Extended Memory Semantics on Hybrid FPGA-x86 Architectures

Jace A. Mogill

Pacific Northwest National Laboratory
Richland, WA, USA
jace.mogill@pnl.gov

ABSTRACT:

The principal bottleneck in conventional High Performance Computing (HPC) clusters derives from data movement, not inefficient use of computational cores. Multi-Threaded Architectures (MTAs) such as the Tera MTA and Cray XMT attempt to overcome this weakness using parallelism to mask latency and maximize bandwidth utilization. We propose the use of hybrid (FPGA) Field Programmable Gate Array/x86 systems to implement on commodity hardware the key architectural features of those two systems. When integrated through a CPU socket, the FPGA becomes part of the CPU load/store memory model making it possible to alter the performance and semantics of ordinary x86 memory operations. Extended memory semantics such as full/empty bits can be managed by the FPGA and enforced through manipulation of the existing cache coherency protocol, latency hiding CPU context switches can by forced by the FPGA issuing CPU interrupts, and distributed shared memory can be implemented by extending the inter-processor protocol over Scalable Coherent Interface (SCI).

KEYWORDS: FPGA, Hybrid Architecture, Programming Models, Cray XMT, Parallelism

1 Introduction

Data intensive computing requires systems which make possible programming and execution models that are efficient on irregular data structures and workloads. These characteristics are captured in graph problems, such as traversal of power law graphs like those found in social networks, which do not yield to traditional data or work decomposition techniques. Decomposition by graph vertex is not possible when the number of out bound edges exceeds the size of single compute node, communication and computation bottlenecks form on nodes with high degree vertexes, and any vertex may connect to any other vertex, defeating decomposition techniques which exploit data locality [1]. For applications with these traits, extended memory semantics (EMS) multi-threaded architectures like the Tera MTA and Cray XMT are demonstrated to outperform high performance computer clusters, are programmed using a simpler programming model, and dissipate about $1/6^{th}$ the power of an x86 processor [2–5].

The advantages of the Tera MTA and Cray XMT systems stem from three architectural features: Globally shared memory, fast context switches, and extended memory semantics. In figure 1 these three key architectural features define a design space for multi-node systems with a data-centric, not thread-centric, programming and execution model. The custom hardware Tera MTA and Cray XMT are in the center of this design space, components of hybrid hard-

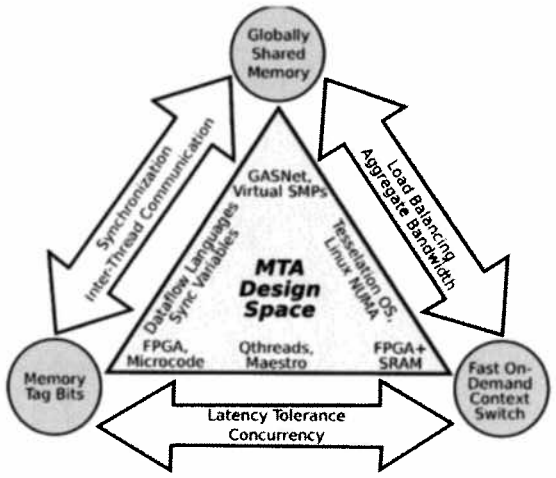

Figure 1: Key architectural features of the Tera MTA and Cray XMT.

ware implementations fall into the corners of the triangle, and software constructs tie discrete the hardware features together.

The Cray XMT is the successor system to the Tera MTA, implementing the same instruction set and extended memory semantics in different technology and packaging. Both systems share a data-centric programming model which more closely resembles a Dennis data flow execution model than the multiple vonNeumann sequential processors of a multicore x86 [6]. Tera MTA and Cray XMT processors both feature a single-issue fully pipelined execution core shared by 128 thread contexts called *streams*, which the processor automatically context switches between, every clock period, without OS participation. Each stream may issue up to 8 instructions out of order for a total of 1024 outstanding memory operations per processor. With hundreds of in-flight instructions the processor is able to maintain high utilization because most clock cycles it is able to find an instruction whose dependencies have all been satisfied and may be retired. The need for a cache hierarchy meant to minimize latency is eliminated by the latency hiding capability, and all memory operations effectively have no time cost.

The Tera MTA is an example of early efforts in hardware-software co-design [7], wherein hardware capabilities were designed specifically to support the programming and execution models being concurrently designed to be efficient on the selfsame hardware. In the spirit of co-design, this paper is organized to cover not only the hardware architecture of the hybrid system, but the operating system, runtime libraries, compilers, applications, and integration into existing compute facilities.

1.1 XMT Architectural Features

The individual architectural features of the XMT are each useful in their own right, but when combined emergent properties present alternative programming and execution models. When considered as a single feature, full/empty memory tag bits and shared memory provide equivalent functionality to message passing on clusters, serving as a place for communication between threads, task synchronization, and enforcing atomicity of composed operations. Load balancing is a prerequisite for scalability and full utilization of distributed resources like processors, memory bandwidth, and interconnect bandwidth. Shared memory and fast context switches taken together provide freedom in scheduling tasks and continuations which afford load balancing almost impossible to achieve in message passing programs. Extended memory semantics and fast on-demand context switches combine to provide latency tolerance and increased system utilization by forcing CPU cores to abort the stalled memory operations and context switch to another task which can make progress.

1.2 x86 Architectural Features

Modern x86 processor architectures use a link based interprocessor communication protocol such as HyperTransport (HT) or QuickPath Interconnect (QPI) instead of bus architectures to implement a cache coherent single system image from multiple processors [8, 9]. Link based interconnects improve upon busses by eliminating electrical and snooping protocol limitations which prevent scaling in performance and system

size. The interprocessor communication network carries all signals between processors and the I/O subsystem, including memory loads and stores, peripheral device commands and data, interrupt and low-level system control messages, and side-band traffic.

Both HT and QPI implement design decisions which trade higher performance for lower power and cost implementations by limiting the system size to 8 or fewer sockets. To scale to larger systems, data traffic inside HT or QPI networks can bridge to a Scalable Coherent Interface (SCI) network which does efficiently implement large scale systems [10]. SCI is a functional superset of HT and QPI, defining a processor memory bus, I/O bus, high performance network and switch, and a distributed directory based cache coherency for a global shared memory model.

Ordinary commodity x86 mother boards wire standard DDR3 memory modules directly to a processor where an embedded memory controller services loads and stores from any device on the interprocessor fabric. When a system is booted from a cold start, the devices on the interprocessor fabric negotiate for unique physical address ranges to assign to local resources. The physical address ranges of memory and I/O devices does not change once the system is booted, making it possible for software to reason about and enforce data locality in the non-uniform memory architecture (NUMA). An example showing the physical address ranges for different devices on the interprocessor fabric is shown in figure 2.

Not all devices implement all layers of functionality of the interprocessor protocol. For example, the HyperTransport protocol has both cache-coherent and non-coherent levels of implementations, cache-coherency being necessary for devices which make copies of data that are also held in other memory or caches. Cache-coherent devices are more complicated to design, implement, and test than their non-coherent counterparts, and have additional costs in licensing fees. Memory address ranges managed by non-coherent devices have the uncachable bit set in the memory's Page Attribute Table, meaning loads and stores to the addresses are never satisfied by a copy in cache, and probes are responded to only by the device managing the address.

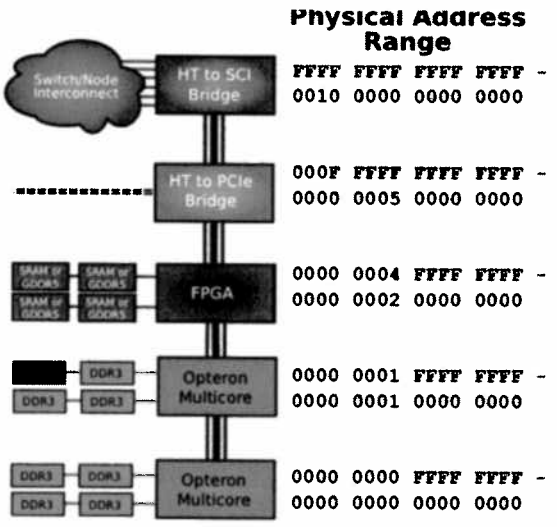

Figure 2: A simplified logical view of the physical address ranges associated with devices on the interprocessor fabric.

2 FPGA-x86 Hybrid Architectures

Several commercially available FPGA/x86 hybrid systems are based on designs in which one or more of the CPU sockets of a standard x86 server are populated with a small board containing one or more FPGAs and a small amount of SRAM memory [11,12]. A logical block diagram of such a system is shown in figure 3. The reconfigurable device is programmed to use the processor interconnect protocol, and can participate as a non-cache-coherent I/O device, or participate fully in the cache coherent virtual memory domain as a specialized processor. Research using this kind of hybrid reconfigurable systems has been limited to application acceleration by encoding a specific algorithm directly in the FPGA's logic, or by implementing a domain specific soft-core processor [13]. In contrast to these models, we introduce the novel use of the FPGA to modify memory performance and semantics independently of the software running on the system.

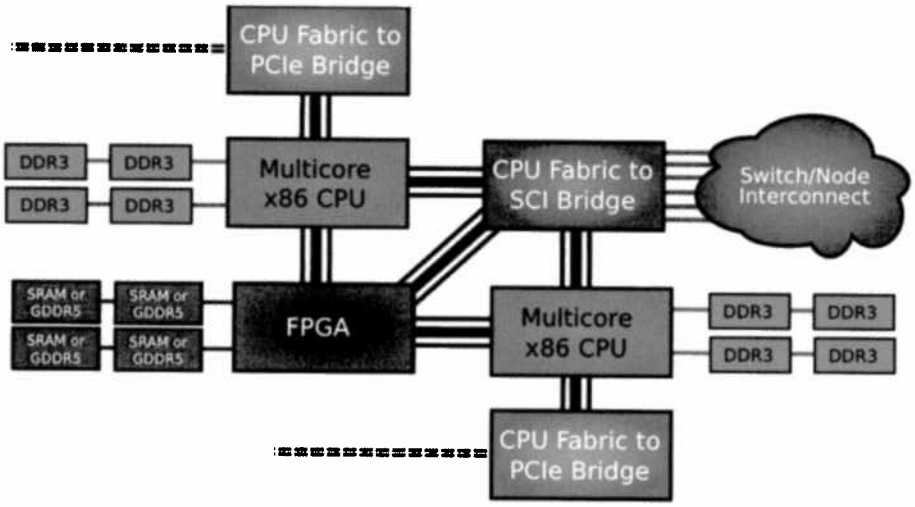

Figure 3: Block diagram of a hybrid multi-threaded architecture. Green blocks represent ordinary commodity devices, blue blocks are custom or reconfigurable devices, red blocks are commercially available Virtual SMP devices, and gray boxes are the part of the motherboard chipset.

2.1 Custom Memory Devices

Processor interconnect protocols do not specify how memory operations are performed, nor do they require any particular implementation of physical memory, meaning devices are free to support many different types of memory modules. The memory controllers built into conventional x86 processors typically support several kinds of memory modules in various densities operating at different speeds, however only the address range is known to other devices. When a FPGA populates a CPU socket, the FPGA must have a memory controller to access any directly connected memory modules, but unlike a x86 processor, a FPGA may be programmed with memory controllers for any kind of commodity or custom memory devices used in any combination. Memory devices must conform to the physical form factor of the board's original memory modules, otherwise they may be customized for random access, low latency, non-volatility, or any other need. To provision dedicated fast bit-level atomic random access memory for tag bits (i.e.: full/empty), custom SRAM or GDDR5 parts might be used in the system's DDR3 memory slots.

The memory provided by modules in the FPGA managed sockets may be exposed to the system as ordinary memory or can be private to the FPGA. Configured as ordinary system memory, unmodified operating systems and software running on the system will transparently enjoy the capabilities of the custom memory (i.e.: cache-like latencies) because loads and stores to the FPGA-managed address range are hidden at the lowest levels of the system behind the interprocessor protocol. Modifications to the software's memory allocator can force specific data such as register spills or context switch states into the modified memory, targeting maximum benefit to performance sensitive data.

2.2 Extended Memory Semantics

The interprocessor fabric implements a shared address space across all devices on the fabric, meaning memory operations originate in any device but may be serviced by another device where the physical address is. Because the remote device is responsible for fulfilling the memory request, the semantics enforced remotely ultimately govern the memory operation, allowing any semantics to be imposed on ordinary x86 memory operations. This property makes it reasonable to think of address bits indicating the

address is physically managed by the FPGA to also be thought of as an access control instruction field for any memory operation.

It is not necessary for real memory to be provisioned for all address ranges because it is possible to separate the concerns of address ranges and the actual storage space of the data through the FPGA stripping address bits which indicate the address is managed by the FPGA and re-issuing the memory operation from the FPGA. In this way, the FPGA can enforce extended semantics on data which is stored in ordinary system memory and is otherwise accessed conventionally.

The full/empty memory tag bit for each word of memory is of paramount importance to the EMS-MTA programming and execution model because it is largely responsible shifting the emphasis of parallel programming from thread management to data synchronization. The full/empty bit semantics can be enforced by a FPGA which provides an address range that encodes access control instructions in two bits, and stores the state bits in dedicated fast memory attached directly to the FPGA. The x86 processors can operate on the data without concern of other processor cores circumventing the extended semantics by finding a cached copy of the data because, as discussed in section 1.2, the FPGA managed address range can be configured as uncachable addresses, forcing all CPUs to always use the FPGA managed copy of the data.

Latency hiding parallelism is exploited by the XMT's barrel processor which performs a thread context switch every clock period without involving the operating system. A FPGA/x86 hybrid multithreaded architecture can achieve similar mechanical advantage by responding to long latency memory operations with an interrupt signal. The instruction with the long latency event is invalidated by the CPU and will be retried by the operating system after an involuntary context switch to another task which can make progress. Modified operating systems can store contexts outside of the operating system in dedicated memory managed by the FPGA, making context switches efficient and fixed cost.

2.3 Large Single System Image

Although x86 implementations are generally limited to 8-socket systems, it is possible to extend the processor interconnect fabric over a Scalable Coherent Interface bridge which converts a snoop, probe, or broadcast based cache coherency protocol into a scalable directory based one. As with extended memory semantics, virtual SMPs are implemented at a low level in the system, beneath the interprocessor fabric, making the system extension transparent to the software running on the system. This means unmodified operating systems and applications are able to use large single system images provided by virtual SMP adapters.

The reasons software cannot directly detect it is running on virtual SMP hardware are the same reasons no other hardware device on the interprocessor fabric can detect it is part of a larger SCI fabric. Virtual SMP hardware can therefore be freely incorporated into a FPGA-x86 hybrid system, and the resulting multi-node FPGA-x86-vSMP hybrid system would function like a single large system. This technique can be used to construct multi-node single system images of nodes with modified memory performance and semantics.

2.4 Address Hashing

Efficient use of physically distributed resources such as memory bandwidth, processors, and interconnect bandwidth is predicated on software distributing data and work across all the system's resources. The XMT strives to evenly distribute data accesses by hashing physical addresses so consecutive addresses are spread across the system. This functionality can be replicated in a FPGA-x86 hybrid system by the FPGA managing a scrambled address range, and hashing physical addresses at cache line granularity across the entire system. Address hashing can be implicitly applied to all address managed by the FPGA, or restricted to memory regions through the use of an explicit scrambling access control bit. A node topology is implicitly defined as part of address hashing through node

vSMP Node	Access Control	Socket ID	Physical Address	Storage Site	EMS Instruction
12 bits	00	00	48 bits	Socket 0	Ordinary load/store
12 bits	00	01	48 bits	Socket 1	Ordinary load/store
12 bits	01	XX	48 bits	Any	`READFE, WRITEEF`
12 bits	10	XX	48 bits	Any	`READFF, WRITEXF`
12 bits	11	XX	48 bits	Any	`Unused, WRITEXE`

Table 1: 64 bit addresses interpreted as virtual SMP node IDs and extended memory semantics access control instructions.

selection by address range, giving some control over data routing regardless of the SCI bridge implementation.

2.5 Other Tag Bits

Other tag bit semantics such as address forwarding and user defined tag bits can be implemented in a similar fashion. The address forwarding bit, if set, causes the data stored at the target address to be used as an address to which the operation is automatically forwarded. The effect is similar to pointer chasing, but without the issuing of additional instructions. Address forwarding is useful for generational garbage collection methods, debugging, and other atomic data structure manipulations. User defined tag bits can be implemented to cause arbitrary user code to be executed whenever marked memory is accessed, providing powerful debugging and evaluation on demand capabilities.

2.6 System Resiliency

Single system image computers are susceptible to catastrophic failure due to failure of any one component, and large single system image computers are even more vulnerable to spontaneous failure due to the larger number of parts they are composed from. FPGA-x86-vSMP hybrid systems are better able to tolerate component failure than standard systems because resiliency can be built into the processor interconnect fabric in several different and complementary ways.

Loads and stores of single cache lines or words can be bit-sliced and distributed across multiple nodes with error correction codes. When a node or connection fails, the missing bits can be reconstructed and the system continues to operate degraded. Individual memory operations can also be cloned and directed to backup memories to serve as a redundant copy of memory. This backing store does not need to be implemented using the same kind of memory as the primary memory, affording the use of non-volatile memory to provide an additional kind of system resiliency.

All I/O devices in conventional x86 systems are memory mapped, and device drivers in the operating system control the devices through loads and stores to command and data memory addresses which are fixed at boot-time. The technique of rewriting memory operations in the FPGA and re-issuing them can be used to proxy for I/O devices, maintaining a copy of any device state and redirecting I/O operations to redundant devices upon failure of a primary I/O device without participation of the device driver or any software.

3 Software

In an extended memory semantics hybrid system the only processor kind is the x86, and the memory address space is shared, presenting a vastly simplified programming model over hybrids which use multiple processor kinds connected to different memory address spaces. Preservation of the legacy x86 programming model and software stack provides a critical incremental path to port applications to the hybrid system – programs which do not use the

extended memory semantics are not affected by their presence, and software optimized for conventional x86 systems will still perform in an optimized way on hybrid systems.

Unlike porting applications to distributed memory clusters where all data dependencies must manually be identified and structures must be decomposed by hand with appropriate data exchanges inserted throughout the application, parallelizing applications for shared memory parallel processing can be done incrementally, on a loop-by-loop basis, without completely restructuring data structures throughout the entire application. The reduced risk and complexity of developing for the hybrid relative to a cluster is augmented by using a programming model which does not require different implementations of critical computational routines for each target platform.

3.1 Operating System

Operating systems which are not modified to take advantage of extended semantic features can be used on a FPGA-x86 hybrid system and the hardware will act as an ordinary unmodified system. FPGAs configured to be cache coherent and expose their attached memory as ordinary system memory will function as if a x86 processor with conventional memory modules were installed, treating the special memory as part of the normal system memory. Any characteristics of the modified memory will be enjoyed by any software accessing that memory, whether the modification are performance or resiliency related. Device drivers work as they would in unmodified systems, permitting the use of commercially available storage and network devices in the hybrid system, providing a wealth of options for further customization and integration into existing compute facility infrastructure. Direct Memory Access (DMA) operations performed by unmodified devices drivers, such as transferring disk blocks to memory, may be performed to the FPGA managed address range, enabling ordinary I/O operations to implicitly use the extended semantics.

3.2 Applications and Runtime System

Processing occurs only on conventional x86 processors, maintaining a single-source programming model and preserving the large existing catalog of programming, debugging, and performance tools available for x86, meaning there is a mature and rich tool set for the hybrid system from the outset. The semantic extensions are accessible through manipulation of address bits, giving direct, low overhead, syntactically economical access to the new features from most natively executed programming languages. Figure 4 illustrates using the extended semantics from ANSI-C.

The two dominant programming models for shared memory systems are Pthreads and OpenMP, both of which are suited for a hybrid multithreaded architecture. The Cray XMT extends ANSI-C with intrinsics to manage the full/empty tag bits, and non-loop parallel constructs such as *future* variables for unstructured parallelism and data flow algorithms. Several software tools [14,15] implement a MTA-like programming and execution model by emulating in software full/empty tag bits and fast on-demand context switches, and compiler tools such as ROSE [16] provide additional integration of directives to guide execution of parallel regions with particular scheduling techniques, or to automatically restructure non-parallel loops into parallel regions. It is easy to see how all of these tools can be modified to exploit hybrid hardware implementations of functionality they presently support through software emulation.

Both hardware latency in which data must travel through several stages of a memory hierarchy and network, and algorithmic latency in which the producer and consumer of data are far apart in time can be masked using parallelism to overlap computation and latency. As noted in section 2.2, the FPGA can force context switches on demand due to long latency memory events, providing a means to expose parallelism greatly exceeding the number of computational cores, and increasing the number of outstanding memory operations beyond the x86's built-in capabilities. For modest numbers of threads,

```
#define READFE(addr) *(0x0001000000000000|addr)
#define READFF(addr) *(0x0002000000000000|addr)
#define WRITEXF(addr, val) *(0x0001000000000000|addr) = val
#define WRITEXE(addr, val) *(0x0002000000000000|addr) = val
#define WRITEEF(addr, val) *(0x0003000000000000|addr) = val
...
int d; // Unannotated variable declaration
d = x + z; // Ordinary store of x+z in d
fprintf(fh, "d = %d", d); // Ordinary load of a value stored in d
y = d + y; // Ordinary load of a value stored in d
...
WRITEXF(&d, 0); // Unconditionally and atomically set d to 0 and mark full
...
x = READFE(&d); // When d is full, atomically mark as empty and load the value
...
WRITEEF(&d, x + y); // When d is empty, atomically mark as full and store x+y
```

Figure 4: Access to extended memory semantics from ANSI-C using macro address modification.

naïve over-subscription and relying on the operating system's thread scheduling may perform adequately, but to tolerate long latencies hundreds of threads per CPU core may be required. Existing operating systems are not optimized to manage large numbers of lightweight threads and lack insight into application execution to reason about scheduling of task continuations, creating opportunities to optimize applications and runtime systems.

Memory operation completion notification from the FPGA provides information required by the runtime to reason about which thread(s) should be scheduled to execute next. Direct communication between the FPGA and the OS or runtime is possible through a combination of interrupts and messages communicated via load and stores. Modified operating systems can dedicate one CPU core per socket to coordinating thread scheduling according to request completion notifications. This mechanism can also be used to marshal data from main system memory to the appropriate cache, reducing interprocessor memory bandwidth and latency. Alternatively, the data marshaling functionality can be hidden in a virtual machine layer, making it transparent to operating systems and applications running on the hybrid.

4 Economics of the EMS Hybrid Architecture

Reconfigurable hybrid system designs are unlike custom hardware system designs in that they do not require the same high level of planning, modeling, testing, and verification because they are based on already working implementations and are incremental manipulations of existing proven designs. To this end, hybrid reconfigurable systems present the possibility that for some configurations, real prototype systems may be easier to implement than to model, fundamentally changing the economics of architectural research and creating the potential for hardware and architectural experimentation with software-like development time, cost and risk.

5 Summary

An example combination of the components of a hybrid system is shown in figure 3 and consists of two conventional multicore x86 microprocessors, a FPGA acting as an extended seman-

tics memory controller, and a interprocessor fabric to Scalable Coherent Interface bridge which extends the single system image over multiple nodes.

The utility of hybrid systems of this class are twofold: in addition to being useful for multithreaded data intensive algorithm and software development, they also serve as a vehicle for computer architecture research, allowing researchers to implement memory with any semantics or performance characteristics they require. The EMS system described herein may serve as a viable alternative target platform for applications and research currently limited to the Cray XMT. Specifically, data intensive computing requires systems which offer programming and execution models that are efficient on irregular data structures and workloads, however conventional systems lack fine grained synchronization needed for these purposes, stifling research in algorithms and methods.

Extended memory semantics hybrids overcome many hybrid system limitations and deficiencies, namely: elimination of the performance penalty for serial execution; elimination of the mixed processor types with different instruction sets, byte storage orders, and address spaces; support for native I/O to external devices; and access to the entire x86 software stack including operating systems, compilers, debuggers, performance analysis tools, legacy applications, etc. The lack of multiple inherent weaknesses in EMS hybrids may be enough to overcome other benefits provided by accelerators and custom system implementations.

Unlike FPGA hybrid approaches which require complicated logic design for each application, this use of the FPGA modifies the system, not the software, freeing users to concentrate on parallelizing software instead of mapping an algorithm into a circuit. Hybrids need not be limited to MTA-style extended memory semantics, but could easily be specialized for another need such as transactional memory or dual-load-link/store-conditional instructions for updating doubly linked lists atomically. With costs and risks much closer to software than custom hardware design and a lower barrier to entry than logic design approaches, we believe this system design can increase the user base for FPGA hybrid systems.

Acknowledgments

This report was funded under the Center for Adaptive Supercomputing Software - Multithreaded Architectures (CASS-MT) at the Dept. of Energy's Pacific Northwest National Laboratory. Pacific Northwest National Laboratory is operated by Battelle Memorial Institute under Contract DE-ACO6-76RL01830.

About the Author

Jace A Mogill, Research Scientist, studies hybrid-microparallel computer architectures and parallel programming models at the Center for Adaptive Supercomputer Software and Multi-Threaded Architectures at Pacific Northwest National Laboratory. He was formerly an applications analyst at Tera Computer Company, and he eventually took Alan Kay's advice to learn to build his own hardware. He can be reached at: Pacific Northwest National Laboratory, 902 Battelle Boulevard, P.O. Box 999, MSIN J4-30, Richland, WA, 99352, USA, Email: `jace.mogill@pnl.gov`.

References

[1] D. Bader, G. Cong, and J. Feo, "On the Architectural Requirements for Efficient Execution of Graph Algorithms," in *The 34th International Conference on Parallel Processing*, Jun. 2005, pp. 547–556.

[2] D. Bader and K. Madduri, "Designing Multithreaded Algorithms for Breadth-First Search and st-connectivity on the Cray MTA-2," in *The 35th International Conference on Parallel Processing*, Aug. 2006.

[3] J. Crobak, J. Berry, K. Madduri, and D. Bader, "Advanced Shortest Path Algorithms on a Massively-Multithreaded Ar-

chitecture," in *First Workshop on Multithreaded Architectures and Applications*, Mar. 2007.

[4] K. Jiang, D. Ediger, and D. A. Bader, "Generalizing k-Betweenness Centrality Using Short Paths and a Parallel Multithreaded Implementation," in *The 38th International Conference on Parallel Processing*, Sep. 2009, pp. 542–549.

[5] S. Kang and D. Bader, "Large scale complex network analysis using the hybrid combination of a mapreduce cluster and a highly multithreaded system," in *4th Workshop on Multithreaded Architectures and Applications*, Apr. 2010.

[6] J. B. Dennis and G. R. Gao, "An efficient pipelined dataflow processor architecture," in *Supercomputing '88: Proceedings of the 1988 ACM/IEEE conference on Supercomputing*. Los Alamitos, CA, USA: IEEE Computer Society Press, 1988, pp. 368–373.

[7] B. Smith, "Concurrent Design of a Compiler and an Architecture," in *Parallel Architectures and Compilation Techniques Keynote Address*, 1997.

[8] HyperTransport Consortium, *HyperTransport I/O Link Specification*, 3rd ed. HyperTransport Technology Consortium, Jun. 2009.

[9] Intel Corp., *An Introduction to the Intel QuickPath Interconnect*, 320412th ed. Intel Press, Jan. 2009.

[10] IEEE, "IEEE Standard 1596-1992," Los Alamitos, CA, USA, 1992, David B. Gustavsson, Comittee Chair.

[11] XtremeData Inc., *XD2000-F FPGA In-Socket Accelerator for AMD Socket F.* XtremeData Inc., Oct. 2009.

[12] DRC Computer, *DRC Reconfigurable Processor Unit RPU110 Family.* DRC Computer, 2007.

[13] T. Brewer, "Instruction Set Innovations for the Convey HC-1 Computer," in *Hot Chips*, 2007.

[14] K. Wheeler, R. Murphy, and D. Thain, "Qthreads: An API for Programming with Millions of Lightweight Threads," in *22nd IEEE International Parallel & Distributed Processing Symposium*. Los Alamitos, CA, USA: IEEE Computer Society Press, 2008.

[15] A. Porterfield, "Maestro: Program thread and synchronization interface, version 0.1," RENCI, North Carolina, Tech. Rep. TR-08-01, March 2008. [Online]. Available: http://www.renci.org/publications/techreports/TR0801.p

[16] C. Liao, D. J. Quinlan, T. Panas, and B. de Supinski, "A ROSE-based OpenMP 3.0 Research Compiler Supporting Multiple Runtime Libraries," in *International Workshop on OpenMP (IWOMP) 2010*, no. LLNL-CONF-422873, 2010.