Do's and Don'ts of Parallel Programming

Paul Lu
Associate Professor
Dept. of Computing Science
University of Alberta
Edmonton, Alberta, Canada
paullu@cs.ualberta.ca

Intended Audience, Introduction

- Beginner-to-intermediate parallel programmers
  - Advanced programmers might find this too basic
- Main ideas apply to shared-memory programming or distributed-memory programming
  - Pthreads, OpenMP
  - MPI
- In fact, many of these ideas also apply to sequential programming

Overview

- Don't
  - …optimize pre-maturely
  - …guess about bottlenecks. Benchmark. Measure. Consider Amdahl's Law and program phases.
  - …focus on granularity and high-overhead operations (e.g., I/O and memory-to-memory copies)
- Do
  - …write test programs to learn new ideas
  - …work with the system: compilers, libraries, debuggers, tools
  - …tap all available experts

Do: Write test programs

- When learning how to use Pthreads, OpenMP, MPI, etc., write lots of small test programs to learn new concepts
  - Trying to learn a new idea within a 5,000+ line application is difficult
  - Some new concepts (e.g., non-blocking sends in MPI, synchronization) are very different
  - If working from an example, do not modify the code before it is working correctly
- When debugging for correctness, consider forking your code and simplifying it to triangulate the bug

Don’t optimize pre-maturely

- As with sequential programming, do not optimize code until you know it will be a bottleneck
  - Focus on “the 20% of the code where you spend 80% of your time”
- Be clear, instead of being clever
- When it comes time to optimize, clear, modular, well-factored code will be easier to work with

Don’t guess about bottlenecks

- Use intuition to come up with ideas, but measure performance to prove or disprove your hypothesis
  - Don’t give in to “that section of the code **must** be the problem…”
- See later discussion about Amdahl’s Law and tools

Aside: Top 3 Lessons of Computer Performance

1. Granularity is king
   - I/O of any kind, including messages, will hurt performance
2. Optimize for the common case
3. Advanced: Avoid copying data (e.g., memory-to-memory) if at all possible

Aside: Amdahl's Law

- Many programs naturally break down into phases: read I/O, compute, communicate, compute, write I/O
  - Often, these phases are delimited by barrier synchronizations
- Do a scalability test (i.e., speedup or problem-size tests) and record the size (i.e., in seconds/minutes) of each phase
- Focus on the phases that do not speed-up linearly
- When debugging or optimizing **add** phases or barriers to narrow your focus

Do: Work with the system

- Do not try to out-smart the compiler
  - Especially with vector and new/old SIMD code, the compilers will be important
- Try different compilers
  - gcc is not always the best choice
  - Try the Intel compilers, Portland Group, etc.
- Consider standard libraries (e.g., FFTW, BLAS)
- Explore performance analysis tools

Do: Tap all experts

- Learn to ask questions of your local experts
  - Administrators and technical staff
  - Find/build a community of application users
  - Learn to ask good questions
- Use Google and Wikipedia

Do: Use a debugger, etc.

- Find the best debugger for your system. Learn to use it.
  - TotalView
- Learn a few performance monitoring tools

Concluding Remarks

- Don't
  - …optimize pre-maturely
  - …guess about bottlenecks. Benchmark. Measure. Consider Amdahl's Law and program phases.
  - …focus on granularity and high-overhead operations (e.g., I/O and memory-to-memory copies)
- Do
  - …write test programs to learn new ideas
  - …work with the system: compilers, libraries, debuggers, tools
  - …tap all available experts

