

Dense matrix multiplication (OpenMP)

Introduction

The purpose of this assignment is to become familiar with OpenMP by implementing dense matrix-matrix multiplication.

Dense matrix multiplication ($C = AB$)

You need to write a program, ***matmult_omp***, that will take as input two matrices *A* and *B*, and will output their product. A baseline almost complete serial multiplication program, ***matmult***, is provided to you as C code. It is just an example. You can write your OpenMP code in C or C++. Complete the program by initializing matrices with random values and adding a timer solely around the matrix multiplication function call. You should use the *omp_get_wtime* function for timing. Your program should be able to be executed as:

```
matmult 4 5 3 [optional parameters]
```

which would create two matrices with random values, *A* of size 4 x 5 and *B* of size 5 x 3 and would multiply them, storing the result into a matrix *C* of size 4 x 3. The program should print out execution time in microseconds with up to 4 decimal points. You may optionally add a 4th parameter for nthreads (number of threads the program should execute the multiplication with).

General program

Design your programs so that they follow the following steps:

1. A single thread creates the matrices.
2. Multiple threads work together to do the matrix multiplication.
3. At the end, one thread reports the execution time.

These are just guidelines, and you should think carefully about step 2. Also, make sure the matrix multiplication computation is correct.

Parallelization strategy

Implement 2 parallelization strategies: with and without *tiling*. The goal of the experiments will be to compare the execution time for the *tiling* and *block* (without tiling) strategies. For the tiling case, you may want to experiment with different tile sizes to find the optimum values.

Baseline code

Serial baseline code in C is provided for you. After you finish the program, execute 'make' to compile the code, and 'make clean' to remove the executable and artifacts.

Testing

Test your code on the HPC compute nodes (the cpu queue) with the following parameters. For each execution, store results in log file which you can later use to create tables or figures for your report.

Parallelization: tiling or block

Number of threads: 1, 2, 4, 8, 12, 14, 16, 20, 24, 28.

Matrixes:

A nrows	A ncols	B ncols
1000	1000	1000
1000	2000	5000
9000	2500	3750

Note that the evaluation of your code may be done using a different set of matrices than the ones above. As such, do not over-specify your code to work perfectly on those matrices and less so on others not in that set.

What you need to turn in

1. The source code of your program.
2. A short report that includes the following:
 - a. A short description of how you went about parallelizing dense matrix multiplication. One thing that you should be sure to include is how you decided what work each thread would be responsible for doing.
 - b. Timing results for your parallel executions. These results should be reported in a weak scaling chart reporting the speedup at different numbers of threads vs the serial execution (1 thread).
 - c. A brief analysis of your results. Some things to consider might be:
 - i. How does the number of threads affect the runtime? Why do you think that is?
 - ii. How does the size of the problem affect the runtime? Why do you think that is?
 - iii. How well does your program scale, i.e., if you keep increasing the number of threads, do you think the performance will keep increasing at the same rate?
 - iv. Are performance characteristics consistent for the different shapes of the matrices? Why do you think that is?

Submission specifications

- A makefile must be provided to compile and generate the executable file.
- The executable file should be named 'matmult_omp'.
- All files (code + report) MUST be in a single directory and the directory's name MUST be your university student ID. Your submission directory MUST include at least the following files (other auxiliary files may also be included):

```
<Student ID>/matmult_omp.cpp  
<Student ID>/Makefile  
<Student ID>/report.pdf
```

- Submission MUST be in .tar.gz
- The following sequence of commands should work on your submission file:

```
tar xzvf <Student ID>.tar.gz  
cd <Student ID>  
make  
ls -ld matmult_omp
```

This ensures that your submission is packaged correctly, your directory is named correctly, your makefile works correctly, and your output executable files are named correctly. If any of these does not work, modify it so that you do not lose points. I can answer questions about correctly formatting your submission BEFORE the assignment is due. Do not expect questions to be answered the night it is due.

Evaluation criteria

The goal for this assignment is for you to become familiar with the OpenMP API and develop an efficient parallel program. As such, the following things will be evaluated:

1. follows the assignment directions,
2. solve the problem correctly,
3. do so in parallel,
4. achieve speedup:
 - 5 / 10 points will be reserved for this criterion.
 - points will be awarded by comparing the relative performance of your solution to the solution of the benchmark program provided.
 - The speedups obtained will probably depend on the matrix sizes. It is not expected that you to get good speedups for small matrices, but you should be able to get good speedups for large ones.