System Software Challenges and Opportunities on Asymmetric Multi-core Systems

Dimitris Nikolopoulos
Center for High-End Computing Systems
Department of Computer Science
Virginia Tech
dsn@cs.vt.edu
http://www.cs.vt.edu/~dsn
Asymmetric Multi-Core Systems

- Why asymmetric?
  - Buzz on Cell BE
  - Buzz on accelerator-based architectures (FPGA, GPGPU)

- More pragmatic reasons
  - Matchmaking between computational kernels and specialized cores (e.g. data-parallel, SIMD)
  - Amdahl’s law: few heavy cores with sub-linear speedup vs. many specialized cores with super-linear speedup
  - More flexibility in power management, opportunities beyond clock gating and voltage/frequency scaling
Challenges for system software

- **Taming heterogeneity**
  - Programming languages/libraries, user-level scheduler, OS, communication mechanisms
- **Working with multiple ISAs**
  - Single source (challenge for compilers)
  - Multiple sources (challenge for programmers)
- **Working with heterogeneity in operating systems (or hypervisors)**
  - Commodity OS running as host
  - Monolithic OS controlling or cores?
  - Specialized thin OS running on accelerators?
  - Inter-OS interfaces?
  - No OS at all? (a thin library instead…)
Challenges for optimization

- Seemingly endless choices of programming models
  - Any combination of SIMD, data-parallel, task-parallel, streaming,…
  - Any combination of MPI, OpenMP, SIMD,…
  - Parallelism in many layers (intra-core, inter-core, intra-accelerator, inter-accelerator, memory-level, interconnect-level,…)
  - Need new models of parallel computation, maybe new language constructs (or revised ones), robust libraries, operating system suppor
Polymorphic Parallelism on the Cell BE

- How can we synthesize an efficient parallel execution model?
  - Focus on efficiency, then think about programming models
- Bottom-up approach to synthesis: modeling
  - Synthesize a model of parallel computation
    - MMGP (Blagojevic et. Al. HiPEAC’2008)
    - Models DMA, SIMD, cross-SPU execution, cross-PPU execution
    - Model (simple) user-level schedulers
    - Error < 10% (typically < 5%)
  - Optimal (non-trivial) layered program decomposition given a sequential profile and generic annotations of parallelism
  - Still limited (does not model load imbalance, detailed memory access/contention, needs more than one programming models with more than one abstractions of parallelism)
Polymorphic Parallelism on the Cell BE

- Bottom-up approach to synthesis: execution
  - Event-driven thread scheduler (PPoPP’07)
  - PPU-SPU thread events visible to user-level and kernel-level scheduler
  - Events trigger:
    - Dynamic dependence-driven scheduling of PPUs
    - Dynamic allocation of SPU data-parallel tasks
    - 2.7x over Cell Linux

Diagram:
- Task completion
  - concurrency control/loop distribution
  - DMA scheduling buffer allocation
  - Task offload
  - co-scheduling
  - dependence-driven execution
Polymorphic Parallelism on the Cell BE

- **Key technique: adaptation**
  - Runtime phase analysis
    - Timing analysis
    - Characterization using hardware event counters
    - Memory traffic analysis
    - Power/thermal characterization (ongoing research)

- **Why adaptation?**
  - Search space for program configurations can be large
    - 4 layers of parallelism in the hardware (2-8 parallel execution units per layer, 1-2 options for communication per layer,)
    - 6-110 phases in typical benchmarks, with widely varying scalability/synchronization properties
    - Other dependencies (input, real or artificial load imbalance…)}
Future challenges

- **Vision**: one programming model unifying all forms of parallelism
- **Porting MPI, OpenMP,…**
  - Hard without synthesis of 2+ models, maybe inefficient…
- **Explicit parallel programming models**
  - Need a collection of simple, processor-independent constructs, expressing patterns of parallel computation
  - Use smart compiler/runtime system to map parallel computation
- **Topology awareness**
  - Mechanisms for localizing data and computation,
- **Hypervisors and virtualization**
  - Really thin OS for higher performance
  - Acceleration of system-level tasks (e.g. encryption, compression)