Blue Gene/L

- Scalable high performance distributed memory machine, with interesting design decisions
  - main goal is high performance with low power consumption, and high reliability
  - idea is to scale to large configurations of low power, less powerful individual components
  - Distributed memory system, with up to 64K nodes
- Each node is a dual processor chip, with integrated memory controller, network interface, caches, etc.
  - System on chip (SoC) design
  - 2 PowerPC cores, split L1 cache per core, L2 cache/prefetch buffer per core, 1 shared L3 cache, 512MB shared memory
    - L1 caches not hardware coherent, so need software help, other cache levels are coherent
  - Floating point multiply-add instructions for improved power/performance (when used)
  - Plus link chip for network component, for all routing and other network ops between nodes

Blue Gene/L

- 5 distinct networks, all interfaced to node through link chip
  - 3D torus is main message passing network – each node has 6 bi-directional nearest-neighbor links, with cut-through routing
  - Collective network for broadcasts, reductions, etc. – each node has 3 links (parent, 2 children, in a binary tree)
    - also forwards I/O requests to I/O nodes
  - Barrier network – for global sync operations (or other OR or AND operations from a set of nodes)
  - Gigabit Ethernet for file system access (I/O nodes only)
  - Fast Ethernet for initialization, diagnostics, debugging
- Programming model allows for either using each core for a separate process, or use 1 core as a communication co-processor

Notes

- Need volunteers for new posted papers
- Midterm currently scheduled for April 8
  - can we move to April 15 (day after interim project report due)?
- Sample topics for group project posted
  - proposal due March 24 (Wed. after break)
  - other due dates also posted
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