CS194-15: Engineering Parallel Software Assignment 2

September 12, 2013

1 Introduction

The goal of this assignment is to introduce OpenMP and **pThreads** programming by parallelizing matrix multiply and image blurring.

1.1 Matrix multiply

Matrix multiply, shown in Listing 1, is a short and simple kernel.

```
void matmuld(double a[1024][1024], double b[1024][1024], double c[1024][1024])
{
  for(int i=0;i<1024;i++)
    for(int j=0;j<1024;j++)
      for(int k=0;k<1024;k++)
        c[i][j] += a[i][k]*b[k][j];
}
```

Listing 1: Three-nested loop matrix multiply

1.2 Image blurring

Image blurring (two-dimensional convolution [1]) is a slighty more complex kernel. The source for the image processing kernel you'll be working with is slightly too large to comfortabilty fit on a single page. Instead, refer to the function **blur_frame** in the source file `conv2d.cpp` as the baseline. You should note that the blurring radius can vary from pixel to pixel.

2 Getting the assignment

The source for the assignment is on both Piazza and bSpace.

3 Matrix multiply

3.1 Programming problems

3.1.1 OpenMP parallel for-loops

Add the appropriate pragmas to the file `omp_matmul_for.cpp` to enable parallel for-loops.

3.1.2 OpenMP parallel tasks

Add the appropriate pragmas to the file `omp_matmul_task.cpp` to parallelize matrix-multiply using OpenMP tasks.

3.1.3 pThreads

Add the appropriate function calls to the file `pthread_matmul.cpp` to parallelize matrix-matrix using **pThreads**.

[1] See Wikipedia if you're unfamilar with convolution

3.2 Matrix multiply questions

Please answer each of the following questions:

3.2.1 Scaling plot

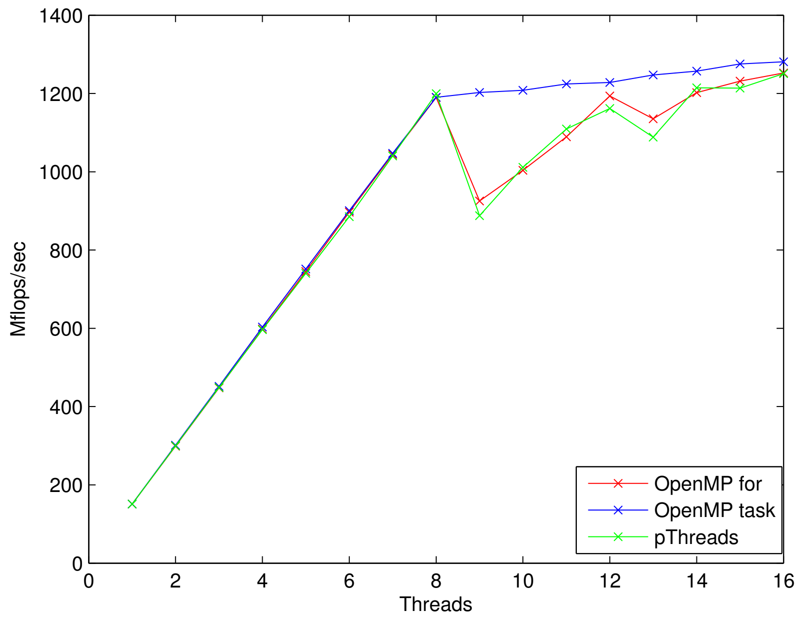

Figure 1: An example scaling plot

For each of the three approaches, make a scaling plot. An example of a scaling plot is shown in Figure 1. The framework included with this assignment calls your code with a varying number of threads. Performance (in mflops/sec) is reported. Please plot these results.

3.2.2 Peak performance

How many times faster was your best parallel implementation compared to the scalar baseline? Did you achieve linear scaling? Can you explain your performance results?

3.2.3 Measuring up

The **hive** machines have a peak performance of 76.8 double-precision gigaflops per second. How does your matrix-multiply performance compare to the peak? Why are you achieving less than peak performance? What sorts of optimizations would you propose to improve performance? (don't implement, just tell us what you're thinking)?

3.2.4 Personal preferences

Which do you like using more: OpenMP or **pThreads**? Why?

3.2.5 How does OpenMP work?

The purpose of this question is to better understand how OpenMP works.

3.2.6 How do you think OpenMP works?

Try compiling your version OpenMP parallel-task matrix-multiply code with following command:

```
g++ -O3 -fopenmp -fdump-tree-ssa -c matmul.cpp
```

This command will dump GCC's internal representation of your program (the intermediate representation, for those of you who have taken a compilers course). On my machine, the intermediate representation is saved in the file `matmul.cpp.017t.ssa`. The filename may change on your machine, but will always end in `ssa`.

Look for functions and structures with "omp" in their name. Explain how you think OpenMP launches parallel work when you use parallel tasks.

4 Image blurring

Choose either OpenMP or **pThreads** and parallelize the **blur_frame** function in the file `conv2d.cpp`.

4.1 Image blurring questions

4.1.1 Make a scaling plot with fixed blurring radius

Make a scaling plot (like in Section 3.2.1) for the parallelized image blurring code when the blurring radius is fixed to a distance of 1. This blur radius is enabled by calling the executable with the parameter `-n1`.

4.1.2 Make a scaling plot with a variable blurring radius

Rerun your image blurring code with a variable radius blur with a maximium radius of 10. The variable radius blur is enabled by calling the executable with the parameter `-n10`. Does the code scale as well as with a fixed blurring radius. If it does not scale as well as before, can you think of a reason why scaling is reduced? Also, propose solutions to improve scalability (but you do not need to implement them).

4.2 How much time did you spend on this assignment?

This assignment is designed to be a straightforward assignment. How much time did you spend on it?

5 What to turn in

Please answer the questions to Sections 3.2 and 4.1 and include your answers as a PDF. Use bSpace to submit your answers. No code please.