Programming for Performance: Assignment 3 (v1)

Patrick Lam

Due: March 7, 2011

Now that you've manually written parallel code using pthreads or a glib interface to it, we'll move on to 1) automatic parallelization (L12) and 2) manually specifying parallelization with OpenMP (L13 v2), both 2a) for array-based programs (for-loop parallelization) and 2b) task-based OpenMP parallelization.

Gentle reminder: you need to hand in code that compiles. Otherwise you get 0 points for that question.

Part 1: automatic parallelization (40 points)

In this part of the assignment, you'll play with automatic parallelization and try to get it working on some production code. In class, I talked about the Oracle Solaris Studio compiler, which you can download on the Internet. I also found another free (as-in-speech) parallelizing compiler, Open64, which AMD productized[1]. You can read the documentation[2] and find out about the `-apo` (automatic parallelization) and `-mso` (optimize for multicores) options. Figure out how to make it tell you what it did.

Another option is the Portland Group compiler[3], which is not free. You can download a 15-day trial version. It works on Mac OS X. I have no Mac and no experience with the PGI compiler.

Part 1a (10 points). First, consider the `dither.c` file in `lib/a2jpeg`. There are two loops that you might parallelize. Rewrite the loops so that they are parallelizable. (That part is trivial.) Argue that your transformations are correct. For fun, try out and benchmark the parallelized version. You get props, but no credit, for submitting benchmark results.

Part 1b (30 points). The next two tasks are going to parallelize the ancient `meschach` matrix library. I've made a git repository and a tarball available on

[1] `http://developer.amd.com/cpu/open64/pages/default.aspx`
[2] `http://developer.amd.com/cpu/open64/Documents/open64.html`
[3] `http://www.pgroup.com/products/pgiworkstation.htm`

the course website. You need to run my version of the `torture` executable (run `make torture`).

In this task, find two loops that you can rewrite to automatically parallelize. They don't need to be profitable; I just want to see the compiler say "PARALLELIZED". You actually have to do some (possibly-trivial) change, though; it's not good enough to point at the loops the compiler can automatically parallelize. Show me the diff (including the makefile and your C files) and the compiler output showing successful parallelization.

Part 2: parallelization with OpenMP (60 points)

This part of the assignment has two subparts: 1) using OpenMP to parallelize array codes, and 2) using OpenMP to parallelize dynamic data structure traversal, as supported by OpenMP 3.0's tasks.

Task 2a (40 points). Now use profiling to figure out how to best parallelize `torture`, add OpenMP parallelization directives, and show me the runtimes and evidence that you effectively parallelized the code[4]. Think about whether you need to use locks or not to make the code correct; if you think the answer is no, explain why not. Otherwise, submit the code with locks.

Task 2b (20 points). For this part, you will OpenMP `task` specifications (L13 v2) to parallelize some recursive tree traversal code. I've made available the complete `antiword` source code. `antiword` converts Word files into non-proprietary formats. However, you only need the `a3-computepps-code.tar.gz` tarball which I've put up, containing the relevant function from `computepps.c` and its dependencies. Hand in, as diffs, OpenMP task specifications which parallelize the `vComputePPSlevels` function in the `computepps.c` file.

[4]My experience with using the Solaris C compiler is that `torture` reports an error in the matrix-vector multiplies. I encourage you to fix it, but I'm not making that an assignment requirement.