Parallel Matrix Vector Project Description Math 4610

Project Description:

This project involves the implementation of an algorithm that computes the product of a matrix times a vector using multiple cores. If precise, the algorithm will implement a matrix multiplied into a column vector from the left. That is, if A is a matrix with m the number of rows in the matrix and n the number of columns in the matrix, and the vector $\mathbf{x}$ is a column vector of length n compute the vector $\mathbf{y}$ given by

$$\mathbf{y} \leftarrow A\,\mathbf{x}$$

Note that the serial algorithm (no parallelization) is relatively simple to implement in most computer languages. For example, C, C++, Fortran, Java, and so on.

Project Description:

There are a number of ways you can complete this project. However, it may be most efficient if the following steps are used to build code, test examples, and/or create reusable code that can be used in this course and other situations where this type of algorithm is needed, say the power method used to compute the largest eigenvalue of a square matrix.

- If you have not already written a serial version of an algorithm do so in C, C++, or Fortran.
- Test your serial code using known matrices and input/output vectors on small matrices - say under 10 rows and columns.
- Place timing methods/routines in your serial code that will return the amount of time needed to complete the work of multiplying the input matrix into the input vector. For C and C++, you will need to include time.h. In fortran, there are a couple of intrinsic routines that can be used identify the start and end times for any algorithm. Look these things up online.
- Copy your serial code into another file and use the serial algorithm to incorporate directives that will parallelize the algorithm. In this project, you will need to acquaint yourself with OpenMp which is readily available online. Note that most C, C++, and Fortran compilers recognize OpenMP directives and use these to optimize code on multicore computers.
- Using compiler flags, compile the parallel code. For example, for C and C++, use

```
% gcc -fopenmp -o matvec_p matvec_p.c
```

  and in a Fortran setting,

```
% gfortran -fopenmp -o matvec_p matvec_p.f
```

  In these commands, the % indicates a linux/unix prompt. The next token is the compiler being invoked (gcc/gfortran) and the rest involve the name of the file and the executable file.
- Make sure the serial and parallel codes output the same results. In some implementations will cause the computations to be reordered.
- Graph the time each of the codes take to complete the work in the multiplication of the matrix into the vector from the left. Graph the compute time as the size of the matrix increases.
- Document all of your work in your Github repository for the course.

Remarks Students May Need to Consider:

- You may need to add a couple of directives to make sure the parallel code does the correct thing.

- It might make sense to try out optimization flags in the compiler that do not use OpenMP. For example, using

```
% gcc -O1 o matvec_p matvec_p.c
```

  Note that the 1 in the 'O' flag indicates the level of optimization provided. In the gcc compiler you can specify more optimization by increasing the '1'. For example,

```
% gcc -O2 o matvec_p matvec_p.c
```

  and

```
% gcc -O3 o matvec_p matvec_p.c
```

  should work. Although 'O3' may lead to inconsistent results. You should always test these types of codes using known inputs and outputs.