CMSC 714
Lecture 11
IBM Cell BE and GPUs vs. CPUs

Alan Sussman

Notes

- OpenMP project due tomorrow
  - questions?
- Research project info posted later today
  - due dates
  - topics from previous semesters

Cell Broadband Engine (Cell BE)

- GPU style design, initially targeted at graphics and gaming applications
- Each processor has one 64-bit PowerPC processor (PPE) and 8 synergistic processing elements (SPEs)
- SPE is a 128-bit SIMD processor with 256KB local memory
  - 128 bits can be used as 2 64-bit floats or integers, 4 32-bit floats or integers, 8 16-bit integers, or 16 8-bit chars
  - all memory ops are 128 bit, so smaller accesses more complicated – need masks, sometimes read-modify-write
  - up to 2 instructions per cycle, if to both even and odd pipe
  - since branches slow, default is branch not taken, but can use hint (an instruction) to change the default
    - also causes prefetch of instructions on new path
  - explicit movement of instructions and data between main memory and SPE local memories, using 128 byte unit DMAs (max 16KB each)
    - DMA engine is coherent with PPE caches and main memory

Cell Broadband Engine

- Need to compile both for PPE and SPEs
  - PPE is main control, and calls out to SPEs, typically via library calls – user can write code for both
  - need all sorts of optimizations to deal with SPE oddities
    - to operate on parts of the 128 bit data chunks
    - to move data and instructions into and out of local SPE memory
    - to optimize branches, instruction scheduling, etc.
    - align stream accesses properly
  - IBM also does some auto-parallelization (*auto-SIMDization*), so programmer doesn't have to write multiple programs
    - start from OpenMP code
    - compiler generates code sections for PPE and SPEs, and coordinates execution across them, with help from runtime library
    - need to do data transfer and code partitioning optimizations

GPUs vs. CPUs

- Study targeting throughput computing
  - Also called streaming applications sometimes, or data parallel
- Architectural limits to parallelism
  - CPUs have limited number of cores
  - GPUs have limited capabilities, e.g. no caches
- End results, on a set of representative benchmarks, is that GPU performs 2.5X faster than CPU
  - Application kernels include linear algebra (SGEMM from BLAS), Monte Carlo, Convolution, FFT, SAXPY (from BLAS), Lattice Boltzman (CFD), Constraint Solver, Sparse Matrix/Vector Multiply, Collision Detection (virtual environments), Radix Sort, Ray Casting, Index Search, Histogram, Bilateral Filter (image processing)
  - Platforms are Intel Core i7 CPU (4 hyper-threaded cores, 4-wide SIMD units, and caches) and NVIDIA GTX280 GPU (array of 30 SMs, each with 8 scalar processing units and local memory)

GPUs vs. CPUs

- Main advantage of CPU is caches
  - For fast single thread performance, but also helps with multi-threaded apps
  - Disadvantage is complexity, limiting number of cores per chip
  - Also have fast synchronization
- Main advantage of GPU is high throughput
  - each instruction for an SM executes on 8 scalar units (32 data elements)
  - Disadvantage is need to move data explicitly into (small) SM memory from large shared memory
  - Also have support for gather/scatter from memory and special functional units (e.g., texture sampling, math ops)
- Performance measurements for GPU assume data already in GPU memory (from other GPU computations)
- Overall performance of GPU (geometric mean) is 2.5X of CPU (n^{th} root of product of speedups)
  - Why? Because they optimized both CPU and GPU versions of the kernels