CS 133 Lab 2

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

Instructor: Prof. Jason Cong

May 9th, 2016

1 Description

Your assignment is to parallelize dense matrix multiplication with MPI 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 a *blocked matrix multiplication* function using MPI:

```
void mmul(float A[I][K], float B[K][J], float C[I][J], int n)
```

where n is the size of the i, j, k dimension of the matrix multiplication. Note that we will only test *squared matrices*, and the matrix size is always *power of 2*. You should select the block size with the best performance when you submit this lab.

Caution: All three matrices (`float *A`), `float *B`, and `float *C`) have to be created and initialized **ONLY** at processor 0 (Please see `mmul_main.c` for details). That means your code in `mmul()` should explicitly send a part of matrices A and B from processor 0 to each processors. After the computation, processor 0 should receive the data from all processors and store it to matrix C.

2 Preparation

2.1 Provided files

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

```
/u/cs/class/cs133/cs133ta/release/lab2.tar.gz
```

Please untar the file using `tar -xzf lab1.tar.gz`, and you will get a folder "release" 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.
- mmul.c: Please modify this code to perform parallel blocked matrix multiplication.
- Makefile: A simple makefile to compile and run the program.
- setenv.sh: A setup file for MPI related environment settings.
- lab2_test: The submission checking script.

2.2 Compilation and Execution

Before compilation and execution programs using MPI, you need to setup the MPI compiler and runtime system by

```
source setenv.sh
```

The environment setting includes MPI library path and GCC version. Note that you **MUST** use the GCC version we specified in the `setenv.sh` instead of any other version. Once you setup the environment successfully, you should see the following message if you type `mpicc`:

```
gcc: no input files
```

Please note that you need to source the setup file **every time** when you log in the server. You can also add the source command to your `.bashrc` to automatic setup that.

After finishing the lab2, you can compile your program by using provided makefile:

```
make
```

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

```
mpirun -np 16 mmul
```

where 16 is the number of processors that will be running the program.

We also provide three configuration variables for your convenience:

- seq: The sequential execution is used for verifying the correctness of your result. However, you can turn off it by specifying `seq=0` when compiling your program:

  ```
  make seq=0
  ```

  you will get a warning message on this mode for reminding you to verify the correctness later on.

- size: You can change the matrix size by specifying `size` when running the program (default is 1024). For example:

  ```
  make run size=2048
  ```

- np: When running your program, you can specify the number of processors you want to use (default is 16). For example:

  ```
  make run np=8
  ```

Note that the later two variables can be used together (e.g. `make run np=8 size=2048`). Please test your program on our class server: cs133.seas.ucla.edu

3 Submission

3.1 Files

You can use `tar -czf lab2-uid.tar.gz uid` to zip your submission in a tarball. 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:

```
lab2-000000000.tar.gz
└── 000000000
    ├── mmul.c
    └── lab2.pdf
```

We will 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:

```
./lab2_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.
```

Note that we will use our own `mmul_main.c` and makefile, so you have to put all of your program settings to `mmul.c`. Also, you cannot use any OpenMP pragma in `mmul.c`.

3.2 Report

You have to submit a brief lab report named "lab2.pdf" along with your codes, which summarizes:

- The experimental result and discussion. Please report and quantify the impact of all experiments you have done such as block size and number of processors. Please express your performance in GFlop/s ($n \times n \times n \times 2/exectime$).
- Please briefly explain how you have partitioned the data and the computation among the processors. Also, briefly explain how the communication among processors is being done.
- Please provide the runtime using blocking (`MPI_Send`, `MPI_Recv`), buffered blocking (`MPI_Bsend`, `MPI_IRecv`), and non-blocking (`MPI_Isend`, `MPI_Irecv`) communication. Which type of communication gives best result? Please explain why. Note that the final submission file should use the type of communication that gives the best result.
- Please report the scalability of your algorithm using `mpirun np 4, 8, 16, 32`. Do you get linear speedup? If not, why? Note that the final submission file should be optimized for 16.
- How is this MPI implementation when comparing to your OpenMP implementation in lab 1 in terms of the throughput the programming effort?

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%)

The correctness of this lab is defined as follows:

$$\forall C(i,j) \in C, |C(i,j) - C_0(i,j)| < 10^{-4} \tag{1}$$

where C is the result matrix by your program, and C_0 is the matrix by the sequential execution. You can use the routines in `mmul_main.c` to check the correctness of your code. The return value means the number of elements that violate the correctness rule. You have to make sure there has **0** violated element before submitting your program.

In addition, you may try with different matrix size - `512 x 512`, `1024 x 1024`, `2048 x 2048`, and `4096 x 4096` to verify the correctness. Your code will be also tested for correctness using different `mpirun np 4 to 32`.

4.3 Performance (25%)

Your performance will be evaluated based on the **16** processor performance on the multiplication of matrices of size `4096 x 4096` only. The performance point will be added **ONLY** if you have the correct result. Please prioritize the correctness over performance. Your performance will be evaluated based on the ranges of throughput (GFlop/s). We will set five ranges after evaluating all submissions and assign the points as follows:

More than 60 GFlop/s (1.5X of TA's): 25 points + 5 points (bonus)
Range A GFlop/s: 25 points
Range B GFlop/s: 20 points
Range C GFlop/s: 15 points
Range D GFlop/s: 10 points
Speed up lower than range D: 5 points
Slowdown (less than 0.8 GFlop/s): 0 points

4.4 Report (15%)

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