INFO-H-417 : Report

Baudoux Nicolas   Devers Marine   Meire Wouter

11 décembre 2018

Abstract

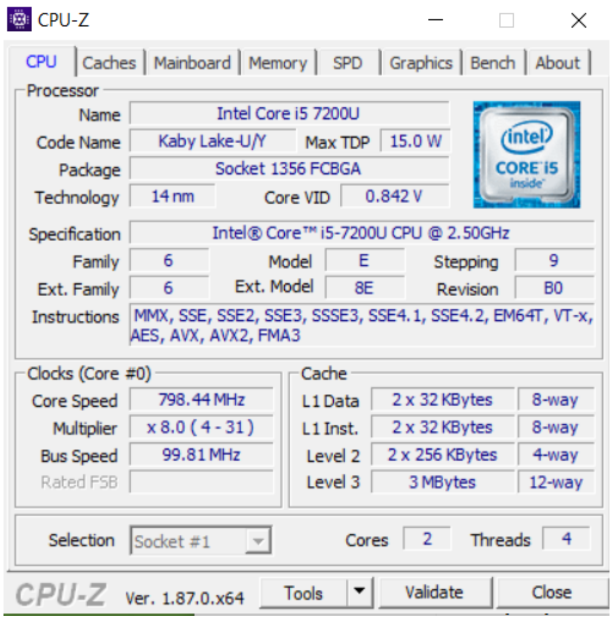
Contents

1 Introduction

For this assignment, we were asked to implement an external-memory merge-sort algorithm and to examine its efficiency under different parameters, such as the input file size, the size of the main memory available, the number of streams, but also many others. Before the implementation itself, we have to explore different ways to read data from, and write data to secondary memory. Then, the multi-way merge and the external multi-way merge-sort are constructed and tested. The main goal of this project is to get real-world experience with the performance of external-memory algorithms. The code of this project is fully written in **C++**.

1.1 Environment

We ran all our tests on one machine, here is the system environment of this machine:


CPU-Z
CPU | Caches | Mainboard | Memory | SPD | Graphics | Bench | About
CPU-Z Ver. 1.87.0.x64
Tools
Validate
Close

Processor

Name	Intel Core i5 7200U		
Code Name	Kaby Lake-U/Y	Max TDP	15.0 W
Package	Socket 1356 FCBGA		
Technology	14 nm	Core VID	0.842 V

Specification: Intel® Core™ i5-7200U CPU @ 2.50GHz

Family	6	Model	E	Stepping	9
Ext. Family	6	Ext. Model	8E	Revision	B0

Instructions: MMX, SSE, SSE2, SSE3, SSSE3, SSE4.1, SSE4.2, EM64T, VT-x, AES, AVX, AVX2, FMA3

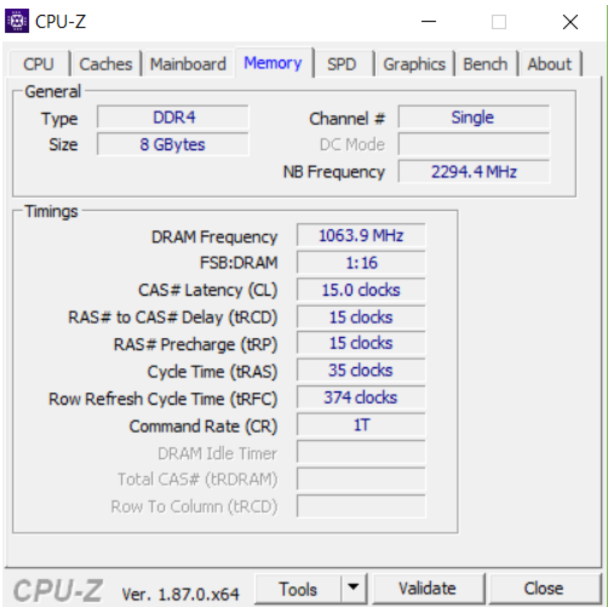
Clocks (Core #0)

Core Speed	798.44 MHz
Multiplier	x 8.0 (4 - 31)
Bus Speed	99.81 MHz
Rated FSB	

Cache

L1 Data	2 x 32 KBytes	8-way
L1 Inst.	2 x 32 KBytes	8-way
Level 2	2 x 256 KBytes	4-way
Level 3	3 MBytes	12-way

Selection: Socket #1 Cores: 2 Threads: 4


CPU-Z
CPU | Caches | Mainboard | Memory | SPD | Graphics | Bench | About
CPU-Z Ver. 1.87.0.x64
Tools
Validate
Close

General

Type	DDR4	Channel #	Single
Size	8 GBytes	DC Mode	
		NB Frequency	2294.4 MHz

Timings

DRAM Frequency	1063.9 MHz
FSB:DRAM	1:16
CAS# Latency (CL)	15.0 clocks
RAS# to CAS# Delay (tRCD)	15 clocks
RAS# Precharge (tRP)	15 clocks
Cycle Time (tRAS)	35 clocks
Row Refresh Cycle Time (tRFC)	374 clocks
Command Rate (CR)	1T
DRAM Idle Timer	
Total CAS# (tRDRAM)	
Row To Column (tRCD)	

Figure 1 – CPU

Figure 2 – Memory

1.2 Libraries used

The following libraries are used in our project [2]:

- **Boost** [1]: this library is used to analyze every execution time of different implementations.
- <**fcntl.h**>: this library is required to use file control options.
- <**stdio**>: this library is required to use the *fread* and *fwrite* functions.
- <**fstream**>: this library is used to operate on files. It is an input/output stream class.
- <**random**>: this library allows us to produce random numbers using combinations of generators and distributions.
- <**bitset**>: a bitset stores bits. Bitsets have the feature of being able to be constructed from and converted to both integer values and binary strings.
- <**windows.h**>: this library is required to use the function *CreateFileMapping*.

2 Observations on streams

2.1 Reading and writing: streams

At first, we were asked to implement different mechanisms for reading data from, and writing data to disk. We were asked to develop *streams* classes in order to read and write disk files consisting of 32-bits integers. Therefore, two types of streams are needed: the *input* streams and the *output* streams. The operations **open** (open an existing file for reading), **read_next** (read the next element from the stream), and **end_of_stream** (a boolean operation that returns true if the end of stream has been reached) should be supported by the input stream. Also, the operations **create** (create a new file), **write** (write an element to the stream), and **close** (close the stream) should be supported by the output stream. Four distinct I/O mechanisms are used to retrieve four different implementations of the input and output streams.

2.1.1 Read and write system calls

For the first I/O mechanism, only one element is read or written at a time. We use the *read* and *write* system calls that are available through the <*io.h*> header. We also use a variable called *fileHandle* that is equal to −1 if we have not been working on a designated file.

2.1.2 Read and write with its buffering mechanism

For the second I/O mechanism, the *read* and *write* are performed with their own buffering mechanism. We use the *fread* and *fwrite* functions from the <*stdio*> library. It reads integers of 32 bits and puts them into its own buffer and its writes them from the same buffer to a file using our *filePointer* that points toward the open file.

2.1.3 Read and write with a buffer of size B in internal memory

For the next I/O mechanism, the read and write are performed as in the first one explained above. However, each stream has its own buffer of size **B** in internal memory now. Instead of just reading/writting one element at a time, we can read/write **B** elements at a time. When the buffer empties itself, the following **B** elements can be read from the file. When the buffer becomes full, the following **B** elements can be written to the open file.

2.1.4 Memory mapping

The last I/O mechanism is implemented by performing *read* and *write* with the mapping and the unmapping of a **B** element portion of the file into internal memory through memory mapping.

2.2 Expected behaviour

2.3 Experimental observations

	Write 1	Read 1	Write 2	Read 2	Write 3	Read 3	Write 4	Read 4
Average (μs)								

Table 2.1 – Read and write of 10 values with a buffer size = 512

	Write 1	Read 1	Write 2	Read 2	Write 3	Read 3	Write 4	Read 4
Average (μs)								

Table 2.2 – Read and write of 10 values with a buffer size = 512

	Write 1	Read 1	Write 2	Read 2	Write 3	Read 3	Write 4	Read 4
Average (μs)								

Table 2.3 – Read and write of 10 values with a buffer size = 512

	Write 1	Read 1	Write 2	Read 2	Write 3	Read 3	Write 4	Read 4
Average (μs)								

Table 2.4 – Read and write of 10 values with a buffer size = 512

	Write 1	Read 1	Write 2	Read 2	Write 3	Read 3	Write 4	Read 4
Average (μs)								

Table 2.5 – Read and write of 10 values with a buffer size = 512

	Write 1	Read 1	Write 2	Read 2	Write 3	Read 3	Write 4	Read 4
Average (μs)								

Table 2.6 – Read and write of 10 values with a buffer size = 512

	Write 1	Read 1	Write 2	Read 2	Write 3	Read 3	Write 4	Read 4
Average (μs)								

Table 2.7 – Read and write of 10 values with a buffer size = 512

	Write 1	Read 1	Write 2	Read 2	Write 3	Read 3	Write 4	Read 4
Average (μs)								

Table 2.8 – Read and write of 10 values with a buffer size = 512

	Write 1	Read 1	Write 2	Read 2	Write 3	Read 3	Write 4	Read 4
Average (μs)								

Table 2.9 – Read and write of 10 values with a buffer size = 512

	Write 1	Read 1	Write 2	Read 2	Write 3	Read 3	Write 4	Read 4
Average (μs)								

Table 2.10 – Read and write of 10 values with a buffer size = 512

	Write 1	Read 1	Write 2	Read 2	Write 3	Read 3	Write 4	Read 4
Average (μs)								

Table 2.11 – Read and write of 10 values with a buffer size = 512

	Write 1	Read 1	Write 2	Read 2	Write 3	Read 3	Write 4	Read 4
Average (μs)								

Table 2.12 – Read and write of 10 values with a buffer size = 512

	Write 1	Read 1	Write 2	Read 2	Write 3	Read 3	Write 4	Read 4
Average (μs)								

Table 2.13 – Read and write of 10 values with a buffer size = 512

	Write 1	Read 1	Write 2	Read 2	Write 3	Read 3	Write 4	Read 4
Average (μs)								

Table 2.14 – Read and write of 10 values with a buffer size = 512

2.4 Discussion of expected behaviour vs experimental observations

3 Observations on multi-way merge sort

3.1 Expected behaviour

3.2 Experimental observations

3.3 Discussion of expected behaviour vs experimental observations

4 Conclusion

References

[1] Rene Rivera, Boost Library Documentation [Internet]. 2005 [Consulted on the 27th November 2018] Available on: `https://www.boost.org/doc/libs/`

[2] Cpp Reference, A list of open source C++ libraries [Internet]. 2018 [Consulted on the 11th December 2018] Available on: `https://en.cppreference.com/w/cpp/links/libs`

[3] Author's Name, Title of the source [where was the source found]. 2018 [Consulted on the 15th November 2018] Available on: `http://google.com`