CITS3402 High Performance Computing Programming Project 1

CITS3402 High Performance Computing

Programming Project

The aim of the programming project is to parallelize a C code on multi-core machines to gain as much speed-up as possible. As such, I expect the coding to be minimal, as understanding is the key to the success in this project.

Background: Though understanding of the problem is not strictly necessary, Finite Elements Method (FEM) is widely used in engineering and science for solving *boundary value problems*. It is very common to model the changes of physical quantities like stress, flield strength etc. by using differential equations or partial differential equations. Usually the values of these physical quantities over some domain (one, two or three dimensional space) are solved subject to some boundary conditions (the value of the physical quantities at the boundary of the domain). The aim is to determine the values of the physical quantities at every point of the domain. However, there are an infinite number of points in the domain and quite often the only possible way to solve these problems is through numerical techniques, using a computer. The domain is usually divided into a *mesh* and the differential equation is converted into a difference equation and the solution is obtained through an error minimizing technique, iteratively. If you are interested in learning more about FEM, this Wikipedia article is a good point to start. There are also numerous tutorials on the web on FEM.

FEM1D is a popular FEM program for solving 1-dimensional finite element problems. There are many versions of this program depending on the exact boundary value problem to solve. We will use one of the versions in this project. Here is the code for FEM1D.

The program is written in a bit of old style C. You may not like the code, but this is often the case that codes for scientific computing are not always written keeping in mind the best practices. You are very welcome to change the code and write it in a nice way, however, you have to ensure that the code does what it is supposed to do (more on that below). You can also complaint about the poor quality of the code, I will listen patiently.

Your task in this project is to improve the runtime performance of this program using OpenMP running it on a multicore machine.

There are two constants in the code that need particular attention from you, NSUB and NL. In particular, you will notice that there are many arrays declared with these constants (or their combinations) as sizes of these arrays. Unfortunately C run time system allocates all arrays declared like this in the stack and there is a rather small limit on how large an array you can allocate in a stack. You will need to experiment with much larger arrays in this project and hence you need to allocate these arrays in the heap. If you are confused, you can listen to this video to understand more about stack and heap.

You can allocate the storage for an array in the heap in the following way:
Suppose the array is `double f[NSUB+1];`
You have to declare a pointer to double:

```
double *f;
```

and then allocate storage inside your function:

```
f=(double *)malloc(sizeof(double)*(NSUB+1));
```

You can now access the elements of the array by referring to `f[0], f[1]` etc.

The program prints many things. Another of your tasks is to print the output of the program in a file so that you can compare the output of the original program with your modified program that you have written using OpenMP. You need to change the `printf` statements to `fprintf` statements for doing this. You have to first declare:

```
FILE *fp, *fopen();
```

then open a file called `out.txt` (you can use any other name)

```
fp=fopen("out.txt","w+");
```

and change a `printf` statement to `fprintf`:

```
fprintf(fp,......);
```

The deliverables:

- The first deliverable is of course your modified C code with appropriate OpenMP directives. You should also comment your code suitably, so that the code itself can be read and your modifications can be understood.
- You have to also submit a (reasonably) small document where you should explain how you have implemented the parallelism in the code and why. It should also contain extensive performance evaluation of your code with different parameters. I am not looking for anything specific here, rather well designed performance evaluation and speed-up results that highlight your work.

- The main emphasis in the evaluation of your project is on how extensively you have done your experiments and how well you have documented the results of your experiments. You should not despair if you do not get spectacular speed up. You should concentrate more on well thought out experiments that you have performed and that you believe should improve the runtime performance of this code.

Marks: The total marks for this project is 25 (15 marks for the quality of your experiments, 8 marks for the quality of the report and 2 marks for innovative solutions). I will evaluate the project through a comparative assessment, i.e., against the best submission from the class.

Submission: The deadline for the submission of the project is 11:59 pm, September 30 through cssubmit. You can submit up to five files for C code and only one file for the report. The report file **must be** a pdf file. The report should mention the names of the group members (if you are doing the project in a group of two) and the student ids. The name **must be** your official name as recorded in the university records, not your nickname.

Note: The project can be done either individually or in a group consisting of a maximum of two students.

Amitava Datta
September 2015

School of Computer Science & Software Engineering
The University of Western Australia
Crawley, Western Australia, 6009.
Phone: +61 8 9380 2716 - Fax: +61 8 9380 1089.
CRICOS provider code 00126G