2009 Symposium on Application Acceleration in High-Performance Computing
July 27-31, 2009
Urbana, Illinois

Investigations of Multicore and Acceleration Technologies at ARSC

Gregory B. Newby,
Chief Scientist

Arctic Region Supercomputing Center
University of Alaska Fairbanks

July 28, 2009
**Presentation Abstract**

- The Arctic Region Supercomputing Center has been investigating multicore and acceleration technologies for several years. This presentation will highlight some of the findings to date. Technologies under investigation have included FPGAs, multicore CPUs, GPUs, the Cell processor, and chip multithreading. Throughout, an emphasis has been on determining the feasibility of the technology and associated software for high performance computing workloads.
Presentation Outline

- About your Presenter & about ARSC
- FPGAs
- Multicore
- Hardware Multithreading
- Cell Processor
- GPU
- Solid State Drives
- Conclusion
Your Humble Narrator: Gregory B. Newby

- Chief Scientist of the Arctic Region Supercomputing Center
- Interests in hardware acceleration, information systems, and high-latitudes geophysical phenomena
- More information: people.arsc.edu/~newby/
Arctic Region Supercomputing Center

Helping scientists to seek understanding of our past, present and future by applying computational technology to advance discovery, analysis and prediction.

ARSC’s state of the technology supercomputers provide 24/7 accessibility for high-performance computing needs.

ARSC’s storage infrastructure provides redundant back-up, swift access and retrieval of archival data.
A brief Advertisement

- **NTMS 2009**: December 20-21, Cairo
  - www.ntms-conf.org
  - Submissions due: August 23
  - Tracks on Mobility, Security, and New Technologies/Services
    - gbn is general co-chair
Theme: FPGAs for HPC

- Utility of FPGAs as coprocessors for HPC applications
- Comparison of programming languages & environments (Mitrion-C, DSP-Logic, Handel-C, others)
- Use of Cray XD1 and standalone systems
Six Cray XD1 chassis with 108 processing elements in the following configuration:
- 12 nodes with 64-bit AMD Opteron 250 processors (Single Core)
- 6 nodes with 64-bit AMD Opteron 250 processors (Single Core) with application acceleration processors,
  Field-programmable gate arrays (FPGAs), that act as coprocessors to the Opteron processors.
- 18 nodes with 64-bit AMD Opteron 260 processors (Dual Core)

Each chassis features six RapidArray processors
- hardware designed by Cray, provided a high-bandwidth, low-latency interface to the high-speed RapidArray interconnect
- network later upgraded to a fat tree topology, improving performance for all-to-one, one-to-all, or all-to-all communication by providing a direct link to each node.
Findings: FPGAs for HPC

- **Limited utility for mainstream HPC applications today**
  - Bandwidth & latency limitations in PCI boards; in-socket solutions will help
  - Too much low-level hardware detail for most HPC programmers to know
  - Not enough circuitry for large-scale HPC applications; very limited double precision floating point capability. **Virtex-6** might change this outlook

- **Good utility for some single node applications, such as genetic sequencing**
Theme: Hypertransport Cache Coherency

- Motivation: many applications on ARSC’s Midnight were much faster on 2-socket versus 8-socket nodes (dual processor Opteron).
- We looked at many applications and benchmarks, from micro-level to large parallel applications (WRF, HYCOM, ...)

2-socket AMD64 topology

One 1 GHz 16x16 Hypertransport link per supported processor with 8GB/second bandwidth
8 socket AMD64 topology

* All HT links are operating on 1GHz and 8GB/s
Challenges: Multicore

- Already commonly deployed
- However, most HPC applications & libraries ignore the differences between:
  - Cores on the same physical CPU socket
  - Cores on different sockets in the same node
  - Cores on separate nodes
- Challenge: shared resources (cache, memory and I/O buses) might lead to contention among processor cores
- Challenge: different applications stress different aspects of the processor & subsystems

ARSC Sun X4600 Schematic
Findings: Hypertransport Cache Coherency

- Finding: the main explanation of Midnight’s varying large- and small-node performance is that cache coherency checking on the Hypertransport does not scale well
- Finding: applications that manage their own multi-threaded work allocation can do nearly as well on the 8-socket as 2-socket. CHARM++ & applications based on it are exemplary
  - Note: we are now replicating our studies with Pingo’s quad-core dual-socket opterons
Theme: Hardware Multithreaded (CMT) Multicore

- Motivation: Utilize early 6- and 8-core multithreaded multicore processors; evaluate for HPC work
- We used Sun T1 and T2+ in single- and dual-socket systems
- Findings are applicable to today’s Xeon (Nehalem) and Opteron processors, but Sun’s processors are not geared towards HPC
Technology: Multithreading

- To benefit from hardware multithreading:
  - Match the # of threads to the proportion of computation vs. communication (that is, don’t deploy too many threads)
  - Use shared memory approach (such as OpenMP) on the processor or node
  - Allow for unevenness of resources (such as fewer FPs than hardware threads)
- Bottom line: Multithreading can be problematic for HPC applications that are computationally bound, since threads slice existing CPU capabilities, rather than adding more CPU capabilities
- Bottom line: On heterogeneous and accelerated hardware, having multiple threads in order to allow memory pre-fetching and otherwise keep the CPU busy is a best practice (but not suitable for all applications)
- Adding threading (or any parallelism) to existing serial applications can be difficult, and potentially error-prone
  - Sun’s T1/T2+
  - Experimental: Intel 80-core; Tilera Tile64
- Based on the idea of allowing computational work to happen within a thread, while other threads wait for I/O
- Such threading in hardware can also be done in software:
  - POSIX threads
  - Software frameworks (i.e., CHARM++)
Findings: Multithreaded Multicore

- The Sun processors are geared towards enterprise computing, not HPC
- Shared cache very good
- Shared FPU is limiting
- This work was from 2007; there is not much new traction from Sun’s T1/T2+ line in HPC
- Tile 64: Under investigation at GWU
**TILE64™ Architecture Overview**

- **8 X 8 grid of identical, general purpose processor cores (tiles)**
  - 3-way VLIW
  - 5MB on-chip cache
  - Low power operating modes

- **Four DDR2 memory controllers**

- **Flexible IO interfaces**
  - XAUI, PCIe, GigE

- **On-chip iMesh™ networks**
  - Memory Dynamic Network (MDN)
    - Access DDR memory
  - User Dynamic Network (UDN)
    - Inter-tile comm; streaming
  - Static Network (STN)
    - Low latency comm for small messages
  - IO Dynamic Network (IDN)
    - IO communications
  - Tile Dynamic Network (TDN)
    - Non-programmable; inter-cache comm.
PGAS Model Semantics on TILE64

- **Architectural advantages on TILE64 for UPC implementations**
  - Physical shared memory space eases memory synchronization
    - Common DDR2 used for private and shared memory area
  - Variety of on-chip, inter-core comm networks may facilitate PGAS implementation
    - MDN, UDN, STN networks used for local & remote memory accesses

- **Performance benefits through UPC**
  - Logically partitioned memory abstraction for all threads
  - Small cache sizes make data locality awareness vital for performance
    - L1 data: 8KB, L2: 64 KB
    - Simple, one-sided comm. mechanisms
Multithreading Software Approaches

- **To benefit from hardware multithreading:**
  - Match the # of threads to the proportion of computation vs. communication (that is, don’t deploy too many threads)
  - Use shared memory approach (such as OpenMP) on the processor or node
  - Allow for unevenness of resources (such as fewer FPUs than hardware threads)

- **Bottom line: Multithreading can be problematic for HPC applications that are computationally bound, since threads slice existing CPU capabilities, rather than adding more CPU capabilities. This also applies to hyperthreading**

- **Bottom line: On heterogeneous and accelerated hardware, having multiple threads in order to allow memory pre-fetching and otherwise keep the CPU busy is a best practice (but not suitable for all applications)**

- **Adding threading (or any parallelism) to existing serial applications can be difficult, and potentially error-prone**
Multicore processors don’t need to be synchronized, but exist together on a socket.

Multicore & Multithreading
Software Emerging Practices

- Consensus for HPC programming style: use OpenMP on node, MPI off-node.
- Forthcoming: libraries that are built with an assumption of the three scenarios (same socket, same node, different node)
- Challenges persist:
  - Cache updates for shared memory
  - Different latency for communication on- and off-node
  - Less memory per core, as # of cores goes up
  - Few provisions for better on-socket resource sharing
Theme: Cell on PS3

- Used in Playstation3; enhanced version in Roadrunner
- PowerPC (hyperthreaded) front end general-purpose CPU; up to 8 synergistic processing elements (limited capability)
- Total theoretical over 200GFLOPs/Cell
- Hybrid processor, but in-socket. SPEs have no direct access to system memory etc.
Theme: Cell on QS22

- Available for research on acceleration technology; testing of codes (Quasar is not a production system)
- www.arsc.edu/resources/quasar.html
- 1 JS21 Login Node: 2 GB of shared memory (1 GB per core); 1 dual core IBM PPC970MP processor; 4X SDR InfiniBand
- 12 QS22 Compute Nodes:
  - 8 GB of shared memory per node (4 GB per core)
  - 2.32 GHz IBM PowerXCell 8i processor per node each with 8 synergistic processing elements per processor.
- 4X SDR InfiniBand
- Red Hat Enterprise Linux 5
- Torque batch scheduler

Quasar: not quite as big as Roadrunner
Findings: Cell Broadband Engine

- Very few ported applications & libraries
- Dual programming model is difficult
- OpenMP XL compiler quite good
- Acceleration of 2X-10X is typical for full applications. Scaling out on CPUs might be a better option for existing applications
- Work on MAGMA and OpenCL, among others, should make programming the Cell easier
- IBM’s SDK is good, but not completely implemented (i.e., many missing parts, or limitations in implementations)
Exemplar: Roadrunner Ported Applications

- See McPhearson 2008 LA-UR-08-4494:
  - “VPIC: fully-relativistic, charge-conserving, 3D explicit particle-in-cell code. 8.5K lines C/C++, 10% modified
  - SPaSM: Scalable parallel short-range molecular dynamics code (was on CM-5). 34K lines C, 20% modified
  - Milagro: Parallel, multi-dimensional, object-oriented code for thermal x-ray transport via implicit Monte Carlo on a variety of meshes. 110K lines C/C++, 30% modified
  - Sweep3D: Simplified 1-group 3D Cartesian discrete originates (Sn) kernel representative of the PARTISN neutron transport code. 2.5K lines C, 50% modified”
  - Note: none of these, or other ported LANL Roadrunner applications, were found for free public download

- A fairly short list, with ~8 person-years of effort combined
- Can we infer anything about other applications of interest? (Level of effort to port; suitability of different applications)
Custom Cell Applications at ARSC DSRC

- Anton Kulchitsky, HPC Specialist, wrote slowsum and mightyfft to test Quasar functionality
  - Slowsum: compute a series
  - Mightyfft: multiple approaches (fftw and IBM SDK) to finding a peak in a 1D series

- people.arsc.edu/~kulchits/quasar/quasar.html
Kulchitsky’s slowsum

- Finds a slowly converging series:

\[
\sum_{n=2}^{\infty} \frac{1}{n \ln^2(n)}
\]

- Uses gcc (dual binaries) and IBM SDK
- Single node; up to 16 SPUs
- Variety of options
<table>
<thead>
<tr>
<th>Time in seconds</th>
<th>Quasar Serial, PPE, double</th>
<th>Quasar 1 SPE, double</th>
<th>Quasar 1 SPE, float</th>
<th>Quasar 16 SPE, double</th>
<th>Quasar 16 SPE, float</th>
<th>Quasar 16 SPE, double, SIMD version</th>
<th>Quasar 16 SPE, IBM XL, OpenMP, double</th>
<th>Quasar 16 SPE, IBM XL, OpenMP, float</th>
</tr>
</thead>
<tbody>
<tr>
<td>77.94</td>
<td>77.26</td>
<td>75.17</td>
<td>5.54</td>
<td>5.54</td>
<td>0.4</td>
<td>0.15</td>
<td>0.94</td>
<td>0.96</td>
</tr>
</tbody>
</table>

**Type Descriptions on Quasar**

Quasar: Slowsum Performance with $N = 10^{28}$
Kulchitsky’s mightyfft

- Calculates the spectrum of the magnetic field output from a satellite (tested with ACE and intended to be used with WIND as well).
- Uses gcc compiler, choice of FFTW or IBM SDK FFT
  - Both are limited, as mentioned earlier
- Single node, up to 16 SPUs. Multi-node planned

• What’s next? CFD. Stay tuned.
mightyfft outcomes

Quasar: FFT1D performance

- c2c data type
- 10000 FFTs
- size of FFT = 1100

Graph showing the relationship between the number of SPUs and the time (in microseconds) for FFT1D performance.
mightyfft outcomes
**Theme: GPU**

- **The idea: utilize graphics processors for general purpose computing**
  - Enabled thanks to vendors providing an API, some libraries, and languages
  - Standardization is emerging via OpenCL (maybe CT, later)

- **Challenge: SIMD programming model can be challenging, and sometimes inefficient**

- **Challenge: GPUs are not well suited for all complete applications. Only part of an application is likely to benefit from GPUs as coprocessors.**

---

**GPU block diagram**
GPU Programming: CUDA & CU-BLAS, Brook++, OpenCL

- Experience with single node, multiple-GPU. Not much multi-node (MPI)
- These are all functional, and all require at least some rewriting + refactoring of code
- OpenCL is not yet accelerated, but has the most promise as a common approach. However, OpenCL is a relatively low-level API, less so than a new language, toolkit, or library
Work on FFT on Cell, CUDA, FFTW

- **GPU**: For small ffts, CUFFT performs much slower than FFTW. But for larger ffts, CUFFT results in up to at least 8X speedup over threaded FFTW.
- **GPU**: for the optimized CUFFT, it is at least 3X faster than CUFFT, it is designed for $n = 512$, also it has $n = 8$ and $n = 64$ working in a special data layout.
- **GPU v. Cell**: For the MightFFT Application, we can get obvious result to compare the GPU performance with large input data file.
  - Cell slower than GPU
FFT performance

Time (ms) vs N

- FFTW
- CUFFT
- CUFFT (Optimized)
Cell v. GPU Timing
Findings: GPU

- Matrix & vector operations are good candidates for GPUs. These are widely used in HPC applications, and to some extent already known by compilers (i.e., SSE instructions)
  - Having efficient communication to the GPU is important
  - GPUs are not as good at double precision math, yet
- New languages are important
- So are libraries and GPU-aware compilers
- We anticipate GPU challenges will be solved, and that these solutions will benefit other acceleration & hybrid technologies
GPU SGEMM benchmark

- One core Q9550 ACML
- ATI-9250 G-ACML
- NV-9800GTX CUBLAS kernel
- NV-9800GTX CUBLAS
- NV-GTX280 CUBLAS kernel
- NV-GTX280 CUBLAS

GFlops vs. # Matrix Dimension N

ACML 4.2, G-ACML 1.0, CUBLAS 2.1
Theme: New Languages

• “High Productivity Computing Systems” (US – DARPA) and other initiatives are pushing development and utilization of parallel global address space (PGAS) languages and other new languages & approaches
  – These include UPC, Chapel, X10, CoArray Fortran, and others
  – These languages present a more uniform hardware landscape to the programmer, so that explicit task parallelism is not necessarily required

• ARSC is hands-on with CAF & Chapel; partners at GWU are emphasizing UPC
Findings: Characteristics and Challenges of New Languages

- These are about providing parallelism across nodes, more easily than (for example) MPI
- Can be accomplished by providing a global memory address space, rather than explicitly programming parallel tasks
  - They provide relatively low level access (such as threads, pointers...), for use when needed
  - They also allow higher level programming which doesn’t need to explicitly program for parallelism. For example, parallel arrays
- Relatively few existing applications in large scale use are written with these languages. This will change over time as the HPCS program proceeds, and as a result of other forces
- Today, choice of a new language for a new application is difficult, due to lack of language or implementation maturity
- Porting existing large-scale applications to such new languages is, in most cases, premature
Theme: Old Languages

- **OpenMP**: Excellent ease of use for Cell BE with IBM’s XL compiler. Many older codes still have OpenMP directives which could be resurrected, to facilitate OpenMP on-node, and MPI off-node

- **CoArray Fortran**: Forthcoming in Fortran 2008 standard; already available from Cray & others
OpenMP Overhead on Cell

Synchronization Overhead

Number of processors

Overhead (microseconds)
Findings: Compiler Technology

- Compilers can help by providing some automatic parallelism, such as loop unrolling
- Source-to-source compilers offer ease of adoption of new languages & hardware
- IBM, Portland Group, Intel and others are providing increasingly sophisticated compilers for multicore. Other acceleration technologies are addressed, though to a smaller extent
- Compiler optimization doesn’t always win. See Lee Higbie’s analysis of Fortran compilers at people.arsc.edu/~higbie/
Theme: Solid State Disks (SSD) for HPC

- ARSC will deploy SSDs as fast disk on various systems in 2010 (so will everyone else!)
- Work in progress: evaluation of filesystem tuning for write cache, optimized for SSDs (Miaoqing Huang)
  - Newby’s laptop (Powerbook with SSD): about 17% faster building a BerkeleyDB than a nearly identical system with a HDD
  - Toshiba Qosmio gaming system with SSD: much faster for both Windows and Linux applications
Findings: SSD

• All HPC vendors are rolling out products with SSDs rather than spinning disks
  – Write speed, hardware exhaustion are issues that will be mitigated
  – Enterprise-class SSDs have better characteristics for HPC (similar to enterprise-class hard drives)

• Very few spinning disks will remain in future years

• OS software & device drivers will benefit more from SSDs if they are modified
Conclusions and Recommendations

- **Heterogeneity is here to stay. Embrace it.**
  - Memory hierarchies
  - Mixed processor types, some with limitations
  - Payoffs for utilizing locality of memory, communication

- **Needed: efforts to determine whether to port or rewrite today’s most important applications**
  - Community efforts needed to insure correct functionality & adoption
  - Writing fault-tolerant code, with new languages (i.e., PGAS) will be challenging

- **Supercomputing time to solution will result partly from faster components, and more of them – as in the past. But the applications that will benefit most will specifically utilize heterogeneous and accelerated hardware**

- **There are many R&D questions. Ultimately, many people and places together will shape the future**
Thanks!

- Many of these slides came from coworkers, colleagues, students and visitors over the past year. Including:
  - ARSC staffers Don Bahls, Anton Kulchitsky, John Mitchell, Lee Higbie, Ed Kornkven, and others
  - ARSC students Wei Dang, John Styers and Jessica Gonowon
  - GWU partner Tarek El-Ghazawi and graduate students including Olivier Serres, Abdullah Kayi, Miaoqing Huang, Lenny Wang and Ahmed Amber
  - Major Alex Mentis, summer 2009 visiting faculty
  - NCSA April 1-3 application acceleration workshop participants
  - 2008-2009 ARSC summer interns including Ying Yu, Vahid Ajimine, Kylie McCormick
  - ARSC’s PS3 & Quasar users
  - Mike Kistler @ IBM
  - And many others!