



Parallel Programming Basics

IBM
June, 2006

Parallel programming is essential to exploit modern computer architectures

- **Single processor performance is reaching limits**
 - Moore's Law still holds for transistor density, but...
 - Frequency is limited by heat dissipation and signal cross talk
 - Multi-core chips are everywhere...
- **Advances in network technology allow for extreme parallelization**

Parallel choices

- **MPI**
 - Good for tightly coupled computations
 - Exploits all networks and all OS
 - No limit on number of processors
 - Significant programming effort; debugging can be difficult
 - Master/Slave paradigm is supported, as well
- **OpenMP**
 - Easy to get parallel speed up
 - Limited to SMP (single node)
 - Typically applied at loop level ← limited scalability
- **Automatic parallelization by compiler**
 - Need clean programming to get advantage
- **pthread = Posix threads**
 - Good for loosely coupled computations
 - User controlled instantiation and locks
- **fork/exec**
 - Standard Unix/Linux technique

Parallel programming recommendations (for scientific and engineering computations)

- **Use MPI if possible**
 - Performance on SMP node is almost always at least as good as OpenMP
 - For 1-D, 2-D domain decomposition: schedule 2 months work
 - For 3-D domain decomposition: schedule 3-4 months
- **OpenMP can get good parallel speed up with minimal effort**
 - 1 week to get 60% efficient on 8 CPUs; 3 weeks to get 80%
 - May get best performance with `-qsmp=omp` instead of relying on compiler to auto-parallelize for older codes
 - Can use `-qsmp -qreport=smlist` to get candidate loops.
- **Hybrid is also possible**
 - OpenMP under MPI
- **threads are fine. Use them if it makes sense for your program.**