Benchmarking and Tuning for Parallel Programs

Charlie Peck
Earlham College

Introduction to Parallel Programming and Distributed Computing
NCSI and Blue Waters
August, 2012 @ OU/OSCER in steaming Norman, OK

Introduction

- Benchmarking and tuning parallel software is just like improving the performance of serial software, just roughly `-np X` 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
  - `MPI_Wtime()` and `MPI_Wtick()` and `MPI_WTIME_IS_GLOBAL`
  - `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 (`-ON`, 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
  `https://cluster.earlham.edu/wiki/index.php/Cluster:Gprof`
- You will need a piece of serial code as one of the inputs to the lab, for today I suggest you use the serial area under a curve. Note that you will need to significantly increase the problem size to see meaningful results. There is a serial version of this code contained in the lab.

Resources

- gprof lab - `https://cluster.earlham.edu/wiki/index.php/Cluster:Gprof`
- `man gprof`
- IBM whitepaper on low–level tuning - `http://www-128.ibm.com/developerworks/library/pa-bigiron3/`
- *High Performance Computing 2e*, Severance and Dowd, O'Reilly, Sebastopol, CA
- *Performance Optimization of Numerically Intensive Codes*, Goedecker and Hoisie, SIAM Publishing, Philadelphia, PA