Final Project Report of C/C++ Programming MA 251

He Liu

January 6, 2016

Instructor: Professor Li

1 Introduction

In this project, we are supposed to write a program which can be used to sort massive amounts of data. Since the size of data may be too large to load into RAM, it is not feasible to sort directly. Generally, there are two methods to do this.

One is creating a memory-mapped file. A memory-mapped file contains the contents of a file in virtual memory. This mapping between a file and memory space enables an application, including multiple processes, to modify the file by reading and writing directly to the memory. With the memory-mapped files that are associated with a source file on a disk, when the last process has finished working with the file, the data are saved to the source file on the disk.

Another is using external sorting, which is an application of divide and conquer strategy. One example of external sorting is the external merge sort algorithm, which sorts chunks that each fit in RAM, then merges the sorted chunks together. The merge algorithm only makes one pass sequentially through each of the chunks, each chunk does not have to be loaded completely.

In this project, I tried to use the first method in the beginning, but failed. Most of the sources programmed in C are based on Linux/UNIX, and I do not have enough time nor enough ability to migrate them into Win32. Then I turned to try the second method and got right result of sorting. This report is based on the second method, external sorting.

2 Program Logic

I made a flow diagram (Figure 1) to show the procedure of my program.

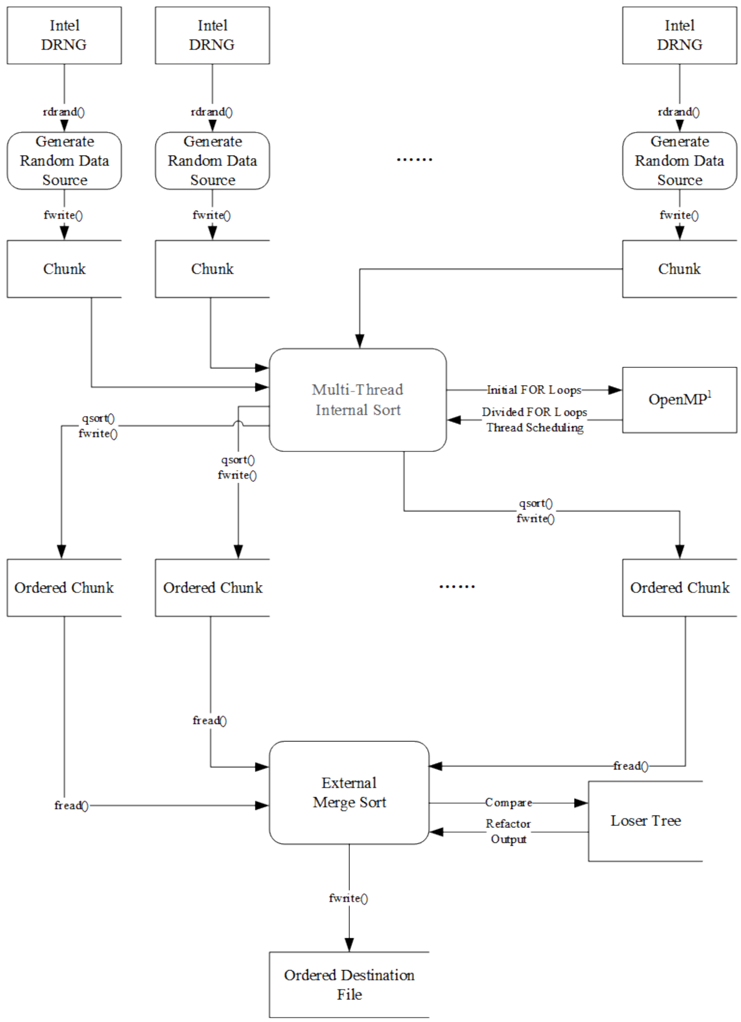

Figure 1: The procedure of my program.

[1]OpenMP (Open Multi-Processing) is an application programming interface (API) that supports multi-platform shared memory multiprocessing.

3 Time Required

The integers involved in sorting are 32-bit integer [2], which have a range of $-2\,147\,483\,648$ to $2\,147\,483\,647$.
The shortest time used in each case is shown in Table 1. [3]

Table 1: Shortest time required

NUM. OF INT32	NUM. OF CHUNKS	TIME REQUIRED (s)
1,024,000	8	0.225
5,120,000	8	0.673
10,240,000	8	1.216
51,200,000	16	5.746
102,400,000	32	11.623
204,800,000	64	23.162

All of the sorting results are checked by viewing them in Notepad++ with HEX-Editor plug-in.

4 Analysis

In this project, running time can be affected by internal sort (Quick Sort), file I/O and external sort. The function qsort() included in C STL has been heavily optimized and we do not need to do anything on this. However, there are many files to be sorted, if they can be sorted at the same time, lots of time can be saved. File I/O is a waste of time but cannot be removed, so the times to read or write files should be reduced as much as possible. The algorithm applied in external sort is important. In this project, we use Loser Tree to improve merge efficiency.

4.1 Parallel Quick Sort

There are 8 threads in the CPU of my computer. The default is each program only establish one process, but we can create multiple processes through programming. My basic idea on this is distributing the files waiting for being sorted in to 8 parts in average. Then let them do Quick Sort at the same time. Since thread scheduling takes some time, the speed up cannot be 8 times, but there is still much. Meanwhile, the time used on File I/O can be reduced.

In parallel coding, to reduce process latency, get the correct results and avoid

[2] The random integers are generated by rdrand(), which is included in Intel Digital Random Number Generator (DRNG), can generate "real" random integers based on CPUID . The initial random integers are 32-bit unsigned integers, which have a range of 0 to $4\,294\,967\,295$. I used (u32 >> 16) & 0x7FFF to convert them to 32-bit signed integers.

[3] The development environment is shown in Appendix A, and all of the raw data are attached in Appendix B.

variables competition, the design of parallel structure and the settings of variables are important.

In addition, this parallel computing follows forkjoin model, as show in Figure 2.

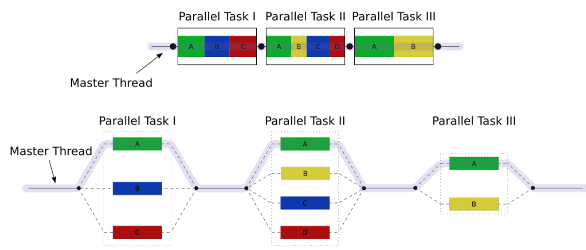

Figure 2: Folk-join Model

4.2 Loser Tree

In K-way merge, we can use Heap Sort or Winner/Loser Tree. When the number of ways is not too large, we often use Winner/Loser Tree to reduce the time used in adjust. Loser Tree and Winner Tree can find the extremum in $O(\log K)$, and the algorithm complexity of Loser Tree is $O((n-1) \cdot \log K)$.

Loser Tree is an improvement from Winner Tree. Updating a Loser Tree is more convenient. The steps of Loser Tree algorithm is:
When a number is output, the number following enters as a leaf, and moves up by levels toward the root. As you move up, you carry the winner along. At each level, you find the loser of the last contest at that level. The node you are carrying challenges this previous loser. If it wins, you take it on to the next level. If it loses, you leave it and take the winner (the number which was there before you came) along to the next level. The winner of the contest at the root is placed the "extra" overall winner slot.

In addition, we can improve it by doing things in parallel: we can do the reading or writing at the same time as the sorting.

4.3 The link between the number of chunks and speed

As we can see in Appendix B, when the size of chunks increase, the speed may not be increased. And the reason is File I/O will consume more time. When the amount of data increase, more chunks will be needed because of the heavy pressure on Quick Sort. If the size of source file is too large to load into RAM, it must be splitted into several chunks. I think there exists a appropriate number range of chunks to be the best one, but I do not have enough time to generate data and make curve fitting.

Since my computer has a 8 threads CPU, the minimal appropriate number should be no less than 8 in this "big data sorting", and the number of chunks is supposed to be multiples of 8.

4.4 Others

- Use Binary File I/O.
- Do not use stack as a buffer for its size limit.
- Use QueryPerformanceCounter() in Win32API to print out the time used in a section.

5 Conclusion

In this project, I am more familiar with C programming, and learned a lot about sort algorithm. In addition, this project stimulated my interest. I tried to use parallel computing, tried to generate and sort 32-bit "real" random integers on Windows platform, tried to use Intel(R) Advisor and Intel(R) VTune(TM) Amplifier to help me optimize my program. I will try to use memory-mapped file on Linux platform later. I think the most valuable treasure I gained in this project is the interest on C programming.

References

[1] `http://www.codingunit.com/writing-memory-to-a-file-and-reading-memory-from-a-file-in-c`
[2]`http://chenkegarfield.blog.163.com/blog/static/62330008200910249526638/`
[3]`https://software.intel.com/en-us/articles/the-drng-library-and-manual`

Appendices

A Development Environment

Table 2: Hardware

Component	Model
CPU	Intel(R) Core(TM) i7 4700HQ
SSD	TOSHIBA(R) Q200 EX.
RAM	HyperX(R) DDR3 1600MHz CL9 12GB

Table 3: Software

Software	Name
OS	Windows 10 TH2 x64
IDE	Microsoft(R) Visual Studio 2015 Community
IDE Compile Mode	Release X86
Compiler	Intel C++ Compiler 16.0
Parallel Coding API	OpenMP 4.0
Assisted optimization tool	Intel(R) Advisor XE 2016
Assisted analysis tool	Intel(R) VTuneTM Amplifier 2016
Random integer generator API	Intel(R) Digital Random Number Generator

B Raw Time Using Data

Table 4: Number of intergers: 1,024,000 Size of source file(MB): 4

Num. of Chunks	Time of internal sort (ms)	Time of merge sort (ms)	Total time(s)	Avg. time(s)
	97	147	0.244	
4	71	152	0.223	0.231
	74	151	0.225	
	86	154	0.240	
8	87	137	0.224	**0.225**
	61	151	0.212	
	115	138	0.253	
16	100	145	0.245	0.250
	109	143	0.252	
	163	140	0.303	
32	172	157	0.329	0.324
	186	154	0.340	
	277	156	0.433	
64	252	158	0.410	0.447
	343	155	0.498	

Table 5: Number of intergers: 5,120,000 Size of source file(MB): 20

Num. of Chunks	Time of internal sort (ms)	Time of merge sort (ms)	Total time(s)	Avg. time(s)
	238	448	0.686	
4	248	463	0.711	0.703
	244	468	0.712	
	176	488	0.664	
8	177	492	0.669	**0.673**
	177	508	0.685	
	220	501	0.721	
16	232	483	0.715	0.731
	242	515	0.757	
	280	505	0.785	
32	309	507	0.816	0.799
	281	514	0.795	
	367	524	0.891	
64	409	506	0.915	0.909
	398	524	0.922	

Table 6: Number of intergers: 10,240,000 Size of source file(MB): 40

Num. of Chunks	Time of internal sort (ms)	Time of merge sort (ms)	Total time(s)	Avg. time(s)
	316	935	1.251	
8	304	900	1.204	**1.216**
	299	893	1.192	
	381	921	1.302	
16	329	920	1.249	1.275
	325	948	1.273	
	427	920	1.347	
32	397	930	1.327	1.336
	383	952	1.335	
	477	945	1.422	
64	467	961	1.428	1.426
	495	934	1.429	

Table 7: Number of intergers: 51,200,000 Size of source file(MB): 200

Num. of Chunks	Time of internal sort (ms)	Time of merge sort (ms)	Total time(s)	Avg. time(s)
	1681	4262	5.943	
8	1595	4295	5.890	5.940
	1697	4289	5.986	
	1381	4310	5.691	
16	1383	4472	5.855	**5.746**
	1374	4317	5.691	
	1445	4421	5.866	
32	1428	4454	5.882	5.870
	1425	4436	5.861	
	1559	4470	6.029	
64	1545	4471	6.016	6.022
	1542	4479	6.021	

Table 8: Number of intergers: 102,400,000 Size of source file(MB): 400

Num. of Chunks	Time of internal sort (ms)	Time of merge sort (ms)	Total time(s)	Avg. time(s)
8	6236	8326	14.562	13.641
	5050	8478	13.528	
	4423	8411	12.834	
16	3078	8632	11.710	12.039
	3489	8762	12.251	
	3264	8892	12.156	
32	2805	8827	11.632	**11.623**
	2798	8946	11.744	
	2705	8789	11.494	
64	2820	8943	11.763	11.636
	2791	8790	11.581	
	2798	8767	11.565	
128	3178	9214	12.392	12.380
	3109	9156	12.265	
	3158	9325	12.483	

Table 9: Number of intergers: 204,800,000 Size of source file(MB): 800

Num. of Chunks	Time of internal sort (ms)	Time of merge sort (ms)	Total time(s)	Avg. time(s)
8	8295	16680	24.975	24.985
	8237	16746	24.983	
	8220	16777	24.997	
16	7521	17241	24.762	25.124
	8596	17301	25.897	
	7617	17095	24.712	
32	7990	17687	25.677	25.912
	8797	17489	26.286	
	8357	17415	25.772	
64	5641	17573	23.214	**23.162**
	5555	17606	23.161	
	5577	17533	23.110	
128	5693	17958	23.651	23.809
	5704	18077	23.781	
	5662	18332	23.994	
256	6238	19218	25.456	26.122
	6172	18823	24.995	
	6237	21677	27.914	