C12 – Invalidity of U.S. Patent No. RE49,461 – Buck '977

U.S. Patent No. RE49,461	Buck '977
	 FIG. 3 Fig. 3. FIG. 1 illustrates an example of a suitable computing system environment in which the GPU optimization system and method may be implemented. [0015]. FIG. 3 is a block diagram illustrating the interaction between the GPU and GPU optimization system shown in FIG. 2. [0017]. FIG. 1 illustrates an example of a suitable computing system environment in which the GPU optimization system and method may be implemented. The computing system environment

Ex. C12 – Invalidity of U.S. Patent No. RE49,461 – Buck ’977

U.S. Patent No. RE49,461	Buck ’977
	100 is only one example of a suitable computing environment and is not intended to suggest any limitation as to the scope of use or functionality of the invention. Neither should the computing environment 100 be interpreted as having any dependency or requirement relating to any one or combination of components illustrated in the exemplary operating environment 100. [0037]. The computer 110 may also include other removable/non-removable, volatile/nonvolatile computer storage media. By way of example only, FIG. 1 illustrates a hard disk drive 141 that reads from or writes to non-removable, nonvolatile magnetic media, a magnetic disk drive 151 that reads from or writes to a removable, nonvolatile magnetic disk 152, and an optical disk drive 155 that reads from or writes to a removable, nonvolatile optical disk 156 such as a CD ROM or other optical media. [0045]. Other removable/non-removable, volatile/nonvolatile computer storage media that can be used in the exemplary operating environment include, but are not limited to, magnetic tape cassettes, flash memory cards, digital versatile disks, digital video tape, solid state RAM, solid state ROM, and the like. The hard disk drive 141 is typically connected to the system bus 121 through a non-removable memory interface such as interface 140, and magnetic disk drive 151 and optical disk drive 155 are typically connected to the system bus 121 by a removable memory interface, such as interface 150. [0046]. Other input devices (not shown) may include a microphone, joystick, game pad, satellite dish, scanner, radio receiver, or a television or broadcast video receiver, or the like. These and other input devices are often connected to the processing unit 120 through a user input interface 160 that is coupled to the system bus 121, but may be connected by other interface and bus structures, such as, for example, a parallel port, game port or a universal serial bus (USB). A monitor 191 or other type of display device is also connected to the system bus 121 via an

Ex. C12 – Invalidity of U.S. Patent No. RE49,461 – Buck ’977

U.S. Patent No. RE49,461	Buck ’977
	interface, such as a video interface 190. In addition to the monitor, computers may also include other peripheral output devices such as speakers 197 and printer 196, which may be connected through an output peripheral interface 195. [0048]. To the extent Plaintiff contends that this claim element is not expressly or inherently disclosed, or is not obvious in view of Buck ’977 and the knowledge of a POSITA, it would have been obvious to combine the teachings of Buck ’977 with any of the prior art references in NVIDIA’s Invalidity Contentions and any supplements thereto and the relevant section of charts for other prior art for the ’461 patent in a manner that would result in the subject matter of this limitation given, at the very least, the person's understanding of the state of the art, the problems addressed and solved in the prior art, and the teachings of Buck ’977. Specific examples of such references are cited and explained in these invalidity contentions, including the cover pleading and charts C1–C15, and Appendix C.
27. The method of claim 21, wherein executing the user interaction stream comprises: parsing elements to be used in the computations representing the artificial neural network.	Buck ’977 discloses, expressly and/or inherently, the method of claim 21, wherein executing the user interaction stream comprises parsing elements to be used in the computations representing the artificial neural network. <i>See claim 21.</i> <i>See e.g.:</i> <i>As a non-limiting example, Buck ’977 discloses parsing elements to be used in computations via loading shaders, initial weights and other learning parameters onto the GPU.</i> The first step is to load the shaders, initial weights, and other learning parameters on the GPU (box 1500). Data transfer from CPU to GPU is relatively expensive, so the CPU pre-loads as much as possible onto the GPU before entering the training loop. In this example, the learning parameters are the weights of each of the neural network layers, and the threshold of each unit.

Ex. C12 – Invalidity of U.S. Patent No. RE49,461 – Buck ’977

U.S. Patent No. RE49,461	Buck ’977
	The learning parameter is a single scalar called the learning rate. The programs P are for the different shaders used for the forward and backward propagation, and for the weight updates (box 1505). [0214]. Next, pre-processing is performed whereby X is transformed into X' and X' is sent to the GPU instead of X (box 1520). The pre-processing can have many different functions such as putting the data in better form (normalization), extracting intelligent or complex features, generating distortions (enriching the data set), and so forth. In theory, the pre-processing could be done either on the GPU or the CPU. In practice, it is much easier to program on the CPU than the GPU. This means that if the preprocessing is computationally inexpensive, it is much easier to run it on the CPU. In some cases, certain operations may not be easily, efficiently, or at all be implementable on the GPU, and must therefore run on the CPU. In our case, the preprocessing is used to generate artificial data (translation, rotation, etc) from the original data. This process is known to improve generalization. [0216]. For training the 2 layer neural networks, the forward and backpropagation correspond to about twenty different shaders (some of which are called multiple times). The number and complexity of shaders can of course vary depending on the algorithm used. The shaders are called for each pattern in a group (500 times in our case). For stochastic gradient descent, the learning parameters are updated after processing each pattern in the group. For batch gradient descent, the learning parameter gradients are accumulated over several patterns before the learning parameters are updated. Whether stochastic or batch gradient descent is used depends heavily on the application and the learning algorithm (and for some learning algorithms, such as SVM, the questions does not arise). For handwriting recognition and neural networks, stochastic gradient descent is preferable, and this is what is implemented in this example. [0219].

Ex. C12 – Invalidity of U.S. Patent No. RE49,461 – Buck ’977

U.S. Patent No. RE49,461	Buck ’977
	The next two processes have been added for generality, so that the CPU can get feedback from the GPU inside the training loop. For instance, the CPU can collect training statistics in order to adjust the learning parameters, or even the frequency of presentation of certain kind of patterns. For instance, with neural networks, it is sometimes desirable to decrease the learning rate as the learning progress. In another class of algorithm called “boosting”, the frequency of certain patterns, or their learning influence, can be changed as a function of the errors made by the system. Trainable parameters in the preprocessing can also be modified as a function of error signals from the GPU, as we previously mentioned in the preprocessing section. Finally, the termination criteria can be a function of the performance. The first box, therefore, corresponds to the CPU collecting training data (e.g. errors, gradients, weights, etc) from the GPU, and the following box corresponds to the CPU sending adjustments to the parameters governing the learning on the GPU. In this example, the current weights are periodically transferred from the GPU to run test statistics on the training. Because moving data from GPU to CPU is expensive, this is done as seldom as possible. [0220]. To the extent Plaintiff contends that this claim element is not expressly or inherently disclosed, or is not obvious in view of Buck ’977 and the knowledge of a POSITA, it would have been obvious to combine the teachings of Buck ’977 with any of the prior art references in NVIDIA’s Invalidity Contentions and any supplements thereto and the relevant section of charts for other prior art for the ’461 patent in a manner that would result in the subject matter of this limitation given, at the very least, the person's understanding of the state of the art, the problems addressed and solved in the prior art, and the teachings of Buck ’977. Specific examples of such references are cited and explained in these invalidity contentions, including the cover pleading and charts C1–C15, and Appendix C.
28. The method of claim 27, wherein the processing unit comprises a graphics processing unit (GPU) and executing the user interaction stream further	Buck ’977 discloses, expressly and/or inherently, the method of claim 27, wherein the processing unit comprises a graphics processing unit (GPU) and executing the user interaction stream further comprises converting the elements into GPU programs. <i>See claim 27.</i>

Ex. C12 – Invalidity of U.S. Patent No. RE49,461 – Buck '977

U.S. Patent No. RE49,461	Buck '977
comprises: converting the elements into GPU programs.	<i>See e.g.:</i> <i>As a non-limiting example, Buck '977 discloses converting elements into GPU programs via executing code fragments to exploit the SIMD processing of the fragment shader.</i> As shown in FIGS. 10A, in actual execution, the code fragment on the of the mad approach exploits the SIMD processing of the fragment shader and the distribution of 4-vector components of each pixel along columns of the matrix is therefore the preferred layout in memory. [0121]. While either memory layout can be used with the associated code fragment shown below it, it has been seen that for the matrix-vector inner product, the layout shown in FIG. 10A leads to a more efficient algorithm. However, the layout shown in FIG. 10B can be used to implement a transposed matrix-vector multiply. This is necessary for the neural net training algorithm, since in-memory transposition of matrices is too expensive. In other words, it involves a complete copy of the largest amount of array data. [0122]. FIG. 15 is a flow diagram illustrating an example of the data flow and control structure of a training session performed on the graphics processing unit in the working example. Note that the FIG. 15 is not specific to neural network training: many other learning algorithms (SVM, K-means, LVQ, EM, etc.) can use the same architecture. In this example, the CPU gives instruction to the GPU and blocks until the GPU complete these instructions. It is also possible to have the CPU and GPU compute at the same time. [0213].

Ex. C12 – Invalidity of U.S. Patent No. RE49,461 – Buck '977

U.S. Patent No. RE49,461	Buck '977
	The first step is to load the shaders, initial weights, and other learning parameters on the GPU (box 1500). Data transfer from CPU to GPU is relatively expensive, so the CPU pre-loads as much as possible onto the GPU before entering the training loop. In this example, the learning parameters are the weights of each of the neural network layers, and the threshold of each unit. The learning parameter is a single scalar called the learning rate. The programs P are for the different shaders used for the forward and backward propagation, and for the weight updates (box 1505). [0214]. Once the training data has been loaded on the GPU, the CPU instructs the GPU to run the various shaders that make up the learning algorithm (box 1540). A typical learning algorithm is represented in a box inside the GPU. The learning algorithm computes a function $G(X', W)$ as a function of the preprocessed input X' and the trainable parameters W. The goal is to make this output as close as possible as the target value T. An error 1555 between $G(X', W)$ and T is computed, and error signals (e.g. gradient with respect to W) are sent back to the learning machine. The weights are then updated in order to reduce the error between $G(X', W)$ and T. [0218]. Next, the CPU instructs the GPU to use the shaders to compute the function $Y=G(X', W)$ (box 1635). A simple diagram illustrates the trained function in the GPU box. The results for each of the patterns of the group are accumulated in the GPU memory (Result Y box in FIG. 16). [0225]. To the extent Plaintiff contends that this claim element is not expressly or inherently disclosed, or is not obvious in view of Buck '977 and the knowledge of a POSITA, it would have been obvious to combine the teachings of Buck '977 with any of the prior art references in NVIDIA's Invalidity Contentions and any supplements thereto and the relevant section of charts for other prior art for the '461 patent in a manner that would result in the subject matter of this limitation given, at the very least, the person's understanding of the state of the art, the problems addressed and solved in the prior art, and the teachings of Buck '977. Specific

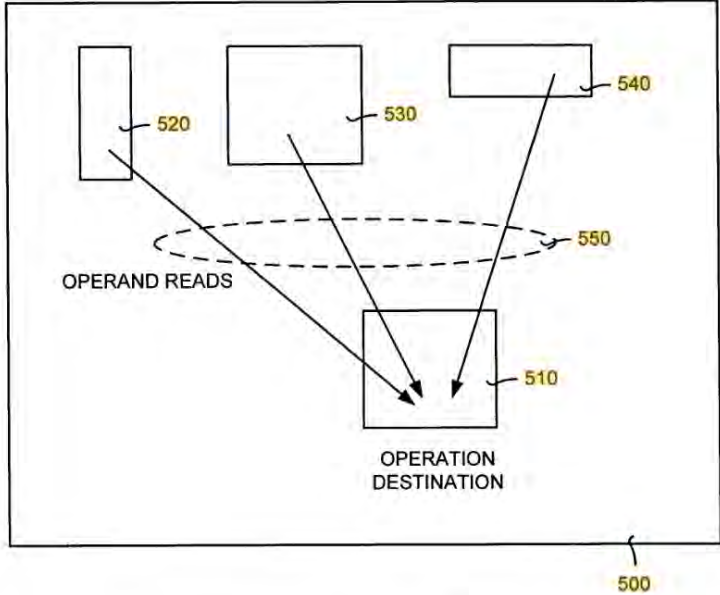
Ex. C12 – Invalidity of U.S. Patent No. RE49,461 – Buck ’977

U.S. Patent No. RE49,461	Buck ’977
	examples of such references are cited and explained in these invalidity contentions, including the cover pleading and charts C1–C15, and Appendix C.
29. The method of claim 28, wherein executing the user interaction stream comprises: compiling the GPU programs.	Buck ’977 discloses, expressly and/or inherently, the method of claim 28, wherein executing the user interaction stream comprises: compiling the GPU programs. <i>See claim 28.</i> <i>See e.g.:</i> <i>As a non-limiting example, Buck ’977 discloses compiling GPU programs via loading of shaders, initial weights, and other learning parameters onto the GPU.</i> FIG. 15 is a flow diagram illustrating an example of the data flow and control structure of a training session performed on the graphics processing unit in the working example. Note that the FIG. 15 is not specific to neural network training: many other learning algorithms (SVM, K-means, LVQ, EM, etc.) can use the same architecture. In this example, the CPU gives instruction to the GPU and blocks until the GPU complete these instructions. It is also possible to have the CPU and GPU compute at the same time. [0213]. The first step is to load the shaders, initial weights, and other learning parameters on the GPU (box 1500). Data transfer from CPU to GPU is relatively expensive, so the CPU pre-loads as much as possible onto the GPU before entering the training loop. In this example, the learning parameters are the weights of each of the neural network layers, and the threshold of each unit. The learning parameter is a single scalar called the learning rate. The programs P are for the different shaders used for the forward and backward propagation, and for the weight updates (box 1505). [0214].

Ex. C12 – Invalidity of U.S. Patent No. RE49,461 – Buck ’977

U.S. Patent No. RE49,461	Buck ’977
	Once the training data has been loaded on the GPU, the CPU instructs the GPU to run the various shaders that make up the learning algorithm (box 1540). A typical learning algorithm is represented in a box inside the GPU. The learning algorithm computes a function $G(X',W)$ as a function of the preprocessed input X' and the trainable parameters W. The goal is to make this output as close as possible as the target value T. An error 1555 between $G(X',W)$ and T is computed, and error signals (e.g. gradient with respect to W) are sent back to the learning machine. The weights are then updated in order to reduce the error between $G(X',W)$ and T. [0218]. Next, the CPU instructs the GPU to use the shaders to compute the function $Y=G(X',W)$ (box 1635). A simple diagram illustrates the trained function in the GPU box. The results for each of the patterns of the group are accumulated in the GPU memory (Result Y box in FIG. 16). [0225]. To the extent Plaintiff contends that this claim element is not expressly or inherently disclosed, or is not obvious in view of Buck ’977 and the knowledge of a POSITA, it would have been obvious to combine the teachings of Buck ’977 with any of the prior art references in NVIDIA’s Invalidity Contentions and any supplements thereto and the relevant section of charts for other prior art for the ’461 patent in a manner that would result in the subject matter of this limitation given, at the very least, the person's understanding of the state of the art, the problems addressed and solved in the prior art, and the teachings of Buck ’977. Specific examples of such references are cited and explained in these invalidity contentions, including the cover pleading and charts C1–C15, and Appendix C.
30. The method of claim 29, wherein executing the user interaction stream comprises: transferring the GPU programs to the second memory partition.	Buck ’977 discloses, expressly and/or inherently, the method of claim 29, wherein executing the user interaction stream comprises transferring the GPU programs to the second memory partition. <i>See claim 29.</i>

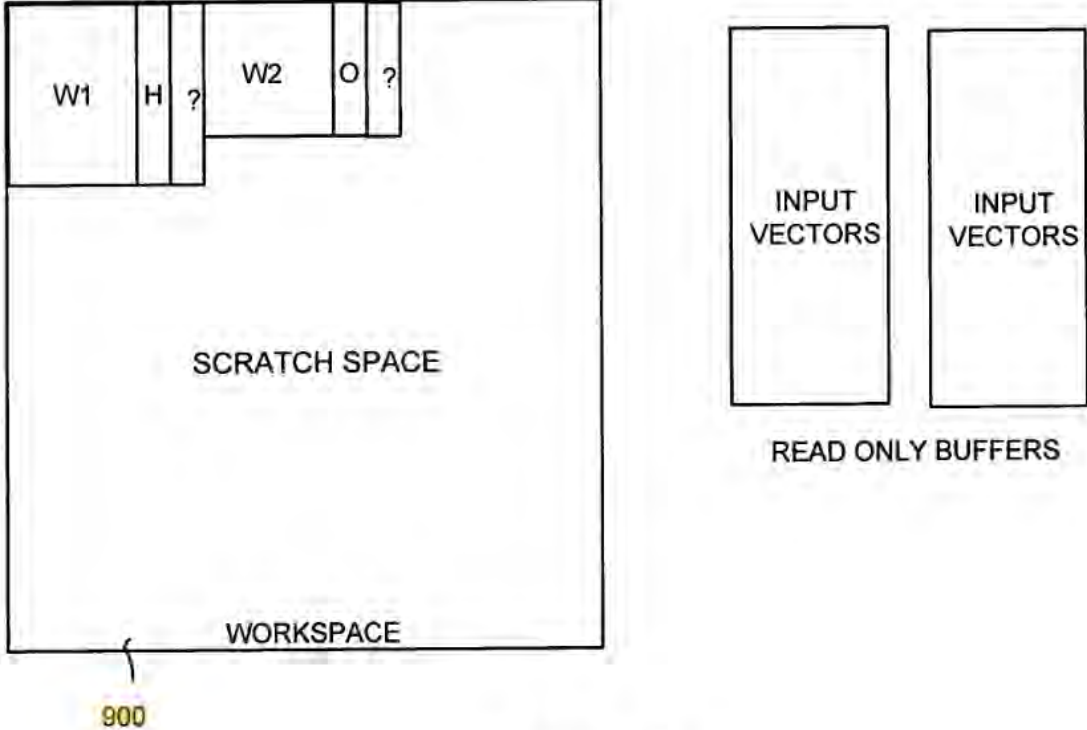
Ex. C12 – Invalidity of U.S. Patent No. RE49,461 – Buck '977

U.S. Patent No. RE49,461	Buck '977
	<i>See e.g.:</i> <i>As a non-limiting example, Buck '977 discloses transferring GPU programs to the second memory partition via sending operand reads to the operation destination.</i>  The diagram, labeled FIG. 5, is enclosed in a rectangular frame 500. It depicts a data flow process. At the top, there are three rectangular boxes labeled 520, 530, and 540. Below these, a dashed oval labeled 550 is positioned. At the bottom center, there is a rectangular box labeled 510. Solid arrows point from each of the top boxes (520, 530, and 540) down to the bottom box (510). A dashed arrow points from the dashed oval 550 to the bottom box 510. The text 'OPERAND READS' is placed to the left of the dashed oval 550, and 'OPERATION DESTINATION' is placed below the bottom box 510. FIG. 5 Fig. 5.

Ex. C12 – Invalidity of U.S. Patent No. RE49,461 – Buck '977

U.S. Patent No. RE49,461	Buck '977
	The diagram, labeled FIG. 6, illustrates a system 600. It features three rectangular blocks at the top: block 620 on the left, block 630 in the center, and block 640 on the right. Below these is a dashed oval labeled 650, with the text 'OPERAND READS' positioned to its left. At the bottom center is a rectangular block labeled 610, with the text 'OPERATION DESTINATION' below it. Arrows point from each of the top blocks (620, 630, and 640) down to block 610. The entire assembly is enclosed in a large rectangle labeled 600 at the bottom right corner. FIG. 6 Fig. 6.

Ex. C12 – Invalidity of U.S. Patent No. RE49,461 – Buck '977

U.S. Patent No. RE49,461	Buck '977
	 The diagram illustrates a workspace layout. A large rectangle is labeled 'WORKSPACE' at the bottom. Inside the top-left corner of the workspace is a smaller rectangle labeled 'SCRATCH SPACE'. Above the scratch space, there are several small rectangular blocks labeled 'W1', 'H', '?', 'W2', 'O', and '?'. To the right of the workspace, there are two vertical rectangles labeled 'INPUT VECTORS'. Below these input vectors, the text 'READ ONLY BUFFERS' is written. A bracket at the bottom of the workspace points to a label '900'. FIG. 9 Fig. 9. The key to optimization of GPU performance is to make maximum use of SIMD parallelism in the GPU pixel shaders. This requires that the operands (the inputs to the overall process) and the intermediate results (the outputs from and inputs to intermediate computing passes) be stored in memory in an arrangement that supports SIMD execution. The 4-component parallelism of GPUs is an additional axis of arrangement that must be considered when

Ex. C12 – Invalidity of U.S. Patent No. RE49,461 – Buck '977

U.S. Patent No. RE49,461	Buck '977
	optimizing for SIMD execution. In addition, this GPU optimization is subject to constraints imposed by current GPU architectures. [0113]. Program access to graphics memory differs in several ways from memory access from conventional CPUs. FIG. 9 is a block diagram illustrating the segmentation of the graphics memory. As shown in FIG. 9, the graphics memory 900 is segmented (such as being accessed in 2K×2K chunks), accessed with two dimensional addresses, and each reference is to a floating point 4-vector. For general purpose programs, the read/write memory is constrained to a 2K×2K workspace. Additional read only buffers (segments) outside this workspace may be used as input data. [0114]. In FIG. 9, part of the scratch space is temporary buffer and part is a permanent buffer. By placing everything into a single scratch space, the space where the optimization GPU method is reading from and writing to never has to change. However, it can vary within the scratch space. [0115]. As shown in FIGS. 10A and 10B, the 4-way parallelism of each memory location can either be distributed across columns (left) or rows (right) of the matrix. In FIG. 10A, M1 is a column vector having 4 elements, and R0 is a column vector having 4 elements. In FIG. 10B, M1 and R0 are row vectors having 4 elements. [0120]. FIG. 15 is a flow diagram illustrating an example of the data flow and control structure of a training session performed on the graphics processing unit in the working example. Note that the FIG. 15 is not specific to neural network training: many other learning algorithms (SVM, K-means, LVQ, EM, etc.) can use the same architecture. In this example, the CPU gives

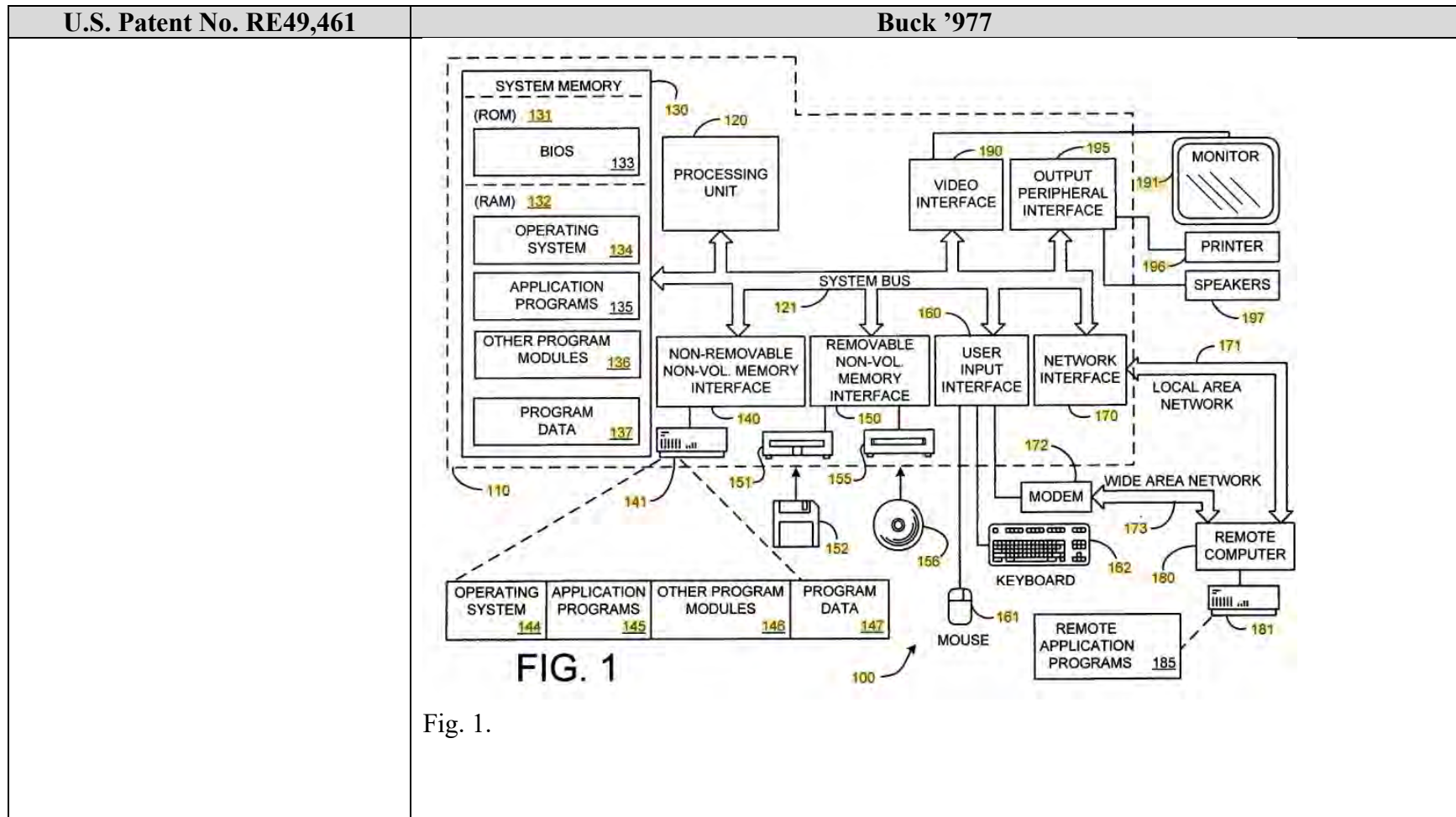
Ex. C12 – Invalidity of U.S. Patent No. RE49,461 – Buck ’977

U.S. Patent No. RE49,461	Buck ’977
	instruction to the GPU and blocks until the GPU complete these instructions. It is also possible to have the CPU and GPU compute at the same time. [0213]. The first step is to load the shaders, initial weights, and other learning parameters on the GPU (box 1500). Data transfer from CPU to GPU is relatively expensive, so the CPU pre-loads as much as possible onto the GPU before entering the training loop. In this example, the learning parameters are the weights of each of the neural network layers, and the threshold of each unit. The learning parameter is a single scalar called the learning rate. The programs P are for the different shaders used for the forward and backward propagation, and for the weight updates (box 1505). [0214]. Once the training data has been loaded on the GPU, the CPU instructs the GPU to run the various shaders that make up the learning algorithm (box 1540). A typical learning algorithm is represented in a box inside the GPU. The learning algorithm computes a function $G(X', W)$ as a function of the preprocessed input X' and the trainable parameters W. The goal is to make this output as close as possible as the target value T. An error 1555 between $G(X', W)$ and T is computed, and error signals (e.g. gradient with respect to W) are sent back to the learning machine. The weights are then updated in order to reduce the error between $G(X', W)$ and T. [0218]. To the extent Plaintiff contends that this claim element is not expressly or inherently disclosed, or is not obvious in view of Buck ’977 and the knowledge of a POSITA, it would have been obvious to combine the teachings of Buck ’977 with any of the prior art references in NVIDIA’s Invalidity Contentions and any supplements thereto and the relevant section of charts for other prior art for the ’461 patent in a manner that would result in the subject matter of this limitation given, at the very least, the person's understanding of the state of the art, the problems addressed and solved in the prior art, and the teachings of Buck ’977. Specific

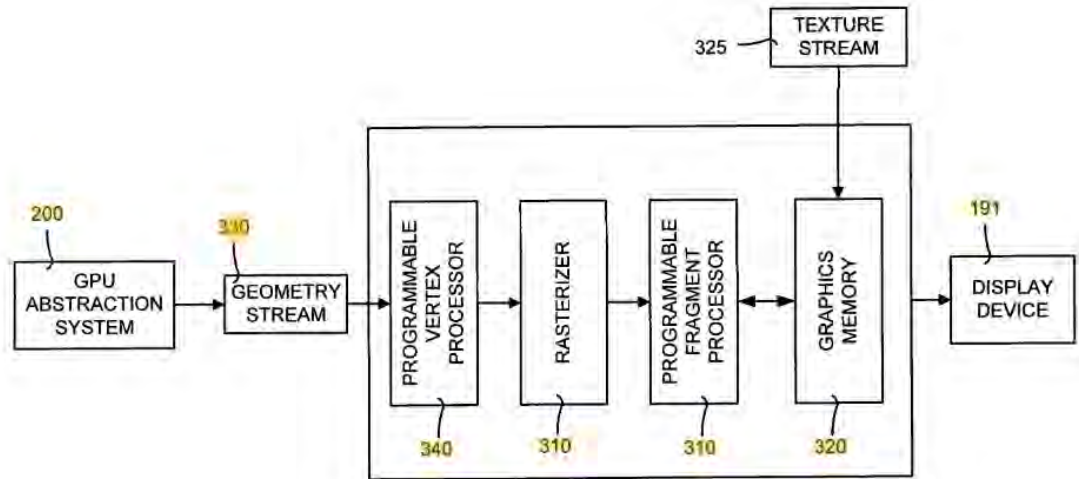
Ex. C12 – Invalidity of U.S. Patent No. RE49,461 – Buck ’977

U.S. Patent No. RE49,461	Buck ’977
	examples of such references are cited and explained in these invalidity contentions, including the cover pleading and charts C1–C15, and Appendix C.
31. The method of claim 21, further comprising: executing, by the at least one CPU, a data output stream, the data output stream controlling transfer of outputs of the computations representing the artificial neural network to disk.	Buck ’977 discloses, expressly and/or inherently, the method of claim 21, further comprising: executing, by the at least one CPU, a data output stream, the data output stream controlling transfer of outputs of the computations representing the artificial neural network to disk. <i>See claim 21.</i> <i>See e.g.:</i> <i>As a non-limiting example, Buck ’977 discloses at least one CPU and the data output stream controlling transfer of outputs to the disk.</i>

Ex. C12 – Invalidity of U.S. Patent No. RE49,461 – Buck '977



Ex. C12 – Invalidity of U.S. Patent No. RE49,461 – Buck '977

U.S. Patent No. RE49,461	Buck '977
	 FIG. 3 Fig. 3. FIG. 1 illustrates an example of a suitable computing system environment in which the GPU optimization system and method may be implemented. [0015]. FIG. 3 is a block diagram illustrating the interaction between the GPU and GPU optimization system shown in FIG. 2. [0017]. FIG. 1 illustrates an example of a suitable computing system environment in which the GPU optimization system and method may be implemented. The computing system environment 100 is only one example of a suitable computing environment and is not intended to suggest any limitation as to the scope of use or functionality of the invention. Neither should the

Ex. C12 – Invalidity of U.S. Patent No. RE49,461 – Buck '977

U.S. Patent No. RE49,461	Buck '977
	computing environment 100 be interpreted as having any dependency or requirement relating to any one or combination of components illustrated in the exemplary operating environment 100. [0037]. The computer 110 may also include other removable/non-removable, volatile/nonvolatile computer storage media. By way of example only, FIG. 1 illustrates a hard disk drive 141 that reads from or writes to non-removable, nonvolatile magnetic media, a magnetic disk drive 151 that reads from or writes to a removable, nonvolatile magnetic disk 152, and an optical disk drive 155 that reads from or writes to a removable, nonvolatile optical disk 156 such as a CD ROM or other optical media. [0045]. Other removable/non-removable, volatile/nonvolatile computer storage media that can be used in the exemplary operating environment include, but are not limited to, magnetic tape cassettes, flash memory cards, digital versatile disks, digital video tape, solid state RAM, solid state ROM, and the like. The hard disk drive 141 is typically connected to the system bus 121 through a non-removable memory interface such as interface 140, and magnetic disk drive 151 and optical disk drive 155 are typically connected to the system bus 121 by a removable memory interface, such as interface 150. [0046]. Other input devices (not shown) may include a microphone, joystick, game pad, satellite dish, scanner, radio receiver, or a television or broadcast video receiver, or the like. These and other input devices are often connected to the processing unit 120 through a user input interface 160 that is coupled to the system bus 121, but may be connected by other interface and bus structures, such as, for example, a parallel port, game port or a universal serial bus (USB). A monitor 191 or other type of display device is also connected to the system bus 121 via an interface, such as a video interface 190. In addition to the monitor, computers may also include

Ex. C12 – Invalidity of U.S. Patent No. RE49,461 – Buck ’977

U.S. Patent No. RE49,461	Buck ’977
	other peripheral output devices such as speakers 197 and printer 196, which may be connected through an output peripheral interface 195. [0048]. Once the training data has been loaded on the GPU, the CPU instructs the GPU to run the various shaders that make up the learning algorithm (box 1540). A typical learning algorithm is represented in a box inside the GPU. The learning algorithm computes a function $G(X', W)$ as a function of the preprocessed input X' and the trainable parameters W. The goal is to make this output as close as possible as the target value T. An error 1555 between $G(X', W)$ and T is computed, and error signals (e.g. gradient with respect to W) are sent back to the learning machine. The weights are then updated in order to reduce the error between $G(X', W)$ and T. [0218]. Next, the CPU instructs the GPU to use the shaders to compute the function $Y=G(X', W)$ (box 1635). A simple diagram illustrates the trained function in the GPU box. The results for each of the patterns of the group are accumulated in the GPU memory (Result Y box in FIG. 16). [0225]. To the extent Plaintiff contends that this claim element is not expressly or inherently disclosed, or is not obvious in view of Buck ’977 and the knowledge of a POSITA, it would have been obvious to combine the teachings of Buck ’977 with any of the prior art references in NVIDIA’s Invalidity Contentions and any supplements thereto and the relevant section of charts for other prior art for the ’461 patent in a manner that would result in the subject matter of this limitation given, at the very least, the person's understanding of the state of the art, the problems addressed and solved in the prior art, and the teachings of Buck ’977. Specific examples of such references are cited and explained in these invalidity contentions, including the cover pleading and charts C1–C15, and Appendix C.