Benchmarking and Tuning for Parallel Programs

Charlie Peck
Introduction to Parallel Programming and Distributed Computing
UW and ISU
NCSI/SC Education Program
June, 2011
Introduction

• Benchmarking and tuning parallel software is just like improving the performance of serial software, just roughly $-n_p \times$ times harder to do...

• What is benchmarking? Accurately measuring the time and resource consumption profile of a program built with particular options with a given set of input data and run-time options.

• What is tuning? Improving the performance and/or resource consumption profile of a program built with particular options with a given set of input data and run-time options.

• Big question - How to use a given set of computational resources to solve a particular problem efficiently?
Resources

- CPU utilization
- Memory utilization (cache, RAM; space, bandwidth)
- Disk utilization (intentional and unintentional (e.g. paging))
- Network utilization (bandwidth, latency)
Overall Process

- Looking at the outside; what resources is it using?
- Looking at the inside; what is it doing to consume those resources?
- Working from the highest level to the lowest level; the most change is possible at the highest level, as you go down less change is possible since the lower layers are all in response to the higher layers.
  - The algorithm
  - The implementation of the algorithm
  - The compiler
  - The operating system
- The 80/20 rule.
- Time/space tradeoffs.
- The effect of the memory hierarchy.
- Style, clarity, generality; then tuning only if necessary.
Benchmarking

• Accurately measuring the time and resource consumption of a program built with particular options with a given set of input data and run-time options to find the nature and location of the bottleneck(s).

• Operating system level

  - `time` - system call, shell built-in and standalone
  - `vmstat`
  - `top`
  - `iostat`

• Program level

  - `printf()` or `cout` statements
  - `gprof` - statistical profiling (lab)
  - `getrusage()` - resource measurement from within the program
  - Performance counters (lab)
Tuning

• Improving the performance and/or resource consumption profile of a program built with particular options with a given set of input data and run-time options.

• Working from the top down because the most change is possible at the highest levels since lower levels are just responses to what happens at the levels above them.

• What work is being done? Where is it being done? Is there a more efficient way to accomplish the task?

• The process: measure, think, change one thing; measure, think, change one thing; measure, think, ...
Tuning - Continued

• Choice of algorithm

• Resource limits (ulimit -a)

• Compiler choice (GNU, Intel, etc.)

• Compiler optimizations (-O\_N, loop unrolling, etc.)

• Find an optimized library, e.g. Goto’s BLAS, that does what you need more efficiently/quickly
Parallel and Distributed Specific Tuning

- Latency and bandwidth; aggregation
- Synchronization
- Memory copies
- Network port contention
- Communication barriers
- Load balancing
The gprof Lab

- The lab is located at

- You will need a piece of serial code as one of the inputs to the lab, for today I suggest you use Henry’s serial N-Body code.
Resources

- gprof lab -

- man gprof

- IBM whitepaper on low–level tuning -

- High Performance Computing 2e, Severance and Dowd, O’Reilly, Sebastopol, CA

- Performance Optimization of Numerically Intensive Codes, Goedecker and Hoisie, SIAM Publishing, Philadelphia, PA