High-Performance Computational Science with Hardware Accelerators: Challenges and Potential Solutions

Tarek El-Ghazawi, Professor of ECE

Director, GW Institute for Massively Parallel Applications and Computing Technologies (IMPACT)

Co-Director, NSF Center for High-Performance Reconfigurable Computing (CHREC)

tarek@gwu.edu
Outline

- Introduction
- Architectural Convergence
- Accelerators and why accelerators
- Productivity Hurdles/Issues
- Potential Directions for Solutions
- Conclusions
Introduction

◆ Our research focus
  – Hardware accelerators productivity

◆ Parallel Programming
  – Partitioned Global Address Space (PGAS) models
  – Unified Parallel C (UPC)
    ◆ upc.gwu.edu
High-Performance Computing Resources at GWU

- Pyramid: 1048 Core SUN Cluster
- 8-node CELL Cluster
- Lyra: 80 Core DELL Cluster
- 8-node Tesla GPU Cluster
High-Performance Reconfigurable Computing Resources at GWU

- sgi Altix-4700
- sgi Altix-350
- CRAY XD1
- SRC-6E
- HC-36
- SRC-6

GaTech/AFRL Workshop on CSE Challenges - August 17, 2009
Architecture Convergence

- CPU / Tile / CLB / Core
  - e.g. Cray XT5m, Intel Paragon
- FPGA
- GPGPU

A Generic Parallel Architecture

GaTech/AFRL Workshop on CSE Challenges - August 17, 2009
Driving Factors for Convergence

- End of road for serial computing technologies
  - Rising power and thermal costs limit achievable clock rates

- Continued density escalation
  - Multi-core is the new performance road/highway
  - Performance from exploiting multiple levels of parallelism

Continued Density Escalation

Source: Kurzweil 1999 – Moravec 1998
Architectural Advances and Integration: From machines to chips!

1971-78
First MIMD System
• CMU C.mmp (16 PDP11s)

1985
SIMD
• CDC STAR-100
• TI ASC
• ILLIAC IV

1996-1998
SIMD
• Altivec
By Apple, IBM, and Motorola

1998
HPRC
SRC

2001
Vector Processor/SIMD CELL BE
GPGPUs NVIDIA and AMD
Multicore CPUs IBM Power 4

2007-2010
Hybrid-Reconfigurable/Chip
Convey
Manycore Tile64 and Larabee

Early 1970s
First Vector and SIMD Systems
• CDC STAR-100
• TI ASC
• ILLIAC IV

1971-78
First MIMD System
• CMU C.mmp (16 PDP11s)

1998
HPRC
SRC

2001
Vector Processor/SIMD CELL BE
GPGPUs NVIDIA and AMD
Multicore CPUs IBM Power 4

2007-2010
Hybrid-Reconfigurable/Chip
Convey
Manycore Tile64 and Larabee

1998
HPRC
SRC

2001
Vector Processor/SIMD CELL BE
GPGPUs NVIDIA and AMD
Multicore CPUs IBM Power 4

2007-2010
Hybrid-Reconfigurable/Chip
Convey
Manycore Tile64 and Larabee

1998
HPRC
SRC

2001
Vector Processor/SIMD CELL BE
GPGPUs NVIDIA and AMD
Multicore CPUs IBM Power 4

2007-2010
Hybrid-Reconfigurable/Chip
Convey
Manycore Tile64 and Larabee
Emergence of Accelerators

- Accelerators present heterogeneity
- Enable higher performance through fine-grained parallelism
- Offer higher computational density than CPUs
Why Accelerators?

- **Amdahl’s law**
  - Speedup limited by the sequential part
  - The sequential part may not be really sequential

- **Different hardware requirements based on the characteristics of the “sequential part”**
  - Fine grained, wide parallelism => GPGUs, FPGAs
  - Control dominated sequential code => Powerful CPU core.

- **Accelerators are required for addressing the different requirements**
Accelerators Asserted their Presence: Reaching the PetaFLOPS Milestone

- Vector Machines
- Massively Parallel Processors
- MPPs with Multicores and Heterogeneous Accelerators

1993-1999: HPCC
2006: End of Moore’s Law in Clocking!

Roadrunner
ASCI Red
## Comparison of Accelerators

<table>
<thead>
<tr>
<th></th>
<th>SP</th>
<th>DP</th>
<th>Max. Power (Watts)</th>
<th>DP GFLOPS /W</th>
<th>Best Suited For</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Cell</strong></td>
<td>204</td>
<td>102</td>
<td>80</td>
<td>0.33</td>
<td>SIMD/Data Parallel operations</td>
</tr>
<tr>
<td><strong>GPU: Nvidia Tesla C1060 1.3 GHz</strong></td>
<td>933</td>
<td>78</td>
<td>187.8</td>
<td>0.42</td>
<td>SIMD/Data parallel operations</td>
</tr>
<tr>
<td><strong>Intel Xeon 3.0 GHz</strong></td>
<td>48</td>
<td>24</td>
<td>80</td>
<td>0.30</td>
<td>Sequential, floating point operations</td>
</tr>
<tr>
<td><strong>AMD Opteron 2.5 GHz</strong></td>
<td>20</td>
<td>10</td>
<td>68</td>
<td>0.15</td>
<td></td>
</tr>
<tr>
<td><strong>Xilinx V5 LX330</strong></td>
<td>-</td>
<td>28</td>
<td>30</td>
<td>0.93</td>
<td>Fine-grained parallelism, Integer operations</td>
</tr>
<tr>
<td><strong>Altera Stratix III EP3S260</strong></td>
<td>-</td>
<td>50.7</td>
<td>30</td>
<td>1.69</td>
<td></td>
</tr>
</tbody>
</table>
Which Accelerators Will Survive?

The ones that can be mass produced have a better chance!

Cell Processor  
Gaming Consoles

GPGPU  
Laptops and Workstations

FPGA  
Telecom Equipment
Heterogeneity in MultiCore Chips

◆ Dedicated special function units such as graphics, crypto, & DSP alongside processor cores
◆ Specialized units enable exploitation of multiple levels of parallelism
◆ Computational density advantage over processors
◆ May ease performance setbacks due to serial bottlenecks as predicted by Amdahl’s law

Intel Tera-scale roadmap

Manycores as Accelerators: Asymmetric Cores

- Pollack’s Rule: If resources from $r$ cores are combined to create a more powerful core, the performance will be proportional to $\sqrt{r}$.
  - Cannot replace all cores by a single, powerful core!
  - Use of only simple cores slows down the serial portion
  - Best of both worlds – asymmetric multicore

Asymmetric Cores – Speedup Projections

Execution Model for Hardware Accelerators

- Transfer of Control
- Input Data

- Output Data
- Transfer of Control

Accelerator

Vector, Systolic Arrays, SIMD, Reconfigurable, ...
1. Application must follow a 90-10 rule
2. Programmer partitions the code across the CPU and accelerator
3. Programmer co-schedules CPU and accelerator
4. Programmer transfers data between CPU and accelerator
5. Transfer time is not negligible, a substantial overhead
6. Using a new accelerator means learning a new programming interface(s)
7. Moving code to a different board (same accelerators) requires substantial programming
8. Changing the ratio of CPUs to accelerators requires also substantial programming
Why Focus on Accelerators

- Important in their own right, but also address general HPC questions
- Most general in exposing the abstract issues that we need to deal with
  - Parallelism
  - Locality
  - Heterogeneity
  - Hierarchy
  - Asynchrony
- Many interesting applications in addition to HPC
  - Video gaming
  - Supercomputing
  - Real-time embedded systems (DSPs, FPGAs, ..)
Issue #1: The 10-90 Rule

- Application dependent, cannot do much about it
- The 10% part that takes 90% of the time may benefit a lot from one kind of accelerators over the other
  - Use a specific accelerator if the target is one specific application, e.g. FPGAs and Cryptography
  - Otherwise, use an accelerator that works well in the general case for most apps

Altix 4700 with 6 V4 FPGAs vs. a Cluster of 2.3 GHz Opterons

<table>
<thead>
<tr>
<th>Application</th>
<th>Speedup</th>
<th>SAVINGS</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Cost</td>
</tr>
<tr>
<td>DES Breaker</td>
<td>38514</td>
<td>96x</td>
</tr>
</tbody>
</table>

Issue #2: Hardware-Software Partitioning

◆ With your bare hands
  – Not very productive!

◆ Library-based
  – The simplest, but restrictive

◆ Compiler based
  – A tough problem
  – Sequential language can hide the gems
  – CUDA, Brook, etc., even OpenCL, are the wrong answer
    ◆ They assume that the code is already partitioned
    ◆ They do not scale up to clusters/system level, cannot see outside the chip
  – New languages may make it easy

User Guidance/Feedback can also help
Library Approach – A Case Study

Compilation Approach – A Case Study

Accelerator code templates (communication…)

Accelerator C code generation

Accelerator C compiler (e.g. Impulse C for FPGA, CUDA for GPU)

Accelerator executable

UPC code

Selected UPC functions

Code partitioning

UPC host code generator (add accelerator control)

UPC compiler

Host executable

"Parallel Programming of High-Performance Reconfigurable Computing Systems with Unified Parallel C", Tarek El-Ghazawi, Olivier Serres, Samy Bahra, Miaoqing Huang and Esam El-Araby (RSSI'08), July 7-9, 2008, Urbana, Illinois, USA.
Issue #3: CPU-Accelerator Co-scheduling

Programmer co-schedules CPU and accelerator

- Grouping and deploying kernels in time to
  - Share the accelerator
  - Reduce and amortize overheads (e.g. reduce/overlap data transfers)
  - Increase throughput (e.g. through forwarding/chaining)

- Use CPU if accelerator is busy (run-time only)

- Some work for automating the process, algorithms can be traced to embedded systems

- Related work also in heterogeneous computing

- More recent work for FPGAs

- Need more to support general applications and systems

- Should rely more on run-time system, based on monitoring and conditions
Heterogeneous Co-scheduling

ReCoS algorithm considers:
- Data Access time
- Resource Availability (e.g. Area)
- Reconfiguration time (if applicable)
- Data dependency

Exploits available resources for higher throughput

Optimized Scheduling for Heterogeneous Processing

Function-level DAG of a given application

System Constraints/Requirements

Scheduler/Optimizer

Architectural-variants Hardware Library

DAG after optimized mapping and scheduling process

Tasks in same background color are scheduled into the same configuration

Approaches

- Efficient selection of task implementation variants [1]
- Reduced Data Movement task scheduling [2]


Issue #4: Explicit Data Transfer

- Explicit CPU and accelerator data transfers required by the programmer

- Explicit transfers can be avoided with:
  - Global address (CPU and accelerator memories)
  - Coherence
Issue #5: Transfer Time Not Negligible

◆ Widening gap between increase in processing capacity vs. data transfer rates
  – As processing speed increases, transfer times quickly become the bottleneck

◆ Various optimizations may reduce transfer times, but may not work
  – Optimizations such as overlapping data transfer and processing has little value in this case

◆ Hardware solutions required to address data transfer bottlenecks
Wavelet-Based Hyperspectral Dimension Reduction
(Execution Profiles)

Total Execution Time = 1.67 sec
(SRC-6E)
Speedup = 12.08 x (without-streaming)
Speedup = 13.21 x (with-streaming)

Total Execution Time = 0.84 sec
(SRC-6)
Speedup = 24.06 x (without-streaming)
Speedup = 32.04 x (with-streaming)

SRC-6 I/O Bandwidth = 1600 MB/sec (1400 MB/sec Sustained)
SRC-6E I/O Bandwidth = 800 MB/sec (350 MB/sec Sustained)
Historical CPU – Accelerator Interconnect Trends

Examples,
- CPU-FPU integration over time
- CPU-GPU integration over time
Issue #6: New Programming Interface

◆ Best is to develop one research agenda for general scalable HPC, Accelerators, and accelerated HPC
◆ Learn from HPC and rely only on consortiums
◆ Adapt and influence HPC programming models
◆ Rework the software stack to accommodate new requirements
  – Heterogeneity
  – Parallelism Hierarchy
  – Locality Hierarchy
  – Dynamic Parallelism?
  – ..
New Programming Models?

- Hierarchy, Heterogeneity and locality
TILE64™ - A PGAS Case Study

- 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 PGAS 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 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 effective
    ◆ L1 data: 8KB, L2: 64 KB
  - Simple, one-sided comm. mechanisms
Issues #7 & #8: Portability and Need for Virtualization

◆ Same accelerator, different board
  – Program has to be rewritten to accommodate different communication interfaces
  – Other on-board hardware changes can also affect accelerator scheduling, such as reduction in on-board memory

◆ Same accelerator, change in CPU to accelerator ratio
  – Program has to be changed
  – Not all program instances (processes) will be able to use accelerator to speed up execution

◆ Same type of accelerator, different family/vendor
  – Recompilation and/or rewriting code
  – E.g. AMD vs. Nvidia GPU, Xilinx Virtex 2 vs. Virtex 5
Virtualization Techniques

Sharing
Examples: LPARs, VMs, virtual disks, VLANs
Benefits: Resource utilization, workload manageability

Aggregation
Examples: Virtual disks, IP routing to clones
Benefits: Management simplification, scalability

Emulation
Examples: Arch. emulators, iSCSI, virtual tape
Benefits: Compatibility, interoperability

Insulation
Examples: Spare CPU subst., CUoD, SAN-VC
Benefits: Continuous availability, investment protection

LPAR = Logical PARTitions
VLAN = Virtual/logical Local Area Network
CUoD = Capacity Upgrade on Demand
SAN-VC = SAN Volume Controller

Virtualization Solutions

The diagram illustrates the concept of virtualization in computing, showing layers of hardware and software components. It highlights the importance of managing virtual machines, hardware, and software layers to optimize performance.

Key points:
- **Management OS**: Device drivers and vendor APIs.
- **Virtual Machine 1**: Application, GWU API.
- **Hypervisor Layer**: (Xen Virtual Machine Monitor).
- **FPGA/GPU** configurations for different processes.
- **Application code running on each process** (CPU core) has the virtual view of an attached accelerator.

Citations:
Conclusions

- Architectural innovations at the chip level, prompted by feature size reduction and the flattening of clocking speed resulted in a rich plethora of many core architectures and accelerators.

- There are differences in the architectures but at a more abstract level there are much more similarities even between the chip and the system levels, all are exhibiting HPC features.

- Development options are too many, laborious, fragmented, confusing and unproductive.

- Application developers deserve a single and integrated interface.

- It is fair and should be sufficient to ask developers to understand the concepts of parallel systems and computing at an abstract level (e.g. parallelism, locality, heterogeneity, hierarchy, ..).

- It is unfair to ask them to be aware of the specific implementation details.
Conclusions

◆ There is need to
  – Provide one programming interface that can scale up and scale down
  – Support implicit data transfers, and
  – Use virtualization and coherence, among other things, at the lower levels to mask out implementation details