GPGPU Computing and Multicore Processors: Exploring The Spectrum

B. Scott Michel, Ph.D.
High Performance Computing Section
Computer Systems Research Department

scottm@aero.org
Organization and Outline

Target Audience: “A little something for everybody”

Talk Outline:

Part I: How Did We Get Here? What Is The Technology Spectrum?

Part II: A Look At The Software Ecosystem

Part III: Acquisition Issues
Part I

How Did We Get Here?
What Is The Technology Spectrum?
I Have A Need… For Speed!

- Ground systems don’t run out of reasons for more processing capability
  - Increasing end-user data needs: transform the raw data into various types of end-user product
  - Increasing requirements for technology insertions, future programs
- Increased number of transistors, modest increases in performance
  - 90% die relatively passive as L1, L2 and L3 cache
  - 10% die actively computing
Intel “Moore’s Law” Trends

(sources: Intel, Wikipedia, K. Olukotun)
Other Important Factors…
Heat and Feeding The Processor

- Physics is winning!
  - $I_{ss}$ vs $I_{dd}$ and shrinking feature size: static leakage current approaching switching current
  - Heat proportional to clock frequency
  - Static current leakage contributes significant idle heat

- The Memory Wall: Feeding the processor
  - High latency penalty for off-processor fetch
  - Cache nondeterminism: little or no control over cache replacement actions or policies

- Superscalar “bag o’ tricks” exhausted: instruction level parallelism, deeper pipelines, etc…

- Compiler optimizers aren’t the programmer’s friend
Today’s Two Solutions: SIMD And Multiple Cores

• SIMD: Single Instruction, Multiple Data
  – Basis of Cray’s architecture
  – Found in graphics processor units
  – Intel/AMD SSE2, SSE3 instructions, PowerPC Altivec
  – Upside: Compute multiple results per instruction
  – Downside: Requires data structure refactoring

• Multiple cores
  – Invert the 90/10 rule: 90% active, 10% passive
  – Slower clock speed: decrease heat with comparable or higher problem throughput as single core processor
  – SWAP improvements: hibernate idle cores (power management), shift workload between cores (thermal management)
Multicore Taxonomy: Homogeneous vs. Heterogeneous

• Homogeneous multicore
  – “The Traditional Approach”: duplicate execution units as needed
  – Intel/AMD dual core, quad core
  – Sun UltraSparc T1, T2 (aka Niagara, Niagara II)

• Heterogeneous multicore
  – General-purpose and special-purpose elements
  – General-Purpose GPU computing
  – STI Cell Broadband Engine
  – MIT RAW and USC/ISI MONARCH
General-Purpose GPU Computing

- Started as an effort to compute game physics between frames
- Harnesses the GPU’s SIMD stream processing on matrices
  - NVIDIA nv40, G70: 24 and 32 parallel floating point units
  - AMD/ATI Radeon X1K: 48 parallel FP units
- GPU code generally outperforms CPU code by 2-4x
- Delivers higher GFLOPS/W compared to uniprocessors
  - nVidia nv40 @ 400 Mhz $\uparrow$ 0.55 GFLOPS/W
  - Intel x86_64 @ 3 GHz $\uparrow$ 0.11 GFLOPS/W (approx.)
- Reasonably cost effective upgrade, ~$300
GPGPU Application Areas

• Linear algebra acceleration
  – LU decomposition
  – Matrix multiplication
• Signal processing
  – FIR filters
  – Autocorrelation filters
• Scientific computing
  – FEM, ODE, PDE solvers
  – Navier-Stokes solvers
• Database “SELECT” query processing
• GPU-accelerated Folding@Home
• Not particularly good at FFTs
GPGPU Software Development Challenges

• Mapping graphics idioms to the problem
  – Shader languages are designed for graphics, not scientific computing
  – Many shader languages to choose from…

• Single precision floating point
  – Truncates results, no IEEE rounding: Numerical drift
  – Iterative refinement for error compensation: Double precision computation on CPU, feed error correction to GPU

• No double precision floating point

• No arbitrary array or matrix accesses: Reformat data to GPU-friendly format

• Slow GPU-to-CPU result upload: Keep computation on GPU for as long as possible
A Trivial GPGPU Performance Benchmark:
\[ y_{\text{new}} = y_{\text{old}} + \alpha x \times (\text{saxpy}) \]

- **CPU (3.2 GHz AMD x86-64)**
  - Tests execution speed, cache/memory throughput
  - `cpubench-gcc`: “-O3 -Os -funroll-loops”
  - `cpubench-u16`: 16x hand-unrolled loop

- **GPU**
  - Tests parallelism, texture memory throughput
  - Execution only: render/execute only

- **Problem size**: \(32 \leq x \leq 1048576\) (6x6 to 1024x1024 texture sizes), step size 4681

- **Iterated test at each sample point 300x for statistical significance**
GPU vs. CPU: MFLOPS comparison

- **Problem Size (4-component elements)**
- **GPU**
- **CPU, unrolled 16x**
- **CPU, gcc optimize**
STI Cell Broadband Engine

• Sony/Toshiba/IBM co-designed, co-developed processor

• Heterogeneous multicore technology
  – PowerPC-64 Primary Processor Element
  – 8x Symbiotic Processor Elements (SPEs)
    – Vector processor units, based on VMX instruction set
    – 256 GFLOPS peak, single precision FP
    – 26 GFLOPS peak, double precision FP
  – 2.2 GFLOPS/W

• Playstation 3’s processor

• LANL “RoadRunner”: 8,000 Cell-based nodes out of 16,000 total nodes
Reactions to Cell…

• You either love it or hate it!
• “Developers are forced to sweat bullets to take advantage of the Cell Platform” -John Carmack, ID Software
• “[Software developers] are tearing their hair out over multi-core” - Tom Halfhill, Microprocessor Report
• Valve’s Steve Bond isn’t particularly impressed, efforts focused on consumer Intel/AMD multicore
• “What’s so hard about doing non-graphics programming on a GPU?” - John Stokes, Ars Technica

  In a blog entry on another site that links O’Sullivan’s post approvingly, parallel programming researcher Michael Suess reports that a student of his who worked on both Cell and CUDA found Cell to be much easier.

• “[The Cell’s] architecture is very well suited toward running a game and not terribly suited toward running a desktop computer” -Alex Hastings, Insomniac Games (IEEE Spectrum, Dec. 2006).
Cell BE Software Development Challenges

• Software tool ecosystem is evolving…
  – Programming using GCC intrinsics: Glorified assembly language
  – Cell SDK has a lot of code, but is it just a starting point for ideas?
• 256K Local Store: All code + data in a compact space
  – Message orchestration: Get the next tile, work unit into LS when it’s needed, reassemble results on PPE
  – Double buffering: Hides latency, cuts available LS memory
  – Data orchestration: Get the data into a SIMD-friendly format, arrays of structures vs. structures of arrays, avoid accessing singletons (“unaligned”) data
• Not dissimilar to GPGPU software development, but maybe a little easier
Sun UltraSparc T1 and T2

• Originally code named “Niagara”
• 4, 6 and 8 core flavors
• Architecturally designed for thread-heavy applications: 8 cores x 4 threads/core
• UltraSparc T1 is not designed for numeric applications, has one shared floating point unit
• UltraSparc T2 enhances numeric capabilities, thread execution
• Runs existing code
• “Thundering herd” lock contention problem requires minor software redesign
Accelerators

• ClearSpeed Advance
  – Primary market: high performance technical computing, floating point computation

• Aegia PHYSX physics accelerator
  – Initial market: compute physics during game play
  – Branching out to the HPC market… stay tuned…

• FGPAs and reconfigurable computing
MONARCH:
John Granaki, USC/ISI
Part II

A Look At The Software Ecosystem
Common Issues: Vectorization and Parallelization

• 4-element \( <x, y, z, w> \) is the most common: SSE2/3, Altivec, GPU, Cell

• Structures of Arrays
  – Operates on multiple elements together
  – Example: multiply-add (saxpy)

• Arrays of Structures
  – Treats vector components individually
  – Example: Comparison to constant, filtering data

• Primary effort is data refactoring
  – Sometimes it’s OK to take this hit when data isn’t organized as a stride-1 array of vectors -- YMMV…
  – Shuffle/permute primitives reformat individual vectors
  – Lots of algorithm literature from Cray, late 80’s and early 90’s research to rely on and resurrect…
SIMD Structure of Arrays

\[
\begin{align*}
&x_1 & x_2 & x_3 & x_4 \\
&y_1 & y_2 & y_3 & y_4 \\
&x_1+y_1 & x_2+y_2 & x_3+y_3 & x_4+y_4 \\
&z_1 & z_2 & z_3 & z_4 \\
&z_1\cdot a_1 & z_2\cdot a_2 & z_3\cdot a_3 & z_4\cdot a_4
\end{align*}
\]
SIMD Array Of Structures

x y z w

0/1 0/1 0/1 0/1
GPGPU Software Tools

• Major Players: RapidMind and Peakstream

• RapidMind
  – Startup by Mike McCool and graduate students, U. Waterloo.
  – Outgrowth from fragment shader language research (libsh)
  – Embedded functional language, just-in-time compilation to host’s GPU

• Peakstream
  – C++ software library, classes and their operators structure the computation
  – Just-in-time compilation to proprietary virtual machine, excellent debugging facilities
  – Platform-limited: Linux and AMD Stream processor
Hardware Vendor GPGPU Software Tools

• AMD/ATI CTM (“Close To the Metal”)
  – Designed to be general-purpose from the ground up, works with X1K GPUs, AMD Stream processor
  – Been around for approx. one year

• NVIDIA CUDA
  – NVIDIA’s relatively new general-purpose GPU programming toolkit
  – Targeted to G70 GPU line

• Competition between the major GPU vendors can only improve their respective toolkit offerings
Cell BE Software Tools

• Currently, very primitive and rapidly evolving
• Cell SDK v1 and v2
  – gcc 3.3 has limited autovectorization, improved in 4.2 and 4.3 but hand-rolled is generally better
  – SDK libraries and code: a good idea launch pad
  – SDK v2 program chaining: data stays in place, keeps SPU busier
• IBM efforts:
  – Contracted ports of VSIPL/VSIPL++ and other libraries
  – xIC/C++ compilers: research versions have advanced optimizers, OpenMP support, not in general availability (yet)
• RapidMind generates code targeted to Cell
• Mercury Computer Systems offers their own version of a Cell SDK
• Message orchestration, SPU buffer and memory management is the developer’s problem
  – Remember the Apple ][ and TRS-80s?
**Selected Multicore Research Areas**

- **Software Transactional Memory (STM)**
  - Memory regions with acquire, operate, commit and rollback semantics; nested transactions
  - Controversial: Is STM feasible? Is STM really deadlock avoiding or lock-free? How heavy are transactions? Is STM really the right paradigm? How does a STM transaction recover?

- **Parallelizing, auto-vectorizing compiler research and languages**
  - Interpreted languages, virtual machines are easier to transform
  - Explicit vs. implicit parallelism in a language: Is explicit necessary?
  - Important to see the high-level sequence of operations and recognize patterns, e.g., matrix multiply, and combine operations
  - Functional languages making a comeback?

- **Re-evolution vs. revolution and evolution:** lots of work done in the 80’s and early 90’s in multi-processor systems
Part III

Acquisition and Program Issues
Technology Refresh: It’s Inevitable

- Long timeline programs don’t like moving targets but want to leverage new capabilities

- **GPU**: Low-to-medium short-term risk
  - Cost effective: $300 - $500 hardware upgrade
  - Software recode required, but performance payoff is 2x - 4x better than uniprocessor, numerical convergence issues

- **Cell**: Medium-to-high short-term risk
  - PS-3 hardware relatively cost effective, IBM QS20 cluster and Mercury blades are investments
  - Developing immature software ecosystem, but with potentially high performance gains in single precision FP, numerical convergence issues
  - Incrementally migrate functionality to Cell SPUs (LANL approach)

- **UltraSparc T1, Intel/AMD Duo and Quad core**: Low short term risk
  - Highly threaded applications see most benefit
  - Runs existing code
Concept Stage Programs

• **GPU**: Low-to-medium risk
  – NVIDIA and AMD/ATI recognize a marketplace. compete with Cell and other multicore technologies
  – Toolkits will evolve, less management burden on developer

• **Cell**: Low-to-medium risk
  – Software ecosystem evolving and will stabilize
  – Leverage today’s graphics fragment shader expertise to bootstrap efforts, develop in-house expertise
  – Encourage multiple versions of code, benchmark, develop “rules of thumb”

• **UltraSparc T1, Intel/AMD Duo and Quad core**: Low risk
Resources
Selected Resources

GPGPU:

http://www.gpgpu.org: GPGPU resources

http://www.gpgpu.org/sc2006/workshop: SC’06 workshop


Cell BE:

http://www.ibm.com/developerworks/power/cell/: IBM’s Cell developer resources

MONARCH: John Granaki (granaki@isi.edu)

Me: scottm@aero.org