ECE 459: Programming for Performance
Assignment 2 (revision 2)[*]

Patrick Lam

February 4, 2013 (Due: February 25, 2013)

Important Notes:

- **Make sure your working directory is called "assignment-02".**
- **Make sure you run your program on `ece459-1.uwaterloo.ca`.**
- **Use the command "`OMP_NUM_THREADS=4;export OMP_NUM_THREADS`" to set 4 threads.**
- **Run "`make report`" and "`make tar`" and submit your `assignment-02` directory.**
- **Make sure you don't change the behaviour of the program.**

Automatic Parallelization (40 marks)

The Riemann Hypothesis is an important open question in pure mathematics about where Riemann zeta function may have zeros. I've provided some code to compute the Riemann zeta function at various points in `zeta.c`. This code takes a number of parameters: a number of iterations `r`, a number of points `N`, and a starting point `d`. Benchmark your sequential program such that the execution time is approximately 4 seconds; if you're initially not running on `ece459-1` then you may want to adjust the parameters, but my default parameters work there.

Note that the compiler does manage to optimize the computation of the zeta function quite a bit. Report the speedup due to the compiler and speculate about why that is the case. Compare all subsequent numbers to the optimized version `zeta_opt`.

Your first programming task is to modify your program so you can take advantage of automatic parallelization. Figure out why it won't parallelize as is, and make any changes necessary. Preserve behaviour and make all your changes to `zeta_auto.c`.

I put Solaris Studio 12.3 on `ece459-1`. The provided `Makefile` calls that compiler with the relevant flags. Your compiler output should look something like the following (the line numbers don't have to match, but you **must** parallelize the critical loop):

```
Compiling Part 1 Automatic Parallelization
/opt/oracle/solarisstudio12.3/bin/cc  -fast -xautopar -xloopinfo -xreduction -xbuiltin
    src/zeta_auto.c -o bin/zeta_auto
"src/zeta_auto.c", line 170: not parallelized, not profitable
"src/zeta_auto.c", line 175: not parallelized, not profitable
"src/zeta_auto.c", line 180: not parallelized, not profitable
"src/zeta_auto.c", line 170: not parallelized, not profitable (inlined loop)
"src/zeta_auto.c", line 170: not parallelized, not profitable (inlined loop)
"src/zeta_auto.c", line 175: not parallelized, not profitable (inlined loop)
"src/zeta_auto.c", line 180: not parallelized, not profitable (inlined loop)
"src/zeta_auto.c", line 200: PARALLELIZED, reduction, and serial version generated
"src/zeta_auto.c", line 206: PARALLELIZED, reduction, and serial version generated
```

[*]No major changes to deliverables; see git history for details.

```
"src/zeta_auto.c", line 170: not parallelized, not profitable (inlined loop)
"src/zeta_auto.c", line 175: not parallelized, not profitable (inlined loop)
"src/zeta_auto.c", line 180: not parallelized, not profitable (inlined loop)
"src/zeta_auto.c", line 200: PARALLELIZED, reduction, and serial version generated (inlined loop)
"src/zeta_auto.c", line 206: PARALLELIZED, reduction, and serial version generated (inlined loop)
"src/zeta_auto.c", line 222: not parallelized, loop has multiple exits
"src/zeta_auto.c", line 240: not parallelized, unsafe dependence (d)
"src/zeta_auto.c", line 245: PARALLELIZED, reduction, and serial version generated
```

Clearly and concisely explain your changes. Explain why the code would not parallelize initially, why your changes are correct, and why they preserve the behaviour of the sequential version. Run your benchmark again and calculate your speedup. My speedup is less than 1, although I've gotten reports of better speedups. Speculate about why you got your speedup—keep in mind that, for the next part, the speedup will be greater than 3.5.

- **Minimum expected speedup:** 0.5
- **(my) initial solution speedup:** 0.61

Manual Parallelization with OpenMP (30 marks)

Now, it's time to show the compiler who's boss. Copy the original `zeta.c` file over to `zeta_omp.c` (I've done that already in the provided tarball). Now, using OpenMP pragmas and minor code structure changes that you'll need to support them, increase the performance of the program over automatic parallelization. Benchmark your manual parallelization with OpenMP and calculate the speedup over the sequential version. If applicable, using your explanation from part 1 (as to why the performance was so bad), explain why your manual parallelization so drastically improved performance. Be clear and concise.

- **Minimum expected speedup:** 3.5
- **(my) initial solution speedup:** 5.03

Using OpenMP Tasks (30 marks)

We saw briefly how OpenMP tasks allow us to easily express some parallelism. In this part, we apply OpenMP tasks to a Newton's method approximation to the golden ratio, φ. Benchmark the sequential version with a parameter that takes around 10 seconds (31 on `ece459-1`). Now, try to run the provided version with OpenMP tasks (let it run for at least 20 seconds to prove a point, let it run to completion to drive the point home [but not on `ece459-1` near the submission deadline!]). It should be much slower than the sequential version. In your report, clearly and concisely explain why the performance is so dreadful.

Next, we'll modify the code so that there's actually a speedup. Without modifying the structure of the naive calculation (it has to remain recursive with no caching, etc), change the code so that you get some improvement. A Google search reveals that a cutoff that decides whether or not to create tasks should work. The only modifications in this case should be a variable and an if statement. Benchmark your modified program and calculate the speedup over the initial sequential version. Clearly and concisely explain why your changes improved performance.

Note that `gcc` is really good at optimizing this code, so that if you give it `-O2`, the resulting code will basically run instantly. Don't use `-O2`.

- **Minimum expected speedup:** 1
- **(my) initial solution speedup:** 3

Setup

As with Assignment 1, the provided material is available here:

`http://patricklam.ca/p4p/assignments/provided-assignment-02.tar.gz`

as well as in the course SVN repository. Use the provided `Makefile`; do not modify it. Submit your results to your SVN repository. You can submit either a tarball or just your whole directory structure. Grading will be done by running `make`, running your programs, looking at the source code and reading the report.