Lecture 9
Future Directions and Conclusions
Why is programming many-core processors costly today?

• Separate structure from CPU
  – Data isolation and marshalling with pressure to optimize away overhead
• Lack of standardized programming interface
  – Each has its own app development models and tools
• Management of specialized execution and memory resources
• Multi-dimensional optimizations required for achieving performance goals
A different approach from the past

- Simple parallelism
  - Focus on simple forms of parallelism for programmers
  - Trade some generality and performance for productivity
- Power tools
  - Leverage and strengthen app development frameworks
  - Empower tools with specification, analysis and domain information
Reduced Tuning Efforts
Tools need to do more heavy lifting

**Sum of Absolute Differences**

- More complex than matrix multiplication, but still relatively small (hundreds of lines of code)
- Many performance discontinuities

- Search spaces can be huge! Exhaustive search with smaller-than-usual data sets still takes significant time.


© David Kirk/NVIDIA and Wen-mei W. Hwu
Urbana, Illinois, August 18-22, 2008
Analytical Models Help Reduce Search Space.

Sum of Absolute Differences

By selecting only Pareto-optimal points, we pruned the search space by 98% and still found the optimal configuration.

High-level Frameworks for GPU

- Programming many-core GPUs requires restructuring computations around its coordination capabilities
- Global communication is very complicated
- Approach: put this complication in a code generation framework
  - Coordination is made explicit by expressing computation as MapReduce
- User specifies set of reduction functions, map & cleanup functions
- Framework generates efficient multistage reductions implemented in CUDA kernels

© David Kirk/NVIDIA and Wen-mei W. Hwu
Urbana, Illinois, August 18-22, 2008

Keutzer, UCB
Reaching deeper into apps.

- Many data parallel apps have a small number of simple, dominating components
  - Low hanging fruit for parallel computing (meat)

- Small computation components often dominate after the low hanging fruits are picked
  - Usually much more difficult to parallelize (pit)
What does it take to reach deeper?

• MRI: Launching multiple kernels
  – 3D FFT formulated as multiple 2D FFTs.
  – Multiple kernel types beneficial in some apps

• Global Synchronization
  – Some apps require global synch at time step boundaries

• Atomic memory operations
  – Shared memory, device global memory

• Less tedious tuning process for kernels
  – Developers run out of time!
High-level, Implicitly parallel programming with data structure and algorithm property annotations to enable auto parallelization

Locality annotation programming to eliminate need for explicit management of memory types and data transfers, potential ATI entry point

Parameterized CUDA programming using auto-tuning and optimization space pruning

1st generation CUDA programming with explicit, hardwired thread organizations and explicit management of memory types and data transfers
Summary – A multi-level attack on the parallel programming beast

- Simple programmer-level parallelism with power tools
- New Algorithm Frameworks
  - MapReduce (UCB), Convolution (UIUC), etc.
- New Application Frameworks
  - Video (UIUC), Game Physics (NVIDIA), etc.
- More consistent programming across HW platforms
  - MCUDA (IA Multi-core/Many-core), CUDA-lite (ATI GPU, FPGA)
- Better heavy-lifting tools
  - CUDA-tune, CUDA-auto
A Great Opportunity for Many

• GPU parallel computing allows
  – Drastic reduction in “time to discovery”
  – 1st principle-based simulation at meaningful scale
  – New, 3rd paradigm for research: computational experimentation

• The “democratization” of power to discover
  – $2,000/Teraflop SPFP in personal computers today
  – $5,000,000/Petaflops DPFP in clusters in two years
  – HW cost will no longer be the main barrier for big science
  – You will make the difference!

• Join the international CUDA research, education and development community