Merrimac: Supercomputing with Streams

William J. Dally, Patrick Hanrahan, Mattan Erez, Timothy J. Knight
François Labonté, Jung-Ho Ahn, Nuwan Jayasena, Ujval J. Kapasi
Abhishek Das, Jayanth Gummaraju, Ian Buck

ABSTRACT

Merrimac uses stream architecture and advanced interconnection networks to give an order of magnitude more performance per unit cost than cluster-based scientific computers built from the same technology. Organizing the computation into streams and exploiting the resulting locality using a register hierarchy enables a stream architecture to reduce the memory bandwidth required by representative applications by an order of magnitude or more. Hence a processing node with a fixed bandwidth (expensive) can support an order of magnitude more arithmetic units (inexpensive). This in turn allows a given level of performance to be achieved with fewer nodes (a 1-PFLOPS machine, for example, with just 8,192 nodes) resulting in greater reliability, and simpler system management. We sketch the design of Merrimac, a streaming scientific computer that can be scaled from a $20K 2 TFLOPS workstation to a $20M 2 PFLOPS supercomputer and present the results of some initial application experiments on this architecture.

1. Introduction

Modern semiconductor technology makes arithmetic inexpensive and bandwidth expensive. To exploit this shift in cost, a high-performance computer system must exploit locality, to raise the *arithmetic intensity* (the ratio of arithmetic to bandwidth) of the application as well as parallelism to keep a large number of arithmetic units busy. Expressing an application as a *stream program* fulfills both of these requirements. It exposes large amounts of parallelism across stream elements and reduces global bandwidth by expressing locality within and between kernels.

A stream processor exploits the parallelism exposed by a stream program, by providing 100s of arithmetic units, and exploits the locality of a stream program, by providing a deep register hierarchy. In particular, memory bandwidth is reduced by capturing short-term producer-consumer locality in large *local register files*, and long-term producer-consumer locality in a *stream register file*. This locality might not be captured by a reactive cache. More importantly, the stream register file is *aligned* with individual ALUs and requires only local on-chip communication while a cache requires global on-chip communication.

We are designing Merrimac[1], a scientific computer system tailored to exploit the parallelism and locality of streams. The core of Merrimac is a single-chip (90nm CMOS) stream processor that is expected to have 128 GFLOPS peak performance. This processor chip along with 16 high-bandwidth DRAM chips (2G Bytes of memory) form a single Merrimac node. Application experiments suggest that this single-node Merrimac will sustain up to half of peak performance on a range of scientific applications. With an estimated parts cost of less than $1K per 128 GFLOPS node (including network), we expect a Merrimac machine to provide both capability and capacity — being more cost effective than machines based on commodity microprocessors.

Merrimac employs a *high-radix* interconnection network to connect 16 nodes (2 TFLOPS) on a single board, 512 nodes (64 TFLOPS) in a cabinet, and 8K nodes (1 PFLOPS) in 16 cabinets. The network provides a flat shared address space across the multi-cabinet system with flat bandwidth across a board (16 nodes) and a global bandwidth of 1/8 the local bandwidth anywhere in the system.

We have coded three representative scientific applications as stream programs and measured their performance on a simulated Merrimac node. These initial experiments show that typical scientific applications cast as stream programs maintain a high arithmetic to memory bandwidth ratio and achieve a high fraction of peak performance. The applications simulated have computation-to-memory ratios in the range of 7:1 to 50:1, achieving between 18% and 52% of the peak performance of the machine, with less than 1.5% of data references traveling off-chip.

The remainder of this paper describes stream processors and the Merrimac project in more detail. In Section 2 we see that modern VLSI technology makes arithmetic cheap and bandwidth expensive. Section 3 shows how a stream processor exploits the application locality using a bandwidth hierarchy and application parallelism by using large numbers of ALUs. Merrimac, a supercomputer based on streams, is described in Section 4. We show the performance of a simulated stream processor on a number of applications in Section 5. Issues related to scientific computing with streams are discussed in Section 6

This work was supported in part by the Department of Energy, NNSA, under the ASCI Alliances program (contract LLL-B341491), in part by National Science Foundation Fellowships, in part by Stanford Graduate Fellowships, in part by the DARPA Smart Memories Project (contract MDA904-98-R-S855), and in part by the DARPA Polymorphous Computing Architectures Project (contract F29601-00-2-0085).

SC'03, November 15-21, 2003, Phoenix, Arizona, USA
Copyright 2003 ACM 1-58113-695-1/03/0011...$5.00

[1] Merrimac is a Native American word meaning "fast moving stream".

2. VLSI enables inexpensive arithmetic making bandwidth the limiting factor

Modern VLSI fabrication processes make it very inexpensive in terms of both area and power to put large amounts of arithmetic capability on a chip. With arithmetic almost free, global bandwidth, both on-chip and off-chip, becomes the factor limiting performance.

In $0.13\mu m$ CMOS technology, a 64-bit floating-point unit (FPU) (multiplier and adder) has an area of less than 1mm^2 and dissipates about 50pJ of energy per operation [1]. Over 200 such FPUs can fit on a 14mm × 14mm chip that can be manufactured in volume (including testing and packaging) for less than $100. Even at a conservative operating frequency of 500MHz this gives a cost of 64-bit floating-point arithmetic of less than $1 per GFLOPS and a power of less than 50mW per GFLOPS. Even though one cannot completely fill a chip with FPUs, modern graphics chips come close to realizing these cost performance levels. For example, the nVidia NV30 sustains 100 GFLOPS (32-bit floating point) [2].

The already low cost of arithmetic is decreasing rapidly as technology improves. We describe a CMOS technology by its drawn gate length L. Most chips today are manufactured with $L = 0.13\mu m$. Historical trends show that L decreases at about 14% per year [3]. The cost of a GFLOPS of arithmetic scales as L^3 and hence decreases at a rate of about 35% per year [4]. Every five years, L is halved, four times as many FPUs fit on a chip of a given area, and they operate twice as fast — giving a total of eight times the performance for the same cost. Of equal importance, the switching energy also scales as L^3 so every five years, we get eight times the arithmetic performance for the same power.

Global bandwidth, not arithmetic is the factor limiting the performance and dominating the power of modern processors. The cost of bandwidth grows at least linearly with distance in terms of both availability and power [4]. To keep distances constant across technology generations, we express distance in units of *tracks*. One track (or 1χ) is the distance between two minimum width wires on a chip. In $0.13\mu m$ technology, $1\chi \approx 0.5\mu m$. We can put ten times as many $10^3\chi$ wires on a chip as we can $10^4\chi$ wires. More importantly, moving a bit of information over a $10^3\chi$ wire takes only $1/10^{\text{th}}$ the energy as moving a bit over a $10^4\chi$ wire. In an $0.13\mu m$ technology, for example, transporting the three 64-bit operands for a 50pJ floating point operation over global $3 \times 10^4\chi$ wires consumes about 1nJ, 20 times the energy required to do the operation. In contrast, transporting these operands on local wires with an average length of $3 \times 10^2\chi$ takes only 10pJ, much less than the cost of the operation.

Contemporary architectures are not yet tuned to these developing VLSI constraints. These architectures are unable to use more than a few arithmetic units because they are designed for applications with limited parallelism and are hindered by a low bandwidth memory system. Their main goal is to provide high performance for mostly serial code that is highly sensitive to memory latency and not bandwidth. To exploit the capabilities of today's VLSI technology requires an architecture that can exploit parallelism — to keep large numbers or arithmetic units busy while hiding the ever increasing latency to memory, and locality — to increase the ratio of arithmetic, which is inexpensive, to global bandwidth, which is the limiting factor.

3. Stream Architecture exploits the characteristics of VLSI

A *Stream Processor* is able to take advantage of the large number of arithmetic units that VLSI technology enables without exceeding the bandwidth limitations of the technology by using a *register hierarchy* to exploit locality in the application. This greatly reduces the average distance an operand must travel to reach a FPU. As shown in Figure 1, a stream architecture consists of an array of clusters, each with a set of FPUs, a set of *local register files* (LRFs), and a bank of a *stream register file* (SRF). Each FPU in a cluster reads its operands out of an adjacent LRF over very short, ($\approx 100\chi$), wires. FPU results are distributed to the other LRFs in a cluster and accesses to the local SRF bank are made via the cluster switch over short ($\approx 1,000\chi$) wires. While the SRF is similar in size to a cache, SRF accesses are much less expensive than cache accesses because they are aligned and do not require a tag lookup. Each cluster accesses its own bank of the SRF over short wires. In contrast, accessing a cache requires a global communication over long ($\approx 10,000\chi$) wires. The SRF also plays another crucial role in keeping the arithmetic units busy by allowing the software to hide long memory latencies. An entire stream is transferred between the SRF and the memory with a single instruction. These stream memory operations generate a large number of memory references to fill the very deep pipeline between processor and memory, allowing memory bandwidth to be maintained in the presence of latency. Arithmetic units are kept busy by overlapping the execution of arithmetic kernels with these stream memory operations.

To see how a stream processor exploits locality, consider a simple application expressed as a stream program (Figure 2). This figure shows a synthetic application that is designed to have the same bandwidth demands as the StreamFEM application (Section 5). Each iteration, the application streams a set of 5-word grid cells into a series of four kernels. The kernels operate on the data, performing the number of operations indicated, and pass intermediate results on to the next kernel. To perform a table lookup, kernel K1 generates an index stream that is used to reference a table in memory generating a 3-word per element stream into kernel K3.

Figure 3 shows how the stream program of Figure 2 maps to the register hierarchy of a stream processor. The grid cells start in memory and are read a *strip* at a time into a buffer in the SRF. A typical strip might be 1024 5-word records.[2] Once a strip of cells is in the SRF, kernel K1 is run generating a strip of indices and a strip of intermediate results in the SRF. Kernel K2 is run on the results, generating a second set of intermediate results while the indices are applied to memory to read a strip of table values into the SRF. Table values that are repeatedly accessed are provided by the cache. The process continues until the updates to the strip of grid cells, generated by kernel K4, are written back to memory. Each strip is software pipelined so that the loading of one strip of cells is overlapped with the execution of the four kernels on the previous strip of cells and the storing of the strip before that.

This synthetic application shows how the stream architecture exploits locality. In Section 5 we shall see that actual applications exploit locality in a similar manner. Kernels K1...K4 perform all of their 300 operations out of LRFs, performing 900 LRF accesses per grid point. The streams between the kernels are passed through the SRF generating 58 words of SRF bandwidth per grid point. Finally memory accesses total 12 words. This gives us a bandwidth ratio of 75:5:1, 75 LRF references and 5 SRF references for every mem-

[2] The strip size is chosen by the compiler to use the entire SRF without any spilling.

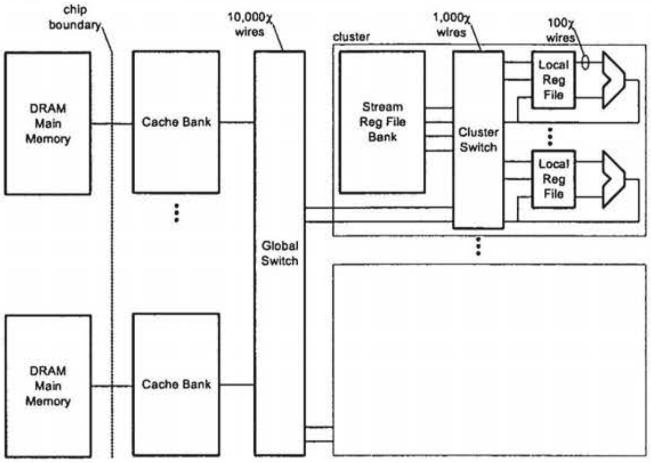


Figure 1: A stream processor consists of an array of clusters each having a number of functional units with local register files and a stream register file bank connected by a cluster switch. The clusters are connected to each other and to cache banks by a global switch. At each level of this hierarchy — local register, intra-cluster, and inter-cluster — the wires get an order of magnitude longer.

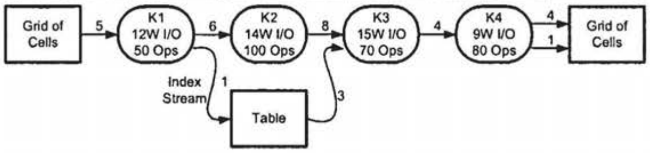


Figure 2: A synthetic stream application, modeled after StreamFEM (Section 5) consists of a set of *kernels* K1 . . . K4 that pass *streams* of data between them.

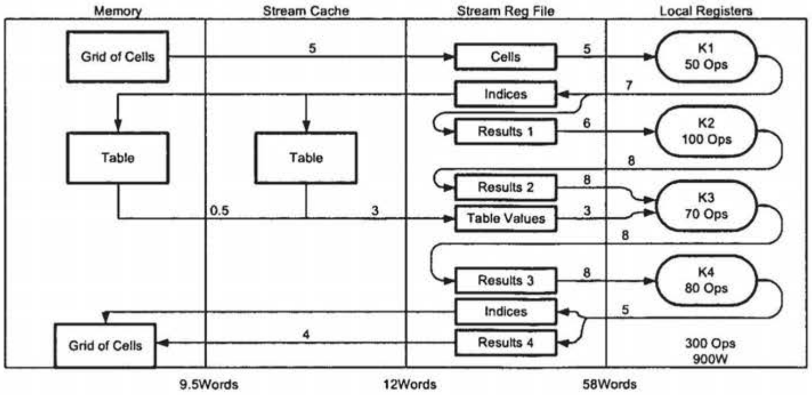


Figure 3: The stream program of Figure 2 is mapped to the bandwidth hierarchy of a stream processor.

ory reference. Put differently, 93% of all references are made from the LRFs, where bandwidth is very inexpensive, and only 1.2% of references are made from the memory system, where bandwidth is expensive for cache hits and very expensive for misses.[3]

A stream processor executes a stream instruction set. This instruction set includes scalar instructions, that are executed on a conventional scalar processor, stream execution instructions, that each trigger the execution of a kernel on one or more strips in the SRF, and stream memory instructions that load and store (possibly with gather and scatter) a stream of records from memory to the SRF. This stream instruction set closely follows that of the Imagine streaming media processor [5, 6].

Merrimac also provides hardware support for a *scatter-add* instruction. This instruction is an example of a new architectural feature that is enabled by programming in streams. A scatter-add acts as a regular scatter, but adds each value to the data already at each specified memory address rather than simply overwriting the data. This type of operation was discussed from a parallel algorithm perspective in [7].

4. Sketch of Merrimac: a Streaming Scientific Computer

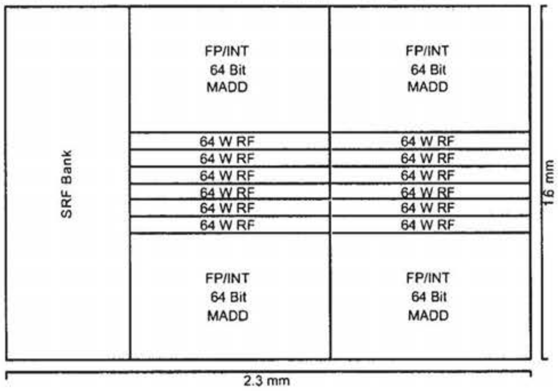


Figure 4: Floorplan of a Merrimac cluster.

Each Merrimac node contains a stream processor (as illustrated in Figure 1) with 16 arithmetic clusters. Each cluster contains four floating-point multiply-add (MADD) units, 768 64-bit words of local registers, and 8K words of stream register file. The entire stream register file has a capacity of 128K 64-bit words, distributed across the 16 clusters. A floorplan for one cluster is shown in Figure 4. Each MADD unit measures 0.9mm × 0.6mm and the entire cluster measures 2.3mm × 1.6mm. We conservatively plan to operate with a clock cycle of 1ns (37 FO4 inverters in 90nm[3]) giving a performance of 8 GFLOPS per cluster and 128 GFLOPS across the 16 clusters.

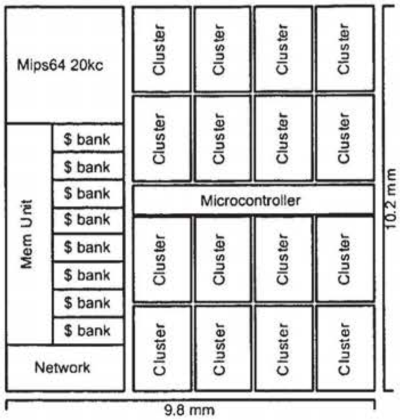


Figure 5: Floorplan of a Merrimac stream processor chip.

A floorplan of the entire Merrimac stream processor chip is shown in Figure 5. The bulk of the chip is occupied by the 16 clusters. The left edge of the chip holds the remainder of the node. A scalar processor [8] fetches all instructions, executes the scalar instructions itself, and dispatches stream execution instructions to the clusters (under control of the microcontroller) and stream memory instructions to the memory system. The node memory system consists of a set of address generators (not shown), a line-interleaved eight-bank 64K-word (512KByte) cache, and interfaces for 16 external DRAM chips. A network interface directs off-node memory references to the routers. We estimate that each Merrimac processor will cost about $200 to manufacture and will dissipate a maximum of 31W of power. Area and power estimates in a standard cell process in 90nm technology are derived from models based on a previous implementation of stream processor [1].

Figure 6 illustrates a single Merrimac board containing 16 nodes — 16 128GFLOPS stream processors (Figure 5) each with 2 GBytes of DRAM — and four router chips. The router chips interconnect the 16 processors on the board, providing flat memory bandwidth on board of 20 GBytes/s per node. The routers also provide a gateway to the inter-board network, with a 4:1 reduction in memory bandwidth (to 5 GBytes/s per node), for inter-board references.

Larger Merrimac systems are interconnected by a five-stage folded-Clos [9] network[4] using high-radix routers as illustrated in Figure 7. The routers on each 16-node board serve as the first and last stage of this network. The basic building block of this network is a 48-input × 48-output router chip. Each bidirectional router channel (one input and one output) has a bandwidth of 2.5 GBytes/s (four 5Gb/s differential signals) in each direction. On each 16-processor board, each of four routers has two 2.5 GByte/s channels to/from each of the 16 processor chips and eight ports to/from the backplane switch. The remaining eight ports are unused. Thus each node provides a total of 32 channels to the backplane. At the backplane level, 32 routers connect one channel to each of the 32 boards and connect 16 channels to the system-level switch. A total of 512 2.5 GByte/s channels traverse optical links to the system-level switch where 512 routers connect all 48 ports to up to 48 backplanes (the figure shows just 32 backplanes).

Table 1 shows the estimated cost of a streaming supercomputer. The processor and router chip are modest-sized (10mm × 11mm) ASICs in 1000-pin flip-chip BGA packages that are expected to

[3] Many of our applications have very large kernels that in effect combine several smaller kernels — passing intermediate results through LRFs rather than SRFs. While this increases the fraction of LRF accesses, it also stresses LRF capacity. Ideally, the compiler will partition large kernels and combine small kernels to balance these two effects. We have not yet implemented this optimization.

[4] This topology is sometimes called a Fat Tree [10].

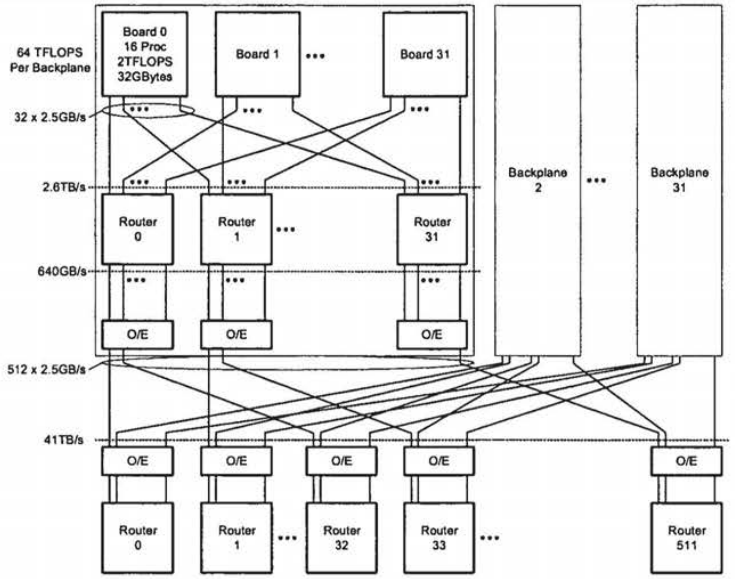


Figure 7: A 2 PFLOPS Merrimac system uses a high-radix interconnection network.

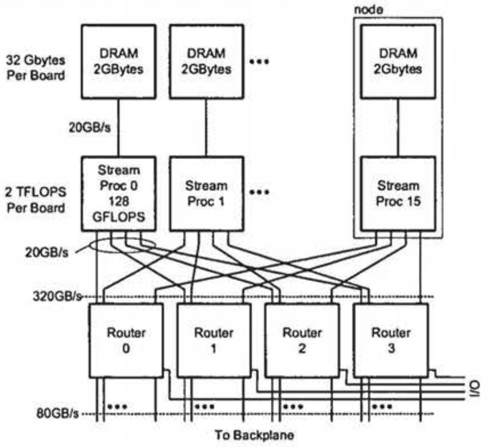


Figure 6: Sixteen 128 GFLOPS stream processors each with 2 GBytes of DRAM memory can be packaged on a single board. The board has a total of 2 TFLOPS of arithmetic and 32 GBytes of memory. Such a board is useful as a stand-alone scientific computer and as a building-block for larger systems.

Item	Cost($)	Per Node Cost ($)
Processor Chip	200	200
Router Chip	200	69
Memory Chip	20	320
Board	1000	63
Router Board	1000	2
Backplane	5000	10
Global Router Board	5000	5
Power	1	50
Per Node Cost		718
$/GFLOPS (128/Node)		6
$/M-GUPS (250/Node)		3

Table 1: Rough Per-Node Budget. Parts cost only, does not include I/O.

cost $200 each in moderate quantities (1000s). DRAM chips are projected to cost $20 each, making DRAM, at $320 the largest single cost item. Board and backplane costs, including connectors, capacitors, regulators, and other components is amortized over the 16 nodes on each board and the 512 nodes in each backplane. The router board and global router board costs reflect the costs of the intra-cabinet and inter-cabinet networks respectively. Supplying and removing power costs about $1 per W or about $50 per 50W node. Overall cost is less than $1K per node, which translates into $6 per GFLOP of peak performance and $3 per M-GUPS[5].

[5] GUPS or *global updates per second* is a measure of global unstructured memory bandwidth. It is the number of single-word read-modify-write operations a machine can perform to memory locations randomly selected from over the entire address space.

Application	Sustained GFLOPS	FP Ops / Mem Ref	LRF Refs	SRF Refs	Mem. Refs
StreamFEM (Euler, quadratic)	32.2	23.5	169.5M (93.6%)	10.3M (5.7%)	1.4M (0.7%)
StreamFEM (MHD, cubic)	33.5	50.6	733.3M (94.0%)	43.8M (5.6%)	3.2M (0.4%)
StreamMD	14.2	12.1	90.2M (97.5%)	1.6M (1.7%)	0.7M (0.8%)
StreamFLO	11.4	7.4	234.3M (95.7%)	7.2M (2.9%)	3.4M (1.4%)

Table 2: Performance measurements of streaming scientific applications

5. Applications exploit the locality of a stream processor

Three scientific applications were used to evaluate the single node performance of the Merrimac stream processor: StreamFEM, StreamMD, and StreamFLO. These applications feature a number of characteristics which are common in scientific applications in general, including regular and irregular multidimensional meshes, multigrid techniques, and particle-in-cell computations.

StreamFEM is a finite element application designed to solve systems of first-order conservation laws on general unstructured meshes. The StreamFEM implementation has the capability of solving systems of 2D conservation laws corresponding to scalar transport, compressible gas dynamics, and magnetohydrodynamics (MHD) using element approximation spaces ranging from piecewise constant to piecewise cubic polynomials. StreamFEM uses the discontinuous Galerkin (DG) method developed by Reed and Hill [11] and later popularized by Cockburn, Hou and Shu [12]. In the present StreamFEM implementation, the limiting procedure of Cockburn et al. has been replaced by variational discontinuity capturing terms as discussed in Jaffre, Johnson and Szepessy [13] with further overall algorithmic simplifications as discussed in Barth [14].

StreamMD is a molecular dynamics solver [15, 16] that is based on solving Newton's equations of motion. The velocity Verlet method (or Leap-frog) is used to integrate the equations of motion in time; using this method, it is possible to simulate the complex trajectories of atoms and molecules for very long periods of time. The present StreamMD implementation simulates a box of water molecules, with the potential energy function defined as the sum of two terms: electrostatic potential and the Van der Waals potential. A cutoff is applied so that all particles which are at a distance greater than r_{cutoff} do not interact. A 3D gridding structure is used to accelerate the determination of which particles are close enough to interact – each grid cell contains a list of the particles within that cell, and each timestep particles may move between grid cells. StreamMD makes use of the scatter-add functionality of Merrimac by computing the pairwise particle forces in parallel and accumulating the forces on each particle by scattering them to memory.

StreamFLO [17] is a finite volume 2D Euler solver that uses a non-linear multigrid algorithm. It is based on the FLO82 code [18][19], which influenced many industrial and research codes. The choice of the code is motivated by the need for an application that is representative of a typical computational fluid dynamics application, without unnecessary complexity. A cell-centered finite-volume formulation is used to solve the fluid equations together with multigrid acceleration. Time integration is performed using a five stage Runge-Kutta scheme.

Table 2 presents measurements from running these three applications on a cycle-accurate simulator of one Merrimac node. These simulations were run on a version of the simulator that included four 2-input multiply/add units per cluster (for a peak performance of 64GFLOPS/node) rather than the four integrated 3-input MADD units (128GFLOPS/node) that is the current design.

The Sustained GFLOPS and FP Ops / Mem Ref columns illustrate the arithmetic intensity of the applications; they are able to sustain from 18% to 52% of the node's peak arithmetic performance, by performing from 7 to 50 floating point operations for each global memory access. Note that only "real" ops are counted in this figure, such as floating point add/mul/compare instructions, and not non-arithmetic ops such as branches. Divides are counted as single floating point operations, even though each divide requires several multiplication and addition operations when executed on the hardware. This leads to the lower performance numbers for StreamMD and StreamFLO – for example, the sustained performance of StreamFLO would double if we counted all the multiplies and adds required for divisions as well.

The right-most three columns list the respective numbers of LRF, SRF, and memory references made by the program, along with the percentage of references satisfied by each level. Note that only a small fraction of references, usually less than 1%, require communication over global ($> 10,000\chi$ or off-chip) wires, and that over 95% of all data movement is on local (100χ) wires (at the LRF level). The register hierarchy of a stream processor exposes costly global communication and allows the locality inherent in applications to be exploited to keep communications local.

Exploiting locality using a register hierarchy increases performance and reduces power dissipation. By performing less data movement per arithmetic operation, we can support a much larger number of arithmetic units before saturating the limited global bandwidth. At the same time power per operation is dramatically reduced by eliminating much of the global communication that dominates power.

6. Discussion

6.1 Streams vs Vectors

Stream processors share with vector processors, like the Cray1 through Cray C90 [20][21], the ability to hide latency, amortize instruction overhead, and expose data parallelism by operating on large aggregates of data. In a similar manner, a stream processor, such as Merrimac, hides memory latency by fetching a stream of records with a single stream load instruction. A kernel is performed on one or more streams of records in the stream register file (SRF) with a single operate instruction. This both amortizes the overhead of the operate instruction and exposes data parallelism.

Stream processors extend the capabilities of vector processors by adding a layer to the register hierarchy, and adding a layer of instruction sequencing that enables them to operate in record (rather than operation) order. The functions of the vector register file (VRF) of a vector processor is split between the local register files (LRFs) and the stream register file (SRF) of a stream processor. The LRFs stage data between ALU operations to exploit fine-grained producer-consumer locality (sometimes called *kernel* locality). To support a large number of ALUs, they have a very high aggregate bandwidth. Because they exploit only kernel locality, their capacity can be modest, a few thousand words - about the same size as a modern VRF. The stream register file (SRF) of a stream processor stages data to and from memory and stages data to and from the LRFs to exploit coarse-grained (sometimes called outer-loop) producer-consumer locality. Because it is relieved of the task of forwarding data to/from the ALUs, its bandwidth is modest (an order of magnitude less than the LRFs) which makes it economical to build SRFs large enough to exploit coarse-grained locality.

6.2 Balance

The ratios between arithmetic rate, memory bandwidth, and memory capacity on Merrimac are balanced based on cost and utility — so that the last dollar spent on each returns the same incremental improvement in performance. This balancing by diminishing returns gives ratios quite different from the common approach of fixing the ratio of GFLOPS to GBytes irrespective of cost. If we took this approach with Merrimac, we would have to provide 128 GBytes of memory (costing about $20K) for each $200 processor chip making our processor to memory cost ratio 1:100. If one needs 128 GBytes of memory, it is more efficient to provide 64 nodes, even if the additional processors are not required — their cost is small compared to the memory.

A similar argument applies to the ratio of arithmetic to memory bandwidth. Merrimac provides only 20 GBytes/s (2.5 GWords/s) of memory bandwidth for 128 GFLOPS, a FLOP/Word ratio of over 50:1. Many vector machines have FLOP/Word ratios of 1:1 [21], and conventional microprocessors have ratios between 4:1 and 12:1 [22][23]. Providing even a 10:1 ratio on Merrimac would be prohibitively expensive. We would need 80 external DRAMs rather than 16. Interfacing to this large number of DRAMs would require at least 5 external memory interface chips (pin expanders). As with memory capacity, taking this fixed-balance approach to memory bandwidth causes the cost of bandwidth to dominate the cost of processing. Its more efficient to just use Merrimac processor chips to directly interface to 16 DRAMs each. For memory bandwidth dominated computations (e.g., sparse vector-matrix product) most of the arithmetic will be idle. However, even for such computations the Merrimac approach is more cost effective than trying to provide a much larger memory bandwidth for a single node.

6.3 High-Radix Routers

In the 1980s and early 90s, when routers had pin bandwidth in the range of 1-10Gb/s, torus networks gave high throughput while balancing serialization latency against network diameter. For this reason, torus networks were quite popular during this period [24, 25, 26]. Today, with router chip pin bandwidths between 100Gb/s and 1Tb/s possible, a torus can no longer make effective use of this bandwidth. A topology with a higher node degree (or radix) is required. When used in conjunction with *channel slicing*, slicing each node's 20GB/s of network bandwidth across eight 2.5GB/s channels, building routers with high degree (48 for Merrimac) enables a network with very low diameter (2 hops to 16 nodes, 4 hops to 512 nodes, and 6 hops to 24K nodes) compared to a 3-D torus (with a node degree of 6).[6] The use of a Clos network has the added advantage that its hierarchical nature facilitates the use of optical links to cover the long distances required at the top level [27].

7. Conclusion

Modern VLSI technology makes arithmetic very cheap (100s of 64-bit FPUs per chip) and bandwidth very expensive (a few words/cycle of off-chip bandwidth). Expressing an application as a stream program exposes parallelism — to take advantage of the large number of arithmetic units and to hide the ever increasing memory latencies — and locality — to reduce the demand on the limited bandwidth. A stream processor exploits this parallelism and locality by providing a deep bandwidth hierarchy that exposes communication so it can be optimized by a compiler. By capturing short-term producer-consumer locality in local registers and long-term producer-consumer locality in a stream register file, a stream processor significantly reduces an application's demand on memory bandwidth.

Merrimac is a stream processor tailored for scientific applications. Merrimac is scalable from a 2 TFLOPS single-board workstation to a 2PFLOPS supercomputer. A 90nm CMOS stream processor chip with a peak performance of 128 GFLOPS enables Merrimac to sustain a high ratio of arithmetic operations to external bandwidth. This allows Merrimac to achieve an efficiency of 128 MFLOPS/$ peak and 23-64 MFLOPS/$ sustained on our pilot applications[7]. A high-radix network gives Merrimac a flat global address space with only an 8:1 (local:global) bandwidth ratio. This gives Merrimac a memory efficiency of 250 K-GUPS/$. This relatively flat global memory bandwidth simplifies programming by reducing the importance of partitioning and placement.

Three representative scientific applications have been converted to stream programs, compiled for Merrimac, and executed on a cycle-accurate simulation of a Merrimac node. These applications all exhibit high locality, maintaining arithmetic to memory access ratios from 7 to 50. Across these applications, over 96% of all data accesses are from local register files and less than 1.5% are to memory. These experiments verify that streams can extract sufficient locality from representative scientific codes to sustain high arithmetic rates with limited memory bandwidth.

The results we present here establish the feasibility of using stream processing for scientific computing — by showing that stream locality exists in representative scientific codes — and suggests that a stream processor can significantly improve the performance per unit cost of scientific computing.

Scientific stream processing raises many interesting questions for future research. Our initial experiments used programs that were manually restructured into stream programs. The development of compilation methods to automate this process of partitioning a vectorized or parallelized code into kernels would make it easier to apply stream processing to enhance the locality of a large volume of existing code. We are also interested in compilation methods that perform transformations on stream programs, splitting and merging kernels to balance register use, and rescheduling kernels and memory operations to most efficiently stage data through the stream register file.

On the architecture front, we are exploring alternative stream register file organizations that appear to offer even greater reductions in required memory bandwidth. We are investigating how to best use a cache in combination with a stream register file and how to give the compiler more control over caching policies. We are also investigating global communication and synchronization

[6] If we employed a butterfly rather than a Clos topology these diameters would be nearly halved. Unfortunately a butterfly network is not practical because of its poor performance routing certain permutations.

[7] Projected from the experiments of Section 5.

mechanisms that are suitable for use with streams. This includes our scatter-add operation, which reduces the need for synchronization in many applications.

Finally our initial experiments used relativley simple 2D codes running on a single node of a simulated machine. We are currently exploring the properties of larger and more complex 3D codes running across multiple nodes of a simulated machine. Initial indications are positive — that these codes exhibit at least as much 'stream' locality as their simpler counterparts.

8. Acknowledgements

We would like to thank Massimiliano Fatica and Eric Darve from the Mechanical Engineering Department at Stanford, as well as Timothy J. Barth and Alan Wray from NASA Ames for their great contribution to the project, and specifically for providing applications and working with us on their Merrimac implementation. We would also like to thank Bill Reynolds, Parviz Moin, and Juan J. Alonso from Stanford University for their support of this work as part of the Center for Integrated Turbulence Simulation.

9. REFERENCES

[1] Khailany, B., Dally, W. J., Rixner, S., Kapasi, U. J., Owen, J. D., and Towles, B., "Exploring the VLSI Scalability of Stream Processors," *Proceedings of the Ninth Symposium on High Performance Computer Architecture*, Anaheim, California, USA, February 2003, pp. 153–164.

[2] *n*VIDIA®, "*n*VIDIA® GeFORCE^{TM} FX," http://www.nvidia.com/docs/lo/2430/SUPP/PO_GFFX_Consumer_030503.pdf.

[3] Semiconductor Industry Association, *The International Technology Roadmap for Semiconductors*, 2001 Edition.

[4] Dally, W. J. and Poulton, W., *Digital Systems Engineering*, Cambridge University Press, 1998.

[5] Khailany, B., Dally, W. J., Rixner, S., Kapasi, U. J., Mattson, P., Namkoong, J., Owens, J. D., Towles, B., and Chang, A., "Imagine: Media Processing with Streams," *IEEE Micro*, March/April 2001, pp. 35–46.

[6] Kapasi, U. J., Rixner, S., Dally, W. J., Khailany, B., Ahn, J. H., Mattson, P., and Owens, J. D., "Programmable Stream Processors," *IEEE Computer*, August 2003.

[7] Kallinderis, Y. and Vidwans, A., "Generic Parallel Adaptive-Grid NavierStokes Algorithm," *AIAA Journal*, Vol. 32, 1994, pp. 54–61.

[8] MIPS Technologies, *MIPS64 20Kc Core*, http://www.mips.com/ProductCatalog/P_MIPS6420KcCore.

[9] Clos, C., "A Study of Non-Blocking Switching Networks," *Bell System Technical Journal*, Vol. 32, 1953, pp. 406–424.

[10] Leiserson, C. E., "Fat-Trees: Universal Networks for Hardware Efficient Supercomputing," *IEEE Transactions on Computers*, Vol. 34, No. 10, October 1985, pp. 892–901.

[11] Reed, W. H. and Hill, T. R., "Triangular mesh methods for the neutron transport equation," Tech. Rep. LA-UR-73-479, Los Alamos National Laboratory, Los Alamos, New Mexico, 1973.

[12] Cockburn, B., Hou, S., and Shu, C., "TVB Runge-Kutta Local Projection Discontinuous Galerkin Finite Element Method for Conservation Laws IV: The multidimensional case," *Math. Comp.*, Vol. 54, 1990, pp. 545–581.

[13] Jaffre, J., Johnson, C., and Szepessy, A., "Convergence of the Discontinuous Galerkin Finite Element Method for Hyperbolic Conservation Laws," *Math. Models and Methods in Appl. Sci.*, Vol. 5, No. 3, 1995, pp. 367–386.

[14] Barth, T., "Simplified Discontinuous Galerkin Methods for Systems of Conservation Laws with Convex Extension," *Discontinuous Galerkin Methods*, edited by Cockburn, Karniadakis, and Shu, Vol. 11 of *Lecture Notes in Computational Science and Engineering*, Springer-Verlag, Heidelberg, 1999.

[15] Darve, E. and Pohorille, A., "Calculating Free Energies using Average Force," *Chemical Physics*, Vol. 115, No. 20, 2001, pp. 9169–9183.

[16] Darve, E., Wilson, M., and Pohorille, A., "Calculating Free Energies using a Scaled-Force Molecular Dynamics Algorithm," *Molecular Simulation*, Vol. 28, No. 1–2, 2002, pp. 113–144.

[17] Fatica, M., Jameson, A., and Alonso, J. J., "STREAMFLO: an Euler solver for streaming architectures," *submitted to AIAA Conference*, Reno, Nevada, USA, 2004.

[18] Jameson, A., "Analysis and design of numerical schemes for gas dynamics 1. Artificial diffusion, upwind biasing, limiters and their effects on accuracy and multigrid convergence," *International Journal of Computational Fluid Dynamics*, Vol. Volume 4, 1995, pp. 171–218.

[19] Jameson, A., "Analysis and design of numerical schemes for gas dynamics 2. Artificial diffusion and discrete shock structure," *International Journal of Computational Fluid Dynamics*, Vol. Volume 5, 1995, pp. 1–38.

[20] Russell, R. M., "The CRAY-1 Computer System," *Communications of the ACM*, Vol. 21, No. 1, Jan. 1978, pp. 63–72.

[21] Simmons, M. L., Wasserman, H. J., Lubeck, O. M., Eoyang, C., Mendez, R., Harada, H., and Ishiguro, M., "A performance comparison of four supercomputers," *Communications of the ACM*, Vol. 35, No. 8, Aug. 1992, pp. 116–124.

[22] Intel®, "Intel® 850E Chipset," http://www.intel.com/design/chipsets/850e/index.htm.

[23] Intel®, "Intel® Pentium® 4 Processor," http://www.intel.com/products/desktop/processors/pentium4/index.htm.

[24] Dally, W. J., "Performance Analysis of k-ary n-cube Interconnection Networks," *IEEE Transactions on Computers*, Vol. 39, No. 6, June 1991, pp. 775–785.

[25] Kessler, R. E. and Schwarzmeier, J. L., "Cray T3D: a new dimension for Cray Research," *Proc. of the IEEE Computer Society International Conferrence (COMPCON)*, Feb. 1993, pp. 176–182.

[26] Scott, S. L. and Thorson, G. M., "The Cray T3E Network: Adaptive Routing in a High Performance 3D Torus," *Proc. of the Symposium on Hot Interconnects*, Aug. 1996, pp. 147–156.

[27] Gupta, A. K., Dally, W. J., Singh, A., and Towles, B., "Scalable Opto-Electronic Network (SOENet)," *proceedings of Hot Interconnects (HotI) X*, Stanford, California, USA, August 2002.

A Streaming Supercomputer

Bill Dally, Pat Hanrahan, and Ron Fedkiw

September 18, 2001

1 Introduction

1.1 We are starving in an era of plenty

We are in an era where computational building blocks are plentiful and inexpensive. A single chip today can hold over 100 1GHz floating-point units for a total performance of 100 GFLOPS/chip. Many graphics chips achieve 80GFLOPS and over 1TOP rendering performance, and cost less than $100. Embedded processors are less powerful, but incredibly cheap. It is fair to say that a raw GFLOPS costs less than $1. Memory is currently selling for less than 20 cents a MByte. Bandwidth has become less expensive as well. Chips with a Tb/s of aggregate bandwidth have recently been demonstrated.

In this era of plenty, however, we have not developed technology to cost effectively scale computing. Supercomputers cost significantly more per GFLOPS and GByte than their low-end counterparts. For example, it is estimated that total cost of future large-scale ASCI machines with 10's of thousands of nodes is greater than $1,000 per GFLOPS. This factor of a 1000:1 in cost effectiveness is paradoxical: it should be possible to reap economies of scale with computing, just as in other major acquisitions. Although scalability has long been a focus of computer science research, it has not been transferred into practical commercial systems. Now more than ever we need to build the technological infrastructure to cost-effectively scale computation.

In addition to being cost inefficient, contemporary high-end computers, constructed from clusters of workstations or servers, do not deliver their promised performance. They achieve a small fraction of peak performance on many key applications that are dominated by global communication. Critical calculations, such as verifying nuclear weapons, performing signal intelligence, calculating the dynamics of protein folding, and fluid flow through complex turbomachinary, do not map well to these machines.

The performance of the microprocessors from which these clusters are composed is no longer scaling at the historic rate of 50% per year. Microprocessors have reached a point of diminishing returns in terms of gates per clock and clocks per instruction. As we enter an era of billion transistor chips, there is not enough explicit parallelism in conventional programs to efficiently use these resources. For example, a modern graphics processor has at least 64 floating point ALUS and 1000's of integer ALUs, almost a hundred times the arithmetic density of a microprocessor. In contrast, most of the chip area in a microprocessor is devoted to cache memory or the support infrastructure (e.g. supporting out-of-order execution) to keep a few ALUS running at their peak clock rate. It is expected that without new innovations in parallel processor designs, microprocessor performance will only increase with the increase in gate speed, at a rate of about 20% per year. Such as change would have a major effect on the computer business, and the entire economy.

Cluster supercomputers, like the microprocessors they are constructed from, are inefficient because they are poorly matched to the technology from which they are constructed and the applications which they run. They are unable to efficiently exploit the large numbers of floating-point units that can be fabricated on a chip. They also have low global bandwidth and have register and cache architectures that do not capture large amounts of application locality and hence make excessive demands on this bandwidth. Because these systems are not well-designed, they are difficult to program. Programmers spend all their time working around the limitations of the machine, rather than on developing efficient algorithms for their application.

1.2 Streaming processors leverage emerging technology

Recent developments enable streaming architectures that efficiently convert the capabilities of emerging technology into realized performance on scientific applications. We envision a streaming supercomputer that delivers orders of magnitude more performance per dollar than clusters of servers ($50 per GFLOPS and \$2 per million GUPS[1]) and is scalable to a machine with one PFLOPS of peak performance and 10^{13} GUPS. We expect that applications will achieve a large fraction of the peak FLOPS (at least 25%) on arithmetic-limited code sections and a large fraction of peak GUPS on memory-limited code sections.

Streaming supercomputers with this level of performance and efficiency are made possible by the confluence of three recent innovations: stream architecture, high-speed signaling, and efficient interconnection network architecture. Stream architectures expose and exploit parallelism and locality in applications. This in turn enables architectures with a high degree of *arithmetic intensity*, that is applications with a high ratio of arithmetic to memory bandwidth. Streams also offer an easy way to hide the inherent latency of global memory references. Stream architectures have been proven on signal- and image-processing applications. Our initial investigations show that they are equally applicable to a broad class of scientific applications. High-speed signaling and efficient network architecture together enable economical memory systems with very high global bandwidth.

The streaming architecture we envision leverages commodity technology to economically achieve high performance. It does not, however, use commodity processors. Processors, in fact, are not really a commodity - they are not interchangeably available from multiple vendors at prices close to cost. Commodity technology is leveraged in three ways. First, the main memory of the system is built entirely out of commodity high-bandwidth memory chips. Such memory chips truly are a commodity - they are available in volume from a number of different suppliers at competitive prices. Second, the streaming processor chips are fabricated using a standard CMOS process. As with memories, CMOS wafers are a commodity - being available in volume from multiple vendors. Finally, the system interconnect is constructed using off-the-shelf connectors and backplane technology.

Realizing the performance of a streaming supercomputer, of course, requires recoding applications in a streaming style - as *streams* of records passing through networks of arithmetic *kernels*. Coding applications in this style makes communication explicit, making it easy for the software tools to efficiently map the application to a streaming architecture.

1.3 Domain-specific languages simplify mapping problems to streaming supercomputers

We envision a three-level programmign system that will simplify the mapping of applications to a stream architecture and at the same time make the resulting code more portable. At the top level, several domain specific languages will target specific classes of applications, e.g., Monte-Carlo integration, ODEs, PDEs, etc.... Each of these languages will enable a scientist to describe their problem's equations, its geometry, constraints on its solution, and solution methods. A compiler then uses this description to map the problem to a streaming programming model. Domain specific languages have proven successful in many applications. In particular, graphics shading languages have been effective in describing complex shading calculations in terms of high-level primitives and mapping these calculations to a variety of hardware, including stream processors.

The target of the domain-specific language compiler is a stream language that describes the application in terms of streams of records passing through kernels of computation. We envision generalizing streams so they can describe not just linear sequences of records but also unordered collections of records, higher-dimensional arrays of records, and arbitrary graph structures (e.g., to describe a finite-element mesh). These collections will be operated on by kernels that *map* a function over the collection, *filter* the collection, selecting certain elements, *expand* a collection, producing several results for each input, or *reduce* a collection, combining several inputs into a smaller set - or even a single - result. By describing the application at an abstract stream level, this stream language will be completely hardware independent but yet will expose available parallelism and locality. A stream compiler will accept such a stream program and a machine description and generate output in our low-level programming language.

The output of the stream compiler is a program in a low-level stream language. In addition to streams, this language includes constructs to describe DSP and SIMD operations, threads and synchronization, and memory management. We anticipate writing several back-ends for the low-level stream compiler

[1] A GUPS (global updates per second) is the number of single word memory references to random locations across its entire memory space that a machine can support per second.

that will enable us to map programs not only to a streaming supercomputer, but also to conventional hardware.

1.4 Paper outline

The remainder of this paper describes our vision of a streaming supercomputer in more detail. We start by sketching the architecture of a streaming supercomputer in Section 2. Section 3 describes our vision of a three-level programming system in more detail. Applications and the domain-specific languages to describe them are discussed in Section 4. Finally, we outline a plan to accomplish this research in Section 5.

2 Architecture of a Streaming Supercomputer

In this section we sketch a possible architecture of a streaming superocomputer to demonstrate the feasibility of this approach. There are many details that remain to be worked out and many parameters and ratios may change by as much as a factor of two. However, the sketch here demonstrates the feasibility of a machine of the class we propose.

2.1 Overview

Figure 1 shows a block diagram of a streaming supercomputer. Each node contains a streaming processor with 64 1-GHz floating-point units (FPUs) and a local memory with 16 1Gb DRDRAM chips with a bandwidth of 2.4GB/s each[2]. The local memory capacity is 2GBytes and the local memory bandwidth is 38GB/s. Each node has a 20GB/s channel to the global interconnection network. Nodes can sustain simultaneous accesses to the memory of adjacent nodes at this rate - half the local memory bandwidth.

The global network enables any processor to access any memory location in the system. The network has a bisection bandwidth of $4N$GB/s, that is 4GB/s *per node*. Thus, each node can simultaneously sustain accesses to global memory at greater than 10% of the local memory bandwidth of the node. We expect that a global memory access in a N =16,384 node machine, including a round trip over the global network and remote memory access time will have a total latency of less than 500ns - 500 processor cycles.

To sustain full global bandwidth - 4GB/s or one word every two 1ns cycles - while tolerating this latency, the processors make streaming memory references- stream loads and stream stores. A *stream load* operation loads a stream of records. The individual records may be addressed with unit-stride, arbitrary-stride, or indexed addressing modes. An indexed stream load gathers individual records (possibly as small as a single word) from arbitrary global locations. A single stream load can request thousands of multi-word records, more than enough to fill the 250-word deep memory pipeline. By fetching contiguous multi-word records, rather than individual words (like a vector load), stream loads result in more efficient access to modern memory chips.

We plan to package 16 nodes of the streaming supercomputer on a circuit card measuring 300mm by 400mm. This card will contain 16 processor chips, 256 DRAM chips, and will have a peak performance of 1TFLOPS[3]. Each cabinet will hold 64 of these cards, in 4 rows of 16, along with associated power supplies and cooling for a total of 1K nodes with 64TFLOPS and 2TBytes of memory per cabinet. Machines larger than 1K nodes are assembled by cabling cabinets together with optical fibers. We anticipate being able to scale the machine to 16K nodes (1PFLOPS) while maintaining our global latency and global to local bandwidth ratio.

The major properties of the streaming supercomputer we envision are summarized in Table 1.

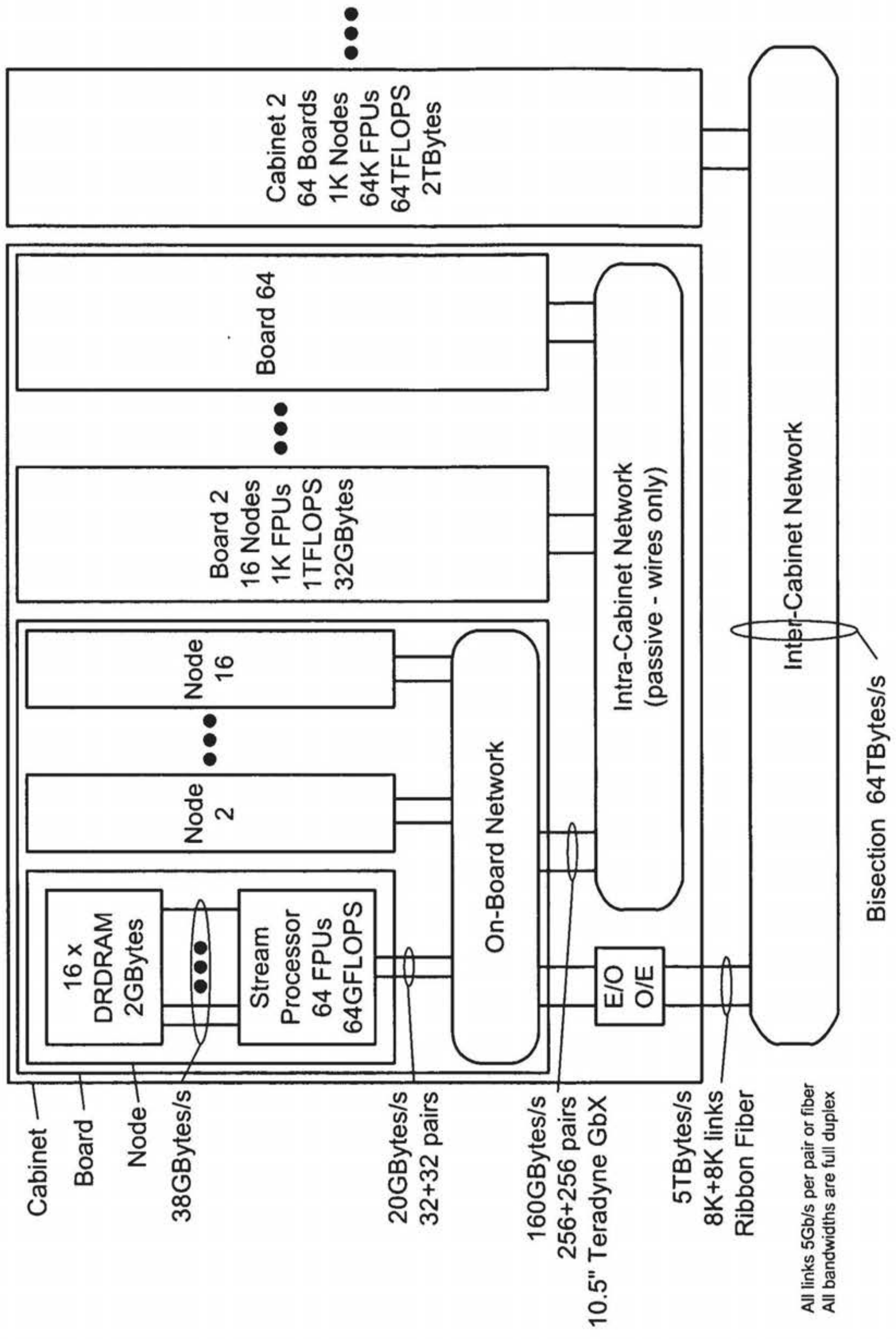


Figure 1: Block diagram of a streaming supercomputer

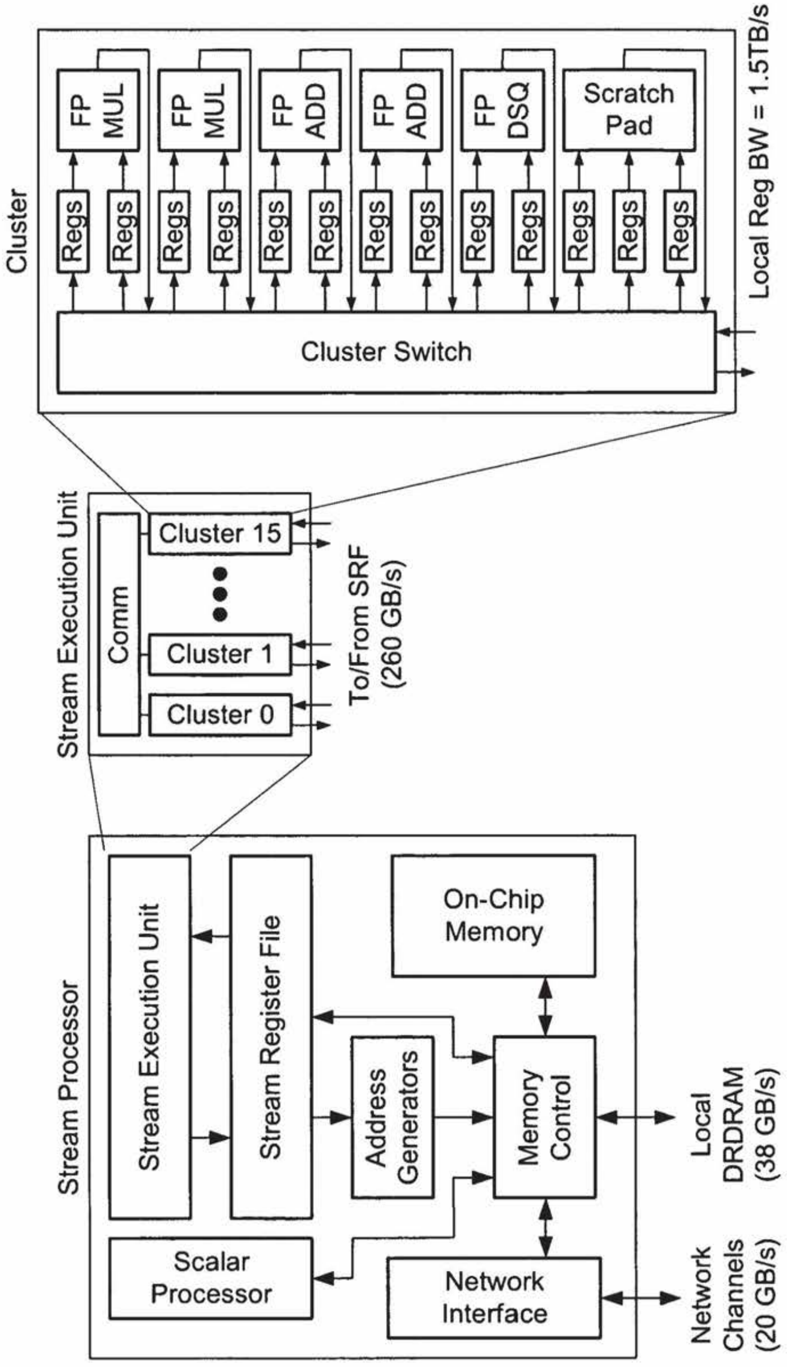


Figure 2: Block diagram of a streaming processor

Parameter	f(N)	N=4,096	N=16,384	Units
Memory Capacity	$2 \times 10^9 N$	2.8×10^{12}	3.3×10^{13}	Bytes
Local Memory BW	$3.8 \times 10^{10} N$	1.6×10^{14}	6.3×10^{14}	Bytes/sec
Global Memory BW	$3.8 \times 10^9 N$	1.6×10^{13}	6.3×10^{13}	Bytes/sec
Global Memory Accesses	$4.8 \times 10^8 N$	2.0×10^{12}	7.9×10^{12}	GUPS
Peak Arithmetic	$6.4 \times 10^{10} N$	2.6×10^{14}	1.0×10^{15}	FLOPS
Processor Chips	N	4,096	16,384	
Memory Chips	$16N$	6.6×10^4	2.6×10^5	
Boards	$N/16$	256	1,024	
Cabinets	$N/1,024$	4	16	
Power (est)	$50N$	2.0×10^5	8.2×10^5	Watts
Parts Cost (est)	$1 \times 10^3 N$	4×10^6	1.6×10^7	2001 Dollars

Table 1: Properties of proposed streaming supercomputer as a function of the number of nodes N, and for N =4,096 and N =16,384

2.2 Streaming Processor

Figure 2 shows a high-level view of the processor chip that provides the arithmetic capability of the streaming supercomputer. The core of the processor is a stream execution unit containing 64 64-bit floating-point units[4]. The stream execution unit also contains 4,096 64-bit *local* registers, 32 on each input of each arithmetic unit, and 8,192 64-bit *scratch-pad* registers for holding intermediate results of arithmetic kernels. The stream execution units read and write streams from a 32K word stream register file that stages stream data to and from memory. The stream execution unit is controlled by stream-operate instructions each of which causes a small subroutine to be executed on each element of the input streams to generate each element of the output streams.

A pair of address generators execute stream load and store instructions to transfer streams between the stream register file and the memory system. The local portion of the memory system is contained on the processor chip. This consists of the DRAM controllers, a network interface, and a cache memory (size to be determined). We plan to make the cache partitionable so some of this on-chip memory space can be used as an explicitly addressed staging memory.

The stream processor exploits a bandwidth hierarchy to efficiently keep such a large number of arithmetic units supplied with data and productively occupied. The levels of the hierarchy are listed in Table 2. For each level of the hierarchy, the table shows both the bandwidth in words/sec and the number of arithmetic operations per word of bandwidth at that level. The 64 arithmetic units in the stream execution unit each consume three 64-bit words of bandwidth each 1ns cycle for an aggregate per node bandwidth of 1.9×10^{11} 64-bit words/sec. The locality exposed by casting applications into kernels keeps most of this bandwidth local, so it can be provided inexpensively out of the local registers with the write bandwidth traversing small per-cluster switches. At the next level, the stream register file provides sufficient global register bandwidth so that one word can be read from each of these levels for every two arithmetic operations. Producer/consumer locality exposed within the stream model is exploited to capture most of inter-kernel bandwidth at this level. There are then three levels to the memory system, with the on-chip memory (staging memory and cache), local DRAM, and global DRAM providing progressively lower amounts of bandwidth.

Across the entire machine, this bandwidth hierarchy spans over two orders of magnitude. Experience with stream architectures on signal- and image-processing applications gives us confidence that the intra-kernel locality and inter-kernel producer-consumer locality will provide sufficient localization of

[2] By convention, lower case 'b' denotes 'bits' while upper case 'B' denotes Bytes.

[3] This board may also hold a number of network chips depending on whether the network is folded into the processor chips or not

[4] These are tentatively arranged as 16 clusters of two adders, two multipliers, and one divide square-root unit each - actually 80 units - but this is subject to change.

Level of hierarchy	BW (Words/s)	(ops/Word)
Local registers	1.9×10^{11}	0.33
Stream registers	3.2×10^{10}	2
On-chip memory	8.0×10^{9}	8
Local DRAM	4.8×10^{9}	13
Global DRAM	4.8×10^{8}	133

Table 2: Bandwidth hierarchy of a streaming supercomputer. Per-processor bandwidth at each level of the hierarchy.

data movement to match many important problems to this hierarchy.

A scalar execution unit executes scalar instructions and dispatches stream instructions to the stream processor and address generators. We plan to leverage an off-the-shelf core for this processor.

2.3 Memory system and network

The streaming supercomputer provides high-bandwidth access at single word granularity across a flat global address space that covers the entire memory of the machine. Memory is accessed via scalar load and store instructions and via stream load and store instructions. Stream load and store instructions hide latency by issuing a stream of memory operations with a single instruction. Each node's memory is implemented as 16 future DRAM chips with a bandwidth of 2.4GB/s each[5]. Memory on remote nodes is accessed via a high-bandwidth interconnection network.

To isolate processes running on the machine without causing performance issues historically associated with TLBs, all memory accesses are translated via a set of eight segment registers. Each segment register specifies the segment length, the subset of nodes over which the segment is mapped (to support space sharing), whether the segment is writeable, the interleave factor for the segment, and the caching options for that segment. Segments are restricted to be aligned in a manner that facilitates fast address formation.

The network employs a hierarchical topology, uses high-speed (5Gb/s per signal) signaling to give high global bandwidth and uses flit-reservation flow control to minimize memory latency. The network organization, sketched in Figure 1, matches the physical packaging hierarchy of the machine. The network is composed of *channels* connected by *routers.* Each channel consists of eight 5Gb/s differential signals giving it a raw bandwidth of 40Gb/s[6]. Messages are switched between channels by routers. There are four routers on each circuit card. Corresponding routers are connected together across the circuit cards to form four completely independent routing planes.

Each router connects to 28 bidirectional channels (eight signal pairs in each direction). Sixteen of the channels are *local* channels, eight of the channels are *backplane* channels, and the remaining four channels are *global* channels. One local channel is connected to each of the sixteen streaming processors on the circuit card. Processors on a circuit card can communicate directly with one another by traversing one router and two local channels and, using all four planes, have a raw bandwidth of 20GB/s over each of these connections. This permits processors to access the memory of other processors on the same circuit card with half the bandwidth that they can access their own memory. The local channels of the 64 circuit cards in a backplane are connected together in a backplane interconnection network (details remain to be worked out). This permits all nodes in a cabinet to sustain a usable bandwidth of 10GB/s each to random locations in the cabinet. Finally, the global channels are converted on the backplane to ribbon fibers and connected in a global interconnection network that permits all nodes in a system to sustain 4GB/s of global memory bandwidth each. Table 3 summarizes how this network tapers bandwidth as more distant memory is referenced.

[5] Rambus' roadmap indicates that DRDRAM chips will have this bandwidth in the appropriate timeframe.

[6] The usable bandwidth will be substantially less than this due to address and control overhead.

Level	Size (Bytes)	Bandwidth (Bytes/s)
Node	2.0×10^9	3.8×10^{10}
Circuit Card	3.2×10^{10}	2.0×10^{10}
Backplane	2.0×10^{12}	1.0×10^{10}
System (16 backplanes)	3.3×10^{13}	4.0×10^9

Table 3: Memory bandwidth vs. accessible memory size

To simplify the task of coordinating operation between the nodes, the network and memory system incorporate a set of synchronization mechanisms. Presence tags can be allocated for each record in memory to synchronize producers and consumers of data. The producing store (scalar or stream) sets the tag to a present state, a consuming load (scalar or stream) blocks until the tag is in this state. Atomic remote operations including fetch and (integer) add or compare and swap are also implemented by the memory controllers to permit common synchronization constructs to be implemented without traversing the network multiple times. More complex remote operations can be implemented using a memory-mapped message send that is received by a message handling thread on the remote scalar processor.

2.4 Input/Output and Mass Storage

I/O and mass storage are attached to the machine via four 12-wide (30Gb/s) infiniband I/O channels on each processor card. Off-the shelf disk arrays, network interfaces, and user input and output are expected to be available to interface with the infiniband network.

3 Programming Models

As mentioned previously, the streaming supercomputer achieves high performance because of two key ideas: data parallelism and arithmetic intensity. A streaming computation involves passing the records of streams through a network of *kernels*. For the problems we envision there are 10^8 to 10^{10} records (and potentially even more) providing large amounts of data parallelism. All calculation within a kernel are local to a processor as are streams that are passed between a pipeline of related kernels. This locality maps well to the bandwidth hierarchy of a streaming computer. Finally, data dependencies are explicitly managed through stream buffers. This prevents read-write hazards and allows data to be efficiently moved throughout the system.

A key challenge is to develop a programming environment for the streaming supercomputer. This programming environment should naturally reflect the capabilities of the machine, so programmers are encouraged to program in an efficient way. Thus, the programming environment must expose parallelism and make data dependencies explicit. It also must encourage local calculations with high compute to memory ratios.

Since a significant investment will need to be made in recoding algorithms for such computers, the programming environment must be carefully designed to be portable and to run on future hardware of this type. Low-level machine parameters that are expected to change over time should be hidden from the programmers and managed by the compiler.

Compiler technology is critical to the success of the programming environment. However, the compiler technology that is needed is quite different than the existing focus of parallel compiler research. The goal of our compiler is not to *discover* parallelism hidden in sequential codes. This has proved to be a difficult task in the past and limits the ultimate scalability of the system. We will assume the programming environment makes parallelism explicit. The goal of our compiler is to *map* the calculation onto the machine in the most efficient way. This is more in the spirit of code generation, a problem that has proved tractable.

To support the streaming supercomputer we envision a three-level system.

1. A low-level language close to the virtual machine that manages platform specific resource constraints. This low-level system would be analogous to UPC or StreaMIT.
2. A mid-level progamming language that supports data parallel calculations. This level would be analogous to the Connection Machine programming environment C*.
3. Finally, we envision a high-level, domain-specific set of languages that make it easy to support the development of specific applications. This would be analogous to the Stanford real-time shading language for programming photorealistic rendering effects.

All these languages would be based on C. There would be a single low-level and mid-level language. A metacompiler (or an extendable source-to-source compiler) would convert the mid-level to the low-level. We envision multiple high-level domain-specific languages. The same metacompiler infrastructure would be used to translate these domain-specific languages to the mid-level language.

In the following subsections we describe the properties of each of these languages in more detail.

3.1 Low-level language presents an abstract view of the hardware

The goal of the low-level language is to expose the features of future streaming computers to the programmer. We believe this is an entirely new class of machines that will have a long life-span. Just like C originally exposed the features of PDP-11 class minicomputers to programmers, our language exposes the features of streaming computers. However, streaming computers are different than current microprocessors, and need additional language support.

The design of the low-level language is incorporates ideas from several parallel programming environments, most notably the Imagine StreamC and KernelC languages, the CILK run-time environment, the StreaMIT language, and the Stanford DSP-C and Smart Memories Virtual Machine. Finally, this language would leverage the current efforts underway to design UPC.

This language is based on the communicating sequential processes (CSP) model of programming, but enhanced to support streams, thread management and scheduling, and memory placement and data management. We include with the language the run-time environment, partly just for convenience, but also because we see a tighter and tighter coupling between the parallel run-time environment and the hardware architecture.

The language and run-time environment must provide the following capabilities:

- Explicit support for streams.

  As in StreaMIT and Stream/Kernel C, streams will be explicitly declared and kernels explicitly identified. This makes all of the communication in the program explicit and exposes it to the metacompiler so it can be optimized.

- Support for modern DSP and SIMD instruction sets.

  This includes support for fixed point calculation and segmented instruction sets. The language should be able to express and generate code for current microprocessors such as the Intel Pentium family with SSE, and the AMD Athlon family with 3DNow; and for current programmable streaming graphics chips such as the NVIDIA GeForce3 with vertex programs.

- Support for threads and synchronization primitives.

  The language should also provide control over the scheduling of threads. It should be possible to express data-affinity; that is, schedule threads after the data has been prefetched or on the processor which has a local copy of the data. The thread scheduler should also be smart. For example, the CILK run-time system is designed to perform load-balancing by assigning tasks to threads and then running those threads.

- Support for memory management and communication primitives.

  Our inspiration for these features are the memory management primitives in UPC. Memory management includes partitioning between global shared memory and local memory. The language should also support different memory consistency models. It should also be possible to explicitly manage the cache, both by prefetching data and by segmenting the cache into subcaches. The language should also support the management of stream register files and stream buffers. It should be possible to express strided access, and to coordinate prefetched gathers with the execution of different kernels. This requires tight coupling between thread execution and data movement.

3.2 Mid-level collection oriented program expresses data parallelism

The goal of the mid-level programming language is to provide support for data parallelism. We believe almost all the major applications of high-performance computing are data-parallel; in particular, signal processing, image or media processing, scientific computing and database engines are data parallel.

Although data parallel computations may be (and are) written using a CSP or thread + communication programming model, we believe it is better to use a data parallel programming model. The data parallel programming model explicitly exposes parallelism and communication at a high-level. A metacompiler can than map this program onto a particular machine, relieving the programmer from dealing with particular machine parameters or limitations. This makes these programs significantly more portable, and hence long-living, encouraging programmers to rewrite their algorithms in this language.

The design of our mid-level data parallel programming environment is based on languages such as C* (and its Lisp counterpart Lisp*), the Connection Machine programming language. Other features are derived from signal, array and vector programming languages and libraries such as ZPL and VSIPL, as well as matlab and APL. Finally, we are motivated to include recent research on high-order functional languages such as Haskell and Scheme, and collection-oriented languages such as NESL.

The language will have the following features:

- Support for collections of records of various types.

  A record may be a primitive type such as a float, or a struct. Programmers will be encouraged to use records as the basic type. An example of a record might the state variables associated with a finite element mesh.

  Collections represent many records. We propose that there be at least three types of collections: sets, lists (or streams), and arrays. Sets are meant to capture the idea of an unordered collection. Certain parallel operations may take advantage of this unordered semantics. Lists or streams capture the idea of an ordered set. Lists and streams are meant to be processed in order. Finally, arrays capture the idea of random accessing or indexable calculations. Collections may have fixed or variable length.

  We may also want to support multidimensional arrays and a *graph* collection that can represent the connectivity of an arbitrary graph, e.g., a finite-element mesh. Later versions might also allow collections of collections (as is done in NESL).

- Kernel functions that represent operations on records.

  In some sense these kernel functions are the atomic operations in the language. Kernels take one or more records as input and produce one or more records (or as we will see a set of records) as output. Kernels represent entirely local calculations. However, unlike the Imagine KernelC programming model, kernels may contain loops and conditionals. Kernels may also have read-only or write-only access to global arrays. These arrays are declared as parameters to to kernel.

- High-level operators that apply kernels to collections.

  These operators would include:

  MAP: map applies a kernel to each element of a collection. For example, we could apply a transformation by a matrix to each vertex in a polygon mesh. Map is polymorphic across collection types, but respects the properties of the collection. For example, mapping a kernel onto an ordered collection will result in an ordered collection in the same order. However, mapping a kernel onto an unordered collection allows the order of the result to be different than the order of the input.

REDUCE: reduce applies a kernel to each element of a collection in the process producing a single result. Examples of reductions include max, sum, any, all, etc. Reduction operations may or may not be associative or commutative. Programmers are encouraged to use the least strict semantics. Reduce is often called SCAN or FOLD and corresponds to the Lisp* β operator.

EXPAND: expand creates a collection from a single record, or from a collection. For example, expand could be used by a rasterization kernel to produce a set of fragments from a triangle.

FILTER: filter produces a subset of a collection; the elements of the subset is determined by the value of a predicate kernel.

SCATTER, GATHER and PERMUTE: these operators rearrange collections.

In time, and as required, additional data parallel operations will be added. However, it has already been demonstrated that many important scientific applications may be written using these operators. In the final section on applications, we will discuss our research plan for different application areas.

Taking inspiration from modern functional programming languages such as Haskell, we would like to formally specify the semantics of these data parallel operators. For example, the expression MAP(f, MAP(g, c)) would be formally equivalent to MAP(f o g, c). That is, two consecutive maps involving f and g is equivalent to a single map of the composition of f and g. The reverse would also be possible: a complex function could be broken into two separate functions. This formal analysis would be very powerful. For example, by composing two functions we increase the arithmetic intensity, since both functions are executed although only a single read is performed. Formally breaking apart functions is also useful. Some implementations may want to break functions with global references into separate kernels with an intervening gather (as, for example, required by the Imagine architecture). Splitting functions might be useful for load balancing. Splitting functions creates more tasks, or potentially more uniform-sized tasks, which could be useful on future fine-grained, massively parallel machines. Reasoning about arithmetic intensity as machines evolve over time is one of the major research challenges for streaming computers.

One natural question is whether the mid-level and the low-level languages could be merged. Although that is possible, we prefer our proposed design because it allows us to leverage commodity compiler infrastructure. In particular, the low-level language is entirely responsible for code generation and of the additional capabilities that we need could be added through run-time libraries, as is traditionally done with dynamic memory management and threads libraries. In fact, it should be easy to develop a low-level language for existing machines, and hence most of our efforts would be on the mid-level language.

3.3 Domain-specific high-level languages map applications to collections

Finally, we envision building several high-level domain-specific languages for important application areas. These high-level languages will expose the capabilities of the streaming supercomputer to application programmers in an easy-to-use way, thus encouraging the adoption of the technology. We describe candidate areas for domain-specific languages in the section on applications.

Domain-specific languages have a long history in computer science. Our goal of providing this layer is motivated by the shading language that we have recently developed for programmable graphics hardware such as the NVIDIA GeForce3 or the ATI Radeon 8500. Shading languages have been developed by graphics researchers to describe the appearance of different objects, materials and environments. Shading languages have built-in functions for common operations, for example, to compute the light reflection from a matte surface using Lambert's Law. They also provide data types unique to that application, for example, vertices, fragments and textures.

Our shading language compiler maps this language onto the data parallel programming model. In terms of the primitives described in the last section, the compiler produces three kernels. A kernel to applied to each primitive, a kernel to be applied to each vertex, and a kernel to be applied to each fragment. The resulting data-parallel calculation is then the sequential execution of three MAPS, one for each collection. (Note in this case programmable graphics hardware handles automatically the conversion of one type of record to the other. However, in our new system, these conversions could be handled by the EXPAND operator.)

Besides providing a high-level programming environment for applications, we believe domain-specific languages will lead to very efficient implementations. For example, many applications require solving linear systems of equations. However, the types of matrices generated by the application varies. In some cases, asynchronous iteration may be used to solve for the unknowns. These leads to an efficient parallel algorithm since updates may occur out of order. Another advantage of application specific languages is that certain sections of code may be difficult to parallelize. By encapsulating them in highly optimized, built-in functions or libraries, these difficulties may be avoided. An example of this is in the shading language; execution paths that involve complex data dependencies such as clipping and rasterization are handled by built-in functions that use specially designed algorithms (and, in the case of graphics chips, hardware).

3.4 Metacompilation

A critical enabling component of the programming environment is the metacompiler. The metacompiler is an extensible compiler infrastructure that performs source to source translation. The same metacompiler will be used to map the high-level language to the mid-level as well as map the mid-level to the low-level language.

The metacompiler performs source-to-source translations. The first stage of the metacompiler is to read in the source language and build a suitable intermediate form. The last step is to translate the intermediate form back into the source language. The core of the metacompiler is extendible methods for analyzing and manipulating the intermediate form. In order to do this, we will build a *program transformation language*. This language will make it easy to match source patterns in the input, and to rewrite that part of the code. This part of the system will be based on the existing metacompiler framework developed by Dawson Engler as part of his research on for checking programs.

The most interesting metacompiler will be the one that transforms from the mid-level language to the low-level langauge. This compiler will also read in a machine description file. This machine description file will include key parameters of the machines, in particular, the computational and communication resources available in the machine (similar to the table presented in previous sections). The size of different parts of the memory hierarchy will also be available. The metacompiler will perform similar analysis to that performed by the Imagine StreamC compiler. It will allocate memory for collections, it will break up collections into smaller chunks to take advantage of producer-consumer locality, it will schedule data transfers from global memory to local stream register files, and it will schedule kernels when the data is available.

However, unlike the Imagine StreamC compiler this compiler will be retargetable. By changing the machine configuration file, future variants of our architecture can be used. It will also be possible to use our programming environment on current clusters and shared memory multiprocessors. Since even conventional shared memory and message-passing multiprocessors benefit from regular access patterns and locality, our system should run efficiently on these machines. That is, the resources of these machines would still be used efficiently, even though such machines would be much less cost-effective than our streaming supercomputer architecture. The ability to use existing machines as platforms will allow us to begin development of the programming environment before the architecture is complete.

4 Applications

In previous work, we have demonstrated that streaming architectures perform extremely well on media applications, include signal processing, image processing and graphics. The Imagine architecture yields an order of magnitude performance increase over conventional processors [?]. Even complex algorithms such as MPEG encoding, depth extraction through correlation, and the conventional graphics pipeline can be mapped onto streaming architectures. Graphics is a particular success story. Last year several vendors have introduced programmable graphics processors that resemble special-purpose stream processors. These graphics processors have created a major new generation of graphics chips with new capabilities. However, since these special-purpose streaming processors are not as general as our streaming computer, there is great interest in industry in more general streaming processor designs.

A major goal of this project is to extend our application domain from media processing to scientific computing. We will explore three major classes of scientific computing problems in roughly increasing degree of difficulty (that is, difficulty for efficient adaption into the streaming pipeline): Monte Carlo (MC) algorithms, ordinary differential equations (ODE), and partial differential equations (PDE). Numerical techniques for these types of problems underlie many important application areas. For example, Monte Carlo algorithms are regularly used in radiation transport and solid state physics, ODEs are used in molecular dynamics, astrophysics and rigid body dynamics, and PDEs are necessary in mechanical and structural design as well as fluid flow. Since simulating complex multi-physics processes is a major goal of the ASCI University Program, we are particularly interested in multi-physics applications, such as turbomachinary simulation which couples combustion with fluid flow and compressor turbine motion.

Some of these applications are embarassingly parallel, and we expect it will be easy to map them onto streaming computers. In fact, embarassingly parallel applications run well on current clusters, since they involve very little communication. However, even in these cases it is worthwhile to map them onto the stream programming model, since this will allow these applications to be run on much more cost-effective and scalable machines.

Other applications are more challenging to map onto parallel computers. These typically involve extensive global communication. These applications do not run well on existing clusters because of their limited global memory bandwidth. Our hypothesis is that they will run much better on the streaming computer because of its high-bandwidth interconnection network and memory system. However, these algorithms will need to be carefully mapped onto the stream programming model, and this will involve close interaction with colleagues in scientific computing. Successfully mapping any of these applications onto the streaming supercomputer could potentially open up whole new areas of computational science.

4.1 Monte-Carlo Radiation Transport

The simplest scientific computing problem that we will tackle is Monte Carlo integration, in particular, Monte Carlo simulation of transport equations. The key application of this technique is radiation transport, which is important in heat transfer and the design of nuclear devices. Monte Carlo of course has many other applications; for example, Markov Chain Monte Carlo or the Metropolis Algorithm is widely user in Bayesian inference, which is a major method used in statistical computing and artificial intelligence.

The basic Monte Carlo algorithm is particle tracing. Particles are created in certain states according to a source distribution functions. These particles make transitions to other states using a scattering distribution function. Finally, particles are terminated according to absorption distribution function. In classic Monte Carlo, each particle or sample is independent of the others and thus the algorithm is easily parallelized. Further, the inner loops involve generating random variables according to probability distribution functions. Although in toy problems these distribution functions are simple (e.g. uniform), in physical simulation they can be quite complex. Thus, not only are the calculations local, but they have high arithmetic intensity.

Monte Carlo is complicated if the model cannot be replicated. For example, when doing radiative transport in complex geometries, the geometric database may be very large and not easily replicated. Also, the interactions of particles with the database is not localized, since particles may be in different parts of the environment. In related work, we have shown how ray tracing may be mapped onto a streaming architecture. The key idea is to formulate the problem of finding a ray-surface intersection as a streaming calculation. Thus, we believe even in this case, we will be able to map radiation transport onto a streaming computer.

4.2 Ordinary Differential Equations

A more complicated problem is the solution of ordinary differential equations, in particular, coupled differential equations. Classic applications of these techniques include molecular dynamics and astrophysics, or N-body problems, which involve pairwise forces between particles. Other applications include chemical kinetics which involves solving chemical rate equations for the concentrations of different species. Another example is rigid body dynamics of articulated figures, or robotics.

As a first example consider chemical reactions as might take place in combustion. Sometimes an overall time step is determined in order to gurantee stability and accuracy, but many times Godonuv (or Strang) time splitting is used to separate the chemical kinetics update from the fluid dynamics. While the fluid dynamics is "frozen", one has to solve a possibly stiff system of coupled ode's in each element of the computational mesh. One can easily have tens of species and hundreds of governing reactions, but these are usually simplified as much as possible employing a reduced chemical mechanism in order to defray the computational cost. These reduced mechanisms can be based on asymptotic theory or experiments and are currently an area of active research (notable methods include using reduced manifolds [Maas and Pope]), especially since they are not generally robust and many times need to be adjusted on a case by case basis. These type of computations are ideal for a streaming computer where one thrives with the full reaction mechanism with a high arithmetic cost per node. Hopefully, our new design will alleviate some of the need for reduced mechanisms allowing full mechanisms to be employed more often.

Another important example is solving equations of motion of large particle systems. This can be done in linear time using the fast multipole method developed by Greengard and Rocklin. The idea of this method is to approximate the field in a cell with a k-term multipole expansion. The field is then propagated up a hierarchy by translating and scaling the coefficients of the expansion. Computationally, this is a linear transformation on the coefficients and may be performed by a matrix-vector multiply. This field is pulled up the hierarchy in this way, and then pushed back down to the leaves. Again, this calculation may be easily mapped onto a streaming computer, and in fact has been efficiently implemented on the connection machine and other data parallel machines.

As a final example, in molecular dynamics a simpler method is often used. Since molecular forces fall off faster than $1/r^2$, it is typical to only compute forces amongst some small set of neighbors. Streams would need to access their neighbors with a GATHER, and then compute forces. But again, since these force terms are reasonably complicated, this would have high arithmetic intensity. Of course, there are still many details to work out, such as how to update the spatial data structure as particles move.

Problems of this type would benefit from high-level languages. In particular, Sussman and Wisdom have recently developed a high-level language based on Scheme for classical mechanics. They are able to specify the Lagrangian of a system from that declaration automatically derive the equations of motion. They then map the equations of motion onto a set of solvers. A similar approach could be used for coupled ODEs and highly parallel systems of ODEs. In addition to choosing methods for solving the ODEs, this new system could automatically parallelize the application.

4.3 Partial Differential Equations

The more complex problem domain we intend to study is methods for solving partial differential equations including finite volume, finite element and finite difference methods. Example applications include elastic and inelastic deformations of solids, and fluid flow, and fluid-solid coupling problems similar to those at the Caltech and Illinois ASCI centers. We further subdivide these techniques into Lagrangian (where the mesh is attached to a local reference frame moving with the material) and Eulerian (where the reference frame is global and the material moves between mesh elements). Of course, ALE schemes (where the mesh has and arbitrary velocity in between the Eulerian and Lagrangian mesh) are also of interest, especially at interfaces, but we intend to first take the approach of the Caltech ASCI center (and researchers such as Noh) and couple Eulerian and Lagrangian schemes directly at a fluid solid interface.

There are two major classes initial-value partial differential equations: Hyperbolic and parabolic equations (here we classify elliptic equations as boundary-value problems). However, for our purposes we will subdivide the problems into those that are *stiff*, and those that are not. Stiffness implies that some part of the problem requires much smaller steps than others in order to guarantee stability, and that this smaller time step is not needed to obtain the desired accuracy. Practically, this means that in stiff systems *implicit* methods are used for efficiency reasons. Implicit methods involve solving a matrix equation - for example a pressure solver in low Mach number or incompressible flow - and are similar (for our purposes) to elliptic partial differential equations. If implicit methods are not needed, then *explicit* methods, which only require local neighborhood computation, can be used.

Explicit methods may be mapped efficiently onto parallel computers using domain decomposition. A typical time step in an explicit method requires a small neighborhood around a position in space. These neighborhoods are required in order to estimate spatial derivatives. So each step involves fetching information from your neighbors, and then updating your value. In many PDEs, the calculation involved is minimal and only involves a few arithmetic operations. Thus, the communication- to-compute requirements are large, although this ratio reduces as one applies more complex numerical methods, e.g. nonlinear limiters to treat discontinuous phenonmena or special algorithms for interface treatment. But, fortunately, there is an easy solution to this case: domain decomposition. By choosing as the unit of computation a region of space, then neighbors need only be communicated at the boundary of the domain. Since the boundary grows as n^2 and the interior grows as n^3 (for 3D problems), the ratio of communication to bandwidth decreases as larger domains as chosen.

Domain decomposition works well even on clusters, although the domains need to be large. The downside is that if the domains become too large, than there are fewer independent tasks. As the number of processors becomes large there may not be enough tasks for each processor, of if there is variability in the amount of work between tasks, load balancing problems may arise. Choosing the right domain size is a well-studied problem, and we again note that the situation improves when using more complex algorthims, e.g. those that are higher order accurate or those needed for the treatment of interfaces and discontinuities. We should be able to use these algorithms for the streaming computer. Again, this could be done by the compiler of a higher-level language.

Methods that involve implicit techniques are more complex. Fundamentally they involve solving linear systems of equations at each time step. This linear system of equations is usually sparse, which means that it can have a very irregular memory access pattern. A typical method involves first applying a preconditioner to the matrix and then using an iterative algorithm such as the conjugate gradient algorithm. In some cases, e.g., incompressible fluid flow, this matrix solve is the most expensive part of the calculation, consuming as much as 90% of the CPU time. Another approach to solving the matrix equation arising from this approach is to use a multigrid algorithm.

Mapping this matrix solve onto a streaming computer is a major research challenge. We should do much better than traditional clusters and shared memory multicomputers because we have greater global bandwidth and can tolerate the latency of irregular accesses. However, we believe we can do better. We will work with the numerical analysts at Stanford to develop new algorithms that map well to streaming computers. This will probably lead to a family of algorithms, since the algorithm of choice will depend on the application domain.

5 Research Plan

There are many challenging problems that must be solved to realize our vision of a streaming supercomputer. Many architecture issues must be resolved, there are many unknowns in the design of a layered stream programming system, and methods for most efficiently mapping our target applications to the stream model remain to be discovered.

To address these problems, we propose a six year research effort that proceeds through phases of *exploration*, *refinement*, and, if the early phases are successful *prototyping*. The overall program is illustrated in Figure 3. Our effort has three main threads: *architecture*, *programming systems*, and *applications*. The threads are tightly coupled with the results of one thread being used by and influencing the other threads.

During the first two years of the program we focus on solving the fundamental issues involved with architecture (e.g., network topologies, the interaction of streaming and cache coherence, and control mechanisms for streaming processors), streaming programming systems (e.g., collection-oriented languages, optimization methods, and program transformation technology), and applications (e.g., numerical methods that are best suited for streaming, and mapping techniques). At the end of this period we expect to be able to run simple Monte-Carlo radiation transport and ODE applications, compiled using our three-layer software system, on our architectural simulator.

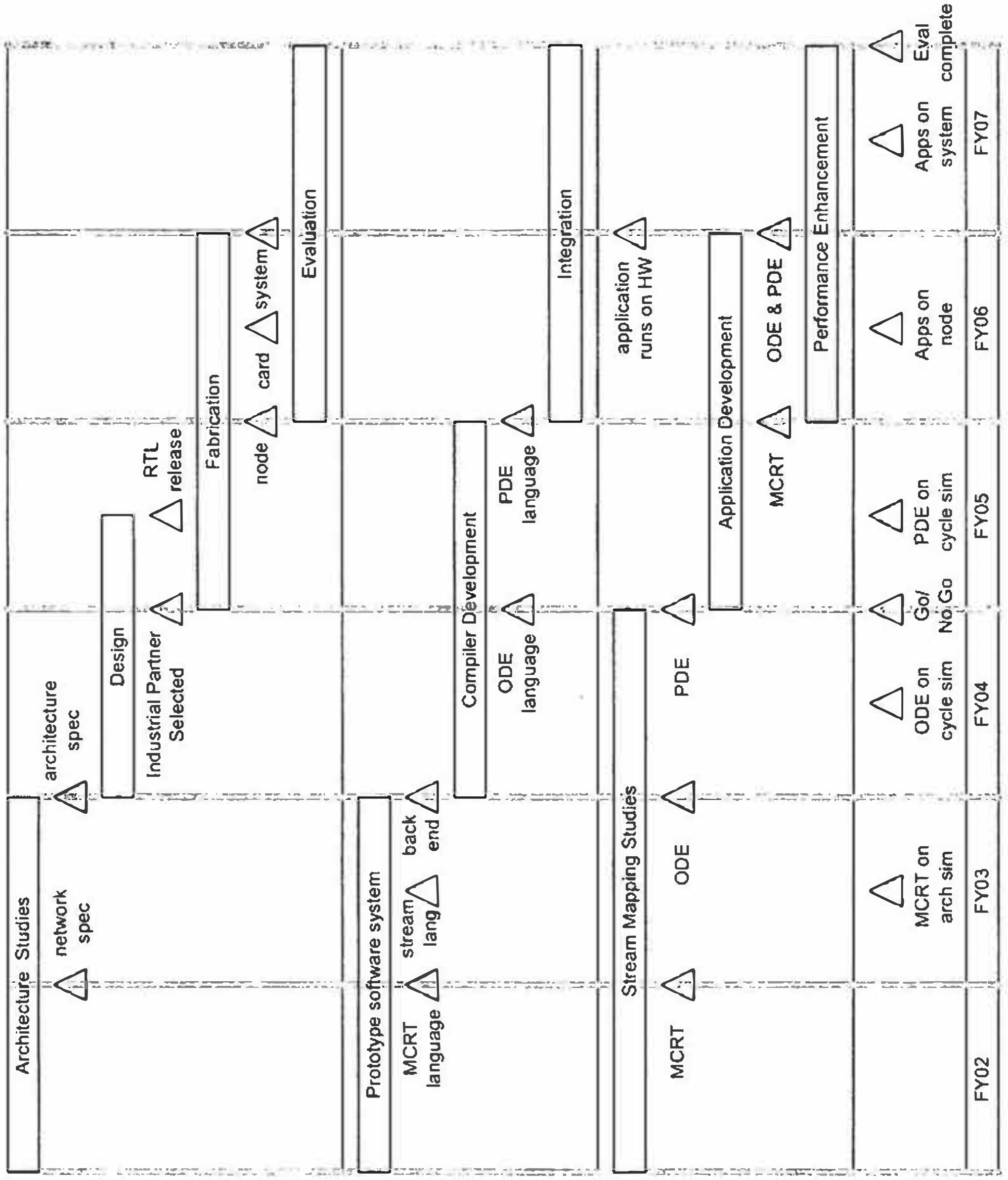


Figure 3: Plan for streaming supercomputer research and development

The next 18 months are a refinement period during which feedback from early simulation experiments will be used to guide the development of the architecture and programming system. During this period, the architecture will be reduced to a more detailed design and the programming system will be expanded. Toward the end of this refinement period we plan to make a go/no-go decision as to whether to continue the streaming supercomputer program into the prototyping phase or not. This decision will be based on whether our experiments suggest we can meet our cost-performance goals for the system and whether our architecture and programming system can yield high sustained performance on meaningful applications.

If the decision is made to proceed, an industrial partner will be selected to take on most of the detailed design, physical design, and fabrication of the streaming supercomputer. At Stanford we will work with our industrial partner on these hardware tasks. At the same time we will be increasing the range, size, and sophistication of applications that our programming system can handle. Our goal is to evolve our system to the point where we can handle the combined turbomachinery/combustion code.

The machine will be brought up in stages: first a single streaming processor node with local memory - no network, then a 16-processor card, and then systems of increasing size. As we bring up the hardware, we will integrate our programming system and evaluate the hardware on our application suite. The final period of the program is devoted to evaluation of the machine to learn what works and what doesn't and to discover how streaming supercomputers should be built in the future.

Key personnel

- Bill Dally
- Pat Hanrahan
- Ron Fedkiw
- Dawson Engler

[Do we want to add Mendel? Mark?]

References

PATENT APPLICATION FEE DETERMINATION RECORD

Effective October 1, 2003

Application or Docket Number
10869200

CLAIMS AS FILED - PART I

	(Column 1)	(Column 2)
TOTAL CLAIMS	24	
FOR	NUMBER FILED	NUMBER EXTRA
TOTAL CHARGEABLE CLAIMS	24 minus 20=	* 4
INDEPENDENT CLAIMS	4 minus 3 =	* 1
MULTIPLE DEPENDENT CLAIM PRESENT		□

* If the difference in column 1 is less than zero, enter "0" in column 2

SMALL ENTITY TYPE □ OR OTHER THAN SMALL ENTITY

RATE	FEE		RATE	FEE
BASIC FEE	385.00	OR	BASIC FEE	770.00
XS 9=		OR	X$18=	72
X43=		OR	X86=	86
+145=		OR	+290=	—
TOTAL		OR	TOTAL	928

CLAIMS AS AMENDED - PART II

4-1-05

AMENDMENT A

	(Column 1) CLAIMS REMAINING AFTER AMENDMENT		(Column 2) HIGHEST NUMBER PREVIOUSLY PAID FOR	(Column 3) PRESENT EXTRA
Total	* 24	Minus	** 24	=
Independent	* 4	Minus	*** 4	=
FIRST PRESENTATION OF MULTIPLE DEPENDENT CLAIM				□

SMALL ENTITY OR OTHER THAN SMALL ENTITY

RATE	ADDITIONAL FEE		RATE	ADDITIONAL FEE
X$ 9=		OR	X$18=	
X43=		OR	X86=	
+145=		OR	+290=	
TOTAL ADDIT. FEE		OR	TOTAL ADDIT. FEE	

AMENDMENT B

	(Column 1) CLAIMS REMAINING AFTER AMENDMENT		(Column 2) HIGHEST NUMBER PREVIOUSLY PAID FOR	(Column 3) PRESENT EXTRA
Total	*	Minus	**	=
Independent	*	Minus	***	=
FIRST PRESENTATION OF MULTIPLE DEPENDENT CLAIM				□

RATE	ADDITIONAL FEE		RATE	ADDITIONAL FEE
X$ 9=		OR	X$18=	
X43=		OR	X86=	
+145=		OR	+290=	
TOTAL ADDIT. FEE		OR	TOTAL ADDIT. FEE	

AMENDMENT C

	(Column 1) CLAIMS REMAINING AFTER AMENDMENT		(Column 2) HIGHEST NUMBER PREVIOUSLY PAID FOR	(Column 3) PRESENT EXTRA
Total	*	Minus	**	=
Independent	*	Minus	***	=
FIRST PRESENTATION OF MULTIPLE DEPENDENT CLAIM				□

RATE	ADDITIONAL FEE		RATE	ADDITIONAL FEE
X$ 9=		OR	X$18=	
X43=		OR	X86=	
+145=		OR	+290=	
TOTAL ADDIT. FEE		OR	TOTAL ADDIT. FEE	

* If the entry in column 1 is less than the entry in column 2, write "0" in column 3.
** If the "Highest Number Previously Paid For" IN THIS SPACE is less than 20, enter "20."
*** If the "Highest Number Previously Paid For" IN THIS SPACE is less than 3, enter "3."
The "Highest Number Previously Paid For" (Total or Independent) is the highest number found in the appropriate box in column 1.

Certificate of Transmission under 37 CFR 1.8

Serial No. 10/869,200

Application of: Daniel Poznanovic, David E. Caliga, and Jeffrey Hammes

Filed: June 16, 2004

Art Unit: 2186

Examiner: Thomas, Shane M.

Attorney Docket No. SRC028

For: SYSTEM AND METHOD OF ENHANCING EFFICIENCY AND UTILIZATION OF MEMORY BANDWIDTH IN RECONFIGURABLE HARDWARE

Confirmation No.: 5929

Customer No.: **25235**

I hereby certify that this correspondence is being facsimile transmitted to the United States Patent and Trademark Office

1. Information Disclosure Statement based on an International Search report.

on 6 - June 2005 3
Date No. of Pages
(incl. Coversheet)

to centralized fax number: 703-872-9306

Signature

Julie Lange
Typed or printed name of person signing Certificate

Note: Each paper must have its own certificate of transmission, or its certificate must identify each submitted paper.

Client Reference No. 80404.0033.001 Fax No. 719-448-5922

Attorney Docket No. SRC028
Client/Matter No. 80404.0033.001
Via Facsimile

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re Application of:

Daniel Poznanovic, David E. Caliga, Jeffrey Hammes

Serial No. 10/869,200

Filed: June 16, 2004

For: SYSTEM AND METHOD OF ENHANCING EFFICIENCY AND UTILIZATION OF MEMORY BANDWIDTH IN RECONFIGURABLE HARDWARE

Confirmation No.: 5929

Examiner: Thomas, Shane M.

Art Unit: 2186

INFORMATION DISCLOSURE STATEMENT BASED ON AN INTERNATIONAL SEARCH REPORT

MAIL STOP AMENDMENT
Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

Sir:

Pursuant to 37 C.F.R. § 1.97 the Examiner may wish to consider the references listed on the attached Form PTO/SB/08A. In submitting these references for the Examiner's consideration, no representation is made or implied that the references are or are not material to the examination of the application. The Examiner is encouraged to make his or her own determination of materiality.

Pursuant to 37 C.F.R. § 1.97(c), it is hereby certified that each item in this Information Disclosure Statement was cited in a communication from a foreign patent office (copy enclosed) in counterpart European application, PCT/US04/19663, mailed 31 MAY 2005, not more than three months prior to the filing of the statement (37 C.F.R. Section 1.97(e)). No petition fee is believed required, however, any fees associated with this communication may be made to Deposit Account No. 50-1123.

Date: 06 JUN 2005

Respectfully submitted,

William J. Kubida, Reg. No. 29,664
HOGAN & HARTSON
One Tabor Center
1200 17th Street, Suite 1500
Denver, Colorado 80202
(719) 448-5909 Tel
(303) 899-7333 Fax

Substitute for form 1449A/PTO			Application Number	10/869,200
INFORMATION DISCLOSURE STATEMENT BY APPLICANT			Filing Date	June 16, 2004
			First Named Inventor	Daniel Poznanovic et al.
			Art Unit	2186
(Use as many sheets as necessary)			Examiner Name	Thomas, Shane M.
Sheet	1	of 1	Attorney Docket No.	SRC028

U.S. PATENT DOCUMENTS

Examiner Initials	Cite No.[1]	Document No. No. – Kind Code[2]	Publication Date MM-DD-YYYY	Name of Patentee or Applicant of Cited Doc	Pages, Columns, Lines, Where Relevant Passages or Relevant Figures Appear
		US-2003/0084244 A1	05/01/2003	Paulraj	Entire Document
		US-2003/0046530 A1	03/06/2003	Poznanovic	
		US-			
		US-			
		US-			
		US-			
		US-			
		US-			
		US-			
		US-			
		US-			
		US-			

FOREIGN PATENT DOCUMENTS

Examiner Initials	Cite No.[1]	Foreign Patent Document Country Code[3] Number[4] Kind Code[5]	Publication Date MM-DD-YYYY	Name of Patentee or Applicant of Cited Doc	Pages, Columns, Lines Where Relevant Passages or Relevant Figures Appear	T[6]

EXAMINER SIGNATURE		**DATE CONSIDERED**	

EXAMINER: Initial if reference considered, whether or not citation is in conformance with MPEP 609. Draw line through citation if not in conformance and not considered. Include copy of this form with next communication to applicant. [1] Applicant's unique citation designation number (optional). [2] See Kinds Codes of USPTO Patent Documents at www.uspto.gov or MPEP 901.04. [3] Enter Office that issued the document, by the two-letter code (WIPO Standard ST.3). [4] For Japanese patent documents, the indication of the year of the reign of the Emperor must precede the serial number of the patent document. [5] Kind of document by the appropriate symbols as indicated on the document under WIPO Standard ST. 16 if possible. 6 Applicant is to place a check mark here if English language Translation is attached.

This collection of information is required by 37 CFR 1.97 and 1.98. The information is required to obtain or retain a benefit by the public which is to file (and by the USPTO to process) and application. Confidentiality is governed by 35 U.S.C. 122 and 37 CFR 1.14. This collection is estimated to take 2 hours to complete, including gathering, preparing, and submitting the completed application form to the USPTO. Time will vary depending upon the individual case. Any comments on the amount of time you require to complete this form and/or suggestions for reducing this burden, should be sent to the Chief information Officer, U.S. Patent and Trademark Office, U.S. Department of Commerce, P.O. Box 1450, Alexandria, VA 22313-1450. DO NOT SEND FEES OR COMPLETED FORMS TO THIS ADDRESS. SEND TO: Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-1450.

Certificate of Transmission under 37 CFR 1.8

Serial No. 10/869,200

Application of: Daniel Poznanovic, David E. Caliga, and Jeffrey Hammes

Filed: June 16, 2004

Art Unit: 2186

Examiner: Thomas, Shane M.

Attorney Docket No. SRC028

For: SYSTEM AND METHOD OF ENHANCING EFFICIENCY AND UTILIZATION OF MEMORY BANDWIDTH IN RECONFIGURABLE HARDWARE

Confirmation No.: 5929

Customer No.: **25235**

I hereby certify that this correspondence is being facsimile transmitted to the United States Patent and Trademark Office

1. Information Disclosure Statement based on an International Search report.

on 6 June 2005
Date

13
No. of Pages
(incl. Coversheet)

to centralized fax number: 703-872-9306

Signature

Julie Lange
Typed or printed name of person signing Certificate

Note: Each paper must have its own certificate of transmission, or its certificate must identify each submitted paper.

Client Reference No. 80404.0033.001 Fax No. 719-448-5922

Attorney Docket No. SRC028
Client/Matter No. 80404.0033.001
Via Facsimile

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re Application of:
Daniel Poznanovic, David E. Caliga, Jeffrey Hammes
Serial No. 10/869,200
Filed: June 16, 2004
For: SYSTEM AND METHOD OF ENHANCING EFFICIENCY AND UTILIZATION OF MEMORY BANDWIDTH IN RECONFIGURABLE HARDWARE

Confirmation No.: 5929
Examiner: Thomas, Shane M.
Art Unit: 2186

INFORMATION DISCLOSURE STATEMENT BASED ON AN INTERNATIONAL SEARCH REPORT

MAIL STOP AMENDMENT
Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

Sir:

Pursuant to 37 C.F.R. § 1.97 the Examiner may wish to consider the references listed on the attached Form PTO/SB/08A. In submitting these references for the Examiner's consideration, no representation is made or implied that the references are or are not material to the examination of the application. The Examiner is encouraged to make his or her own determination of materiality.

Pursuant to 37 C.F.R. § 1.97(c), it is hereby certified that each item in this Information Disclosure Statement was cited in a communication from a foreign patent office (copy enclosed) in counterpart European application, PCT/US04/19663, mailed 31 MAY 2005, not more than three months prior to the filing of the statement (37 C.F.R. Section 1.97(e)). No petition fee is believed required, however, any fees associated with this communication may be made to Deposit Account No. 50-1123.

Date: 06 May 2005

Respectfully submitted,

William J. Kubida, Reg. No. 29,664
HOGAN & HARTSON
One Tabor Center
1200 17th Street, Suite 1500
Denver, Colorado 80202
(719) 448-5909 Tel
(303) 899-7333 Fax

PTO/SB/08a(08/03)
Approved for use through 07/31/2006. OMB 0651-0031
Patent and Trademark Office; U.S. DEPARTMENT OF COMMERCE
Under the Paperwork Reduction Act of 1995, no persons are required to respond to a collection of information unless it displays a valid. OMB control number.

Substitute for form 1449A/PTO				Application Number	10/869,200
				Filing Date	June 16, 2004
INFORMATION DISCLOSURE STATEMENT BY APPLICANT				First Named Inventor	Daniel Poznanovic et al.
				Art Unit	2186
(Use as many sheets as necessary)				Examiner Name	Thomas, Shane M.
Sheet	1	of	1	Attorney Docket No.	SRC028

U.S. PATENT DOCUMENTS

Examiner Initials	Cite No.[1]	Document No. No. – Kind Code[2]	Publication Date MM-DD-YYYY	Name of Patentee or Applicant of Cited Doc	Pages, Columns, Lines, Where Relevant Passages or Relevant Figures Appear
		US-2003/0084244 A1	05/01/2003	Paulraj	Entire Document
		US-2003/0046530 A1	03/06/2003	Poznanovic	
		US-			
		US-			
		US-			
		US-			
		US-			
		US-			
		US-			
		US-			
		US-			
		US-			

FOREIGN PATENT DOCUMENTS

Examiner Initials	Cite No.[1]	Foreign Patent Document	Publication Date MM-DD-YYYY	Name of Patentee or Applicant of Cited Doc	Pages, Columns, Lines Where Relevant Passages or Relevant Figures Appear	T[6]
		Country Code[3] Number[4] Kind Code[5]				

EXAMINER SIGNATURE		DATE CONSIDERED	

EXAMINER: Initial if reference considered, whether or not citation is in conformance with MPEP 609. Draw line through citation if not in conformance and not considered. Include copy of this form with next communication to applicant. [1] Applicant's unique citation designation number (optional). [2] See Kinds Codes of USPTO Patent Documents at www.uspto.gov or MPEP 901.04. [3] Enter Office that issued the document, by the two-letter code (WIPO Standard ST.3). [4] For Japanese patent documents, the indication of the year of the reign of the Emperor must precede the serial number of the patent document. [5] Kind of document by the appropriate symbols as indicated on the document under WIPO Standard ST. 16 if possible. 6 Applicant is to place a check mark here if English language Translation is attached.

This collection of information is required by 37 CFR 1.97 and 1.98. The information is required to obtain or retain a benefit by the public which is to file (and by the USPTO to process) and application. Confidentiality is governed by 35 U.S.C. 122 and 37 CFR 1.14. This collection is estimated to take 2 hours to complete, including gathering, preparing, and submitting the completed application form to the USPTO. Time will vary depending upon the individual case. Any comments on the amount of time you require to complete this form and/or suggestions for reducing this burden, should be sent to the Chief Information Officer, U.S. Patent and Trademark Office, U.S. Department of Commerce, P.O. Box 1450, Alexandria, VA 22313-1450. DO NOT SEND FEES OR COMPLETED FORMS TO THIS ADDRESS. SEND TO: Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-1450.

PATENT COOPERATION TREATY

From the INTERNATIONAL SEARCHING AUTHORITY

To:
CAROL W. BURTON
HOGAN & HARTSON, LLP
1200 17TH STREET, SUITE 1500
DENVER, CO 80202

PCT

NOTIFICATION OF TRANSMITTAL OF THE INTERNATIONAL SEARCH REPORT AND THE WRITTEN OPINION OF THE INTERNATIONAL SEARCHING AUTHORITY, OR THE DECLARATION

(PCT Rule 44.1)

	Date of mailing (*day/month/year*) 31 MAY 2005
Applicant's or agent's file reference SRC028 PCT	**FOR FURTHER ACTION** See paragraphs 1 and 4 below
International application No. PCT/US04/19663	International filing date (*day/month/year*) 17 June 2004 (17.06.2004)
Applicant SRC COMPUTERS, INC.	

1. ☒ The applicant is hereby notified that the international search report and the written opinion of the International Searching Authority have been established and are transmitted herewith.

Filing of amendments and statement under Article 19:
The applicant is entitled, if he so wishes, to amend the claims of the international application (see Rule 46):

When? The time limit for filing such amendments is normally two months from the date of transmittal of the international search report.

Where? Directly to the International Bureau of WIPO, 34 chemin des Colombettes
1211 Geneva 20, Switzerland, Facsimile No.: +41 22 740 14 35

For more detailed instructions, see the notes on the accompanying sheet.

2. □ The applicant is hereby notified that no international search report will be established and that the declaration under Article 17(2)(a) to that effect and the written opinion of the International Searching Authority are transmitted herewith.

3. □ **With regard to the protest against payment of (an) additional fee(s) under Rule 40.2, the applicant is notified that:**

□ the protest together with the decision thereon has been transmitted to the International Bureau together with the applicant's request to forward the texts of both the protest and the decision thereon to the designated Offices.

□ no decision has been made yet on the protest; the applicant will be notified as soon as a decision is made.

4. **Reminders**

Shortly after the expiration of 18 months from the priority date, the international application will be published by the International Bureau. If the applicant wishes to avoid or postpone publication, a notice of withdrawal of the international application, or of the priority claim, must reach the International Bureau as provided in Rules 90*bis*.1 and 90*bis*.3, respectively, before the completion of the technical preparations for international publication.

The applicant may submit comments on an informal basis on the written opinion of the International Searching Authority to the International Bureau. The International Bureau will send a copy of such comments to all designated Offices unless an international preliminary examination report has been or is to be established. These comments would also be made available to the public but not before the expiration of 30 months from the priority date.

Within 19 months from the priority date, but only in respect of some designated Offices, a demand for international preliminary examination must be filed if the applicant wishes to postpone the entry into the national phase until 30 months from the priority date (in some Offices even later); otherwise, the applicant must, within 20 months from the priority date, perform the prescribed acts for entry into the national phase before those designated Offices.

In respect of other designated Offices, the time limit of 30 months (or later) will apply even if no demand is filed within 19 months.

See the Annex to Form PCT/IB/301 and, for details about the applicable time limits, Office by Office, see the *PCT Applicant's Guide*, Volume II, National Chapters and the WIPO Internet site.

Name and mailing address of the ISA/ US Mail Stop PCT, Attn: ISA/US Commissioner for Patents P.O. Box 1450 Alexandria, Virginia 22313-1450 Facsimile No. (703) 305-3230	Authorized officer for Michelle R-Eason Vincent Trans Telephone No. (703)305-9750

Form PCT/ISA/220 (January 2004) (*See notes on accompanying sheet*)

PATENT COOPERATION TREATY

PCT

INTERNATIONAL SEARCH REPORT

(PCT Article 18 and Rules 43 and 44)

Applicant's or agent's file reference SRC028 PCT	**FOR FURTHER ACTION** see Form PCT/ISA/220 as well as, where applicable, item 5 below.	
International application No. PCT/US04/19663	International filing date (*day/month/year*) 17 June 2004 (17.06.2004)	(Earliest) Priority Date (*day/month/year*) 18 June 2003 (18.06.2003)
Applicant SRC COMPUTERS, INC.		

This international search report has been prepared by this International Searching Authority and is transmitted to the applicant according to Article 18. A copy is being transmitted to the International Bureau.

This international search report consists of a total of 3 sheets.

☒ It is also accompanied by a copy of each prior art document cited in this report.

1. **Basis of the Report**
   a. With regard to the language, the international search was carried out on the basis of the international application in the language in which it was filed, unless otherwise indicated under this item.
   ☐ The international search was carried out on the basis of a translation of the international application furnished to this Authority (Rule 23.1(b)).
   b. ☐ With regard to any nucleotide and/or amino acid sequence disclosed in the international application, see Box No. I.
2. ☐ Certain claims were found unsearchable (See Box No. II)
3. ☐ Unity of invention is lacking (See Box No. III)
4. With regard to the title,
   ☒ the text is approved as submitted by the applicant.
   ☐ the text has been established by this Authority to read as follows:
5. With regard to the abstract,
   ☒ the text is approved as submitted by the applicant.
   ☐ the text has been established, according to Rule 38.2(b), by this Authority as it appears in Box No. IV. The applicant may, within one month from the date of mailing of this international search report, submit comments to this Authority.
6. With regard to the drawings,
   a. the figure of the drawings to be published with the abstract is Figure No. ____
   ☐ as suggested by the applicant.
   ☐ as selected by this Authority, because the applicant failed to suggest a figure.
   ☐ as selected by this Authority, because this figure better characterizes the invention.
   b. ☒ none of the figures is to be published with the abstract.

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US04/19663

A. CLASSIFICATION OF SUBJECT MATTER
IPC(7) : G06F 012/00
US CL : 711/137, 213
According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
U.S. : 711/137, 213. 170-173; 712/15

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)
Please See Continuation Sheet

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category *	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	US 2003/0084244 A1 (PAULRAJ) 01 May 2003 (01.05.2003), see entire document.	1-24
A	US 2003/0046530 A1 (POZNANOVIC) 06 March 2003 (06.03.2003).	1-24

□ Further documents are listed in the continuation of Box C. □ See patent family annex.

* Special categories of cited documents:

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier application or patent published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art

"&" document member of the same patent family

Date of the actual completion of the international search 28 April 2005 (28.04.2005)	Date of mailing of the international search report 31 MAY 2005
Name and mailing address of the ISA/US Mail Stop PCT, Attn: ISA/US Commissioner for Patents P.O. Box 1450 Alexandria, Virginia 22313-1450 Facsimile No. (703) 305-3230	Authorized officer for Michelle R. Vincent Trans Telephone No. (703)305-9750

Form PCT/ISA/210 (second sheet) (January 2004)

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US04/19663

Continuation of B. FIELDS SEARCHED Item 3:
EAST
microprocessor, reconfigurable

PATENT COOPERATION TREATY

From the
INTERNATIONAL SEARCHING AUTHORITY

To:
CAROL W. BURTON
HOGAN & HARTSON, LLP
1200 17TH STREET, SUITE 1500
DENVER, CO 80202

PCT

WRITTEN OPINION OF THE INTERNATIONAL SEARCHING AUTHORITY

(PCT Rule 43*bis*.1)

Date of mailing (*day/month/year*) 31 MAY 2005

Applicant's or agent's file reference SRC028 PCT	FOR FURTHER ACTION See paragraph 2 below	
International application No. PCT/US04/19663	International filing date (*day/month/year*) 17 June 2004 (17.06.2004)	Priority date (*day/month/year*) 18 June 2003 (18.06.2003)

International Patent Classification (IPC) or both national classification and IPC

IPC(7): G06F 012/00 and US Cl.: 711/137, 213

Applicant

SRC COMPUTERS, INC.

1. This opinion contains indications relating to the following items:

☒	Box No. I	Basis of the opinion
□	Box No. II	Priority
□	Box No. III	Non-establishment of opinion with regard to novelty, inventive step and industrial applicability
□	Box No. IV	Lack of unity of invention
☒	Box No. V	Reasoned statement under Rule 43*bis*.1(a)(i) with regard to novelty, inventive step or industrial applicability; citations and explanations supporting such statement
□	Box No. VI	Certain documents cited
□	Box No. VII	Certain defects in the international application
□	Box No. VIII	Certain observations on the international application

2. **FURTHER ACTION**

If a demand for international preliminary examination is made, this opinion will be considered to be a written opinion of the International Preliminary Examining Authority ("IPEA") except that this does not apply where the applicant chooses an Authority other than this one to be the IPEA and the chosen IPEA has notified the International Bureau under Rule 66.1*bis(b)* that written opinions of this International Searching Authority will not be so considered.

If this opinion is, as provided above, considered to be a written opinion of the IPEA, the applicant is invited to submit to the IPEA a written reply together, where appropriate, with amendments, before the expiration of 3 months from the date of mailing of Form PCT/ISA/220 or before the expiration of 22 months from the priority date, whichever expires later.

For further options, see Form PCT/ISA/220.

3. For further details, see notes to Form PCT/ISA/220.

Name and mailing address of the ISA/ US
Mail Stop PCT, Attn: ISA/US
Commissioner for Patents
P.O. Box 1450
Alexandria, Virginia 22313-1450
Facsimile No. (703) 305-3230

Authorized officer
for Michelle R. Eason
Vincent Trans
Telephone No. (703)305-9750

Form PCT/ISA/237 (cover sheet) (January 2004)

WRITTEN OPINION OF THE INTERNATIONAL SEARCHING AUTHORITY	International application No. PCT/US04/19663

Box No. I Basis of this opinion

1. With regard to the language, this opinion has been established on the basis of the international application in the language in which it was filed, unless otherwise indicated under this item.

   □ This opinion has been established on the basis of a translation from the original language into the following language ____, which is the language of a translation furnished for the purposes of international search (under Rules 12.3 and 23.1(b)).

2. With regard to any nucleotide and/or amino acid sequence disclosed in the international application and necessary to the claimed invention, this opinion has been established on the basis of:

   a. type of material

   □ a sequence listing

   □ table(s) related to the sequence listing

   b. format of material

   □ in written format

   □ in computer readable form

   c. time of filing/furnishing

   □ contained in international application as filed.

   □ filed together with the international application in computer readable form.

   □ furnished subsequently to this Authority for the purposes of search.

3. □ In addition, in the case that more than one version or copy of a sequence listing and/or table relating thereto has been filed or furnished, the required statements that the information in the subsequent or additional copies is identical to that in the application as filed or does not go beyond the application as filed, as appropriate, were furnished.

4. Additional comments:

WRITTEN OPINION OF THE INTERNATIONAL SEARCHING AUTHORITY

International application No.
PCT/US04/19663

Box No. V Reasoned statement under Rule 43 *bis*.1(a)(i) with regard to novelty, inventive step or industrial applicability; citations and explanations supporting such statement

1. Statement

Novelty (N)	Claims	NONE	YES
	Claims	1-24	NO
Inventive step (IS)	Claims	NONE	YES
	Claims	1-24	NO
Industrial applicability (IA)	Claims	1-24	YES
	Claims	NONE	NO

2. Citations and explanations:

Please See Continuation Sheet

WRITTEN OPINION OF THE INTERNATIONAL SEARCHING AUTHORITY

International application No.
PCT/US04/19663

Supplemental Box
In case the space in any of the preceding boxes is not sufficient.

V. 2. Citations and Explanations:
Claims 1-24 lack novelty under PCT Article 33(2) as being anticipated by Paulraj (US Pat. 2003/0084244).

As per claim 1, Paulraj shows a reconfigurable processor in figure 6 and a first memory (L1) having a first characteristc memory type (line size, blocking factor, associativity, etc.) and a second memory (L2) having a second characteristic memory type (line size, blocking factor, associativity, etc.). Refer to paragraph 23. Paulraj further teaches a functional unit 102 that executes applications using the memories L1 and L2 (paragraph 9). As is known in the art, a cache memory controller is often used to access and move data between a memory hierarchy. The Examiner is considering a data prefetch unit to be the logic assocatied with the moving, and only the moving, of data between the first and second memories (L1 and L2) since Paulraj shows a connection between the levels of cache in figure 6. This logic as well as the first and second memory types (L1 and L2) are configued by a program - refer to paragraphs 23-24. The data prefetch unit as defined by the Examiner must be configued as well by the program when moving data since the cache line size and blocking factor can change, so different amounts of data can be exchanged for the same access when different programs run.

As per claims 2 and 13, as taught in paragraphs 23 and 29 of Paulraj, no specific cache is present in the system of Paulraj. Rather, an FPGA is utilized as representing a caching hierarchy and is optimized based on the memory needs of a specific program running on the reconfigurable processor.

As per claims 3 and 14, Paulraj teaches in paragraph 23 that a specific cache line size of contiguous data is not retrieved since the data line size is optimized based on the memory needs of the program when executing on the reconfigurable processor. Refer also to paragraph 29.

As per claim 4, Paulraj teaches that a load/store unit is used to access the caches (L1-L3) in order to determine if cache data is present in the cache hierarchy (paragraph 6). Since the functional unit 102 (figure 6) is responsible for accessing the programmable memory unit 104, the Examiner is therefore considering the load/store unit logic of the programmable memory unit that is responsible for for accessing the L1 and L2 caches (first and second memory types) to be a memory controller. It can be seen that the memory controller, as defined by the Examiner, controls the transfer of data between the memory (assuming second memory L2) and the data prefetch unit, since the memory controller (load/store unit logic) is responsible for retrieving the data from the cache if a hit occurs (paragraph 4).

As per claim 5, as taught in paragraph 1, an external memory (element 18, figure 1) is generaly coupled to a microprocessor and holds data to be used by the microcontroller during program execution. The Examiner is considering the process of writing data back to the external memory from the FPGA memory 104 containing the caches (on-board memory), such as during a write-back scheme as known in the art, to be performed by the data prefetch unit portion of the functional logic as defined above by the Examiner. The data prefetch logic, as defined above, is responsible for all of the transfer of data into, out of, and between the FPGA memory 104.

As per claim 6, the Examiner is regarding a --register-- in its broadest reasonable sense and it thus considering it he to be a unit of logic. Therefore, the portion of the function logic that is responsible for the movement of data (as defined above to be the data prefetch unit) is being considered by the Examiner as containing a --register-- portion of the reconfigurable processor since, for

Form PCT/ISA/237 (Supplemental Box) (January 2004)

WRITTEN OPINION OF THE
INTERNATIONAL SEARCHING AUTHORITY

International application No.
PCT/US04/19663

Supplemental Box
In case the space in any of the preceding boxes is not sufficient.

instance, the blocking factor and line size of the programmable memory 112 can change, a --register-- or portion of the reconfigurable processor must be set in order to indicate the curret line size and blocking factor when a given application is being run on the reconfigurable processor at a given point in time. Refer to paragraph 23.

As per claim 7, the Examiner is considering the process of --disassembling the data prefetch unit-- as modifying the data prefetch unit logic of the function logic 102 every time the program being executed by the reconfigurable processor changes. It can be seen that the data prefetch unit changes during these intervals since the cache line size, blocking factor, and associativity of the FPGA changes when optimal for the next program to be executed (refer to paragraph 23). Thus it can be seen that the data prefetch unit logic is --disassembled-- when another program is executed by the reconfigurable processor of Paulraj.

As per claim 8, as can be seen that the FPGA memory 112, that comprises the first and second memories (L1 and L2) and which is accessed by the data prefetch unit of the functional unit 102 as discussed above, is a --processor memory-- (part of cpu 110). Therefore, since the data pretech unit can access the L2 cache as discussed above in the rejection of claim 1, the data prefetch unit can retrive data from the L2 portion of --processor memory--112.

As per claim 9, as shown in figure 1 and taught in paragraph 1 of Paulraj, the system 10 is actually a microprocessor, which contains a memory controller 14. The main difference between the prior art of figure 1 and the invention of Paulraj in figure 6 is that the memroy hierarchy is configurable and accessed by a functional unit in lieu of a separate memory controller logic (paragraph 9). Therefore, since the memory controller logic for accessing the cache hierarchy is still contained within cpu 110 of figure 6, it can be seen that the cpu 110 is actually a microprocessor. It follows that the --processor memory-- 112 is therefore a --microprocessor memory--.

As per claim 10, since the cpu 110 of figure 6 is a reconfigurable processer (able to reconfigure its memory heirarchy to match the needs of the application it is currently running), it can be seen that the cpu memory 112 is a reconfigurable processor memory.

As per claim 11, Paulraj depicts a reconfigurable hardware system in figure 6. Paulraj further teaches in paragraph 26 that when a particular application is to be run by the reconfigrable processor 110, a configuration vector is retrieved to program the programmable memory 112 (figure 6). As shown in figure 6, the step of accesing the configuration vector is executed outside of the reconfigurable processor 110. Therefore, the Examiner is considering the memory that contains the configuration vectors to be a --common memory-- and a data prefetch unit (reconfiguration unit 106 executing on the reconfigurable processor 110) accessing the common memory in order to determine how to program the memory 112 (paragraph 29). The data prefetch unit 106 is --configured-- by an application to be excuted on the sysem 110 since when a new application is to be executed, the data prefetch unit is called upon (or configured) to access the configuration vector for the particular application.

As per claim 12, the Examiner is considering a --memory controller-- to be the system portion utilized when creating a new configuration vector for an application. Such a process occurs in figure 5 and taught in paragraghs 23-25 of Paulraj. When a new configuration vector is created by analizing performance information that has been collected for the application. The Examiner is thereby considering the --memory controller-- to be the element of the reconfigurable hardware system that is associated with storing the new configuration vector into the common memory so that the vector can be accessed later when the same application is run again.

As per claim 15, the Examiner is considering the reconfiguration module 106 of the reconfigurable processsor 110, as comprising two distinct elements: a --computational unit-- and a --data access unit--. The data access unit is the element that is responsible for accessing the configuration vector as taught in paragraph 29 of Paulraj; or in other words, the Examiner is considering the --data access unit-- to be the same as the --memory controler-- defined in the rejection of claim 12. The Examiner is further considering the --computational unit-- of the reonfiguration module 106 to be the element that sets up the programmable memory module 104 using the configuration vector that was accessed by the --data access unit-- (paragraph 29).

As per claim 16, as taught by Paulraj in paragraph 29, the --data access unit-- supplies the configuration vector to the --computational unit-- in order to set up the programmable memory 104 as required by the application to be run on the reconfurable processor 110.

As per claim 17, the Examiner is considering a --data prefetch unit-- to be the reconfiguration unit 106 of reconfigurable processor 110 (figure 6). As taught in paragraph 26 and 29 of Paulraj, the --data prefetch unit-- accesses a memory in order to determine if a configuration vector is known for a given application, and if so, the vector is retrieved (from the memory). If this --data-- (configuration vector) is not known then a simulation is performed with the application in order to collect performance information. The Examiner is considering the element that executes and collects the performance data as being a --computational unit-- and the element of Paulraj that stores the configuration vector, once determined, to be a --data access unit-- since it stores the vector into the --memory-- from which it can be later retrieved (step 212 of figure 5). The --computational unit--, --data access unit--, and the --data prefetch unit-- are all --configured-- by a program (application) since (1) a new application configures the computational unit portion of the reconfiguration unit to perform a simulation in order to determine the optimal memory hierarchy organization; (2) the new application configures the --data access unit-- to store and retrieve (step 212) the configuration vector for that particular application; and (3) the --data prefetch unit-- is configured by the application to determine if a configuration file exists for the application and if so, the data prefetch unit is configured by the program the programmable memory 112 in order to optimize the programmable memory for that particular application.

As per claim 18, the --data-- (configuration vector) is transferred from the
--computational unit-- to the --data access unit-- when the configuration unit has created a configuration vector (step 208 of figure 5).
The --data-- is written to the memory --from-- the
--data prefetch unit-- since the data prefetch unit (reconfiguration unit 106) is the element that executed the beginning of the configuration vector creation process (step 200 of figure 5). Refer to paragraph 26. Thus the Examiner is considering the data as being written --from-- the data prefetch unit.

WRITTEN OPINION OF THE
INTERNATIONAL SEARCHING AUTHORITY

International application No.
PCT/US04/19663

Supplemental Box
In case the space in any of the preceding boxes is not sufficient.

As per claim 19, as taught in paragraph 26, if the configuration vector is known, the vector is retrieved from the memory to the data prefetch unit (reconfiguration unit 106). The data is read directly from the data prefetch unit when a request to create a configuration vector is made for a new application as shown in figure 6 since the data prefetch unit is responsible for being the vector creation process. The data is directed from the data prefetch unit (reconfigure logic) to be read from the memory by the data access unit to the computational unit where it is processed to produce a configuration vector.

As per claim 20, as stated above, the configuration vector (--data--) is created by the computational unit via acquired simulation data. The configuration vector is the resultant product that is transferred from the memory to the data prefect unit when it is determined that the configuration vector for the application is available (paragraph 26). Thus --all-- of the data that is transferred is processed by the computational unit (albeit before the transfer occurs) since the data prefetch unit required the entire configuration vector in order to set up the programmable memory 112.

As per claim 21. Paulraj shows in paragraph 26 that an explicit request for the configuration vector for the current application results in the data (if it exists) selected for the optimal configuration of the programmable memory 112 for that application.

As per claim 22, the Examiner is not considering the data (configuration vector) to be the size of a complete cache line since the data is used to create a cache hierarchy. In other words, the caches (L1-L3) of the programmable memory 112 are not programmed when the data is transferred from the memory to the data prefetch unit; therefore, the data cannot be a complete cache line.

As per claim 23, since the Examiner defined the portion of the reconfiguration unit that accesses the configuration file (data) from the memory, the Examiner is defining the logic that controls the actual transfer of that data to the data prefetch unit (portion of the reconfiguration unit that executes the fetch of the configuration vector and then programs the programmable memory 112) to be a --memory controller--. Thus the data access unit determines whether a configuration vector exists for an application and if so, the memory controller sends that data to the data prefetch unit.

As per claim 24, The Examiner is considering the element that executes and collects the performance data as being a --computational unit-- and the element of Paulraj that stores and retrieves the configuration vector, once determined, to be a --data access unit-- since it stores the vector into the --memory-- from which it can be later retrieved (step 212 of figure 5). The --computational unit-- and --data access unit -- are --configured-- by a program (application) since (1) a new application causes in the configuration of the computational unit portion of the reconfiguration unit to perform a simulation in order to determine the optimal memory hierarchy organization for the application and (2) the new application causes the configuration of the --data access unit-- to store and retrieve (step 212) the configuration vector for that particular application. Refer to paragraphs 25-27.

Claims 1-24 meet the criteria set out in PCT Article 33(4), and thus have industrial applicability because the subject matter claimed can be made or used in industry.

Ref #	Hits	Search Query	DBs	Default Operator	Plurals	Time Stamp
L1	117	reconfigur$3 adj (processor micro-processor CPU processor)	US-PGPUB; USPAT	OR	ON	2005/07/06 13:54
L3	143	reconfigur$3 adj (processor micro-processor CPU microprocessor)	US-PGPUB; USPAT	OR	ON	2005/07/06 13:54
L4	6	reconfigur$3 adj (processor micro-processor CPU microprocessor) and "711".clas.	US-PGPUB; USPAT	OR	ON	2005/07/06 13:54
L5	0	reconfigur$3 adj (processor micro-processor CPU microprocessor) and prefetch	US-PGPUB; USPAT	OR	ON	2005/07/06 13:54
L7	12	711/170-173.ccls. and dynamic near3 logic	US-PGPUB; USPAT	OR	ON	2005/07/06 13:54
L8	909	smc.as.	US-PGPUB; USPAT	OR	ON	2005/07/06 13:54
L9	0	smc.as. and "711".clas.	US-PGPUB; USPAT	OR	ON	2005/07/06 13:54
L10	0	smc.as. and "712".clas.	US-PGPUB; USPAT	OR	ON	2005/07/06 13:54
L11	0	(smc and computers) .as. and "712".clas.	US-PGPUB; USPAT	OR	ON	2005/07/06 13:54
L12	0	(smc and computers) .as.	US-PGPUB; USPAT	OR	ON	2005/07/06 13:54
L13	0	(smc and computers).as.	US-PGPUB; USPAT	OR	ON	2005/07/06 13:54
L14	10	(src and computers).as.	US-PGPUB; USPAT	OR	ON	2005/07/06 13:54
L15	87	711/170.ccls. and dynamic$4 near3 configur$5	US-PGPUB; USPAT	OR	ON	2005/07/06 13:54
L16	3	711/170.ccls. and dynamic$4 near3 configur$5 with cache	US-PGPUB; USPAT	OR	ON	2005/07/06 13:54
L18	260	reconfigurable adj processor	US-PGPUB; USPAT	OR	ON	2005/07/06 13:54
L19	4	"206189".ap.	US-PGPUB; USPAT	OR	ON	2005/07/06 13:54
L20	1	"5024031".pn.	US-PGPUB; USPAT	OR	ON	2005/07/06 13:54
L21	6	"869200".ap.	US-PGPUB; USPAT	OR	ON	2005/07/06 13:54
L22	1694	711/170.ccls.	US-PGPUB; USPAT	OR	ON	2005/07/06 13:54
L23	449	711/170.ccls. and (reconfigur$5 rearrang$4 application adj specific)	US-PGPUB; USPAT	OR	ON	2005/07/06 13:54

L24	102	711/170.ccls. and matrix	US-PGPUB; USPAT	OR	ON	2005/07/06 13:54
L25	58	711/170.ccls. and fpga	US-PGPUB; USPAT	OR	ON	2005/07/06 13:54
L26	197	712/15.ccls.	US-PGPUB; USPAT	OR	ON	2005/07/06 13:54
L27	276	711/170.ccls. and (application near2 specific application-specific)	US-PGPUB; USPAT	OR	ON	2005/07/06 13:54
L28	260	reconfigurable adj processor	US-PGPUB; USPAT	OR	ON	2005/07/06 13:54
L29	179	L28 and fpga	US-PGPUB; USPAT	OR	ON	2005/07/06 13:54
L30	9	L29 and memory with reconfiguring	US-PGPUB; USPAT	OR	ON	2005/07/06 13:54
L31	43	711/170.ccls. and ((reconfigur$5 rearrang$4) and application adj specific)	US-PGPUB; USPAT	OR	ON	2005/07/06 13:54
L32	58	711/170.ccls. and FPGA	US-PGPUB; USPAT	OR	ON	2005/07/06 13:54
L33	251	711/170.ccls. and reconfig$7	US-PGPUB; USPAT	OR	ON	2005/07/06 13:54
L34	1	"6779131".pn.	US-PGPUB; USPAT	OR	ON	2005/07/06 13:54
L35	0	("6779131").URPN.	USPAT	OR	ON	2005/07/06 13:54
L36	9	("5892896" | "6060339" | "6081463" | "6154851" | "6204562" | "6363502" | "6405324" | "6483755" | "6530005").PN.	US-PGPUB; USPAT; USOCR	OR	ON	2005/07/06 13:54
L37	16	direct adj execution adj logic	US-PGPUB; USPAT; USOCR	OR	ON	2005/07/06 13:54
L38	4	711/170.ccls. and programmable adj logic adj blocks	US-PGPUB; USPAT; USOCR	OR	ON	2005/07/06 13:54
L39	6	"869200".ap.	US-PGPUB; USPAT; USOCR	OR	ON	2005/07/06 13:54
L40	4	711/171-172.ccls. and FPGA	US-PGPUB; USPAT	OR	ON	2005/07/06 13:54
L41	58	711/170.ccls. and FPGA	US-PGPUB; USPAT	OR	ON	2005/07/06 13:54
L42	58	711/170.ccls. and FPGA	US-PGPUB; USPAT; USOCR	OR	ON	2005/07/06 13:54

L43	4	711/171-172.ccls. and FPGA	US-PGPUB;
			USPAT;
			USOCR
				OR	ON	2005/07/06 13:54

L44	9	711/173.ccls. and FPGA	US-PGPUB;
			USPAT;
			USOCR
				OR	ON	2005/07/06 13:54

L45	88	711/170.ccls. and reprogram$5	US-PGPUB; USPAT	OR	ON	2005/07/06 13:54
L46	81	711/171-172.ccls. and (reprogram$5 reconfig$6)	US-PGPUB; USPAT	OR	ON	2005/07/06 13:54
L47	72	L46 not L45	US-PGPUB; USPAT	OR	ON	2005/07/06 13:54
L48	402	711/170-172.ccls. and ((configur$5).ti. (configur$6).ab.)	US-PGPUB; USPAT	OR	ON	2005/07/06 13:54
L49	16	711/170-172.ccls. and ((configur$5).ti. (configur$6).ab.) and prefetch	US-PGPUB; USPAT	OR	ON	2005/07/06 13:54
L50	90	711/170-172.ccls. and ((configur$5).ti. (configur$6).ab.) and bandwidth	US-PGPUB; USPAT	OR	ON	2005/07/06 13:54
L51	6	711/170-172.ccls. and ((configur$5).ti. (configur$6).ab.) and vhdl	US-PGPUB; USPAT	OR	ON	2005/07/06 13:54
L52	39	711/170-172.ccls. and ((configur$5).ti. (configur$6).ab.) and matrix	US-PGPUB; USPAT	OR	ON	2005/07/06 13:54
L53	13	711/170-172.ccls. and ((configur$5).ti. (configur$6).ab.) and parallelism	US-PGPUB; USPAT	OR	ON	2005/07/06 13:54
L54	1	"6553477".pn.	US-PGPUB; USPAT	OR	ON	2005/07/06 13:54

L55	3	711/170-173.ccls. and reconfigurable adj processor	US-PGPUB;
			USPAT;
			JPO
				OR	ON	2005/07/06 13:54

L56	9	("20030046530" | "5737524" | "5872919" | "5915104" | "5953512" | "6000014" | "6104415" | "6216219" | "6339819").PN.	US-PGPUB;
			USPAT;
			USOCR
				OR	ON	2005/07/06 13:54

L57	264	reconfigurable adj processor	US-PGPUB;
			USPAT;
			JPO
				OR	ON	2005/07/06 13:54

L58	589	reconfigurable adj2 processor	US-PGPUB;
			USPAT;
			JPO
				OR	ON	2005/07/06 13:54

L59	325	L58 not L57	US-PGPUB;
			USPAT;
			JPO
				OR	ON	2005/07/06 13:54

L60	113	L58 and ("711" "713").clas.	US-PGPUB;
			USPAT;
			JPO
				OR	ON	2005/07/06 13:54

L61	8	(adaptive adj processor) and ("711" "713").clas.	US-PGPUB;
			USPAT;
			JPO
				OR	ON	2005/07/06 13:54

L62	4	L61 not L60	US-PGPUB;
			USPAT;
			JPO
				OR	ON	2005/07/06 13:54

L63	0	"008128".pa.	US-PGPUB;
			USPAT;
			JPO
				OR	ON	2005/07/06 13:54

L64	6	"008128".ap.	US-PGPUB;
			USPAT;
			JPO
				OR	ON	2005/07/06 13:54

L65	37	src adj computers	US-PGPUB;
			USPAT;
			JPO
				OR	ON	2005/07/06 13:54

L66	7	711/117.ccls. and reconfigurable near3 (memory cache RAM random adj access adj memory processor)	US-PGPUB; USPAT	OR	ON	2005/07/06 13:54
L67	15	711/118.ccls. and reconfigurable near3 (memory cache RAM random adj access adj memory processor)	US-PGPUB; USPAT	OR	ON	2005/07/06 13:54
L68	4	"859051".ap.	US-PGPUB; USPAT	OR	ON	2005/07/06 13:54
L69	1	"6678790".pn.	US-PGPUB; USPAT	OR	ON	2005/07/06 13:54
L70	2	"021492".ap.	US-PGPUB; USPAT	OR	ON	2005/07/06 13:54
L71	1	"6563746".pn.	US-PGPUB; USPAT	OR	ON	2005/07/06 13:54
L72	2	("6574682" "5860111").pn.	US-PGPUB; USPAT	OR	ON	2005/07/06 13:54
L73	4	("6026402" "6633515").pn. "20030169283" "20030136846"	US-PGPUB; USPAT	OR	ON	2005/07/06 13:54
L74	2	"20030208658" "20030194458"	US-PGPUB; USPAT	OR	ON	2005/07/06 13:54
S63	15	711/118.ccls. and reconfigurable near3 (memory cache RAM random adj access adj memory processor)	US-PGPUB; USPAT	OR	ON	2005/01/03 13:19
S64	6	711/117.ccls. and reconfigurable near3 (memory cache RAM random adj access adj memory processor)	US-PGPUB; USPAT	OR	ON	2005/01/03 11:58

S65	4	"859051".ap.	US-PGPUB; USPAT	OR	ON	2005/01/03 12:06
S66	1	"6678790".pn.	US-PGPUB; USPAT	OR	ON	2005/01/03 12:29
S67	2	"021492".ap.	US-PGPUB; USPAT	OR	ON	2005/01/10 07:41
S68	1	"6563746".pn.	US-PGPUB; USPAT	OR	ON	2005/07/05 17:20
S69	2	("6574682" "5860111").pn.	US-PGPUB; USPAT	OR	ON	2005/07/05 17:21
S70	4	("6026402" "6633515").pn. "20030169283" "20030136846"	US-PGPUB; USPAT	OR	ON	2005/07/05 17:22
S71	2	"20030208658" "20030194458"	US-PGPUB; USPAT	OR	ON	2005/07/05 17:22

PALM INTRANET

Day : Wednesday
Date: 7/6/2005
Time: 14:00:10

Inventor Name Search Result

Your Search was:

Last Name = POZNANOVIC
First Name = DANIEL

Application#	Patent#	Status	Date Filed	Title	Inventor Name 11
60479339	Not Issued	159	06/18/2003	BANDWIDTH EFFICIENCY AND UTILIZATION USING DIRECT EXECUTION LOGIC	POZNANOVIC, DANIEL
60422722	Not Issued	159	10/31/2002	GENERAL PURPOSE RECONFIGURABLE COMPUTING HARDWARE AND SOFTWARE	POZNANOVIC, DANIEL
60286979	Not Issued	159	04/30/2001	DELIVERING ACCELERATION: THE POTENTIAL FOR INCREASED HPC APPLICATION PERFORMANCE USING RECONFIGURABLE LOGIC	POZNANOVIC, DANIEL
11140718	Not Issued	020	05/31/2005	INTERFACE FOR INTEGRATING RECONFIGURABLE PROCESSORS INTO A GENERAL PURPOSE COMPUTING SYSTEM	POZNANOVIC, DANIEL
10869200	Not Issued	071	06/16/2004	SYSTEM AND METHOD OF ENHANCING EFFICIENCY AND UTILIZATION OF MEMORY BANDWIDTH IN RECONFIGURABLE HARDWARE	POZNANOVIC, DANIEL
10285389	Not Issued	080	10/31/2002	DEBUGGING AND PERFORMANCE PROFILING USING CONTROL-DATAFLOW GRAPH REPRESENTATIONS WITH RECONFIGURABLE HARDWARE EMULATION	POZNANOVIC, DANIEL
10285299	Not Issued	092	10/31/2002	PROCESS FOR CONVERTING PROGRAMS IN HIGH-LEVEL PROGRAMMING LANGUAGES TO A UNIFIED EXECUTABLE FOR HYBRID COMPUTING PLATFORMS	POZNANOVIC, DANIEL
10285298	Not Issued	094	10/31/2002	SYSTEM AND METHOD FOR PARTITIONING CONTROL-DATAFLOW GRAPH REPRESENTATIONS	POZNANOVIC, DANIEL
10278345	Not Issued	041	10/23/2002	SYSTEM AND METHOD FOR EXPLICT COMMUNICATION OF MESSAGES BETWEEN PROCESSES RUNNING ON DIFFERENT NODES IN A CLUSTERED MULTIPROCESSOR SYSTEM	POZNANOVIC, DANIEL
10011835	Not Issued	071	12/05/2001	INTERFACE FOR INTEGRATING RECONFIGURABLE PROCESSORS INTO A GENERAL PURPOSE COMPUTING SYSTEM	POZNANOVIC, DANIEL

Inventor Search Completed: No Records to Display.

PALM INTRANET

Day : Wednesday
Date: 7/6/2005
Time: 14:00:25

Inventor Name Search Result

Your Search was:

Last Name = CALIGA
First Name = DAVID

Application#	Patent#	Status	Date Filed	Title	Inventor Name 7
60479339	Not Issued	159	06/18/2003	BANDWIDTH EFFICIENCY AND UTILIZATION USING DIRECT EXECUTION LOGIC	CALIGA, DAVID E.
60422722	Not Issued	159	10/31/2002	GENERAL PURPOSE RECONFIGURABLE COMPUTING HARDWARE AND SOFTWARE	CALIGA, DAVID E.
10869200	Not Issued	071	06/16/2004	SYSTEM AND METHOD OF ENHANCING EFFICIENCY AND UTILIZATION OF MEMORY BANDWIDTH IN RECONFIGURABLE HARDWARE	CALIGA, DAVID E.
10285318	Not Issued	030	10/31/2002	MULTI-ADAPTIVE PROCESSING SYSTEMS AND TECHNIQUES FOR ENHANCING PARALLELISM AND PERFORMANCE OF COMPUTATIONAL FUNCTIONS	CALIGA, DAVID E.
10278345	Not Issued	041	10/23/2002	SYSTEM AND METHOD FOR EXPLICT COMMUNICATION OF MESSAGES BETWEEN PROCESSES RUNNING ON DIFFERENT NODES IN A CLUSTERED MULTIPROCESSOR SYSTEM	CALIGA, DAVID

Inventor Search Completed: No Records to Display.

	Last Name	First Name	
Search Another: Inventor	CALIGA	DAVID	Search

To go back use Back button on your browser toolbar.

Back to PALM | ASSIGNMENT | OASIS | Home page

PALM INTRANET

Day : Wednesday
Date: 7/6/2005
Time: 14:00:34

Inventor Name Search Result

Your Search was:

Last Name = HAMMES
First Name = JEFFREY

Application#	Patent#	Status	Date Filed	Title	Inventor Name 10
60479339	Not Issued	159	06/18/2003	BANDWIDTH EFFICIENCY AND UTILIZATION USING DIRECT EXECUTION LOGIC	HAMMES, JEFFREY
60422722	Not Issued	159	10/31/2002	GENERAL PURPOSE RECONFIGURABLE COMPUTING HARDWARE AND SOFTWARE	HAMMES, JEFFREY
10869200	Not Issued	071	06/16/2004	SYSTEM AND METHOD OF ENHANCING EFFICIENCY AND UTILIZATION OF MEMORY BANDWIDTH IN RECONFIGURABLE HARDWARE	HAMMES, JEFFREY
10345082	Not Issued	030	01/14/2003	MAP COMPILER PIPELINED LOOP STRUCTURE	HAMMES, JEFFREY
10285401	Not Issued	094	10/31/2002	EFFICIENCY OF RECONFIGURABLE HARDWARE	HAMMES, JEFFREY
10285399	Not Issued	061	10/31/2002	SYSTEM AND METHOD FOR CONVERTING CONTROL FLOW GRAPH REPRESENTATIONS TO CONTROL-DATAFLOW GRAPH REPRESENTATIONS	HAMMES, JEFFREY
10285389	Not Issued	080	10/31/2002	DEBUGGING AND PERFORMANCE PROFILING USING CONTROL-DATAFLOW GRAPH REPRESENTATIONS WITH RECONFIGURABLE HARDWARE EMULATION	HAMMES, JEFFREY
10285299	Not Issued	092	10/31/2002	PROCESS FOR CONVERTING PROGRAMS IN HIGH-LEVEL PROGRAMMING LANGUAGES TO A UNIFIED EXECUTABLE FOR HYBRID COMPUTING PLATFORMS	HAMMES, JEFFREY
10285298	Not Issued	094	10/31/2002	SYSTEM AND METHOD FOR PARTITIONING CONTROL-DATAFLOW GRAPH REPRESENTATIONS	HAMMES, JEFFREY

Inventor Search Completed: No Records to Display.

	Last Name	First Name	
Search Another: Inventor	HAMMES	JEFFREY	Search

To go back use Back button on your browser toolbar.

Back to PALM | ASSIGNMENT | OASIS | Home page

UNITED STATES PATENT AND TRADEMARK OFFICE

/-\

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
10/869,200	06/16/2004	Daniel Poznanovic	SRC028	5929

25235 7590 07/12/2005

HOGAN & HARTSON LLP
ONE TABOR CENTER, SUITE 1500
1200 SEVENTEENTH ST
DENVER, CO 80202

EXAMINER	
THOMAS, SHANE M	
ART UNIT	PAPER NUMBER
2186	

DATE MAILED: 07/12/2005

Please find below and/or attached an Office communication concerning this application or proceeding.

PTO-90C (Rev. 10/03)

Office Action Summary	**Application No.** 10/869,200	**Applicant(s)** POZNANOVIC ET AL.	
	Examiner Shane M. Thomas	**Art Unit** 2186	

-- The MAILING DATE of this communication appears on the cover sheet with the correspondence address --

Period for Reply

A SHORTENED STATUTORY PERIOD FOR REPLY IS SET TO EXPIRE *3* MONTH(S) FROM THE MAILING DATE OF THIS COMMUNICATION.

- Extensions of time may be available under the provisions of 37 CFR 1.136(a). In no event, however, may a reply be timely filed after SIX (6) MONTHS from the mailing date of this communication.
- If the period for reply specified above is less than thirty (30) days, a reply within the statutory minimum of thirty (30) days will be considered timely.
- If NO period for reply is specified above, the maximum statutory period will apply and will expire SIX (6) MONTHS from the mailing date of this communication.
- Failure to reply within the set or extended period for reply will, by statute, cause the application to become ABANDONED (35 U.S.C. § 133). Any reply received by the Office later than three months after the mailing date of this communication, even if timely filed, may reduce any earned patent term adjustment. See 37 CFR 1.704(b).

Status

1)☒ Responsive to communication(s) filed on *11 April 2005*.

2a)☒ This action is **FINAL**. 2b)☐ This action is non-final.

3)☐ Since this application is in condition for allowance except for formal matters, prosecution as to the merits is closed in accordance with the practice under *Ex parte Quayle*, 1935 C.D. 11, 453 O.G. 213.

Disposition of Claims

4)☒ Claim(s) *1-24* is/are pending in the application.

4a) Of the above claim(s) _____ is/are withdrawn from consideration.

5)☐ Claim(s) _____ is/are allowed.

6)☒ Claim(s) *1-24* is/are rejected.

7)☐ Claim(s) _____ is/are objected to.

8)☐ Claim(s) _____ are subject to restriction and/or election requirement.

Application Papers

9)☐ The specification is objected to by the Examiner.

10)☒ The drawing(s) filed on *11 April 2005* is/are: a)☒ accepted or b)☐ objected to by the Examiner.

Applicant may not request that any objection to the drawing(s) be held in abeyance. See 37 CFR 1.85(a).

Replacement drawing sheet(s) including the correction is required if the drawing(s) is objected to. See 37 CFR 1.121(d).

11)☐ The oath or declaration is objected to by the Examiner. Note the attached Office Action or form PTO-152.

Priority under 35 U.S.C. § 119

12)☐ Acknowledgment is made of a claim for foreign priority under 35 U.S.C. § 119(a)-(d) or (f).

a)☐ All b)☐ Some * c)☐ None of:

1.☐ Certified copies of the priority documents have been received.

2.☐ Certified copies of the priority documents have been received in Application No. _____.

3.☐ Copies of the certified copies of the priority documents have been received in this National Stage application from the International Bureau (PCT Rule 17.2(a)).

* See the attached detailed Office action for a list of the certified copies not received.

Attachment(s)

1) ☒ Notice of References Cited (PTO-892)

2) ☐ Notice of Draftsperson's Patent Drawing Review (PTO-948)

3) ☒ Information Disclosure Statement(s) (PTO-1449 or PTO/SB/08)
Paper No(s)/Mail Date *4/11/05 & 6/6/05*.

4) ☐ Interview Summary (PTO-413)
Paper No(s)/Mail Date. _____ .

5) ☐ Notice of Informal Patent Application (PTO-152)

6) ☐ Other: _____.

DETAILED ACTION

This Office action is responsive to the amendment filed 4/11/2005.

All previously outstanding objections and rejections to the Applicant's disclosure and claims not contained in this Action have been respectfully withdrawn by the Examiner hereto.

Information Disclosure Statement

The information disclosure statement (IDS) submitted on 4/11/2005 has NOT been considered by the Examiner as the Application Number field on the Form 1449 reflects application number 10/809,200.

The information disclosure statement (IDS) submitted on 6/6/2005 was filed after the mailing date of the non-final Office action on 1/14/2005. The submission is in compliance with the provisions of 37 CFR 1.97. Accordingly, the information disclosure statement is being considered by the examiner.

Response to Amendment

The rejections of claims 1,3,8, and 14 have been modified to reflect the amendments to the respective claims.

Response to Arguments

Applicant's arguments filed 4/11/2005 have been fully considered but they are not persuasive.

Claims 2,3,13, and 14 remain rejected under 35 U.S.C. 112, first paragraph. While the Applicant's response on page 8 has verified the Examiner's assumption regarding the claim limitations of claims 2,3,13, and 14, no correction or amendment has been executed by the Applicant to overcome the rejection. The Applicant's specification does not disclose that a reconfigurable processor cannot have a cache nor a cache line-sized unit of contiguous data. As such the Examiner has maintained the rejections.

As per the Applicant's arguments regarding claim 1, the Applicant states on page 9 of the Response that Paulraj shows a reconfigurable cache but not a reconfigurable processor. The Examiner disagrees. While the caching system 112 (figure 6) of Paulraj is configurable (step 214, figure 5), it is also shown as being an element of CPU 110. Therefore since, the cache 112 is reconfigurable, it is justified that the processor 110, itself, is also reconfigurable as the reconfiguration of the FPGA module 112 occurs *within* the processor. As such, the CPU 110 can be construed as a --reconfigurable-- processor.

As per the Applicant's arguments regarding claim 11, the Examiner has shown in above in the discussion of claim 1 that Paulraj teaches a reconfigurable processor as claimed by the Applicant.

As per the Applicant's arguments regarding claim 17, the Examiner has shown above in the discussion of claim 1 that Paulraj teaches a reconfigurable processor as claimed by the Applicant. Further, the data prefetch unit, as defined in the rejection by the Examiner, is the portion of the reconfiguration unit that accesses the memory; the memory stores a vector corresponding to an optimal configuration for a particular application program (¶26). Data is transferred between the memory and the data prefetch unit in a reconfigurable processor since the reconfiguration unit 106 can be part of a reconfigurable processor 100 as shown in figure 4 (¶22).

As per the Applicant's arguments regarding claim 24, the Examiner has modified the rejection to better explain how the prior art reference of Paulraj teaches the limitations of claim 24.

Claim Rejections - 35 USC § 112

The following is a quotation of the first paragraph of 35 U.S.C. 112:

> The specification shall contain a written description of the invention, and of the manner and process of making and using it, in such full, clear, concise, and exact terms as to enable any person skilled in the art to which it pertains, or with which it is most nearly connected, to make and use the same and shall set forth the best mode contemplated by the inventor of carrying out his invention.

Claims 2,3,13, and 14 are rejected under 35 U.S.C. 112, first paragraph, as failing to comply with the written description requirement. The claims contains subject matter which was not described in the specification in such a way as to reasonably convey to one skilled in the relevant art that the inventor(s), at the time the application was filed, had possession of the claimed invention.

As per claims 2 and 13, the Applicant's disclosure does not explicitly mention that the reconfigurable processors cannot have a cache. The disclosure mentions in the Background section, and specifically in paragraphs 16-17, the drawbacks of having a hard-wired cache in a system; however, the Detailed Description does not explicitly state that the reconfigurable processor as taught by the Applicant *cannot* contain a cache. It appears to the Examiner that no specific (hard-wired) cache memory is included in the reconfigurable processor as taught in the disclosure; rather an on-board memory and user-logic can be configured based on a program (paragraph 52). Therefore, for the purposes of examination, the Examiner shall interpret the claim such that the reconfigurable processor of claim 1 does not contain a *hard-wired* (specific) cache.

As per claims 3 and 14, it follows from the rejection for claims 2 and 13, that since Applicant's disclosure does not explicitly state that a reconfigurable processor *cannot* have a cache, the disclose further does not explicitly teach that the reconfigurable processor cannot have a cache line-sized unit of contiguous data. For the purposes of examination and based on the discussion of claim 2 above, the Examiner shall interpret the limitation of claim 3 such that the reconfigurable processor of claim 1 does not have a *hard-wired* (specific) cache line-sized unit of contiguous data being retrieved from the second memory.

Claim Rejections - 35 USC § 102

The following is a quotation of the appropriate paragraphs of 35 U.S.C. 102 that form the basis for the rejections under this section made in this Office action:

> A person shall be entitled to a patent unless –
>
> (e) the invention was described in (1) an application for patent, published under section 122(b), by another filed in the United States before the invention by the applicant for patent or (2) a patent granted on an application for patent by another filed in the United States before the invention by the applicant for patent, except that an international application filed under the treaty defined in section 351(a) shall have the effects for purposes of this subsection of an application filed in the United States only if the international application designated the United States and was published under Article 21(2) of such treaty in the English language.

Claims 1-24 are rejected under 35 U.S.C. 102(e) as being anticipated by Paulraj (U.S. Patent Application Publication No. 2003/0084244).

As per claim 1, Paulraj shows a reconfigurable processor in figure 6 and a first memory (L1) having a first characteristc memory utilization and a second memory (L2) having a second characteristic memory utilization. It is well known in the art that L1 caches have a higher utilziation rate than a lower-level cache such as L2. Paulraj teaches in ¶1 that upon a command from a processor, a search for the requested data is begines with the highest level cache (L1) and [if a miss occurs] continues next to the next level cache (L2). Thus it is inherent that the memory utilziation characteristc of the L1 cache of the reconfigurable processor 110 in figure 6 is greater than the memory utilziation characteristic of the L2 cache (and likewise for the L3 cache) as the L2 cache would only be utilzied when a miss to the L1 cache occurred. In other words, the reconfigurable processor *always* utilizes the L1 cache for a memory access and the *only* utilzies the L2 cache for requested data when the data is not in the L1 cache. Therefore, the cache utilziation characteristics of the --first memory-- and the --second memory-- are different.

Paulraj further teaches a functional unit 102 that executes applications using the memories L1 and L2 (paragraph 9). As is known in the art, a cache memory controller is often used to access and move data between a memory hierarchy. The Examiner is considering a data prefetch unit to be the logic assocatied with the moving, and only the moving, of data between the first and second memories (L1 and L2) since Paulraj shows a connection between the levels of cache in figure 6. This logic as well as the first and second memory types (L1 and L2) are configued by a program – refer to paragraphs 23-24. The data prefetch unit as defined by the Examiner must be configued as well by the program when moving data since the cache line size and blocking factor can change, so different amounts of data can be exchanged for the same access when different programs run.

As per claims 2 and 13, as taught in paragraphs 23 and 29 of Paulraj, no specific cache is present in the system of Paulraj. Rather, an FPGA is utilized as representing a caching hierarchy and is optimized based on the memory needs of a specific program running on the reconfigurable processor.

As per claims 3 and 14, Paulraj teaches in paragraph 23 that a specific [cache] line size of contiguous data is not retrieved since the data line size is optimized based on the memory needs of the program when executing on the reconfigurable processor. Refer also to paragraph 29. Further, it is therefore inherent that the second memory have a charactersitic line size since Paulraj teaches in ¶¶22-23 that a best line size for the memory arrangement for a particular program is determined and utilzied when that program is run. For example, a line-size characteristic would be ultized when transferring data from the L2 cache to the L1 cache.

As per claim 4, Paulraj teaches that a load/store unit is used to access the caches (L1-L3) in order to determine if cache data is present in the cache hierarchy (paragraph 6). Since the functional unit 102 (figure 6) is responsible for accessing the programmable memory unit 104, the Examiner is therefore considering the load/store unit logic of the programmable memory unit that is responsible for for accessing the L1 and L2 caches (first and second memory types) to be a memory controller. It can be seen that the memory controller, as defined by the Examiner, controls the transfer of data between the memory (assuming second memory L2) and the data prefetch unit, since the memory controller (load/store unit logic) is responsible for retrieving the data from the cache if a hit occurs (paragraph 4).

As per claim 5, as taught in paragraph 1, an external memory (element 18, figure 1) is generaly coupled to a microprocessor and holds data to be used by the microcontroller during program execution. The Examiner is considering the process of writing data back to the external memory from the FPGA memory 104 containing the caches (on-board memory), such as during a write-back scheme as known in the art, to be performed by the data prefetch unit portion of the functional logic as defined above by the Examiner. The data prefetch logic, as defined above, is responsible for all of the transfer of data into, out of, and between the FPGA memory 104.

As per claim 6, the Examiner is regarding a --register-- in its broadest reasonable sense and it thus considering it be to be a unit of logic. Therefore, the portion of the function logic that is responsible for the movement of data (as defined above to be the data prefetch unit) is being considered by the Examiner as containing a --register-- portion of the reconfigurable processor since, for instance, the blocking factor and line size of the programmable memory 112 can change, a --register-- or portion of the reconfigurable processor must be set in order to indicate the currnet line size and blocking factor when a given application is being run on the reconfigurable processor at a given point in time. Refer to paragraph 23.

As per claim 7, the Examiner is considering the process of --disassembling the data prefetch unit-- as modifying the data prefetch unit logic of the fucntion logic 102 every time the program being executed by the reconfigurable processor changes. It can be seen that the data prefetch unit changes during these intervals since the cache line size, blocking factor, and associativity of the FPGA changes when optimal for the next program to be executed (refer to paragraph 23). Thus it can be seen that the data prefetch unit logic is --disassembled-- when another program is executed by the reconfigurable processor of Paulraj.

As per claim 8, as can be seen that the FPGA memory 112, that comprises the first and second memories (L1 and L2) and which is accessed by the data prefetch unit of the functional unit 102 as discussed above, is a --processor memory-- (part of cpu 110). It can also be seen that the --second memory-- (L2) is also a --processor memory-- since it is contained within reconfigurable processor 110. Therefore, since the data pretech unit can access the L2 cache as discussed above in the rejection of claim 1, the data prefetch unit can retrive data from the L2 portion of --processor memory--112.

As per claim 9, as shown in figure 1 and taught in paragraph 1 of Paulraj, the system 10 is actually a microprocessor, which contains a memory controller 14. The main difference between the prior art of figure 1 and the invention of Paulraj in figure 6 is that the memroy hierarchy is configurable and accessed by a fucntional unit in lieu of a separate memory controller logic (paragraph 9). Therefore, since the memory controller logic for accessing the cache hierarchy is still contained within cpu 110 of figure 6, it can be seen that the cpu 110 is actually a microprocessor. It follows that the --processor memory-- 112 is therefore a --microprocessor memory--.

As per claim 10, since the cpu 110 of figure 6 is a reconfigurable processer (able to reconfigure its memory heirarchy to match the needs of the application it is currently running), it can be seen that the cpu memory 112 is a reconfigurable processor memory.

As per claim 11, Paulraj depicts a reconfigurable hardware system in figure 6. Paulraj further teaches in paragraph 26 that when a particular application is to be run by the reconfigrable processor 110, a configuration vector is retrieved to program the programmable memory 112 (figure 6). As shown in figure 6, the step of accesing the configuration vector is executed outside of the reconfigurable processor 110. Therefore, the Examiner is considering the memory that contains the configuration vectors to be a--common memory-- and a data prefetch unit (reconfiguration unit 106 executing on the reconfigurable processor 110) accessing the common memory in order to determine how to program the memory 112 (paragraph 29). The data prefetch unit 106 is --configured-- by an application to be excuted on the sysem 110 since when a new application is to be executed, the data prefetch unit is called upon (or configured) to access the configuration vector for the particular application.

As per claim 12, the Examiner is considering a --memory controller-- to be the system portion utilized when creating a new configuration vector for an application. Such a process occurs in figure 5 and taught in paragraghs 23-25 of Paulraj. When a new configuration vector is created by analizing performance information that has been collected for the application. The Examiner is thereby considering the --memory controller-- to be the element of the reconfigurable hardware system that is associated with storing the new configuration vector into the common memory so that the vector can be accessed later when the same application is run again.

As per claim 15, the Examiner is considering the reconfiguration module 106 of the reconfigurable processsor 110, as comprising two distinct elements: a --computational unit-- and a --data access unit--. The data access unit is the element that is responsible for accessing the configuration vector as taught in paragraph 29 of Paulraj; or in other words, the Examiner is considering the --data access unit-- to be the same as the --memory controler-- defined in the rejection of claim 12. The Examiner is further considering the --computational unit-- of the rconfiguration module 106 to be the element that sets up the programmable memory module 104 using the configuration vector that was accessed by the --data access unit-- (paragraph 29).

As per claim 16, as taught by Paulraj in paragraph 29, the --data access unit-- supplies the configuration vector to the --computational unit-- in order to set up the programmable memory 104 as required by the application to be run on the reconfurable processor 110.

As per claim 17, the Examiner is considering a --data prefetch unit-- to be the reconfiguration unit 106 of reconfigurable processor 110 (figure 6). As taught in paragraph 26 and 29 of Paulraj, the --data prefetch unit-- accesses a memory in order to determine if a configuration vector is known for a given application, and if so, the vector is retrieved (from the memory). If this --data-- (configuration vector) is not known then a simulation is performed with the application in order to collect performance information. The Examiner is considering the element that executes and collects the performance data as being a --computational unit-- and the element of Paulraj that stores the configuration vector, once determined, to be a --data access unit-- since it stores the vector into the --memory-- from which it can be later retrieved (step 212 of figure 5). The --computational unit--, --data access unit--, and the --data prefetch unit-- are all --configured-- by a program (application) since (1) a new application configures the computational unit portion of the reconfiguration unit to perform a simulation in order to determine the optimal memory hierarchy organization; (2) the new application configures the --data access unit-- to store and retrieve (step 212) the configuration vector for that particular application; and (3) the --data prefetch unit-- is configured by the application to determine if a configuration file exists for the application and if so, the data prefetch unit is configured by the program the programmable memory 112 in order to optimize the programmable memory for that particular application.

As per claim 18, the --data-- (configuration vector) is transferred from the --computational unit-- to the --data access unit-- when the configuration unit has created a configuration vector (step 208 of figure 5). The --data-- is written to the memory --from-- the --data prefetch unit-- since the data prefetch unit (reconfiguration unit 106) is the element that executed the beginning of the configuration vector creation process (step 200 of figure 5). Refer to paragraph 26. Thus the Examiner is considering the data as being written --from-- the data prefetch unit.

As per claim 19, as taught in paragraph 26, if the configuration vector is known, the vector is retrieved from the memory to the data prefetch unit (reconfiguration unit 106). The data is read directly from the data prefetch unit when a request to create a configuration vector is made for a new application as shown in figure 6 since the data prefetch unit is responsible for being the vector creation process. The data is directed from the data prefetch unit (reconfigure logic) to be read from the memory by the data access unit to the computational unit where it is processed to produce a configuration vector.

As per claim 20, as stated above, the configuration vector (--data--) is created by the computational unit via acquired simulation data. The configuration vector is the resultant product that is transferred from the memory to the data prefect unit when it is determined that the configuration vector for the application is available (paragraph 26). Thus --all-- of the data that is transferred is processed by the computational unit (albeit before the transfer occurs) since the data prefetch unit required the entire configuration vector in order to set up the programmable memory 112.

As per claim 21, Paulraj shows in paragraph 26 that an explicit request for the configuration vector for the current application results in the data (if it exists) selected for the optimal configuration of the programmable memory 112 for that application.

As per claim 22, the Examiner is not considering the data (configuration vector) to be the size of a complete cache line since the data is used to create a cache hierarchy. In other words, the caches (L1-L3) of the programmable memory 112 are not programmed when the data is transferred from the memory to the data prefetch unit; therefore, the data cannot be a complete cache line.

As per claim 23, since the Examiner defined the portion of the reconfiguration unit that accesses the configuration file (data) from the memory, the Examiner is defining the logic that controls the actual transfer of that data to the data prefetch unit (portion of the reconfiguration unit that executes the fetch of the configuration vector and then programs the programmable memory 112) to be a --memory controller--. Thus the data access unit determines whether a configuration vector exists for an application and if so, the memory controller sends that data to the data prefetch unit.

As per claim 24, Paulraj shows a reconfigurable processor in figure 6 that comprises a computation unit 110 and a data access unit (elements 120 and 114, which comprise the reconfiguration unit 106 of figure 4 - ¶28). In figure 6, the data access unit can be seen as being coupled to the computational unit. The data access unit retrieves data (configuration vector) from a memory internal to the data access unit (i.e. reconfiguration unit) and supplies the data to the computation unit in the form of modifications to the cache FPGA module 112. Refer to ¶23.

The computation unit is configured by the program (application) that is to be executed on it by the run-time profile that is created and stored by the reconfiguration unit (¶22), thereby creating the optimal configuration of the different caches. The data access unit (specifically the memory portion used to store configuration profiles for the different application programs) is configured by the program that is to run on the reconfigurable processor. When a new program is to be run, [as a result] the program configures the reconfiguration unit to collect statistics regarding the memory usages (caches L1, L2, and L3) of the program and a configuration vector is associated with the respective program and stored in the reconfiguration unit. Refer to ¶¶23-24. When a program is known, the program [as a result] configures the data access unit (reconfiguration unit) to retrieve the associated configuration vector and apply it to the FPGA memory of the reconfigurable processor (¶29).

Conclusion

The prior art made of record and not relied upon is considered pertinent to applicant's disclosure.

Magoshi (U.S. Patent Application Publication No. 2003/0208658) teaches the memory utilization characteristics of an L1 and an L2 cache in figure 2. As shown, the L1 cache is always accessed (high memory utilization) upon an access request from a processor and the L2 cache is only accessed (lower memory utilization) when a miss occurs with respect to the L1 cache.

Any inquiry concerning this communication or earlier communications from the examiner should be directed to Shane M. Thomas whose telephone number is (571) 272-4188. The examiner can normally be reached on M-F 8:30 - 5:30.

If attempts to reach the examiner by telephone are unsuccessful, the examiner's supervisor, Matt M. Kim can be reached on (571) 272-4182. The fax phone number for the organization where this application or proceeding is assigned is 703-872-9306.

Information regarding the status of an application may be obtained from the Patent Application Information Retrieval (PAIR) system. Status information for published applications may be obtained from either Private PAIR or Public PAIR. Status information for unpublished applications is available through Private PAIR only. For more information about the PAIR system, see http://pair-direct.uspto.gov. Should you have questions on access to the Private PAIR system, contact the Electronic Business Center (EBC) at 866-217-9197 (toll-free).

Shane M. Thomas

HONG CHONG KIM
PRIMARY EXAMINER

PTO/SB/08a(08/03)
Approved for use through 07/31/2006. OMB 0651-0031
Patent and Trademark Office; U.S. DEPARTMENT OF COMMERCE
Under the Paperwork Reduction Act of 1995, no persons are required to respond to a collection of information unless it displays a valid. OMB control number.

Substitute for form 1449A/PTO

INFORMATION DISCLOSURE STATEMENT BY APPLICANT

(Use as many sheets as necessary)

Sheet 1 of 2		Application Number	10/809,200
		Filing Date	June 16, 2004
		First Named Inventor	Daniel Poznanovic et al.
		Art Unit	2186
		Examiner Name	Thomas, Shane M.
		Attorney Docket No.	SRC028

U.S. PATENT DOCUMENTS

Examiner Initials	Cite No.[1]	Document No. No. – Kind Code[2]	Publication Date MM-DD-YYYY	Name of Patentee or Applicant of Cited Doc	Pages, Columns, Lines, Where Relevant Passages or Relevant Figures Appear
		~~US-6,076,152~~	~~06/13/2000~~	~~Huppenthal et al.~~	
		~~US-6,247,110~~	~~06/12/2001~~	~~Huppenthal et al.~~	
		~~US-6,356,983~~	~~03/12/2002~~	~~Parks~~	
		~~US-6,594,736~~	~~06/15/2003~~	~~Parks~~	
		US-			
		US-			
		US-			
		US-			
		US-			
		US-			
		US-			
		US-			

FOREIGN PATENT DOCUMENTS

Examiner Initials	Cite No.[1]	Foreign Patent Document Country Code[3] Number[4] Kind Code[5]	Publication Date MM-DD-YYYY	Name of Patentee or Applicant of Cited Doc	Pages, Columns. Lines Where Relevant Passages or Relevant Figures Appear	T[6]

EXAMINER SIGNATURE	Shane M Thom	DATE CONSIDERED	7/11/05

EXAMINER: Initial if reference considered, whether or not citation is in conformance with MPEP 609. Draw line through citation if not in conformance and not considered. Include copy of this form with next communication to applicant. [1] Applicant's unique citation designation number (optional). [2] See Kinds Codes of USPTO Patent Documents at www.uspto.gov or MPEP 901.04. [3] Enter Office that issued the document, by the two-letter code (WIPO Standard ST.3). [4] For Japanese patent documents, the indication of the year of the reign of the Emperor must precede the serial number of the patent document. [5] Kind of document by the appropriate symbols as indicated on the document under WIPO Standard ST. 16 if possible. 6 Applicant is to place a check mark here if English language Translation is attached.

This collection of information is required by 37 CFR 1.97 and 1.98. The information is required to obtain or retain a benefit by the public which is to file (and by the USPTO to process) and application. Confidentiality is governed by 35 U.S.C. 122 and 37 CFR 1.14. This collection is estimated to take 2 hours to complete, including gathering, preparing, and submitting the completed application form to the USPTO. Time will vary depending upon the individual case. Any comments on the amount of time you require to complete this form and/or suggestions for reducing this burden, should be sent to the Chief Information Officer, U.S. Patent and Trademark Office, U.S. Department of Commerce, P.O. Box 1450, Alexandria, VA 22313-1450. DO NOT SEND FEES OR COMPLETED FORMS TO THIS ADDRESS. SEND TO: Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-1450.

PTO/SB/08a(08/03)
Approved for use through 07/31/2006. OMB 0651-0031
Patent and Trademark Office; U.S. DEPARTMENT OF COMMERCE
Under the Paperwork Reduction Act of 1995, no persons are required to respond to a collection of information unless it displays a valid. OMB control number.

Substitute for form 1449A/PTO				Application Number	10/809,200
				Filing Date	June 16, 2004
INFORMATION DISCLOSURE STATEMENT BY APPLICANT				First Named Inventor	Daniel Poznanovic et al.
				Art Unit	2186
(Use as many sheets as necessary)				Examiner Name	Thomas, Shane M.
Sheet	2	of	2	Attorney Docket No.	SRC028

NON PATENT LITERATURE DOCUMENTS

Examiner Initials*	Cite No.[1]	Include name of the author (in CAPITAL LETTERS), title of the article (when appropriate), title of the item (book, magazine, journal, serial, symposium, catalog, etc.), date, page(s), volume-issue number(s) publisher, city and/or country where published	T[2]
		~~DALLY, BILL, HANRAHAN, PAT, FEDKIW, RON, "A Streaming Supercomputer", September 18, 2001, pp. 1-17.~~	
		~~DALLY, WILLIAM J. et al., "Merrimac: Supercomputing with Streams", SC'03, November 16-21, 2003, Phoenix, AZ, 7 pages.~~	
		~~"Code Development and Porting Issues", SRC Computer, Inc., SRC-6E C Programming Environment v1.3 Guide, April 11, 2003, pp. 17-26.~~	

EXAMINER SIGNATURE		DATE CONSIDERED	7/11/05

EXAMINER: Initial if reference considered, whether or not citation is in conformance with MPEP 609. Draw line through citation if not in conformance and not considered. Include copy of this form with next communication to applicant.

[1] Applicant's unique citation designation number (optional). [2] Applicant is to place a check mark here if English language Translation is attached. This collection of information is required by 37 CFR 1.97 and 1.98. The information is required to obtain or retain a benefit by the public which is to file (and by the USPTO to process) and application. Confidentiality is governed by 35 U.S.C. 122 and 37 CFR 1.14. This collection is estimated to take 2 hours to complete, including gathering, preparing, and submitting the completed application form to the USPTO. Time will vary depending upon the individual case. Any comments on the amount of time you require to complete this form and/or suggestions for reducing this burden, should be sent to the Chief information Officer, U.S. Patent and Trademark Office, U.S. Department of Commerce, P.O. Box 1450, Alexandria, VA 22313-1450. DO NOT SEND FEES OR COMPLETED FORMS TO THIS ADDRESS. SEND TO: Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-1450.

PTO/SB/08a(08/03)
Approved for use through 07/31/2006. OMB 0651-0031
Patent and Trademark Office; U.S. DEPARTMENT OF COMMERCE
Under the Paperwork Reduction Act of 1995, no persons are required to respond to a collection of information unless it displays a valid OMB control number.

Substitute for form 1449A/PTO				Application Number	10/869,200
				Filing Date	June 16, 2004
INFORMATION DISCLOSURE STATEMENT BY APPLICANT				First Named Inventor	Daniel Poznanovic et al.
				Art Unit	2186
(Use as many sheets as necessary)				Examiner Name	Thomas, Shane M.
Sheet	1	of	1	Attorney Docket No.	SRC028

U.S. PATENT DOCUMENTS

Examiner Initials	Cite No.[1]	Document No. No. – Kind Code[2]	Publication Date MM-DD-YYYY	Name of Patentee or Applicant of Cited Doc	Pages, Columns, Lines, Where Relevant Passages or Relevant Figures Appear
SMT		US-2003/0084244 A1	05/01/2003	Paulraj	Entire Document
SMT		US-2003/0046530 A1	03/06/2003	Poznanovic	
		US-			
		US-			
		US-			
		US-			
		US-			
		US-			
		US-			
		US-			
		US-			
		US-			

FOREIGN PATENT DOCUMENTS

Examiner Initials	Cite No.[1]	Foreign Patent Document Country Code[3] Number[4] Kind Code[5]	Publication Date MM-DD-YYYY	Name of Patentee or Applicant of Cited Doc	Pages, Columns, Lines Where Relevant Passages or Relevant Figures Appear	T[6]

EXAMINER SIGNATURE	Shane M. Thomas	DATE CONSIDERED	7/5/05

EXAMINER: Initial if reference considered, whether or not citation is in conformance with MPEP 609. Draw line through citation if not in conformance and not considered. Include copy of this form with next communication to applicant. [1] Applicant's unique citation designation number (optional). [2] See Kinds Codes of USPTO Patent Documents at www.uspto.gov or MPEP 901.04. [3] Enter Office that issued the document, by the two-letter code (WIPO Standard ST.3). [4] For Japanese patent documents, the indication of the year of the reign of the Emperor must precede the serial number of the patent document. [5] Kind of document by the appropriate symbols as indicated on the document under WIPO Standard ST. 16 if possible. 6 Applicant is to place a check mark here if English language Translation is attached.

This collection of information is required by 37 CFR 1.97 and 1.98. The information is required to obtain or retain a benefit by the public which is to file (and by the USPTO to process) and application. Confidentiality is governed by 35 U.S.C. 122 and 37 CFR 1.14. This collection is estimated to take 2 hours to complete, including gathering, preparing, and submitting the completed application form to the USPTO. Time will vary depending upon the individual case. Any comments on the amount of time you require to complete this form and/or suggestions for reducing this burden, should be sent to the Chief information Officer, U.S. Patent and Trademark Office, U.S. Department of Commerce, P.O. Box 1450, Alexandria, VA 22313-1450. DO NOT SEND FEES OR COMPLETED FORMS TO THIS ADDRESS. SEND TO: Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-1450.

Notice of References Cited

Application/Control No. 10/869,200	Applicant(s)/Patent Under Reexamination POZNANOVIC ET AL.	
Examiner Shane M. Thomas	Art Unit 2186	Page 1 of 1

U.S. PATENT DOCUMENTS

*		Document Number Country Code-Number-Kind Code	Date MM-YYYY	Name	Classification
	A	US-2003/0208658	11-2003	Magoshi, Hidetaka	711/122
	B	US-			
	C	US-			
	D	US-			
	E	US-			
	F	US-			
	G	US-			
	H	US-			
	I	US-			
	J	US-			
	K	US-			
	L	US-			
	M	US-			

FOREIGN PATENT DOCUMENTS

*		Document Number Country Code-Number-Kind Code	Date MM-YYYY	Country	Name	Classification
	N					
	O					
	P					
	Q					
	R					
	S					
	T					

NON-PATENT DOCUMENTS

*		Include as applicable: Author, Title Date, Publisher, Edition or Volume, Pertinent Pages)
	U	
	V	
	W	
	X	

*A copy of this reference is not being furnished with this Office action. (See MPEP § 707.05(a).)
Dates in MM-YYYY format are publication dates. Classifications may be US or foreign.

Search Notes	**Application/Control No.** 10/869,200	**Applicant(s)/Patent under Reexamination** POZNANOVIC ET AL.
	Examiner Shane M. Thomas	**Art Unit** 2186

SEARCHED			
Class	Subclass	Date	Examiner
711	170-173	7/6/2005	
712	15	7/6/2005	

INTERFERENCE SEARCHED			
Class	Subclass	Date	Examiner

SEARCH NOTES (INCLUDING SEARCH STRATEGY)		
	DATE	EXMR
Updated East Search	7/6/2005	
Inventor Name Search	7/6/2005	

Index of Claims

Application No. 10/869,200	Applicant(s) POZNANOVIC ET AL.	
Examiner Shane M. Thomas	Art Unit 2186	

√	Rejected	–	(Through numeral) Cancelled	N	Non-Elected	A	Appeal
=	Allowed	+	Restricted	I	Interference	O	Objected

Claim		Date								
Final	Original	1/5/05	7/6/05							
	1	√	√							
	2	√	√							
	3	√	√							
	4	√	√							
	5	√	√							
	6	√	√							
	7	√	√							
	8	√	√							
	9	√	√							
	10	√	√							
	11	√	√							
	12	√	√							
	13	√	√							
	14	√	√							
	15	√	√							
	16	√	√							
	17	√	√							
	18	√	√							
	19	√	√							
	20	√	√							
	21	√	√							
	22	√	√							
	23	√	√							
	24	√	√							
	25									
	26									
	27									
	28									
	29									
	30									
	31									
	32									
	33									
	34									
	35									
	36									
	37									
	38									
	39									
	40									
	41									
	42									
	43									
	44									
	45									
	46									
	47									
	48									
	49									
	50									

Claim		Date								
Final	Original									
	51									
	52									
	53									
	54									
	55									
	56									
	57									
	58									
	59									
	60									
	61									
	62									
	63									
	64									
	65									
	66									
	67									
	68									
	69									
	70									
	71									
	72									
	73									
	74									
	75									
	76									
	77									
	78									
	79									
	80									
	81									
	82									
	83									
	84									
	85									
	86									
	87									
	88									
	89									
	90									
	91									
	92									
	93									
	94									
	95									
	96									
	97									
	98									
	99									
	100									

Claim		Date								
Final	Original									
	101									
	102									
	103									
	104									
	105									
	106									
	107									
	108									
	109									
	110									
	111									
	112									
	113									
	114									
	115									
	116									
	117									
	118									
	119									
	120									
	121									
	122									
	123									
	124									
	125									
	126									
	127									
	128									
	129									
	130									
	131									
	132									
	133									
	134									
	135									
	136									
	137									
	138									
	139									
	140									
	141									
	142									
	143									
	144									
	145									
	146									
	147									
	148									
	149									
	150									

U.S. Patent and Trademark Office

Part of Paper No. 07052005

UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

BIBDATASHEET

Bib Data Sheet

CONFIRMATION NO. 5929

SERIAL NUMBER 10/869,200	FILING DATE 06/16/2004 RULE	CLASS 711	GROUP ART UNIT 2186	ATTORNEY DOCKET NO. SRC028

APPLICANTS

Daniel Poznanovic, Colorado Springs, CO;

David E. Caliga, Colorado Springs, CO;
Jeffrey Hammes, Colorado Springs, CO;

** CONTINUING DATA **************************
This appln claims benefit of 60/479,339 06/18/2003

YES SMT

** FOREIGN APPLICATIONS ********************

NONE SMT

IF REQUIRED, FOREIGN FILING LICENSE GRANTED
** 08/04/2004

Foreign Priority claimed □ yes ☒ no 35 USC 119 (a-d) conditions met □ yes □ no □ Met after Allowance Verified and Acknowledged ______ Examiner's Signature SMT Initials	STATE OR COUNTRY CO	SHEETS DRAWING 12	TOTAL CLAIMS 24	INDEPENDENT CLAIMS 4

ADDRESS
25235
HOGAN & HARTSON LLP
ONE TABOR CENTER, SUITE 1500
1200 SEVENTEENTH ST
DENVER , CO
80202

TITLE
System and method of enhancing efficiency and utilization of memory bandwidth in reconfigurable hardware

FILING FEE RECEIVED 928	FEES: Authority has been given in Paper No. _________ to charge/credit DEPOSIT ACCOUNT No. _________ for following:	□ All Fees □ 1.16 Fees (Filing) □ 1.17 Fees (Processing Ext. of time) □ 1.18 Fees (Issue) □ Other _______________ □ Credit

PTO/SB/25 (08/03)
Approved for use through 07/31/2006. OMB 0651-0031
Patent and Trademark Office; U.S. DEPARTMENT OF COMMERCE
Under the Paperwork Reduction Act of 1995, no persons are required to respond to a collection of information unless it displays a valid. OMB control number.

Certificate of Transmission under 37 CFR 1.8

Serial No. 10/869,200

Application of: Daniel Poznanovic, David E. Caliga, and Jeffrey Hammes

Filed: June 16, 2004

Art Unit: 2186

Examiner: Thomas, Shane M.

Attorney Docket No. SRC028

For: SYSTEM AND METHOD OF ENHANCING EFFICIENCY AND UTILIZATION OF MEMORY BANDWIDTH IN RECONFIGURABLE HARDWARE

Confirmation No.:

Customer No.: **25235**

I hereby certify that this correspondence with the following documents are being facsimile transmitted to the United States Patent and Trademark Office:

1. Amendment and Response after Final (8 Pages)
2. Certificate of Transmission under 37 CFR 1.8 (1 page)

on 8/26/05 9
Date No. of Pages
(incl. Coversheet)

to centralized fax number: 571.273.8300

for C Schubert
Signature

Joan C. Schubert
Typed or printed name of person signing Certificate

Note: Each paper must have its own certificate of transmission, or its certificate must identify each submitted paper.

Client Matter No.: 80404.0033.001

Client Matter No. 80404.0033.001
Express Mail No.: Via Facsimile

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Serial No. 10/869,200	Confirmation No.: 5929
Application of: Daniel Poznanovic, et al.	Customer No.: **25235**
Filed: June 16, 2004	**EXPEDITED PROCEDURE UNDER 37 C.F.R. 1.116**
Art Unit: 2186	
Examiner: Thomas, Shane M.	
Attorney Docket No. SRC028	
For: SYSTEM AND METHOD OF ENHANCING EFFICIENCY AND UTILIZATION OF MEMORY BANDWIDTH IN RECONFIGURABLE HARDWARE	

AMENDMENT AND RESPONSE PURSUANT TO OFFICE ACTION DATED JULY 12, 2005

MAIL STOP AF
Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

Sir:

In response to the office communication mailed July 12, 2005 please amend the above-identified application as follows:

Amendments to the Claims are reflected in the listing of claims which begins on page 2 of this paper.

Remarks/Arguments begin on page 6 of this paper.

Amendments to the Claims:

This listing of claims will replace all prior versions and listings of claims in the application:

Listing of Claims:

1. (Currently Amended) A reconfigurable processor that instantiates an algorithm as hardware comprising:

a first memory having a first characteristic memory bandwidth and/or memory utilization; and

a data prefetch unit coupled to the first memory, wherein the data prefetch unit retrieves data from a second memory of second characteristic memory bandwidth and/or memory utilization and place the retrieved data in the first memory and wherein at least the first memory and data prefetch unit are configured by a program.

2. (Cancelled)

3. (Cancelled)

4. (Previously Presented) The reconfigurable processor of claim 1, wherein the data prefetch unit is coupled to a memory controller that controls the transfer of the data between the second memory and the data prefetch unit.

5. (Previously Presented) The reconfigurable processor of claim 1, wherein the data prefetch unit receives processed data from on-processor memory and writes the processed data to an external off-processor memory.

6. (Original) The reconfigurable processor of claim 1, wherein the data prefetch unit comprises at least one register from the reconfigurable processor.

7. (Original) The reconfigurable processor of claim 1, wherein the data prefetch unit is disassembled when another program is executed on the reconfigurable processor.

8. (Previously Presented) The reconfigurable processor of claim 1 wherein said second memory comprises a processor memory and said data prefetch unit is operative to retrieve data from the processor memory.

9. (Original) The reconfigurable processor of claim 8 wherein said processor memory is a microprocessor memory.

10. (Original) The reconfigurable processor of claim 8 wherein said processor memory is a reconfigurable processor memory.

11. (Currently Amended) A reconfigurable hardware system, comprising:

a common memory; and

one or more reconfigurable processors that can instantiate an algorithm as hardware coupled to the common memory, wherein at least one of the reconfigurable processors includes a data prefetch unit to read and write data between the data prefetch unit and the common memory, and wherein the data prefetch unit is configured by a program executed on the system.

12. (Original) The reconfigurable hardware system of claim 11, comprising a memory controller coupled to the common memory and the data prefetch unit.

13. (Cancelled)

14. (Cancelled)

15. (Previously Presented) The reconfigurable hardware system of claim 11, wherein the at least one of the reconfigurable processors also includes a computational unit coupled to a data access unit.

16. (Original) The reconfigurable hardware system of claim 15, wherein the computational unit is supplied the data by the data access unit.

17. (Previously Presented) A method of transferring data comprising:

transferring data between a memory and a data prefetch unit in a reconfigurable processor; and

transferring the data between a computational unit and a data access unit, wherein the computational unit and the data access unit, and the data prefetch unit are configured by a program.

18. (Original) The method of claim 17, wherein the data is written to the memory, said method comprising:

transferring the data from the computational unit to the data access unit; and

writing the data to the memory from the data prefetch unit.

19. (Previously Presented) The method of claim 17, wherein the data is read from the memory, said method comprising:

transferring the data from the memory to the data prefetch unit; and

reading the data directly from the data prefetch unit to the computational unit through the data access unit.

20. (Original) The method of claim 19, wherein all the data transferred from the memory to the data prefetch unit is processed by the computational unit.

21. (Original) The method of claim 19, wherein the data is selected by the data prefetch unit based on an explicit request from the computational unit.

22. (Original) The method of claim 17, wherein the data transferred between the memory and the data prefetch unit is not a complete cache line.

23. (Original) The method of claim 17, wherein a memory controller coupled to the memory and the data prefetch unit, controls the transfer of the data between the memory and the data prefetch unit.

24. (Currently Amended) A reconfigurable processor comprising:

a computational unit; and

a data access unit coupled to the computational unit, wherein the data access unit retrieves data from memory and supplies the data to the computational unit, and wherein the computational unit and the data access unit are configured by a program to instantiate an algorithm as hardware.

REMARKS/ARGUMENTS

Claims 1, 4-12 and 15-24 remain in the application. Claims 2, 3, 13 and 14 are cancelled. Claims 1, 11 and 24 are amended to more distinctly describe the subject matter of the invention.

A. Rejections under 35 U.S.C. 112.

The cancellation of claims 2, 3, 13, 14 renders the rejection under 35 U.S.C. 112 moot. However, the concept of a configurable processor that does not have a cache is believed to be supported by the claims themselves, and the subject matter of these claims is not waived.

B. Rejections under 35 U.S.C. 102.

Claims 1-24 were rejected under 35 U.S.C. 102 based upon Paulraj. This rejection is respectfully traversed.

Claim 1 is amended to adopt language from the definition of "reconfigurable processor" appearing in paragraph 39 of the specification as filed. This amendment is not believed to raise any new issues nor require further search because this meaning of reconfigurable processor is consistent with the application as filed and consistent with the definition of that term asserted in prior remarks submitted on April 11, 2005.

As amended, independent claim 1 calls for a reconfigurable processor that instantiates an algorithm as hardware. Although the reference show a reconfigurable cache, Paulraj does not show or suggest a reconfigurable processor that instantiates an algorithm as hardware. Moreover, nothing in Paulraj would suggest the rather significant changes required to replace the CPU with a reconfigurable processor that can instantiate an algorithm as hardware. For at least these reasons claim 1 is not anticipated nor made obvious by Paulraj.

Claims 2-10 that depend from claim 1 are allowable over Paulraj for at least the same reasons as claim 1 as well as the limitations that are presented in those claims.

Claim 11 calls for a reconfigurable hardware system comprising one or more reconfigurable processors that can instantiate an algorithm as hardware. As noted above with respect to claim 1, Paulraj does not show or suggest even one reconfigurable processor that can instantiate an algorithm as hardware. For at least these reasons claim 11 and claims 12-16 that depend from claim 11 are believed to be allowable over Paulraj.

Independent claim 17 calls for, among other things, transferring data between a memory and a data prefetch unit in a reconfigurable processor. Paulraj does not show or suggest a data prefetch unit, nor does Paulraj suggest transferring data between a memory and a data prefetch unit in a reconfigurable processor. The cited portions of Paulraj deal with retrieving a configuration vector but do not use the work "data prefetch unit" or or describe any functional unit that operates in the same way as a data prefetch unit. Moreover, even if the broad construction set out in the Office action is applied, Paulraj does not suggest configuring the computational unit, data access unit and the data prefetch unit by a program. Paulraj simply cannot suggest this configurability because the computational unit in Paulraj is not configurable. For at least these reasons claim 17 and claims 18-23 that depend from claim 17 are allowable over Paulraj.

Claim 24 as amended is believed to clarify that the term "configured" as used in the claims refers to configuration that allows the configured device to instantiate an algorithm as hardware. Loading a software program into a general purpose computational device such as shown in Paulraj does not result in the instantiation of an algorithm as hardware. Accordingly, claim 24 is believed to be allowable over the relied on reference.

C. **Conclusion.**

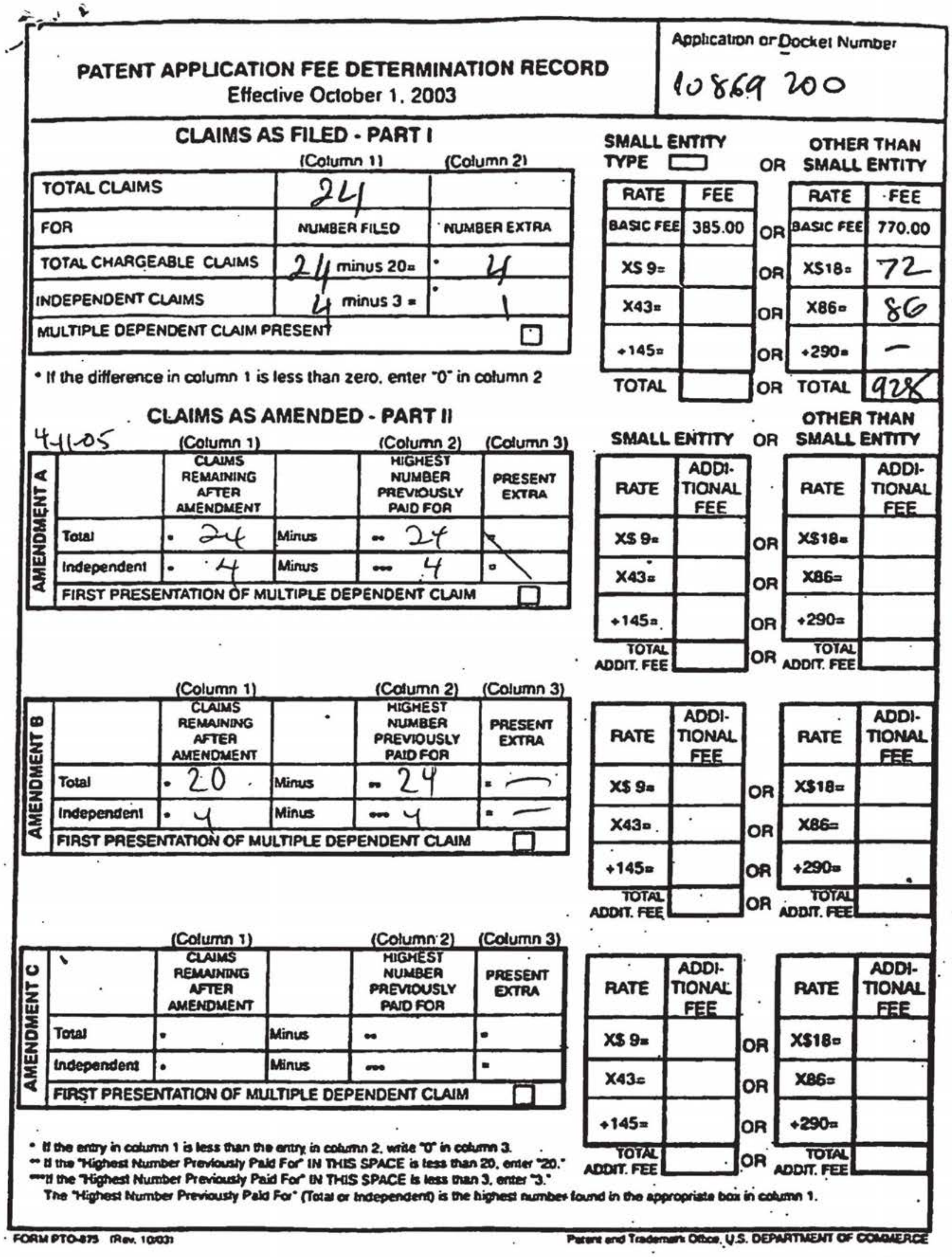
In view of all of the above, the claims are now believed to be allowable and the case in condition for allowance which action is respectfully requested. Should the Examiner be of the opinion that a telephone conference would expedite the prosecution of this case, the Examiner is requested to contact Applicants' attorney at the telephone number listed below.

Any fee deficiency associated with this submittal may be charged to Deposit Account No. 50-1123.

Respectfully submitted,

August 26, 2005

Stuart T. Langley, Reg. No. 33,940
Hogan & Hartson LLP
One Tabor Center
1200 17th Street, Suite 1500
Denver, Colorado 80202
(720) 406-5335 Tel
(303) 899-7333 Fax



PATENT APPLICATION FEE DETERMINATION RECORD

Effective October 1, 2003

Application or Docket Number

10869 200

CLAIMS AS FILED - PART I

	(Column 1)	(Column 2)
TOTAL CLAIMS	24	
FOR	NUMBER FILED	NUMBER EXTRA
TOTAL CHARGEABLE CLAIMS	24 minus 20=	* 4
INDEPENDENT CLAIMS	4 minus 3 =	* 1
MULTIPLE DEPENDENT CLAIM PRESENT		□

* If the difference in column 1 is less than zero, enter "0" in column 2

SMALL ENTITY TYPE □		OR	OTHER THAN SMALL ENTITY	
RATE	FEE		RATE	FEE
BASIC FEE	385.00	OR	BASIC FEE	770.00
XS 9=		OR	XS18=	72
X43=		OR	X86=	86
+145=		OR	+290=	—
TOTAL		OR	TOTAL	928

CLAIMS AS AMENDED - PART II

4-1-05

AMENDMENT A	(Column 1) CLAIMS REMAINING AFTER AMENDMENT		(Column 2) HIGHEST NUMBER PREVIOUSLY PAID FOR	(Column 3) PRESENT EXTRA
Total	* 24	Minus	** 24	=
Independent	* 4	Minus	*** 4	=
FIRST PRESENTATION OF MULTIPLE DEPENDENT CLAIM				□

SMALL ENTITY		OR	OTHER THAN SMALL ENTITY	
RATE	ADDITIONAL FEE		RATE	ADDITIONAL FEE
XS 9=		OR	XS18=	
X43=		OR	X86=	
+145=		OR	+290=	
TOTAL ADDIT. FEE		OR	TOTAL ADDIT. FEE	

AMENDMENT B	(Column 1) CLAIMS REMAINING AFTER AMENDMENT		(Column 2) HIGHEST NUMBER PREVIOUSLY PAID FOR	(Column 3) PRESENT EXTRA
Total	* 20	Minus	** 24	= —
Independent	* 4	Minus	*** 4	= —
FIRST PRESENTATION OF MULTIPLE DEPENDENT CLAIM				□

RATE	ADDITIONAL FEE		RATE	ADDITIONAL FEE
XS 9=		OR	XS18=	
X43=		OR	X86=	
+145=		OR	+290=	
TOTAL ADDIT. FEE		OR	TOTAL ADDIT. FEE	

AMENDMENT C	(Column 1) CLAIMS REMAINING AFTER AMENDMENT		(Column 2) HIGHEST NUMBER PREVIOUSLY PAID FOR	(Column 3) PRESENT EXTRA
Total	*	Minus	**	=
Independent	*	Minus	***	=
FIRST PRESENTATION OF MULTIPLE DEPENDENT CLAIM				□

RATE	ADDITIONAL FEE		RATE	ADDITIONAL FEE
XS 9=		OR	XS18=	
X43=		OR	X86=	
+145=		OR	+290=	
TOTAL ADDIT. FEE		OR	TOTAL ADDIT. FEE	

* If the entry in column 1 is less than the entry in column 2, write "0" in column 3.
** If the "Highest Number Previously Paid For" IN THIS SPACE is less than 20, enter "20."
*** If the "Highest Number Previously Paid For" IN THIS SPACE is less than 3, enter "3."
The "Highest Number Previously Paid For" (Total or Independent) is the highest number found in the appropriate box in column 1.

FORM PTO-875 (Rev. 10/03)

Patent and Trademark Office, U.S. DEPARTMENT OF COMMERCE

43

UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
10/869,200	06/16/2004	Daniel Poznanovic	SRC028	5929

25235 7590 09/01/2005

HOGAN & HARTSON LLP
ONE TABOR CENTER, SUITE 1500
1200 SEVENTEENTH ST
DENVER, CO 80202

EXAMINER	
THOMAS, SHANE M	
ART UNIT	**PAPER NUMBER**
2186	

DATE MAILED: 09/01/2005

Please find below and/or attached an Office communication concerning this application or proceeding.

Advisory Action	**Application No.**	**Applicant(s)**	
Before the Filing of an Appeal Brief	*10/869,200*	POZNANOVIC ET AL.	
	Examiner	**Art Unit**	
	Shane M. Thomas	2186	

--The MAILING DATE of this communication appears on the cover sheet with the correspondence address --

THE REPLY FILED 26 August 2005 FAILS TO PLACE THIS APPLICATION IN CONDITION FOR ALLOWANCE.

1. ☒ The reply was filed after a final rejection, but prior to or on the same day as filing a Notice of Appeal. To avoid abandonment of this application, applicant must timely file one of the following replies: (1) an amendment, affidavit, or other evidence, which places the application in condition for allowance; (2) a Notice of Appeal (with appeal fee) in compliance with 37 CFR 41.31; or (3) a Request for Continued Examination (RCE) in compliance with 37 CFR 1.114. The reply must be filed within one of the following time periods:

a) ☒ The period for reply expires 3 months from the mailing date of the final rejection.

b) □ The period for reply expires on: (1) the mailing date of this Advisory Action, or (2) the date set forth in the final rejection, whichever is later. In no event, however, will the statutory period for reply expire later than SIX MONTHS from the mailing date of the final rejection.

Examiner Note: If box 1 is checked, check either box (a) or (b). ONLY CHECK BOX (b) WHEN THE FIRST REPLY WAS FILED WITHIN TWO MONTHS OF THE FINAL REJECTION. See MPEP 706.07(f).

Extensions of time may be obtained under 37 CFR 1.136(a). The date on which the petition under 37 CFR 1.136(a) and the appropriate extension fee have been filed is the date for purposes of determining the period of extension and the corresponding amount of the fee. The appropriate extension fee under 37 CFR 1.17(a) is calculated from: (1) the expiration date of the shortened statutory period for reply originally set in the final Office action; or (2) as set forth in (b) above, if checked. Any reply received by the Office later than three months after the mailing date of the final rejection, even if timely filed, may reduce any earned patent term adjustment. See 37 CFR 1.704(b).

NOTICE OF APPEAL

2. □ The Notice of Appeal was filed on _____. A brief in compliance with 37 CFR 41.37 must be filed within two months of the date of filing the Notice of Appeal (37 CFR 41.37(a)), or any extension thereof (37 CFR 41.37(e)), to avoid dismissal of the appeal. Since a Notice of Appeal has been filed, any reply must be filed within the time period set forth in 37 CFR 41.37(a).

AMENDMENTS

3. ☒ The proposed amendment(s) filed after a final rejection, but prior to the date of filing a brief, will not be entered because
   (a) ☒ They raise new issues that would require further consideration and/or search (see NOTE below);
   (b) □ They raise the issue of new matter (see NOTE below);
   (c) □ They are not deemed to place the application in better form for appeal by materially reducing or simplifying the issues for appeal; and/or
   (d) □ They present additional claims without canceling a corresponding number of finally rejected claims.
   NOTE: *See Continuation Sheet*. (See 37 CFR 1.116 and 41.33(a)).
4. □ The amendments are not in compliance with 37 CFR 1.121. See attached Notice of Non-Compliant Amendment (PTOL-324).
5. □ Applicant's reply has overcome the following rejection(s): _____.
6. □ Newly proposed or amended claim(s) _____ would be allowable if submitted in a separate, timely filed amendment canceling the non-allowable claim(s).
7. ☒ For purposes of appeal, the proposed amendment(s): a) ☒ will not be entered, or b) □ will be entered and an explanation of how the new or amended claims would be rejected is provided below or appended.
   The status of the claim(s) is (or will be) as follows:
   Claim(s) allowed: _____.
   Claim(s) objected to: _____.
   Claim(s) rejected: *1-24*.
   Claim(s) withdrawn from consideration: _____.

AFFIDAVIT OR OTHER EVIDENCE

8. □ The affidavit or other evidence filed after a final action, but before or on the date of filing a Notice of Appeal will not be entered because applicant failed to provide a showing of good and sufficient reasons why the affidavit or other evidence is necessary and was not earlier presented. See 37 CFR 1.116(e).
9. □ The affidavit or other evidence filed after the date of filing a Notice of Appeal, but prior to the date of filing a brief, will not be entered because the affidavit or other evidence failed to overcome all rejections under appeal and/or appellant fails to provide a showing a good and sufficient reasons why it is necessary and was not earlier presented. See 37 CFR 41.33(d)(1).
10. □ The affidavit or other evidence is entered. An explanation of the status of the claims after entry is below or attached.

REQUEST FOR RECONSIDERATION/OTHER

11. □ The request for reconsideration has been considered but does NOT place the application in condition for allowance because: _____.
12. □ Note the attached Information Disclosure Statement(s). (PTO/SB/08 or PTO-1449) Paper No(s). _____
13. □ Other: _____.

Continuation of 3. NOTE: The amendment to the claims has changed the scope of independent claims 1,11, and 24, and as such, further search and consideration are required.

HONG CHONG KIM
PRIMARY EXAMINER

Client Matter No. 80404.0033.001
Express Mail No.: Via Facsimile

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Serial No. 10/869,200

Application of: Daniel Poznanovic, et al.

Filed: June 16, 2004

Art Unit: 2186

Examiner: Thomas, Shane M.

Attorney Docket No. SRC028

For: SYSTEM AND METHOD OF ENHANCING EFFICIENCY AND UTILIZATION OF MEMORY BANDWIDTH IN RECONFIGURABLE HARDWARE

Confirmation No.: 5929

Customer No.: **25235**

EXPEDITED PROCEDURE UNDER 37 C.F.R. 1.116

Do Not Enter.
SM 8/31/05.

AMENDMENT AND RESPONSE PURSUANT TO OFFICE ACTION DATED JULY 12, 2005

MAIL STOP AF
Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

Sir:

In response to the office communication mailed July 12, 2005 please amend the above-identified application as follows:

Amendments to the Claims are reflected in the listing of claims which begins on page 2 of this paper.

Remarks/Arguments begin on page 6 of this paper.

Index of Claims	**Application No.** 10/869,200	**Applicant(s)** POZNANOVIC ET AL.
	Examiner Shane M. Thomas	**Art Unit** 2186

√	**Rejected**	–	**(Through numeral) Cancelled**	N	**Non-Elected**	A	**Appeal**
=	**Allowed**	÷	**Restricted**	I	**Interference**	O	**Objected**

Claim		Date								
Final	Original	1/5/05	7/6/05	8/30/05						
	1	√	√	√						
	2	√	√	√						
	3	√	√	√						
	4	√	√	√						
	5	√	√	√						
	6	√	√	√						
	7	√	√	√						
	8	√	√	√						
	9	√	√	√						
	10	√	√	√						
	11	√	√	√						
	12	√	√	√						
	13	√	√	√						
	14	√	√	√						
	15	√	√	√						
	16	√	√	√						
	17	√	√	√						
	18	√	√	√						
	19	√	√	√						
	20	√	√	√						
	21	√	√	√						
	22	√	√	√						
	23	√	√	√						
	24	√	√	√						
	25									
	26									
	27									
	28									
	29									
	30									
	31									
	32									
	33									
	34									
	35									
	36									
	37									
	38									
	39									
	40									
	41									
	42									
	43									
	44									
	45									
	46									
	47									
	48									
	49									
	50									

Claim		Date								
Final	Original									
	51									
	52									
	53									
	54									
	55									
	56									
	57									
	58									
	59									
	60									
	61									
	62									
	63									
	64									
	65									
	66									
	67									
	68									
	69									
	70									
	71									
	72									
	73									
	74									
	75									
	76									
	77									
	78									
	79									
	80									
	81									
	82									
	83									
	84									
	85									
	86									
	87									
	88									
	89									
	90									
	91									
	92									
	93									
	94									
	95									
	96									
	97									
	98									
	99									
	100									

Claim		Date								
Final	Original									
	101									
	102									
	103									
	104									
	105									
	106									
	107									
	108									
	109									
	110									
	111									
	112									
	113									
	114									
	115									
	116									
	117									
	118									
	119									
	120									
	121									
	122									
	123									
	124									
	125									
	126									
	127									
	128									
	129									
	130									
	131									
	132									
	133									
	134									
	135									
	136									
	137									
	138									
	139									
	140									
	141									
	142									
	143									
	144									
	145									
	146									
	147									
	148									
	149									
	150									

U.S. Patent and Trademark Office

Part of Paper No. 08302005

09.13.05

RCE
IFW

REQUEST FOR CONTINUED EXAMINATION (RCE) TRANSMITTAL

Address to:
Mail Stop RCE
Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

Application Number	10/869,200
Filing Date	June 16, 2004
First Named Inventor	Daniel Poznanovic, et al.
Group Art Unit	2186
Examiner Name	THOMAS, Shane M.
Attorney Docket Number	SRC028

This is a Request for Continued Examination (RCE) under 37 C.F.R. 1.114 of the above-identified application.
Request for Continued Examination (RCE) practice under 37 CFR 1.114 does not apply to any utility or plant application filed prior to June 8, 1995, or to any design application. See Instruction Sheet for RCEs (not to be submitted to the USPTO) on page 2.

1. Submission required under 37 C.F.R. 1.114 Note: If the RCE is proper, any previously filed unentered amendments and amendments enclosed with the RCE will be entered in the order in which they were filed unless applicant instructs otherwise. If applicant does not wish to have any previously filed unentered amendment(s) entered, applicant must request non-entry of such amendment(s).

a. ☒ Previously submitted. If a final Office Action is outstanding, any amendments filed after the final Office Action may be considered as a submission even if this box is not checked.

i. □ Consider the arguments in the Appeal Brief or Reply Brief previously filed on ______________

ii. □ Other ____

b. □ Enclosed

i. □ Amendment/Reply

ii. □ Affidavit(s)/Declaration(s)

iii. □ Information Disclosure Statement (IDS)

iv. □ Other ____________________

2. Miscellaneous

a. □ Suspension of action on the above-identified application is requested under 37 C.F.R. 1.103(c) for a period of ____________ months. (Period of suspension shall not exceed 3 months; Fee under 37 C.F.R. 1.17(i) **required**)

b. □ Other

3. Fees The RCE fee under 37 C.F.R. 1.17(e) is required by 37 C.F.R. 1.114 when the RCE is filed.

a. ☒ The Director is hereby authorized to charge the following fees, or credit any overpayments, to Deposit Account No. 50-1123

i. □ RCE fee required under 37 C.F.R 1.17(e)

ii. □ Extension of time fee (37 C.F.R 1.136 and 1.17)

iii. ☒ Other: Charge any additional fees or credit any overpayments **for this filing**

b. ☒ Check in the amount of $790.00 enclosed

c. □ Payment by credit card (Form PTO-2038 enclosed)

WARNING: Information on this form may become public. Credit card information should not be included on this form. Provide credit card information and authorization on PTO-2038.

SIGNATURE OF APPLICANT, ATTORNEY, OR AGENT REQUIRED

Name (Print/Type)	Stuart T. Langley	*Registration No. (Attorney/Agent)*	33,940
Signature		*Date*	September 12, 2005

CERTIFICATE OF MAILING OR TRANSMISSION

I hereby certify that this correspondence is being deposited with the United States Postal Service with sufficient postage in an envelope addressed to: Mail Stop RCE, Commissioner For Patents, P.O. Box 1450, Alexandria, VA 22313-1450 or facsimile transmitted to the U.S. Patent and Trademark Office on the date shown below.

Name (Print/Type)	Stuart T. Langley		
Signature		*Date*	September 12, 2005

EXPRESS MAIL NO. EV544475732US
Client/Matter No. 80404.0033.001

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Serial No. 10/869,200	Art Unit: 2186
Application of: Daniel Poznanovic, et al.	Confirmation No.: 5929
Filed: June 16, 2004	Customer No.: **25235**
Examiner: THOMAS, Shane M.	
Attorney Docket No. SRC028	
For: SYSTEM AND METHOD OF ENHANCING EFFICIENCY AND UTILIZATION OF MEMORY BANDWIDTH IN RECONFIGURABLE HARDWARE	

CERTIFICATE OF MAILING BY EXPRESS MAIL

MAIL STOP RCE
Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

Sir:

The undersigned hereby certifies that the following documents:

- Request for Continued Examination;
- Check in the amount of $790.00;
- Certificate of Mailing by Express Mail; and
- Return Receipt Postcard

relating to the above application, were deposited as "Express Mail", Mailing Label No. EV544475732US with the United States Postal Service, addressed to Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-1450.

9/12/05
Date

Stu Langley
Mailer

9/12/05
Date

Stu Langley
Stuart T. Langley, Reg. No. 33,940
HOGAN & HARTSON LLP
One Tabor Center
1200 17th Street, Suite 1500
Denver, Colorado 80202
(720) 406-5335 Tel
(303) 899-7333 Fax

Ref #	Hits	Search Query	DBs	Default Operator	Plurals	Time Stamp
L1	2876	711/170-173.ccls.	US-PGPUB; USPAT	OR	ON	2005/10/15 14:30
L2	30	1 and reconfigurable near3 (processor multiprocessor cache CPU (processing adj unit))	US-PGPUB; USPAT	OR	ON	2005/10/15 14:31
S152	733	((configurable reconfigurable "re-configurable") adj (processor (processing adj unit) CPU microprocessor "micro-processor" cache))	US-PGPUB; USPAT	OR	ON	2005/10/15 09:38
S153	270	S152 and fpga	US-PGPUB; USPAT	OR	ON	2005/10/15 09:38
S154	11	S153 and "711".clas.	US-PGPUB; USPAT	OR	ON	2005/10/15 09:52
S155	20	direct adj execut$3 adj logic	US-PGPUB; USPAT	OR	ON	2005/10/15 09:55
S156	16	memory adj algorithm adj processor	US-PGPUB; USPAT	OR	ON	2005/10/15 11:03
S157	6	"869200".ap.	US-PGPUB; USPAT	OR	ON	2005/10/15 14:30

87

UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
10/869,200	06/16/2004	Daniel Poznanovic	SRC028	5929

25235 7590 10/19/2005

HOGAN & HARTSON LLP
ONE TABOR CENTER, SUITE 1500
1200 SEVENTEENTH ST
DENVER, CO 80202

EXAMINER	
THOMAS, SHANE M	
ART UNIT	**PAPER NUMBER**
2186	

DATE MAILED: 10/19/2005

Please find below and/or attached an Office communication concerning this application or proceeding.

Office Action Summary	**Application No.** 10/869,200	**Applicant(s)** POZNANOVIC ET AL.	
	Examiner Shane M. Thomas	**Art Unit** 2186	

-- The MAILING DATE of this communication appears on the cover sheet with the correspondence address --

Period for Reply

A SHORTENED STATUTORY PERIOD FOR REPLY IS SET TO EXPIRE 3 MONTH(S) OR THIRTY (30) DAYS, WHICHEVER IS LONGER, FROM THE MAILING DATE OF THIS COMMUNICATION.

- Extensions of time may be available under the provisions of 37 CFR 1.136(a). In no event, however, may a reply be timely filed after SIX (6) MONTHS from the mailing date of this communication.
- If NO period for reply is specified above, the maximum statutory period will apply and will expire SIX (6) MONTHS from the mailing date of this communication.
- Failure to reply within the set or extended period for reply will, by statute, cause the application to become ABANDONED (35 U.S.C. § 133). Any reply received by the Office later than three months after the mailing date of this communication, even if timely filed, may reduce any earned patent term adjustment. See 37 CFR 1.704(b).

Status

1)☒ Responsive to communication(s) filed on *12 September 2005*.

2a)□ This action is **FINAL**. 2b)☒ This action is non-final.

3)□ Since this application is in condition for allowance except for formal matters, prosecution as to the merits is closed in accordance with the practice under *Ex parte Quayle*, 1935 C.D. 11, 453 O.G. 213.

Disposition of Claims

4)☒ Claim(s) *1,4-12 and 15-24* is/are pending in the application.

4a) Of the above claim(s) _____ is/are withdrawn from consideration.

5)□ Claim(s) _____ is/are allowed.

6)☒ Claim(s) *1,4-12 and 15-24* is/are rejected.

7)□ Claim(s) _____ is/are objected to.

8)□ Claim(s) _____ are subject to restriction and/or election requirement.

Application Papers

9)□ The specification is objected to by the Examiner.

10)□ The drawing(s) filed on _____ is/are: a)□ accepted or b)□ objected to by the Examiner.

Applicant may not request that any objection to the drawing(s) be held in abeyance. See 37 CFR 1.85(a).

Replacement drawing sheet(s) including the correction is required if the drawing(s) is objected to. See 37 CFR 1.121(d).

11)□ The oath or declaration is objected to by the Examiner. Note the attached Office Action or form PTO-152.

Priority under 35 U.S.C. § 119

12)□ Acknowledgment is made of a claim for foreign priority under 35 U.S.C. § 119(a)-(d) or (f).

a)□ All b)□ Some * c)□ None of:

1.□ Certified copies of the priority documents have been received.

2.□ Certified copies of the priority documents have been received in Application No. _____.

3.□ Copies of the certified copies of the priority documents have been received in this National Stage application from the International Bureau (PCT Rule 17.2(a)).

* See the attached detailed Office action for a list of the certified copies not received.

Attachment(s)

1) □ Notice of References Cited (PTO-892)

2) □ Notice of Draftsperson's Patent Drawing Review (PTO-948)

3) □ Information Disclosure Statement(s) (PTO-1449 or PTO/SB/08) Paper No(s)/Mail Date _____.

4) □ Interview Summary (PTO-413) Paper No(s)/Mail Date. _____ .

5) □ Notice of Informal Patent Application (PTO-152)

6) □ Other: _____.

DETAILED ACTION

This Office action is responsive to the amendment filed 8/26/2005. Claims 1,11, and 24. have been amended; claims 2,3,13, and 14 have been canceled. Claims 1,4-12, and 15-24 are pending.

Continued Examination Under 37 CFR 1.1 1 4

A request for continued examination under 37 CFR 1.114, including the fee set forth in 37 CFR 1.17(e), was filed in this application after final rejection on 9/12/2005. Since this application is eligible for continued examination under 37 CFR 1.1 14, and the fee set forth in 37 CFR 1.17(e) has been timely paid, the finality of the previous Office action has been withdrawn pursuant to 37 CFR 1.114. Applicant's submission filed on 8/26/2005 has been entered.

All previously outstanding objections and rejections to the Applicant's disclosure and claims not contained in this Action have been respectfully withdrawn by the Examiner hereto.

Response to Amendment

The rejections of claims 1,11,17, and 24 have been modified to reflect the amendments and/or Applicant's arguments to the respective claims.

Response to Arguments

Applicant's arguments filed 8/26/2005 have been fully considered but they are not persuasive for the following reasons.

Applicant argues on page 6 of the response that the prior art reference of Paulraj "does not show or suggest a reconfigurable processor that instantiates an algorithm as hardware." The Examiner respectfully traverses. Paulraj teaches in the abstract for one, that the system described determines "an optimal configuration of memory for a particular *application*." The Applicant teaches in ¶55 of the originally filed disclosure that "any computer program [i.e. application] is a collection of algorithms." Therefore it can be seen that since the processor 100 of Paulraj can reconfigure the memory 104 based on the application (or computer program) that is to execute on the processor, that so to can the reconfigurable processor system of Paulraj "instantiate an algorithm (i.e. an application) as hardware (i.e. the FPGA module 104 that is used as a cache memory)."

As per the Applicant's arguments regarding claim 11, the Examiner has shown in above in the discussion of claim 1 that Paulraj teaches a reconfigurable processor 100, as claimed by the Applicant, that instantiates an algorithm as hardware.

As per the Applicant's arguments regarding claim 17 on page 7, the Applicant argues that the prior art reference of Paulraj "does not show or suggest a data prefetch unit, nor suggest transferring data between a memory and a data prefetch unit in a reconfigurable processor. As explained in the Examiner's previous rejection of claim 17, the Examiner is considering the reconfiguration unit 106 of Paulraj to be a --data prefetch unit-- since Paulraj teaches that the unit 106 *prefetches* a configuration vector (i.e. retrieves data from an inherent and non-shown memory) and sets up a programmable memory module 104 (i.e. cache) *before* executing the application relating to the configuration vector (refer to ¶24 and ¶29). Figure 4 of Paulraj clearly shows the --data prefetch unit-- 106 being in a reconfigurable processor 100. Although the cited reference does not explicitly use the phrase "data prefetch unit," and may or may not perform all of the functionality of a "data prefetch unit," as discussed in the Applicants disclosure, the reconfiguration unit 106 performs the ***claimed*** *functionality* of the "data prefetch unit" as discussed above (i.e. merely transferring data between a memory in a reconfigurable processor).

Further, the Applicant argues regarding claim 17 that "Paulraj does not suggest configuring the computational unit, data access unit, and the data prefetch unit by a program. Paulraj simply cannot suggest this configurability because the computational unit in Paulraj is not configurable." The Examiner respectfully traverses. All of the computational, data access, and data prefetch units are configured by a program, as immediately discussed. As defined by the Examiner, the "computational unit" of Paulraj is being considered to be the element of the system of Paulraj that executes and collects the performance data regarding how a specific application utilizes memory in order to determine an optimal memory configuration as discusses in ¶27. Figure 5 of Paulraj shows a method for creating a configuration vector by using the --computational unit-- in steps 204-206. The Examiner is considering the inherent *program* that is being executed in order to perform the steps of figure 5 to be the *program* that configures the computational unit. Therefore, it can be seen that Paulraj *does* suggest configuring the computational unit by a program. The *program* of figure 5 *configures* the computational unit to collect data for a specific application's memory usage statistics in order to create a configuration vector that allows the system of Paulraj to optimally reconfigure the programmable memory module 104. Thus the computational unit can be configured to collect memory usage statistics for a plurality of applications that are to be executed by the reconfigurable processor 100 of Paulraj (¶23).

The same reasoning applies to the data access and data prefetch units. The *program* that is executing the steps of figure 5 (i.e. running on the system of Paulraj that implements the method) *configures* the data access unit to retrieve/store a configuration vector (step 212) based on if a new configuration vector had to be created and further *configures* the data prefetch unit to search for a configuration vector and retrieve that vector if found (steps 200 and 212).

As per the Applicant's arguments regarding claim 24 "that loading a software program into a general purpose computational device such as shown in Paulraj does not result in the instantiation of an algorithm as hardware." The Examiner respectfully traverses. Once the software program has been loaded into the computational unit, a variety of simulations are performed and memory usage statistics are gathered by the computational unit in order to create a configuration vector as taught in ¶¶23-24. This vector allows the programmable memory module 104 of Paulraj to be reconfigured to the most optimal memory configuration for that specific software program (¶26). As discussed supra, a software program or application is a collection of "algorithms"; therefore, the configuration vector for a particular software program allows the system of Paulraj to instantiate a software program as hardware since the configuration vector represents optimal configuration of the hardware (programmable memory module 104 - element 112 of figure 6).

Claim Rejections - 35 USC § 102

The following is a quotation of the appropriate paragraphs of 35 U.S.C. 102 that form the basis for the rejections under this section made in this Office action:

> A person shall be entitled to a patent unless –
>
> (e) the invention was described in (1) an application for patent, published under section 122(b), by another filed in the United States before the invention by the applicant for patent or (2) a patent granted on an application for patent by another filed in the United States before the invention by the applicant for patent, except that an international application filed under the treaty defined in section 351(a) shall have the effects for purposes of this subsection of an application filed in the United States only if the international application designated the United States and was published under Article 21(2) of such treaty in the English language.

Claims 1-24 are rejected under 35 U.S.C. 102(e) as being anticipated by Paulraj (U.S. Patent Application Publication No. 2003/0084244).

As per claim 1, Paulraj shows a reconfigurable processor in figure 6 and a first memory (L1) having a first characteristc memory utilization and a second memory (L2) having a second characteristic memory utilization. It is well known in the art that L1 caches have a higher utilziation rate than a lower-level cache such as L2. Paulraj teaches in ¶1 that upon a command from a processor, a search for the requested data is begines with the highest level cache (L1) and [if a miss occurs] continues next to the next level cache (L2). Thus it is inherent that the memory utilziation characteristc of the L1 cache of the reconfigurable processor 110 in figure 6 is greater than the memory utilziation characteristic of the L2 cache (and likewise for the L3 cache) as the L2 cache would only be utilzied when a miss to the L1 cache occurred. In other words, the reconfigurable processor *always* utilizes the L1 cache for a memory access and the *only* utilzies the L2 cache for requested data when the data is not in the L1 cache. Therefore, the cache utilziation characteristics of the --first memory-- and the --second memory-- are different.

Paulraj further teaches a functional unit 102 that executes applications using the memories L1 and L2 (paragraph 9). As is known in the art, a cache memory controller is often used to access and move data between a memory hierarchy. The Examiner is considering a data prefetch unit to be the logic assocatied with the moving, and only the moving, of data between the first and second memories (L1 and L2) since Paulraj shows a connection between the levels of cache in figure 6. This logic as well as the first and second memory types (L1 and L2) are configued by a program – refer to paragraphs 23-24. The data prefetch unit as defined by the Examiner must be configued as well by the program when moving data since the cache line size and blocking factor can change, so different amounts of data can be exchanged for the same access when different programs run.

The reconfigurable processor of Paulraj has the ability to collect memory usage statistics for a particular application and based on those statistics, create a configuration vector as taught in ¶¶23-24. This vector allows the programmable memory module 104 of Paulraj to be reconfigured to the most optimal memory configuration for that specific software program (¶26). As defined by the Applicant in ¶55 of the originally filed specification, a software program or application is a collection of "algorithms"; therefore, the configuration vector for a particular software program allows the system of Paulraj to instantiate a software program as hardware since the configuration vector represents optimal configuration of the hardware (programmable memory module 104 - element 112 of figure 6).

As per claims 2 and 13, as taught in paragraphs 23 and 29 of Paulraj, no specific cache is present in the system of Paulraj. Rather, an FPGA is utilized as representing a caching hierarchy and is optimized based on the memory needs of a specific program running on the reconfigurable processor.

As per claims 3 and 14, Paulraj teaches in paragraph 23 that a specific [cache] line size of contiguous data is not retrieved since the data line size is optimized based on the memory needs of the program when executing on the reconfigurable processor. Refer also to paragraph 29. Further, it is therefore inherent that the second memory have a charactersitic line size since Paulraj teaches in ¶¶22-23 that a best line size for the memory arrangement for a particular program is determined and utilzied when that program is run. For example, a line-size characteristic would be ultized when transferring data from the L2 cache to the L1 cache.

As per claim 4, Paulraj teaches that a load/store unit is used to access the caches (L1-L3) in order to determine if cache data is present in the cache hierarchy (paragraph 6). Since the functional unit 102 (figure 6) is responsible for accessing the programmable memory unit 104, the Examiner is therefore considering the load/store unit logic of the programmable memory unit that is responsible for for accessing the L1 and L2 caches (first and second memory types) to be a memory controller. It can be seen that the memory controller, as defined by the Examiner, controls the transfer of data between the memory (assuming second memory L2) and the data prefetch unit, since the memory controller (load/store unit logic) is responsible for retrieving the data from the cache if a hit occurs (paragraph 4).

As per claim 5, as taught in paragraph 1, an external memory (element 18, figure 1) is generaly coupled to a microprocessor and holds data to be used by the microcontroller during program execution. The Examiner is considering the process of writing data back to the external memory from the FPGA memory 104 containing the caches (on-board memory), such as during a write-back scheme as known in the art, to be performed by the data prefetch unit portion of the functional logic as defined above by the Examiner. The data prefetch logic, as defined above, is responsible for all of the transfer of data into, out of, and between the FPGA memory 104.

As per claim 6, the Examiner is regarding a --register-- in its broadest reasonable sense and it thus considering it be to be a unit of logic. Therefore, the portion of the function logic that is responsible for the movement of data (as defined above to be the data prefetch unit) is being considered by the Examiner as containing a --register-- portion of the reconfigurable processor since, for instance, the blocking factor and line size of the programmable memory 112 can change, a --register-- or portion of the reconfigurable processor must be set in order to indicate the currnet line size and blocking factor when a given application is being run on the reconfigurable processor at a given point in time. Refer to paragraph 23.

As per claim 7, the Examiner is considering the process of --disassembling the data prefetch unit-- as modifying the data prefetch unit logic of the fucntion logic 102 every time the program being executed by the reconfigurable processor changes. It can be seen that the data prefetch unit changes during these intervals since the cache line size, blocking factor, and associativity of the FPGA changes when optimal for the next program to be executed (refer to paragraph 23). Thus it can be seen that the data prefetch unit logic is --disassembled-- when another program is executed by the reconfigurable processor of Paulraj.

As per claim 8, as can be seen that the FPGA memory 112, that comprises the first and second memories (L1 and L2) and which is accessed by the data prefetch unit of the functional unit 102 as discussed above, is a --processor memory-- (part of cpu 110). It can also be seen that the --second memory-- (L2) is also a --processor memory-- since it is contained within reconfigurable processor 110. Therefore, since the data pretech unit can access the L2 cache as discussed above in the rejection of claim 1, the data prefetch unit can retrive data from the L2 portion of --processor memory--112.

As per claim 9, as shown in figure 1 and taught in paragraph 1 of Paulraj, the system 10 is actually a microprocessor, which contains a memory controller 14. The main difference between the prior art of figure 1 and the invention of Paulraj in figure 6 is that the memroy hierarchy is configurable and accessed by a fucntional unit in lieu of a separate memory controller logic (paragraph 9). Therefore, since the memory controller logic for accessing the cache hierarchy is still contained within cpu 110 of figure 6, it can be seen that the cpu 110 is actually a microprocessor. It follows that the --processor memory-- 112 is therefore a --microprocessor memory--.

As per claim 10, since the cpu 110 of figure 6 is a reconfigurable processer (able to reconfigure its memory heirarchy to match the needs of the application it is currently running), it can be seen that the cpu memory 112 is a reconfigurable processor memory.

As per claim 11, Paulraj depicts a reconfigurable hardware system in figure 6. Paulraj further teaches in paragraph 26 that when a particular application is to be run by the reconfigrable processor 110, a configuration vector is retrieved to program the programmable memory 112 (figure 6). As shown in figure 6, the step of accesing the configuration vector is executed outside of the reconfigurable processor 110. Therefore, the Examiner is considering the memory that contains the configuration vectors to be a--common memory-- and a data prefetch unit (reconfiguration unit 106 executing on the reconfigurable processor 110) accessing the common memory in order to determine how to program the memory 112 (paragraph 29).

The data prefetch unit 106 is --configured-- by an application to be excuted on the sysem 110 since when a new application is to be executed, the data prefetch unit is called upon (or configured) to access the configuration vector for the particular application.

The reconfigurable processor of Paulraj has the ability to collect memory usage statistics for a particular application and based on those statistics, create a configuration vector as taught in ¶¶23-24. This vector allows the programmable memory module 104 of Paulraj to be reconfigured to the most optimal memory configuration for that specific software program (¶26). As defined by the Applicant in ¶55 of the originally filed specification, a software program or application is a collection of "algorithms"; therefore, the configuration vector for a particular software program allows the system of Paulraj to instantiate a software program as hardware since the configuration vector represents optimal configuration of the hardware (programmable memory module 104 - element 112 of figure 6).

As per claim 12, the Examiner is considering a --memory controller-- to be the system portion utilized when creating a new configuration vector for an application. Such a process occurs in figure 5 and taught in paragraghs 23-25 of Paulraj. When a new configuration vector is created by analizing performance information that has been collected for the application. The Examiner is thereby considering the --memory controller-- to be the element of the reconfigurable hardware system that is associated with storing the new configuration vector into the common memory so that the vector can be accessed later when the same application is run again.

As per claim 15, the Examiner is considering the reconfiguration module 106 of the reconfigurable processsor 110, as comprising two distinct elements: a --computational unit-- and a --data access unit--. The data access unit is the element that is responsible for accessing the configuration vector as taught in paragraph 29 of Paulraj; or in other words, the Examiner is considering the --data access unit-- to be the same as the --memory controler-- defined in the rejection of claim 12. The Examiner is further considering the --computational unit-- of the rconfiguration module 106 to be the element that sets up the programmable memory module 104 using the configuration vector that was accessed by the --data access unit-- (paragraph 29).

As per claim 16, as taught by Paulraj in paragraph 29, the --data access unit-- supplies the configuration vector to the --computational unit-- in order to set up the programmable memory 104 as required by the application to be run on the reconfurable processor 110.

As per claim 17, the Examiner is considering a --data prefetch unit-- to be the reconfiguration unit 106 of reconfigurable processor 110 (figure 6). As taught in paragraph 26 and 29 of Paulraj, the --data prefetch unit-- accesses a memory in order to determine if a configuration vector is known for a given application, and if so, the vector is retrieved (from the memory). If this --data-- (configuration vector) is not known then a simulation is performed with the application in order to collect performance information. The Examiner is considering the element that executes and collects the performance data as being a --computational unit-- and the element of Paulraj that stores the configuration vector, once determined, to be a --data access unit-- since it stores the vector into the --memory-- from which it can be later retrieved (step 212 of figure 5).

All of the computational, data access, and data prefetch units are configured by a program, as immediately discussed. As defined by the Examiner, the "computational unit" of Paulraj is being considered to be the element of the system of Paulraj that executes and collects the performance data regarding how a specific application utilizes memory in order to determine an optimal memory configuration as discusses in ¶27. Figure 5 of Paulraj shows a method for creating a configuration vector by using the --computational unit-- in steps 204-206. The Examiner is considering the inherent *program* that is being executed in order to perform the steps of figure 5 to be the *program* that configures the computational unit. Therefore, it can be seen that Paulraj suggests configuring the computational unit by a program. The *program* of figure 5 *configures* the computational unit to collect data for a specific application's memory usage statistics in order to create a configuration vector that allows the system of Paulraj to optimally reconfigure the programmable memory module 104. Thus the computational unit can be configured to collect memory usage statistics for a plurality of applications that are to be executed by the reconfigurable processor 100 of Paulraj (¶23).

The same reasoning applies to the data access and data prefetch units. The *program* that is executing the steps of figure 5 (i.e. running on the system of Paulraj that implements the method) *configures* the data access unit to retrieve/store a configuration vector (step 212) based on if a new configuration vector had to be created and further *configures* the data prefetch unit to search for a configuration vector and retrieve that vector if found (steps 200 and 212).

As per claim 18, the --data-- (configuration vector) is transferred from the --computational unit-- to the --data access unit-- when the configuration unit has created a configuration vector (step 208 of figure 5). The --data-- is written to the memory --from-- the --data prefetch unit-- since the data prefetch unit (reconfiguration unit 106) is the element that executed the beginning of the configuration vector creation process (step 200 of figure 5). Refer to paragraph 26. Thus the Examiner is considering the data as being written --from-- the data prefetch unit.

As per claim 19, as taught in paragraph 26, if the configuration vector is known, the vector is retrieved from the memory to the data prefetch unit (reconfiguration unit 106). The data is read directly from the data prefetch unit when a request to create a configuration vector is made for a new application as shown in figure 6 since the data prefetch unit is responsible for being the vector creation process. The data is directed from the data prefetch unit (reconfigure logic) to be read from the memory by the data access unit to the computational unit where it is processed to produce a configuration vector.

As per claim 20, as stated above, the configuration vector (--data--) is created by the computational unit via acquired simulation data. The configuration vector is the resultant product that is transferred from the memory to the data prefect unit when it is determined that the configuration vector for the application is available (paragraph 26). Thus --all-- of the data that is transferred is processed by the computational unit (albeit before the transfer occurs) since the data prefetch unit required the entire configuration vector in order to set up the programmable memory 112.

As per claim 21, Paulraj shows in paragraph 26 that an explicit request for the configuration vector for the current application results in the data (if it exists) selected for the optimal configuration of the programmable memory 112 for that application.

As per claim 22, the Examiner is not considering the data (configuration vector) to be the size of a complete cache line since the data is used to create a cache hierarchy. In other words, the caches (L1-L3) of the programmable memory 112 are not programmed when the data is transferred from the memory to the data prefetch unit; therefore, the data cannot be a complete cache line.

As per claim 23, since the Examiner defined the portion of the reconfiguration unit that accesses the configuration file (data) from the memory, the Examiner is defining the logic that controls the actual transfer of that data to the data prefetch unit (portion of the reconfiguration unit that executes the fetch of the configuration vector and then programs the programmable memory 112) to be a --memory controller--. Thus the data access unit determines whether a configuration vector exists for an application and if so, the memory controller sends that data to the data prefetch unit.

As per claim 24, Paulraj shows a reconfigurable processor in figure 6 that comprises a computation unit 110 and a data access unit (elements 120 and 114, which comprise the reconfiguration unit 106 of figure 4 - ¶28). In figure 6, the data access unit can be seen as being coupled to the computational unit. The data access unit retrieves data (configuration vector) from a memory internal to the data access unit (i.e. reconfiguration unit) and supplies the data to the computation unit in the form of modifications to the cache FPGA module 112. Refer to ¶23.

The Examiner is considering the inherent *program* that is being executed in order to perform the steps of figure 5 to be the *program* that configures the computational unit. Therefore, it can be seen that Paulraj suggests configuring the computational unit by a program. The *program* of figure 5 *configures* the computational unit to collect data for a specific application's memory usage statistics in order to create a configuration vector that allows the system of Paulraj to optimally reconfigure the programmable memory module 104. Thus the computational unit can be configured to collect memory usage statistics for a plurality of applications that are to be executed by the reconfigurable processor 100 of Paulraj (¶23).

The data access unit (specifically the memory portion used to store configuration profiles for the different application programs) is configured by the *program* that is responsible for running the method of figure 5 of Paulraj as discussed supra. When a new application is to be run, [as a result] the *program* performs the steps 204-206 to configure the reconfiguration unit to collect statistics regarding the memory usages (caches L1, L2, and L3) of the application and a configuration vector is associated with the respective application and stored in the reconfiguration unit. Refer to ¶¶23-24. When an application is known, the program executing the method of figure 5 [as a result] configures the data access unit (reconfiguration unit) to retrieve the associated configuration vector and apply it to the FPGA memory of the reconfigurable processor (¶29).

In other words, once the software program has been loaded into the computational unit, a variety of simulations are performed and memory usage statistics are gathered by the computational unit in order to create a configuration vector as taught in ¶¶23-24. This vector allows the programmable memory module 104 of Paulraj to be reconfigured to the most optimal memory configuration for that specific software program (¶26). As discussed supra, a software program or application is a collection of "algorithms"; therefore, the configuration vector for a particular software program allows the system of Paulraj to instantiate a software program as hardware since the configuration vector represents optimal configuration of the hardware (programmable memory module 104 - element 112 of figure 6).

Conclusion

Any inquiry concerning this communication or earlier communications from the examiner should be directed to Shane M. Thomas whose telephone number is (571) 272-4188. The examiner can normally be reached on M-F 8:30 - 5:30.

If attempts to reach the examiner by telephone are unsuccessful, the examiner's supervisor, Matt M. Kim can be reached on (571) 272-4182. The fax phone number for the organization where this application or proceeding is assigned is 571-273-8300

Information regarding the status of an application may be obtained from the Patent Application Information Retrieval (PAIR) system. Status information for published applications may be obtained from either Private PAIR or Public PAIR. Status information for unpublished applications is available through Private PAIR only. For more information about the PAIR system, see http://pair-direct.uspto.gov. Should you have questions on access to the Private PAIR system, contact the Electronic Business Center (EBC) at 866-217-9197 (toll-free).

Shane M. Thomas

HONG CHONG KIM
PRIMARY EXAMINER

Index of Claims

Application No. 10/869,200	**Applicant(s)** POZNANOVIC ET AL.	
Examiner Shane M. Thomas	**Art Unit** 2186	

√	**Rejected**	**–**	**(Through numeral) Cancelled**	**N**	**Non-Elected**	**A**	**Appeal**
=	**Allowed**	**÷**	**Restricted**	**I**	**Interference**	**O**	**Objected**

Claim		Date								
Final	Original	1/5/05	7/6/05	10/15/05						
	1	√	√	√						
	~~2~~	√	√							
	~~3~~	√	√							
	4	√	√	√						
	5	√	√	√						
	6	√	√	√						
	7	√	√	√						
	8	√	√	√						
	9	√	√	√						
	10	√	√	√						
	11	√	√	√						
	12	√	√	√						
	~~13~~	√	√							
	~~14~~	√	√							
	15	√	√	√						
	16	√	√	√						
	17	√	√	√						
	18	√	√	√						
	19	√	√	√						
	20	√	√	√						
	21	√	√	√						
	22	√	√	√						
	23	√	√	√						
	24	√	√	√						
	25									
	26									
	27									
	28									
	29									
	30									
	31									
	32									
	33									
	34									
	35									
	36									
	37									
	38									
	39									
	40									
	41									
	42									
	43									
	44									
	45									
	46									
	47									
	48									
	49									
	50									

Claim		Date								
Final	Original									
	51									
	52									
	53									
	54									
	55									
	56									
	57									
	58									
	59									
	60									
	61									
	62									
	63									
	64									
	65									
	66									
	67									
	68									
	69									
	70									
	71									
	72									
	73									
	74									
	75									
	76									
	77									
	78									
	79									
	80									
	81									
	82									
	83									
	84									
	85									
	86									
	87									
	88									
	89									
	90									
	91									
	92									
	93									
	94									
	95									
	96									
	97									
	98									
	99									
	100									

Claim		Date								
Final	Original									
	101									
	102									
	103									
	104									
	105									
	106									
	107									
	108									
	109									
	110									
	111									
	112									
	113									
	114									
	115									
	116									
	117									
	118									
	119									
	120									
	121									
	122									
	123									
	124									
	125									
	126									
	127									
	128									
	129									
	130									
	131									
	132									
	133									
	134									
	135									
	136									
	137									
	138									
	139									
	140									
	141									
	142									
	143									
	144									
	145									
	146									
	147									
	148									
	149									
	150									

Search Notes	Application/Control No. 10/869,200	Applicant(s)/Patent under Reexamination POZNANOVIC ET AL.	
	Examiner Shane M. Thomas	Art Unit 2186	

SEARCHED			
Class	Subclass	Date	Examiner

SEARCH NOTES (INCLUDING SEARCH STRATEGY)		
	DATE	EXMR
Updated East Search	10/15/2005	SMT
711/170-173 (text search only - see search printout)	10/15/2005	SMT

INTERFERENCE SEARCHED			
Class	Subclass	Date	Examiner