CS 133 Lab 1

Design and Implement Parallel Dense Matrix Multiplication with OpenMP for Multicore CPUs

Instructor: Prof. Jason Cong

April 14th, 2016

1 Description

Your assignment is to parallelize dense matrix multiplication with OpenMP based on the sequential implementation we provided. Specifically, for matrix multiplication $C(I, J) = A(I, K) \times B(K, J)$, you will need to implement two functions using OpenMP:

```
void mmul1(float A[I][K], float B[K][J], float C[I][J])
void mmul2(float A[I][K], float B[K][J], float C[I][J])
```

`mmul1`: A parallel version of the given matrix multiplication routine using OpenMP pragmas. Please note that some loop permutation might be needed to enable more concurrency. You should provide the result comparing the parallel version with the sequential version. Please use the input size `2048 x 2048` for all three matrices.

`mmul2`: A *blocked matrix multiplication* with OpenMP. You should select the block size with the best performance when you submit this lab. You are welcome to add any other optimization to further increase the performance (please follow Section 3.2 to describe every optimization you performed in the report).

Tip: Start early, and use "top" command before running your program to see the system workload. Your results will be more accurate when there is fewer tasks running at the same time.

2 Preparation

2.1 Provided files

The project code can be found on cs133.seas.ucla.edu:

```
/u/cs/class/cs133/cs133ta/release/lab1.tar.gz
```

Please untar the file using `tar -xzf lab1.tar.gz`, and you will get a folder "lab1" which contains the following files:

- mmul_main.c: Contains main function that generates the input data, measures the execution time, and checks the output data.
- mmul1.c: Please modify this code to perform simple OpenMP matrix multiplication.
- mmul2.c: Please modify this code to perform blocked OpenMP matrix multiplication.

- const.h: You can adjust the matrix size in this file.
- lab1_test: The submission checking script.

2.2 Compilation and Execution

After finishing the modification, you can compile your program by

```
gcc -o mmul *.c -fopenmp -lm -O3
```

then you will get an executable file `mmul`, which can be executed by

```
./mmul
```

3 Submission

3.1 Files

Your submission should be zipped in a tarball named "lab1-uid.tar.gz". In this tarball, you have to create a folder with your UID as the name, and include **ONLY** the following files in the folder. For example, if your UID is 000000000, then your submission would look like:

```
lab1-000000000.tar.gz
└── 000000000
    ├── mmul1.c
    ├── mmul2.c
    └── lab1.pdf
```

Note that we use automated scripts to grade your submission, so you **MUST** run the checking script to make sure your submission format is correct before submitting your files. The checking script can be found in the provided tarball, and you can run it like:

```
./lab1_test <Your UID>
```

Once you get the following message, that means your submission is able to be graded.

```
Pass file checking. Please upload your file to CCLE.
```

3.2 Report

You have to submit a brief lab report named "lab1.pdf" along with your codes, which summarizes:

- Your methods to parallelize and optimize the programs in mmul1.c and mmul2.c.
- The results comparing the sequential version with the optimized parallel version on three different input sizes (`1024 x 1024`, `2048 x 2048`, `4096 x 4096`). If you get significantly different speedup number for the different sizes, please explain why.
- (mmul2 only) Performance results of your OpenMP implementation on cs133.seas.ucla.edu. Please express your performance in GFlop/s ($= ni * nj \times nk \times 2/exec\_time$), and the speedup compared to the serial version. Please report the speedup and the performance number with each optimization techniques you have applied.

- (mmul2 only) For the input size `2048 x 2048`, please quantify the impact of each optimization, including the block size selection and any other optimization you performed.
- (mmul2 only) For the input size `2048 x 2048`, please report the scalability of your optimized code with different number of threads, including 2, 4, 8, 16, and 32.
- A discussion of your result.
- (optional) Challenges you encountered and how did you solve them.

4 Grading Policy

4.1 Submission format (10%)

You might be deducted if your submission cannot pass the checking script. In case of missing reports, missing codes, or compilation error, you might receive 0 for that category.

4.2 Correctness (50%)

You can use the routine in `mmul_main.c` to check the correctness of your code. Try with different matrix size between `256 x 256` and `4096 x 4096`, as well as other rectangular shapes (e.g. `A(1024 x 256)` and `B(256 x 2048)`).

4.3 Performance (25%)

Your performance will be evaluated based on the 32-thread performance on the multiplication of matrices of size either `2048 x 2048` or `4096 x 4096`. The performance point will be added *only if you have the correct result*, so please prioritize the correctness over performance. Your performance will be uniformly distributed from 0% to 100% based on the ranking among fellow student submissions.

4.4 Report (15%)

You might be deducted if your report misses any required section that described above.