

SANDIA REPORT

SAND2009-7291

Unlimited Release

Printed February 2009

Memory Opportunities for High Performance Computing (MOHPC) Final Report

Richard C. Murphy, Arun F. Rodrigues, James A. Ang

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185 and Livermore, California 94550

Sandia is a multiprogram laboratory operated by Sandia Corporation,
a Lockheed Martin Company, for the United States Department of Energy's
National Nuclear Security Administration under Contract DE-AC04-94-AL85000.

Approved for public release; further dissemination unlimited.



Sandia National Laboratories

Issued by Sandia National Laboratories, operated for the United States Department of Energy by Sandia Corporation.

NOTICE: This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government, nor any agency thereof, nor any of their employees, nor any of their contractors, subcontractors, or their employees, make any warranty, express or implied, or assume any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represent that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government, any agency thereof, or any of their contractors or subcontractors. The views and opinions expressed herein do not necessarily state or reflect those of the United States Government, any agency thereof, or any of their contractors.

Printed in the United States of America. This report has been reproduced directly from the best available copy.

Available to DOE and DOE contractors from
U.S. Department of Energy
Office of Scientific and Technical Information
P.O. Box 62
Oak Ridge, TN 37831

Telephone: (865) 576-8401
Facsimile: (865) 576-5728
E-Mail: reports@adonis.osti.gov
Online ordering: <http://www.osti.gov/bridge>

Available to the public from
U.S. Department of Commerce
National Technical Information Service
5285 Port Royal Rd
Springfield, VA 22161

Telephone: (800) 553-6847
Facsimile: (703) 605-6900
E-Mail: orders@ntis.fedworld.gov
Online ordering: <http://www.ntis.gov/help/ordermethods.asp?loc=7-4-0#online>



SAND2009-7291
Unlimited Release
Printed February 2009

Memory Opportunities for High Performance Computing (MOHPC) Final Report

Richard C. Murphy

Arun F. Rodrigues

James A. Ang

Scalable Computer Architectures, 1422

Sandia National Laboratories

P.O. Box 5800, MS-1319

Albuquerque, NM 87185-1319

Abstract

This report summarizes the deliberations and conclusions of the Memory Opportunities for High Performance Computing (MOHPC) workshop held at the Sandia CSRI facility in Albuquerque, NM on January 9-10, 2008.

Acknowledgment

This report would be impossible without the efforts of a large number of people. Each of the participants and presenters provided invaluable insight into our understanding of the state of the art in memory systems, applications, programming models, and computer architecture.

The MOHPC program committee shaped the direction of the workshop's contents and the selection of participants. We would like to thank Almadena Chitchekanova, Sudip Dosanjh, Bill Harrod, Fred Johnson, Dean Klein, Bob Lucas, Bob Meisner, Mike Merrill, Lenore Mullin, Thomas Sterling, and Jeff Vetter for giving so generously of their time.

We are thankful to the chairs and deputy chairs hosting productive breakout and panel sessions: Mark Hill chaired the systems and CPU panel, Jeff Vetter and Mike Heroux chaired the applications panel, Peter Kogge and Arun Rodrigues chaired the architecture panel, and Mike Merrill and Bruce Hendrickson chaired the programming models panel.

We are also tremendously indebted to Tina MacLuso for serving as the official record keeper. Without her skillful notes and revisions, this report would not have been possible.

Richard C. Murphy

Arun F. Rodrigues

James A. Ang

Table 1. MOHPC Participants

Name	Affiliation	Name	Affiliation
Jim Ang	SNL	Bronson Messer	ORNL
Brian Barrett	SNL	Lenore Mullin	NSF
Jon Berry	SNL	Richard Murphy	SNL
Greg Branch	AMD	Rich Oehler	AMD
Maciej Brodowicz	LSU	Doug O’Flaherty	AMD
Sumanta Chatterjee	Oracle	Thomas Olson	Aerospace Corp.
Almadena Chtchelkanova	NSF	Mike Parker	Cray
John Cieslewicz	Columbia Univ.	James Peery	SNL
Carolyn Conner	LANL	Paul Petersen	Micron
Alfred Costantine	SAIC	Steve Poole	ORNL
John Daly	LANL	Dave Resnick	Micron
Erik DeBenedictis	SNL	Rich Ridgely	DOD
Jeff Draper	USC/ISI	Arun Rodrigues	SNL
Rich Dondero	SNL	Glen Rosendale	Nantero
Sudip Dosanjh	SNL	Kevin Ryan	Micron
Guang Gao	Univ. of Delaware	Subhash Shinde	SNL
Bill Harrod	DARPA/IPTO	Dylan Stark	LSU
Scott Hemmert	SNL	Craig Steele	Exogi LLC
Bruce Hendrickson	SNL	Rob Smith	Nantero
Mike Heroux	SNL	Alan Snavley	UCSD
Rick Hetherington	SUN	Richard Stempien	MITRE
Mark Hill	Univ. of Wisconsin	Thomas Sterling	LSU
Stephen Howell	SNL	Jim Sundet	DOD
Bruce Jacob	Univ/ of Maryland	Garret Swart	Oracle
Fred Johnson	DOE/SC	Gerry Taylor	Nantero
Brent Keeth	Micron	Jim Tomkins	SNL
Dean Klein	Micron	Jeff Vetter	ORNL
Peter Kogge	Notre Dame	Pete Vogt	Intel
Bob Lucas	USC/ISI	Harvey Wasserman	LBL
Tina MacLuso	SAIC	Trey White	ORNL
Ray McConnell	Clearspeed	Karl-Heinz Winkler	LANL
Bob Meisner	DOE/NNSA	Rich Witek	AMD
Mike Merrill	DOD	Tom Zipperian	SNL

Table 2. MOHPC Topics

Topic	Speaker
Memory Market Drivers	Dean Klein, Micron
Memory Tutorial (Circuits and Packaging)	Brent Keeth, Micron
Alternative Memory Architectures	Glen Rosendale, Nantero
Systems and CPU Panel	Chair: Mark Hill, Wisconsin Rich Oehler, AMD Mike Parker, Cray Pete Vogt, Intel Ray McConnell, Clearspeed
Breakout Sessions and Critiques: Applications Architecture Programming Models	Co-chairs: Jeff Vetter (ORNL) Mike Heroux (SNL) Peter Kogge (ND) Arun Rodrigues (SNL) Mike Merrill (DOD) Bruce Hendrickson (SNL)

Table 3. Acronyms

Acronym	Meaning
API	Application Programming Interface
AMO	Atomic Memory Operation
CAM	Content Addressable Memory
CAS	Column Address Strobe
CPU	Central Processing Unit
DIMM	Dual In-line Memory Module
DMA	Direct Memory Access
DRAM	Dynamic Random Access Memory
FFT	Fast Fourier Transform
HPC	High Performance Computing
I/O	Input/Output
ISA	Instruction Set Architecture
MOHPC	Memory Opportunities for High Performance Computing
MPI	Message Passing Interface
OS	Operating System
PDE	Partial Differential Equation
PIM	Processing-In-Memory
RAS	Row Address Strobe
RLDRAM	Reduced Latency DRAM
TLB	Translation Lookaside Buffer
TM	Transactional Memory

Contents

Participants	5
Topics	6
Acronyms	7
1 Introduction	13
2 Architecture	15
Metrics	15
Issues & Problems	15
Proposals	16
“Smarter” Memory Controller & Software Interface	16
Radically More Concurrency in the Memory System	17
Other Proposals	18
Cross Pollination	19
Applications to Architecture	19
Programming Models to Architecture	19
3 Applications	21
Application Motifs	21
Sparse Methods	21
Database and Informatics Applications	22
Everything Else	22
Challenges	22

Performance of Indirect Addressing	22
Transparency of Memory Performance	23
Keeping Reliability Above Suspicion	23
Memory Capacity	23
Smart Memory Analysis	23
Adoption	24
Cross Pollination	25
Architecture to Applications	25
Programming Models to Applications	26
4 Programming Models	27
Issues & Problems	27
Proposals	27
Diagnostics	28
Synchronization	28
Enabling Programmer Access	29
Intelligent Memory Controllers	29
Hierarchy	29
Cross Pollination	30
Architecture to Programming Models	30
Applications to Programming Models	31
5 Summary and Recommendations	33

List of Figures

2.1	Highly concurrent memory system example	17
-----	---	----

List of Tables

1	MOHPC Participants	5
2	MOHPC Topics	6
3	Acronyms	7
3.1	Application Programmer Feature Adoption Willingness	24

Chapter 1

Introduction

The memory wall, and more generally in a parallel system, the data movement problem has long dominated system performance. While well known, the implications of these challenges are perhaps poorly understood. The Memory Opportunities for High Performance Computing (MOHPC) workshop met to better quantify the problems of memory performance in the context of supercomputing.

In addition to the performance problems associated with memory, the commodity nature of the part poses unique industry challenges. The overall DRAM market has grown to nearly \$30 billion, but in 2007 invested approximately 2/3 that amount in capital expenditures. Additionally, while standards are required for industry adoption (via JEDEC, which represents over 290 companies), the consensus driven process slows time to market. Furthermore, processor vendors are often motivated to keep the memory system “as dumb as possible”, which may negatively impact overall HPC system performance and power budgets.

Technically, DRAM faces several challenges:

- **Power:** at large scale, DRAM consumes a large fraction of a machine’s power budget. In an environment where moving the data to a computation unit is more energy inefficient than performing a mathematical operation, DRAM’s traditional row/column design wastes power.
- **Latency:** experimental results show that many applications of interest to the community are latency dominated. There are fewer available mechanisms for decreasing memory latency even in an environment where clock cycle times are relatively flat. Few techniques remain to address DRAM latency without fundamentally changing the interface architecture.
- **Bandwidth:** increased packaging costs limit the ability of DRAM devices to deliver bandwidth to processors.
- **Capacity:** memory capacity on a per-core basis is increasingly difficult to deliver as core counts may accelerate faster than available memory channels. This poses a significant challenge for application developers as surface:volume ratios change accordingly. If current trends continue, memory capacity is expected to

grow at a rate approximately 2X slower than the number of cores is expected to grow.

From an application and programming model perspective, the memory system suffers from a lack of expressibility. In fact, typical memory hierarchies provide the programmer with little or no ability to describe the types of data operations being performed (even very simple descriptions such as “this data is to be used once”). Furthermore, most hierarchies are designed to obscure their structure from observation by the programmer.

Typically, a lack of performance feedback from the memory system inhibits the application’s ability to understand performance bottlenecks.

There is significant opportunity to optimize memory system performance, more explicitly define data movement throughout the system, and enable enhanced synchronization in the memory system. However, there are significant challenges to doing so in a fashion that is sustainable long-term. Applications far outlive hardware, and, as a result, application developers are typically unwilling to adopt ephemeral features. Portability across a range of architectures often helps to alleviate these concerns, but, thus far there are no portable programming models that enable the expression of data movement throughout the memory system.

The MOHPC workshop gathered over 60 participants in HPC (see Table 1). The workshop began with three presentations and a panel before splitting into three breakout groups focusing on Applications, Architecture, and Programming Models. This is summarized in Table 2. Each breakout group had two phases of discussion. The first phase concentrated on describing the technical challenges and memory opportunities from the perspective of their specific breakout group. All participants convened for overview presentations from each breakout group. The second phase for the breakout groups was focused on “cross-pollination” discussions to reflect on and respond to the overview presented by the other two groups.

The remainder of this report is organized as follows. Chapter 2 describes the proceedings of the Architecture breakout. Chapter 3 describes the deliberations of the Applications group. Chapter 4 describes the Programming Model’s groups discussions. Finally, Chapter 5 provides a final summary with recommendations.

Chapter 2

Architecture

The Architecture Breakout Group was tasked with finding an architectural perspective on future memory systems. The problems of power, capacity, and performance are all amplified in large scale HPC systems, and often require architectural solutions. At the same time, new solutions still must accommodate existing “legacy” codes, or at least provide an easy transition path.

Discussion began with identification of the key metrics by which a memory system can be judged. This naturally became a discussion on the crucial architectural challenges and problems facing future memory systems. Identifying these challenges led to several proposals to address them, two of which were developed in detail. In later “cross-pollination” discussions, the findings of the applications and programming model groups were examined.

Metrics

The first set of metrics to be defined were the most clear cut: capacity, performance (primarily defined as both bandwidth and latency), and power.

When examining performance, the measurement is more difficult. The question of bandwidth can be reexamined as “effective bandwidth” - i.e. are we wasting the bits we transfer? What is the overhead of address to data bits? Also, performance is best expressed as “per pin” or “per picojoule.”

Other metrics include the complexity of access commands and the desirability of meta-information in the data for synchronization, forwarding, exceptions, and notification.

Issues & Problems

Several issues were noted:

- **Capacity:** It was noted that capacity (in terms of DIMMs per channel) has

largely plateaued or even fallen, and this was unlikely to change in the DDR-based technologies. This problem is exacerbated by multicore. Currently, the DDR pads consume about the area of a single processor core, but unlike cores, pad area is not scaling with Moore's law.

- **Power:** DRAM is the major consumer of power in many commercial environments. A comparison was made with CPUs which have low-power states. Memory parts have similar states, but they are often not used. Power is often wasted due to the granularity of access (e.g. several open rows in a DIMM are activated, but only a few bits from each DRAM access is sent to the pins). Additionally, DRAM refresh is inefficient as it focuses on worst-case performance.
- **Lack of hints:** Software lacks hints to communicate with the memory hierarchy and controller(s). Operations where hints might be useful would include Cache Management (e.g. single touch data), DRAM Row Management (open/closed policy), specifying scratchpad, and the ability to specify an access pattern.
- **Smart memory:** A "Smart" memory brings additional issues. Data placement or affinity becomes more important to avoid hot spots and keep multiple memory controllers, DIMMs, and DRAM banks balanced. TLB reach is insufficient on current architectures and needs OS support to use superpages and non-continuous pages. Integration with the network also becomes an issue: should memory be attached to a processor, or should we have "memory-in-network?"
- **FLASH:** The emergence of FLASH and non-volatile memories raises the question of how it should be integrated into a system.

Proposals

"Smarter" Memory Controller & Software Interface

One proposal to address several of these issues was to construct a "smarter" memory controller and accompany it with a memory-centric software interface. This memory controller would support at least two modes. In one mode, it could perform introspection and prediction, and so act as a very powerful prefetcher. To allow good scheduling decisions, the controller would need to know who generated the request, whether it is a read/write/prefetch or instruction fetch. It would also have to be multi-core and multi-threaded aware. This would allow analysis to enable prefetching and improve data placement. Another mode would allow the programmer explicit control of data transfers and other capabilities.

Some proposed capabilities:

- **Statistics gathering:** A detailed set of memory controller performance registers.
- **Reference hints:** Including usage hints (e.g. single touch data, read-only data, open/closed page hints) and type (instruction, data, prefetch, etc...).
- **Scratchpad:** Provide an I/O scratchpad for staging data or containing frequently used data.
- **Flexible Data Movement:** Allow very flexible programmable prefetch and DMA. This would allow traditional fixed stride accesses, irregular access patterns, and scatter/gather across the memory hierarchy.
- **Asynchronous Notification:** Operations would need to be queued and performed asynchronously to allow overlap of computation and data transfer.

To enable these features, serious support from the OS would need to be provided. For example, a user-visible scratchpad memory has resource management and contention requirements which would need to be examined. Also, to encourage adoption, a cross-compatible emulation library would be needed to allow use on platforms which lack the smarter hardware.

Radically More Concurrency in the Memory System

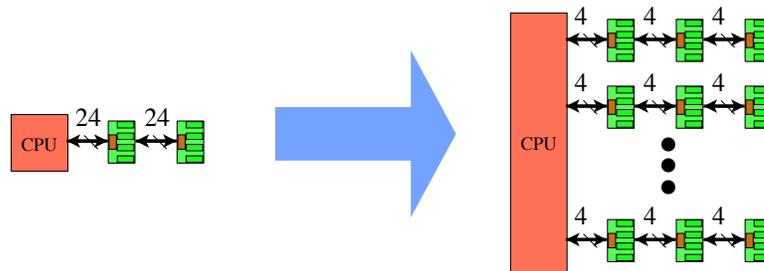


Figure 2.1. Highly concurrent memory system example

A second proposal suggested introducing much more concurrency into the memory system to mask slower, narrower memory channels. In this proposal, rather than a few wide (10s or 100s of pins/bits), fast (100s of MHz), shallow (1-2 DIMMS) channels attached to each memory controller, use many narrower (<10 bits/pins), slower, deeper (8 or more DIMMS) memory channels (Figure 2.1).

This organization would rely on massive concurrency from a large number of processors (and possibly network interfaces) to mask somewhat higher latency and lower

bandwidth. Potentially, it could provide much more memory capacity at a lower cost (due to pin savings and possibly easier board layout) and at less power. It may also be a good fit to processors with a large number of “skinny” cores. Alternately, adding more banks to each DRAM device could achieve similar results (more concurrency).

This organization is not without drawbacks. The biggest being the increase in latency. Also, the memory controllers would have to be more complex to keep track of more accesses “in-flight” at a time. The design would require more (and more complex) memory controllers at a cost in silicon area and power.

Other Proposals

Several other proposals were also voiced, though they were not explored in as much detail.

- **Hierarchy:** The insertion of a new level of the memory hierarchy composed of non-volatile memory or a memory technology denser than DRAM could open new possibilities. The biggest constraint is in the usage model: should it be transparent to the user, or should it be explicitly managed? Should the interface be block-transfer streaming or load/store?
- **TLB improvement:** The improvement of TLB reach and flexibility may be necessary for many smarter memories. The use of superpages or non-contiguous pages would increase reach and flexibility. This would require extensive support from the OS.
- **Scratchpad:** A scratchpad memory or lockable cache could provide improved latency, bandwidth, and more regular timing. However, this would also require OS support to manage resources. It is probable that library developers would be the best end-user for these features, as applications may cause too much contention and require more hand-tuning.
- **Partial Row Activation:** Increased use of posted Column Access Strobe (CAS) would allow greater efficiency by avoiding activating unneeded portions of the open row. This feature is already available in RDRAM.
- **Optical Memory Connections:** Direct optical connections between the memory and processor could improve bandwidth and lower power. Though there are fabrication issues involved with optics on the chip, advanced packaging techniques may overcome them.
- **Tagged memory:** Memory data may be tagged to contain meta-information for synchronization, forwarding, exceptions, and notification. This could be useful for highly multithreaded applications, or for debugging.

Cross Pollination

Applications to Architecture

The application group was unified in the desire for increased control and the ability to express application requirements, while simultaneously expressing disinterest in features that may be short lived or are not portable.

In general, the application group agreed that the architecture must support the graceful evolution of code. Creating ephemeral features, particularly those which may have a major impact in application performance (if the application is not modified) is particularly problematic. Investments in the current code base are huge, and an understanding of the impact of architectural change, as well as a plan for continuing new features is highly desirable.

While the architecture group's metrics are strong, the mapping of those metrics to applications is complex. The ultimate metric is likely application throughput per unit cost, and the proposed architecture metrics have no obvious/direct correlation.

Finally, the architecture group should use relevant applications in the analysis of impact. Analysis with simplistic or un-realistic benchmarks can be very misleading and damaging to the HPC community.

Programming Models to Architecture

The programming models group generally wanted the architecture group to push in more potentially disruptive areas, which tended not to be considered. In general the other two group's theme of a requirement for well supported, portable mechanisms to enable new classes of memory architectures was maintained.

The primary concern of the programming models group to the architecture proposals are that they are very processor-centric (and lack discussion of the Data or Memory/Push model). Additional discussion of data movement possibilities would be desirable. The group also noted a lack of discussion of disruptive technologies, such as Processing-In-Memory (PIM).

A discussion of "Willing To Do" and "Not Willing To Do", similar to the application group's would be profitable in the future. As with the application group, the programming models group is willing to assume more of the burden of overall control of the memory system. For example, a significant loosening of load/store consistency, potentially to the point of making it a software/compiler problem may be of interest. The architecture group expressed distrust of these approaches, noting that the application programmer and compiler developer may be unwilling to accept the responsibility, or that providing that level of control may increase software complexity

too much.

Finally, the group also recommended an extended discussion of virtualization throughout the system.

Chapter 3

Applications

The Applications Breakout Group was charged with evaluating the capabilities of the hardware and how those capabilities might be exposed by the underlying programming model in the context of a mature code base that has undergone extensive development. Indeed, because application life far exceeds machine life, architecture features that do not endure tend to be near impossible to support in the code base.

Discussion began with Application Motifs, focusing on physics, informatics, and other key applications. It progressed to a discussion of challenges presented by various general hardware and programming models options. The group examined the applicability of more intelligent memory technology to those challenges. Then the application group ended by providing a set of things they are both willing to consider and unwilling to consider for the adoption of new technology. Finally, the cross-pollination discussion is summarized.

Application Motifs

The Application group divided the space into three general motifs: sparse methods, database and informatics applications, and everything else.

Sparse Methods

Many sparse methods represent core HPC applications over a long time period. They include:

- Structured Grids with non-unit stride, iterating over different dimensions or visiting different slices
- Unstructured Grids that may benefit from Gather/Scatter capabilities, iterating with indirect references and irregular connectivity patterns
- Spectral (FFTs) with non-unit strides

- Adaptive (structured and unstructured) with time varying data structures

Database and Informatics Applications

Database and informatics represent newer HPC applications areas, and are often applied to the results of simulations or large real-world data sets. Consequently, there is less certainty about core application properties. The following key properties were defined:

- Data is often accessed only once
- Simultaneous heavy access to uniform arrays and irregular random accesses often interfere with each other
- Scratch pad memories could help (e.g., the Cray-2 local memory)
- There may be an as yet unquantified similarity to PDEs

Everything Else

Additional relevant applications include:

- Large Graph Traversals including reorderings, and especially non-PDE graphs
- Monte Carlo methods
- Virtualization that may include additional layers of indirection

Challenges

Each of the above enumerated application areas shares a set of common challenges for the application developer.

Performance of Indirect Addressing

One clear performance bottleneck is the ability of the architecture to perform complex indirect addressing. This is true across all application domains, but may particularly be seen in the database and informatics codes. It has long been observed that real applications tend not to saturate the memory bus on conventional architectures, which

is primarily caused by two factors: first, data dependencies within the application make it difficult to generate addresses quickly enough to achieve full memory bandwidth; and second, the memory system may not support enough concurrency to allow sufficient issue where the dependencies are less complicated.

Transparency of Memory Performance

Complex memory hierarchies increase the complexity of understanding application performance. Automation or sets of tools to facilitate understanding and optimizing memory performance are highly desirable, especially for changing platform hardware targets.

Keeping Reliability Above Suspicion

Increasing machine memory size has produced subsystems (e.g., disks) where silent data corruption is possible, and this should not be allowed in the core compute environment. Both the recovery from and reporting of errors is critical.

Memory Capacity

Finally, although memory capacity per core may be declining in order to achieve full memory bus bandwidth, this is undesirable from an application perspective.

Smart Memory Analysis

Increased programmer control over the memory system is considered highly desirable, particularly as it may improve the effective use of both bandwidth and power. Today's memory systems provide significant waste in cache-sized in application environments with low spatial locality, wasted sharing, and relatively low temporal reuse. Scatter/gather and non-unit stride memory access support could potentially alleviate the compulsory miss problem.

Again, transparency of memory performance is a critical issue. The application breakout group felt that a predictable performance model and deterministic behavior combined with support for introspection (memory usage statistics, overall footprint, hotspots, etc.) would facilitate the development of high performance applications.

Supporting fine-grain memory synchronization (e.g., XMT full/empty bits or transactional memory), and a set of robust atomic memory operations was also considered

Table 3.1. Application Programmer Feature Adoption Willingness

Willing To Do	Not Willing To Do
Add directives/pragmas	Adopt non-portable programming features (Except, possibly specialized kernels)
Evolve to address system design challenges	Abandon the existing code base

of benefit. Potentially coupling those memory operations with better data placement control (e.g., cache bypassing atomics) would also facilitate better overall CPU performance.

A set of simple in-memory operations could be beneficial. These include:

- Memory Clear
- Memory Replication
- Memory Copy
- Garbage Collection
- Simple Pattern Search

Finally, application runtime reconfiguration of memory is highly desirable, including small content addressable memories (CAMs), a runtime declared “scratchpad”, or support for specially placed variables within an application.

Adoption

The group came to a consensus about what application programmers would and would not be willing to do, as well as areas where there may be reluctance. This is summarized in Table 3.1.

As a group, they expressed reluctance to adopt a new language unless it could be done incrementally, and they were also reluctant to examine the current programming model.

The application group felt that their current programming model could be enhanced to support new memory devices, as well as existing systems. In general, the desire is for an increased ability to express locality. Three core examples were discussed: single-use data, which should not be cached; affinity between work and data; and replication (particularly for ghost nodes).

Portable mechanisms for defining a scratchpad memory and CAMs were also considered desirable.

Finally, introspection and performance transparency should be exposed by the language through the runtime system.

Cross Pollination

Architecture to Applications

The architecture group expressed some unifying themes when looking at the programming models and applications group's findings:

- **What does the general multicore community need?** Is it a better MPI? Better threads? Streaming programs?
- **We cannot just look at HPC.** But, we can draw on the HPC community's expertise. There is significant cross over if we look for it. The HPC community has been looking at concurrency and scalability for decades. We can help the commodity computing industry figure out 8 and 16 cores and tell them about the pitfalls at 10,000 or 100,000. By demonstrating the growth path from 8 to 10000 cores and beyond the HPC community can have the most influence.
- **FLOPS are free - but only in bulk.** AMD and Intel can make cheap FLOPS only because they produce millions of processors. From a usage perspective, it might be perceived to be free - but the hours and effort that go into creating them are intensely complex - often measured in man-millennia.

Comments on the applications breakout group:

- **Single-touch data:** This is feasible, however it must be universally supported (in hardware or emulation) to be useful. Also, there are options for the granularity it is performed at. Perhaps the easiest is to do this add the page level (i.e. added to TLB). There are other options for hints, including modifications to the ISA.
- **Scratchpad:** While feasible, a key concern is that saving state is difficult and expensive. Additionally, a scratchpad presents resource contention issues. A lockable cache may be easier to implement and manage. A key question for the application writers is to express why they want a scratch pad. Is it because it makes naming easier? Is it for bandwidth? latency? guaranteed timing? Also, there are the standard concerns about portability.

- **Smarter memory:** The operations proposed (DMA in memory, garbage collection, searching, etc...) are good ideas and broadly feasible. However, they are more difficult to perform across several memory controllers, especially if data is striped across channels (which may be desirable to avoid hotspots). Coordinating a search or garbage collection in hardware across several memory controllers (each running with multiple memory mappings) would be difficult.
- **Applications unwilling to adopt non-portable features:** If this is the case, they may have to accept emulation on some platforms. This also argues for standards across vendors to express extended-memory functionality. This presents a common “chicken and egg” problem - will application writers adopt a new feature if it must be emulated on certain platforms, or must it be universally supported? Will hardware vendors add a new feature before there is broad acceptance amongst application writers?

Programming Models to Applications

The programming models group generally wanted the application group to push in more potentially disruptive areas, which tended not to be considered. In general the other two group’s theme of a requirement for well supported, portable mechanisms to enable new classes of memory architectures was maintained.

The group again noted the productivity of the “Will Do”/”Will Not Do” list, but noted that to have significant impact the discussion will have to be extended to additional communities (outside of the HPC community). Furthermore, future trends in application design are not well understood.

It may be productive for application experts to propose ideal strawman architecture and runtime models to serve as the basis for discussion. This would produce application driven requirements and facilitate further discussion.

The group also noted that developing “easy-to-write” experimental code, where performance may be less critical, vs. more performant code should be enabled in future programming models. Additionally, various organizations are willing to put differing levels of effort into performance improvements. For example, Oracle expressed a willingness to put more effort into a smaller performance gain than did the DOE.

Finally, the applications group also lacked much discussion of disruptive technologies, such as active messages and other forms of continuation-based programming.

Chapter 4

Programming Models

The programming models group was charged with evaluating the potential for enabling architectural innovation to be used by applications. This breakout report describes the issues and problems related to enabling new data movement architectures, enumerates proposals for doing so, and ends with a discussion of “cross-pollinating” ideas from the architecture and applications groups.

Issues & Problems

The group looked at two models for data movement:

1. **Control:** or CPU/Pull model, in which data movement is a function of program control flow and primarily orchestrated by the processor
2. **Data:** or the Memory/Push model, in which the memory system controls data movement into the CPU (similar to the HTMT Percolation model¹).

The group concluded that both perspectives are needed, and that they are in no way mutually exclusive.

It was also observed that computation is now less expensive than data movement, and that named CPUs may inhibit the programmer’s ability to describe data movement. Specifically, current systems may overemphasize the control model.

Proposals

The group provided recommendations in five areas: memory system diagnostics, synchronization, enabling programmer access to enhanced memory systems, more intel-

¹See: Jacquet, Janot, Govindarajan, Leung, Gao, and Sterling, *Executable Performance Model and Evaluation of High Performance Architectures with Percolation*, University of Delaware, CAPSL Technical Memo 43, November 21, 2002.

ligent memory controllers, and issues surrounding memory hierarchy.

Diagnostics

As noted by both the architecture and application groups, enhanced memory introspection is critical to understanding and optimizing system performance. The programming models group went further to note that this is a problem at both the user and runtime levels. They further note a lack of tools available to the programmer for diagnosing memory system performance, specifically because typical systems attempt to hide the hierarchy from the user. Further, while the programmer is often willing and able to provide hints (e.g., “this data is only used once and should not be cached”), programming languages typically lack the ability to express these hints, and architectures typically lack the mechanisms to take advantage of them. For example, architecturally, there are often non-caching memory regions for I/O subsystems, but few convenient mechanisms in a processor’s ISA to perform a non-caching loads.

The lack of diagnostics often leads to an inability to precisely identify data access problems within an application, including hotspots, race conditions, inter-thread data corruption, or other memory mismanagement. These diagnostic capabilities are lacking throughout the system: in the execution, runtime, and application models.

Synchronization

Numerous strong proposals for enhanced memory synchronization mechanisms exist including:

- Atomic Memory Operations (AMOs)
- Full/Empty bits (that appear in the Cray XMT, Cray/Tera MTA, and Denelcor HEP)
- Transactional Memory (TM)
- orderless synchronization

However, these mechanisms tend to be poorly supported in most architectures, and when supported are not typically portable. Consequently, programming model and application support to express synchronization tends to be weak. This is a particularly critical problem in multicore processors that increase concurrency without creating a correspondingly simpler mechanism for synchronization.

Enabling Programmer Access

The group discussed enabling programmer access through a series of memory hints (for managing the hierarchy, data movement, and synchronization). It was further noted that a lack of portability and expressibility inhibits new features. This complements the conclusions of the applications group.

Intelligent Memory Controllers

As with the architecture and application groups, the programming models group agrees that there is significant potential utility in an enhanced memory controller. Critically, such a controller could enable data movement operations (such as scatter/gather), as well as synchronization, particularly in the form of AMOs. AMOs are attractive because many atomic updates should occur “in memory” without the cache pollution that would result from processor-based implementations. The memory controller is the closest place to put these operations.

Once again, the user programmability issue is of significant concern. The question of how to enable these operations portably and so that they act in a predictable fashion across a range of architectures is a challenge for all three breakout groups.

Additionally, prior attempts at intelligent memory controller design, most notably Impulse, ran into problems of virtualization and aliasing because they were not tightly coupled to the processor. Better processor integration for future intelligent memory systems is highly desirable. For example, atomic memory operations issued as an instruction in the processor that occur at the place within the memory hierarchy that “owns” a particular data item are much more transparent to the programmer than those that may only be issued at the memory controller.

Hierarchy

The group noted the difficulty of managing the memory hierarchy, and that these hierarchies are becoming increasingly complex. Additionally, requirements for hierarchy management (both from an architectural and application programmer perspective) may be conflicting. For example, the desire for cache awareness and cache obliviousness are in conflict.

The use of “local” memory (scratch pads, etc.) shows significant promise, but tends to also be performed in a non-portable fashion. Additionally, there tend to be few mechanisms for coping with the expansion of the memory hierarchy.

Cross Pollination

Architecture to Programming Models

The architecture group expressed some unifying themes when looking at the programming models and applications group's findings:

- **What does the general multicore community need?** Is it a better MPI? Better threads? Streaming programs?
- **We cannot just look at HPC.** But, we can draw on the HPC community's expertise. There is significant cross over if we look for it. The HPC community has been looking at concurrency and scalability for decades. We can help the commodity computing industry figure out 8 and 16 cores and tell them about the pitfalls at 10,000 or 100,000. By demonstrating the growth path from 8 to 10000 cores and beyond the HPC community can have the most influence.
- **FLOPS are free - but only in bulk.** AMD and Intel can make cheap FLOPS only because they produce millions of processors. From a usage perspective, it might be perceived to be free - but the hours and effort that go into creating them are intensely complex - often measured in man-millennia.

Comments on the Programming Models breakout group:

- **Instrumentation:** Putting in a counter is easy, but only if you know what you are counting. Also, there is impact for process context switching (and so some extent for thread switching). Hotspots, race conditions, etc. are hard for hardware to detect because it cannot measure program intention. Hardware would need more data, possibly through tagged memory.
- **Memory synchronization:** AMOs are very feasible and relatively easy to implement. Additionally, weak ordering is needed for scalability.
- **Smarter Memory Controller:** Features like scatter/gather, AMOs, user programmable caching, and virtualization are all feasible.
- **Memory Hierarchy:** Though there was general agreement on the utility of abstraction and the desirability of better use of local memory, this raised counter questions. How would the programmer want to handle FLASH or other memory (block addressing, streaming, Load/Store)? Also, how do we (safely) make it "user" visible and not just OS-visible?

Applications to Programming Models

The application group was unified in the desire for increased control and the ability to express application requirements, while simultaneously expressing disinterest in features that may be short lived or are not portable.

The application group uniformly agreed that additional access to diagnostic and performance information is highly desirable, however, historically it has proven very difficult to generate a robust, portable, non-ephemeral API to support these features.

Providing the programmer with options is also desirable, pragmas are often broken or ignored. The interface presented to the programmer should not require the programmer to think deeply about the implementation of the memory system, rather it should allow for the expression of programmer knowledge about data access patterns.

Chapter 5

Summary and Recommendations

The memory wall is a tremendous problem for HPC. The technology problems of power, latency, bandwidth, and capacity will only continue to intensify without investment on the part of the government. However, enabling architectural and technology solutions requires significant cooperation from application and programming models developers. The MOHPC group offers the following eight broad recommendations:

1. **Continued Investment in Memory Technology:** the problem can be addressed by an application requirement driven initiative with industry. Given the challenges of competing in a heavily competitive commodity market place, we recommend supporting on-shore memory capabilities through advanced technology development.
2. **Engaging Other Constituencies:** historically, the HPC community has often relied on technologies from other communities to fulfill mission requirements. In this case, we believe a proactive approach could lead to a broadly acceptable technology with heavy HPC influence. Because the community owns and understands the key application set, we are in a position to turn that understanding into real deliverables.
3. **Application Focus:** relying on simplistic or un-realistic benchmarks can be very misleading and damaging to the HPC community. When in doubt, enabling programmer control should take precedence.
4. **Improved Memory Hierarchy Understanding and Introspection:** each MOHPC breakout group requested less obfuscation of the memory system and an improved ability to identify software problems. This should be supported at all levels from the architecture, through the runtime and programming model, and to the application.
5. **CPU and Memory System Integration:** future systems should support both the traditional “CPU pull” and emerging “memory system push” models. Cooperation from both CPU and memory manufacturers is required for the full solution.
6. **Malice of Forethought:** architectural solutions need to have the potential for broad support from the programming and runtime environments. They must

cleanly support enhanced and unenhanced memory architectures, and provide feedback and understanding to the application developer.

7. **Continued Community Involvement:** further broad-based discussion is required to continue to understand community requirements. A “willing to do”/”not willing to do” list from each constituency may prove a useful tool.
8. **Willingness to Innovate:** each of the breakout groups at MOHPC was rightly conservative in their recommendations, which has the potential to stifle innovation.

DISTRIBUTION:

- 1 Almadena Chtchelkanova, NSF, 4201 Wilson Boulevard, Arlington, VA 22230
- 1 Bill Harrod, DARPA/IPTO, 3701 Fairfax Drive, Arlington, VA 22203-1714
- 1 Fred Johnson, DOE Office of Science, SC-21, Germantown Building, 1000 Independence Ave SW, Washington, DC 20585
- 1 Bob Meisner, NNSA, NA-121.2, Forrestal Building, 1000 Independence Ave SW, Washington, DC 20585

- 1 MS 1322 John Aidun, 1435
- 1 MS 1319 Jim Ang, 1422
- 1 MS 1319 Brian Barrett, 1423
- 1 MS 1319 Ron Brightwell, 1423
- 1 MS 1320 Scott Collis, 1414
- 1 MS 1319 Doug Doerfler, 1422
- 1 MS 1322 Sudip Dosanjh, 1420
- 1 MS 1319 K. Scott Hemmert, 1422
- 1 MS 1320 Mike Heroux, 1416
- 1 MS 1318 Bruce Hendrickson, 1410
- 1 MS 9151 Howard Hirano, 8960
- 1 MS 9158 Curtis Janssen, 8961
- 1 MS 1319 Sue Kelly, 1423
- 1 MS 0801 Rob Leland, 9300
- 1 MS 0321 John Mitchner, 1430
- 1 MS 0321 James Peery, 1400
- 1 MS 1319 Neil Pundit, 1420
- 1 MS 1316 Danny Rintoul, 1409/1412
- 1 MS 1319 Arun Rodrigues, 1423

- 1 MS 0822 David Rogers, 1424
- 1 MS 1318 Suzanne Rountree, 1415
- 1 MS 1318 David Womble, 1540
- 2 MS 9018 Central Technical Files, 8944
- 2 MS 0899 Technical Library, 4536

