HW3 Specification

Multicore and GPU Programming

Prof. Lee Jinho
T.A. Yang Sejong

0 Introduction

In this project, you will implement matrix multiplication or joining tables with multi-threading. The framework can be downloaded from here or YSCEC.

The specification has the following contents.

!!!Caution: Plagiarism!!!

You need to be careful to comply with plagiarism regulations. There is an ambiguity between discussion, idea, and solution. Therefore, be careful not to take away the opportunity of your colleague.

- No code sharing
- **No idea sharing**
- Ok discussion about the project

1 Environment Setup

The same manner in HW1, HW2.

You don't need to set up the environment. We prepared everything you need to do an experiment about multi-threading. Check GNU GCC/G++ 7.5.0, GNU make, and HTCondor documentation for more information.

2 Framework

2-1 Matrix Multiplication

2-1-0 Structure of template code

```
/HW3/matmul$ tree .
.
├── build                  // make file generate objective files to here
│   └── driver.o           // only objective file provided!
├── data                   // Saving data here recommended
├── Makefile               // Compile, Clean, Format, Submit
├── matmul4096.cmd         // Command for condor
├── matmul2048.cmd         // Command for condor
├── result                 // Result from condor
└── src                    // Source Code
    ├── generate.cpp
    ├── matmul.cpp
    └── matmul.h

4 directories, 7 files
```

2-1-1 Generator

You can use my generator to make input, output files for your 'matmul' program. Or you can download matmul_data.zip with some pair of input and output.'

The 'generate' program get four arguments.

- n: matrix size
- inputPath: path for input file containing *n* and values for matirx A and matrix B
- outputPath: path for output file containing *n* and values of matrix C(multiplication of A and B)

```
/HW3/matmul$ make generate
g++ -std=c++11 -pthread -lpthread -fopenmp -Wl,--no-as-needed -c src/generate.cpp
```

```
-o build/generate.o
g++ -std=c++11 -pthread -lpthread -fopenmp -Wl,--no-as-needed -o generate \
      build/generate.o

/HW3/matmul$ ./generate 64 data/input.txt data/output.txt

======== Matrix Multiplication ========
n : 64
inputPath : data/input.txt
outputPath : data/output.txt
=======================================
```

2-1-2 Implement your optimized solution

```
/HW3/matmul$ cat src/matmul.cpp

#include "matmul.h"

void matmul_ref(const int** const matrixA, const int** const matrixB,
                      int** const matrixC, const int n) {
  // You can assume matrixC is initialized with zero
  for (int i = 0; i < n; i++)
    for (int j = 0; j < n; j++)
      for (int k = 0; k < n; k++)
        matrixC[i][j] += matrixA[i][k] * matrixB[k][j];
}

void matmul_optimized(const int** const matrixA, const int** const matrixB,
                            int** const matrixC, const int n) {
  // TODO: Implement your code
}
```

2-1-3 Compile and run matmul

The 'matmul' program get three arguments.

- inputPath: path to input file
- outputPath: path to output file
- sanityCheck: 0 is false and 1 is true, this will check the validation of input, output files, **don't use in condor with large matrix size**.

```
# Compile
/HW3/matmul$ make
g++ -std=c++11 -pthread -lpthread -fopenmp -Wl,--no-as-needed -c src/matmul.cpp -o
build/matmul.o
```

```
g++ -std=c++11 -pthread -lpthread -fopenmp -Wl,--no-as-needed -o matmul \
      build/driver.o \
      build/matmul.o

# Show help
/HW3/matmul$ ./matmul
./matmul inputPath outputPath sanityCheck(0 or 1)
Use sanityCheck=1 only in local

# Local run
/HW3/matmul$ ./matmul data/input.txt data/output.txt 1
====================================
       Matrix Multiplication
====================================
The size of Matrix: 4
Sanity Check(0: false, 1: true): 1
====================================

Read input file(data/input.txt)...
Read output file(data/output.txt)...

Run sanity check for input, output...

matmul_ref took 2.273e-06 sec
Problem(data/input.txt) and
Solution(data/output.txt) are
Matched!

Run your solution...

matmul_optimal took 0.00172086 sec
Correct
```

2-2 Joining tables

2-1-0 Structure of template code

Everything is same without naming.

```
/HW3/jointable$ tree .
.
├── build
│   └── driver.o
├── data
├── jointable_diff.cmd
├── jointable_same.cmd
├── Makefile
```

```
├── result
└── src
    ├── generate.cpp
    ├── jointable.cpp
    └── jointable.h

4 directories, 7 files
```

2-1-1 Generator

You can use my generator to make input, output files for your 'jointable' program. Or you can download jointable_data.zip with some pair of input and output.

The 'generate' program get four arguments.

- R: size for table A
- S: size for table B
- inputPath: path for input file containing R, S and values for tableA and table B
- outputPath: path for output file containing size and values of table C(Intersection of table A and table B, check lecture slide for more information)

```
/HW3/jointable$ make generate
g++ -std=c++11 -pthread -lpthread -fopenmp -Wl,--no-as-needed -c src/generate.cpp
-o build/generate.o
g++ -std=c++11 -pthread -lpthread -fopenmp -Wl,--no-as-needed -o generate \
      build/generate.o

/HW3/jointable$ ./generate 64 64 data/input_6464.txt data/output_6464.txt

============= Join Tables =============
R : 64
S : 64
inputPath : data/input_6464.txt
outputPath : data/output_6464.txt
=======================================
Generate flags for tableA...
Generate flags for tableB...
Fill tableA...
Fill tableB...
Fill tableC(intersection of tableA and tableB)
Write input file...
Write output file...
Free memories...
```

2-1-2 Implement your optimized solution

```
/HW3/jointable$ cat src/jointable.cpp
#include "jointable.h"

void jointable_ref(const int* const tableA, const int* const tableB,
                   std::vector<int>* const solution, const int R, const int S) {
  // You can assume matrixC is initialized with zero
  for (int i = 0; i < R; i++)
    for (int j = 0; j < S; j++)
      if (tableA[i] == tableB[j]) {
        solution->push_back(tableA[i]);
        break;
      }
}
void jointable_optimized(const int* const tableA, const int* const tableB,
                         std::vector<int>* const solution, const int R,
                         const int S) {
  // TODO: Implement your code
}
```

2-1-3 Compile and run jointable

The 'jointable' program get three arguments.

- inputPath: path to input file
- outputPath: path to output file
- sanityCheck: 0 is false and 1 is true, this will check the validation of input, output files, **don't use in condor with large table size**.

```
/HW3/jointable$ make
g++ -std=c++11 -pthread -lpthread -fopenmp -Wl,--no-as-needed -c src/jointable.cpp
-o build/jointable.o
g++ -std=c++11 -pthread -lpthread -fopenmp -Wl,--no-as-needed -o jointable \
      build/driver.o \
      build/jointable.o

/HW3/jointable$ ./jointable data/input_6464.txt data/output_6464.txt 0
====================================
          Join Tables
====================================
The size of tableA: 64
The size of tableB: 64
The size of tableC(Intersection of tableA and table B): 5
Sanity Check(0: false, 1: true): 0
====================================
```

```
Read input file(data/input_6464.txt)...
Read output file(data/output_6464.txt)...

Run your solution...

jointable_optimal took 0.000651653 sec
Correct
```

3 Submission

3-1 Matrix Multiplication

There are two submit function, **submit_2048** and **submit_4096**. Each number means the size of matrix.

Step 0. Download or generate proper input and output files.
Step 1. Submit a job to condor.
Step 2. Wait until finished.(Check **result/matmul.log** for Job terminated)
Step 3. Check the output(**result/matmul.out**). There will be **TIME** and **CORRECTNESS** of your solution.

```
/HW3/matmul$ cat matmul2048.cmd
##################################################
##
## Matrix Multiplication Condor command file
##
##################################################

executable      = matmul
output          = result/matmul.out
error           = result/matmul.err
log             = result/matmul.log
request_cpus = 16
should_transfer_files   = YES
when_to_transfer_output = ON_EXIT
transfer_input_files    = data/input_2048.txt, data/output_2048.txt
arguments               = input_2048.txt output_2048.txt 0
queue

/HW3/matmul$ make submit_2048
condor_submit matmul2048.cmd
Submitting job(s).
1 job(s) submitted to cluster 12738.
```

```
/HW3/matmul$ cat result/matmul.out
==================================
       Matrix Multiplication
==================================
The size of Matrix: 2048
Sanity Check(0: false, 1: true): 0
==================================

Read input file(input_2048.txt)...
Read output file(output_2048.txt)...

Run your solution...

matmul_optimal took TIME sec
CORRECTNESS
```

3-2 Joining table

There are two submit function, **submit_same** and **submit_diff**. In each function, R(size of table A) and S(size of table B) are same or different.

Step 0. Download or generate proper input and output files.
Step 1. Submit a job to condor.
Step 2. Wait until finished.(Check **result/jointable.log** for Job terminated)
Step 3. Check the output(**result/jointable.out**). There will be **TIME** and **CORRECTNESS** of your solution.

```
/HW3/jointable$ cat jointable_diff.cmd
#####################################################
##
## Join Tables Condor command file
##
#####################################################

executable      = jointable
output          = result/jointable.out
error           = result/jointable.err
log             = result/jointable.log
request_cpus = 16
should_transfer_files   = YES
when_to_transfer_output = ON_EXIT
transfer_input_files    = data/input_1m100m.txt, data/output_1m100m.txt
arguments               = input_1m100m.txt output_1m100m.txt 0
queue
```

```
/HW3/jointable$ make submit_diff
condor_submit jointable_diff.cmd
Submitting job(s).
1 job(s) submitted to cluster 12740.

/HW3/jointable$ cat result/jointable.out
====================================
          Join Tables
====================================
The size of tableA: 1000000
The size of tableB: 100000000
The size of tableC(Intersection of tableA and table B): 198048
Sanity Check(0: false, 1: true): 0
====================================

Read input file(input_1m100m.txt)...
Read output file(output_1m100m.txt)...

Run your solution...

jointable_optimal took TIME sec
CORRECTNESS
```

4 Criteria

4-1 Requirements

4-1-0 General

- It has to be parallel (no serial implementation)
- No open source, or parallel library - you can only use STL.
  You can use boost::barrier, but if you want to use anything else, contact us before doing so.
- You can use pthread, std::thread, openMP, or MPI as your parallelization framework. If you want to use something else (e.g., Apache Spark), contact us first.
- Do not override driver.o

4-1-1 Matrix Multiplication

- Problem size: 2048x2048, 4096x4096
- Performance requirements: **1.76**sec (2048x2048), **13.89**sec (4096x4096)
- Correctness: Should work correctly for any square matrix input between size of 64x64 ~ 65536x65536

- You can use techniques such as the Strassen or Winograd algorithm, provided that you parallelize them yourself.

4-1-2 Join

- Problem size: R:10m/S:10m, R:1m/S:20m
- Performance requirements: **4.50**sec (R:10m/S:10m), **3.84**sec (R:5m/S:20m)
- Correctness: Should work correctly for input between any combinations of size 10k~100m
- You must implement either at least partitioned hash join (lecture slide p38) or m-way sort-merge join (lecture slide p54) algorithm, and then add/remove any technique you want. If your final (fastest) implementation is not based on the partitioned hash join or m-way SM join, leave your partitioned hash join algorithm or m-way SM code as a separate file and provide a make target for it

5 Report

Your report should include

- What techniques you have implemented
- How to run your code
- How each technique affected your performance (+ comparison)
- Why you think your technique, or your combination of techniques produced the best result
- max 4 pages (firm)
- **PDF only**
- If 4 pages is to short to contain everything you want to say, use a double-column format (e.g., https://ieeecs-media.computer.org/assets/zip/Trans_final_submission.zip , https://ieeecs-media.computer.org/assets/zip/ieeetran-final_sub.zip)

6 Grading

Correct parallel implementation (20) - finish within 1 minute, produce correct result with parallel implementation
Meet the performance bound (25) - your runtime for both the test size is both below the bar
Report (20) - Refer to 5. Report
Ranking (35) - 35 * (91 - rank)/90. Higher of the two. The ranking is decided by comparing the product of two inputs. For example, if you implemented matrix multiplication, and got 0.1 sec for 2048x2048 and 0.9 sec for 4096x4096, your score is 0.09. We will announce the current top score at least once a week.
Extra point (15) - If you implement both, and achieve the performance bar, you get 15 pts. If you get a correct implementation slower than the performance bar, you get 6 pts. Total points cannot exceed 100 pts.

7 Server Info

tell us if you need anything else!

(base) leejinho@acsys01:~$ cat /proc/cpuinfo
processor : 0
vendor_id : AuthenticAMD
cpu family : 23
model : 113
model name : AMD Ryzen 7 3700X 8-Core Processor
stepping : 0
microcode : 0x8701012
cpu MHz : 2195.879
cache size : 512 KB
physical id : 0
siblings : 16
core id : 0
cpu cores : 8
apicid : 0
initial apicid : 0
fpu : yes
fpu_exception : yes
cpuid level : 16
wp : yes
flags : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush mmx fxsr sse sse2 ht syscall nx mmxext fxsr_opt pdpe1gb rdtscp lm constant_tsc rep_good nopl xtopology nonstop_tsc cpuid extd_apicid aperfmperf pni pclmulqdq monitor ssse3 fma cx16 sse4_1 sse4_2 movbe popcnt aes xsave avx f16c rdrand lahf_lm cmp_legacy svm extapic cr8_legacy abm sse4a misalignsse 3dnowprefetch osvw ibs skinit wdt tce topoext perfctr_core perfctr_nb bpext perfctr_llc mwaitx cpb cat_l3 cdp_l3 hw_pstate sme ssbd ibpb stibp vmmcall fsgsbase bmi1 avx2 smep bmi2 cqm rdt_a rdseed adx smap clflushopt clwb sha_ni xsaveopt xsavec xgetbv1 xsaves cqm_llc cqm_occup_llc cqm_mbm_total cqm_mbm_local clzero irperf xsaveerptr arat npt lbrv svm_lock nrip_save tsc_scale vmcb_clean flushbyasid decodeassists pausefilter pfthreshold avic v_vmsave_vmload vgif umip rdpid overflow_recov succor smca
bugs : sysret_ss_attrs spectre_v1 spectre_v2 spec_store_bypass
bogomips : 7186.40
TLB size : 3072 4K pages
clflush size : 64
cache_alignment : 64
address sizes : 43 bits physical, 48 bits virtual
power management: ts ttp tm hwpstate cpb eff_freq_ro [13] [14]

… (continues until processor 16)

```
leejinho@acsys01:~$ lscpu
Architecture:        x86_64
CPU op-mode(s):      32-bit, 64-bit
Byte Order:          Little Endian
CPU(s):              16
On-line CPU(s) list: 0-15
Thread(s) per core:  2
Core(s) per socket:  8
Socket(s):           1
NUMA node(s):        1
Vendor ID:           AuthenticAMD
CPU family:          23
Model:               113
Model name:          AMD Ryzen 7 3700X 8-Core Processor
Stepping:            0
CPU MHz:             2195.289
CPU max MHz:         3600.0000
CPU min MHz:         2200.0000
BogoMIPS:            7186.40
Virtualization:      AMD-V
L1d cache:           32K
L1i cache:           32K
L2 cache:            512K
L3 cache:            16384K
NUMA node0 CPU(s):   0-15
Flags:               fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat
pse36 clflush mmx fxsr sse sse2 ht syscall nx mmxext fxsr_opt pdpe1gb rdtscp lm
constant_tsc rep_good nopl xtopology nonstop_tsc cpuid extd_apicid aperfmperf pni
pclmulqdq monitor ssse3 fma cx16 sse4_1 sse4_2 movbe popcnt aes xsave avx f16c
rdrand lahf_lm cmp_legacy svm extapic cr8_legacy abm sse4a misalignsse
3dnowprefetch osvw ibs skinit wdt tce topoext perfctr_core perfctr_nb bpext
perfctr_llc mwaitx cpb cat_l3 cdp_l3 hw_pstate sme ssbd ibpb stibp vmmcall fsgsbase
bmi1 avx2 smep bmi2 cqm rdt_a rdseed adx smap clflushopt clwb sha_ni xsaveopt
xsavec xgetbv1 xsaves cqm_llc cqm_occup_llc cqm_mbm_total cqm_mbm_local clzero
irperf xsaveerptr arat npt lbrv svm_lock nrip_save tsc_scale vmcb_clean flushbyasid
decodeassists pausefilter pfthreshold avic v_vmsave_vmload vgif umip rdpid
overflow_recov succor smca
```

```
# dmidecode 3.1
Getting SMBIOS data from sysfs.
SMBIOS 3.2.0 present.
# SMBIOS implementations newer than version 3.1.1 are not
```

fully supported by this version of dmidecode.

Handle 0x0011, DMI type 17, 84 bytes
Memory Device
	Array Handle: 0x0009
	Error Information Handle: 0x0010
	Total Width: Unknown
	Data Width: Unknown
	Size: No Module Installed
	Form Factor: Unknown
	Set: None
	Locator: DIMM 0
	Bank Locator: P0 CHANNEL A
	Type: Unknown
	Type Detail: Unknown
	Speed: Unknown
	Manufacturer: Unknown
	Serial Number: Unknown
	Asset Tag: Not Specified
	Part Number: Unknown
	Rank: Unknown
	Configured Clock Speed: Unknown
	Minimum Voltage: Unknown
	Maximum Voltage: Unknown
	Configured Voltage: Unknown

Handle 0x0013, DMI type 17, 84 bytes
Memory Device
	Array Handle: 0x0009
	Error Information Handle: 0x0012
	Total Width: 64 bits
	Data Width: 64 bits
	Size: 32 GB
	Form Factor: DIMM
	Set: None
	Locator: DIMM 1
	Bank Locator: P0 CHANNEL A
	Type: DDR4
	Type Detail: Synchronous Unbuffered (Unregistered)
	Speed: 2666 MT/s
	Manufacturer: Samsung
	Serial Number: 03AC8A9D
	Asset Tag: Not Specified
	Part Number: M378A4G43MB1-CTD
	Rank: 2
	Configured Clock Speed: 2666 MT/s

Minimum Voltage: 1.2 V
Maximum Voltage: 1.2 V
Configured Voltage: 1.2 V

Handle 0x0016, DMI type 17, 84 bytes
Memory Device
Array Handle: 0x0009
Error Information Handle: 0x0015
Total Width: Unknown
Data Width: Unknown
Size: No Module Installed
Form Factor: Unknown
Set: None
Locator: DIMM 0
Bank Locator: P0 CHANNEL B
Type: Unknown
Type Detail: Unknown
Speed: Unknown
Manufacturer: Unknown
Serial Number: Unknown
Asset Tag: Not Specified
Part Number: Unknown
Rank: Unknown
Configured Clock Speed: Unknown
Minimum Voltage: Unknown
Maximum Voltage: Unknown
Configured Voltage: Unknown

Handle 0x0018, DMI type 17, 84 bytes
Memory Device
Array Handle: 0x0009
Error Information Handle: 0x0017
Total Width: 64 bits
Data Width: 64 bits
Size: 32 GB
Form Factor: DIMM
Set: None
Locator: DIMM 1
Bank Locator: P0 CHANNEL B
Type: DDR4
Type Detail: Synchronous Unbuffered (Unregistered)
Speed: 2666 MT/s
Manufacturer: Samsung
Serial Number: 03AC8ACA
Asset Tag: Not Specified
Part Number: M378A4G43MB1-CTD

Rank: 2
Configured Clock Speed: 2666 MT/s
Minimum Voltage: 1.2 V
Maximum Voltage: 1.2 V
Configured Voltage: 1.2 V

```
leejinho@acsys01:~$ uname -a
Linux acsys01 4.15.0-72-generic #81-Ubuntu SMP Tue Nov 26 12:20:02 UTC 2019
x86_64 x86_64 x86_64 GNU/Linux
```

```
leejinho@acsys01:~$ free --giga
              total        used        free      shared  buff/cache   available
Mem:             65           0          42           0          22          64
Swap:             2           0           2
```

8 Reference

- GNU GCC/G++ 7.5.0
- GNU make
- HTCondor
- Our Lecture Slides
- S. Blanas, Y. Li, and J. M. Patel, "Design and evaluation of main memory hash join algorithms for multi-core CPUs", SIGMOD 2011
- A. Shatdal, C. Kant, and J. F. Naughton, "Cache conscious algorithms for relational query processing," in VLDB, 1994
- S. Manegold et al., Optimizing main-memory join on modern hardware, IEEE TKDE 2002
- Cagri Balkesen et al., MultiCore, MainMemory Joins: Sort vs. Hash Revisited, VLDB 2013
- M.-C. Albutiu et al., Massively Parallel SortMerge Joins in Main Memory MultiCore Database Systems
- Ask Professor, TA(Of course not about the code)
- HW3 Specification
- HW3 Directory
    - HW3_v1.zip
    - matmul_data.zip
    - jointable_data.zip