COMSW4112 Project Report 2

Fan Yang fy2207, Guo YiXuan yc

April 29, 2016

1 Basic Q & A

1.1 Questions

1. Which machines were used?
2. What are the machine's characteristics (clock frequency, amount of RAM, etc.)?
3. Were the machines idle apart from the experiment?
4. What compiler did you use to compile the code? What compiler options were provided?
5. How did you instrument the code to measure the time taken just for phase 2?

1.2 Solution

1. I used my own laptop, the MacBook Pro (Retina, 13-inch, Early 2015) to implement the compile and executive processes.

2. The operating system of my laptop is the OS X Yosemite (Version 10.10.5). The processor of it is the 2.7 GHz Intel Core i5 (two cores). The Memory of it is the 8 GB 1867 MHz DDR3, with L2 Caches of 256 KB per cache and a L3 cache of 3 MB.

3. My machine is idle during the whole process of experiment, except for some indispensable processes like the main process of operating system.

4. I use the gcc (MacPorts gcc5 5.3.0_0) 5.3.0 to compile code. The whole compile line is:
gcc -msse4.2 -O2 -o project2 project2.c
Because none of the SIMD executions in our program associated with the <ammintrin.h> library, so I omit the -msse4a flag, and only adopt the -msse4.2 flag to enable the SIMD related lines. The -O2 flag is for optimization, I have also tried the -O3 flag to it, but it make no further improvement and my makefile could not recognize it strangely (But my terminal could). To make the line more compact, I omit it also.

5. My code could be divided mainly into three phases, as illustrated the project description, the first part is the tree construction, the second part is the probing part, the third part is the output part. Here, I utilized the clock function of <time.h> to get the clock numbers between the executing two lines. Because the utilizing clock numbers are rather large, so the randomness lead by this approximated evaluation could be omitted. The corresponding real time could be easily derived through the simple divide operation.
To accurately test the probing time, I set a t1=clock() function just before the probing circle begin, and set a t2=clock() function just after all of the probing results have been written to the output array. Thus we could be guaranteed to only measure the time taken for the phase2. Also in my program, I test the time taken for phase1, to show that it might be negligible compared with the one of phase2.

2 Test Results

Here, for each K P L Case, I test the running time of phase 2 for five times and get there average as the result, to mostly eliminate the effect of randomness. For each kind of test tree, I test 3 K values and 5 P values. For the 9-5-9 Tree Case, I test both the time consuming of the binary search version, the hardcoded SIMD version and the non-hardcoded SIMD version. For the other two, I only test the binary search version and the non-hardcoded SIMD version. As for the forms, the first row represents the value of P, and the first column represents the value of K. Here I prefer the clock numbers instead of the realistic time measures, because the latter one might depends on the hardware of the machine. But the former one would be rather constant, and the relative size of three version might not changed, no matter which number we adopt. And here I provide them both, it seems that in C the CLOCKS_PER_SEC is fixed.

2.1 9-5-9 Tree Case

K&P	1000000	2000000	3000000	5000000	10000000
100	11163.8, 11.16	23129, 23.13	36441.6, 36.44	60662.6, 60.66	122334.4, 122.33
200	12783.2, 12.78	27980, 27.98	41984.6, 41.99	67245.2, 67.24	138894.2, 138.90
300	13154.2, 13.15	27863.8, 27.86	41567.8, 41.57	69455.2, 69.45	140427.0, 140.43

Table 1: table of 9-5-9 Structure (binary search), unit: clocks, ms.

K&P	1000000	2000000	3000000	5000000	10000000
100	8319.6, 8.32	17190.2, 17.19	26550.0, 26.55	43987.6, 43.99	88508.6, 88.51
200	8608.4, 8.61	17581.0, 17.58	26416.4, 26.42	44195.8, 44.20	90659, 90.66
300	8105.8, 8.11	17620.4, 17.62	27286.4, 27.29	44198.6, 44.20	90999, 90.99

Table 2: table of 9-5-9 Structure (non-hardcoded), unit: clocks, ms.

K&P	1000000	2000000	3000000	5000000	10000000
100	4740.6, 4.74	10123.4, 10.12	15241.8, 15.24	25690.4, 25.69	50645.2, 50.64
200	4952.4, 49.52	10288.2, 10.29	15624.6, 15.62	25557.2, 25.58	51081.8, 51.08
300	4776.0, 47.76	10143.4, 10.14	15284.2, 15.28	25763.2, 25.76	51825.2, 51.83

Table 3: table of 9-5-9 Structure (hardcoded), unit: clocks, ms.

2.2 17-17 Tree Case

The result forms are in the next page.

2.3 9-5-5-9 Tree Case

The result forms are in the last page.

3 Conclusion

Because the datasets experiment results are alike, so I put all the paragraphs here instead of put them separately in the three parts. From the three test above, the result is such vivid. We could directly observe that for all the three structures the time consuming of binary search version would always larger than the non-hardcoded SIMD version. And for the specific 9-5-9 tree, the hardcoded SIMD version is better than the non-hardcoded SIMD version. It might comply with our common sense that the more specific designed version, and the version with lower layer like assembly language would always have a better performance.

To be more precisely, just take the case of 9-5-9 tree as an example, the non-hardcoded version just consume about 65 to 75% time of the binary search version. And the hardcoded version could further decrease it to approximately 35 to 40%, which could directly give us the feeling of performance improvement.

Also, we could find that the probing time of SIMP register might be emphasized less by the number of tree keys to construct, since they need to go through same number of layers (comparisons). And the time spent in probing might seems to be linear to the probing numbers in both three cases. And something interesting is that, as for the alike K number, the detailed structure of tree might emphasize the SIMD non-hardcoded version more than the binary search.

However, to be noticed, even the non-hardcoded SIMD version, it might only support the fan-out of 5, 9 and 17. But for the binary search version, the fan-out could be anything, which provides a more robust utilization in the real life. If you are not executing a tremendous amount of probing and the difference between these two methods is not too large, then the binary search version could still have its own advantage.

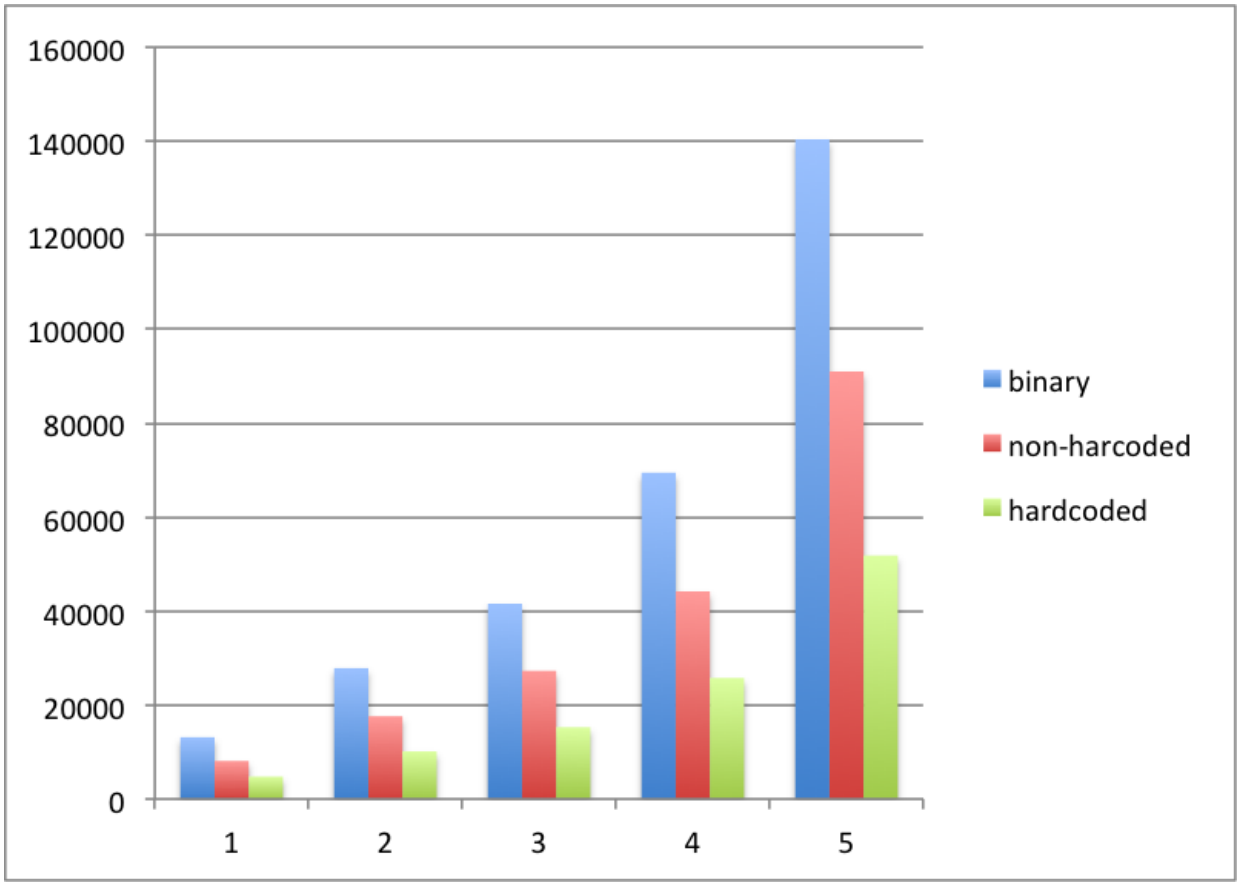
Also, we provide three charts of the tables, charts use the last row data as default, so the corresponding K is 300, 250 and 2000 for the three charts, and the P are 1000000 to 10000000 (corresponding to 1 to 5). The units are clock numbers.

K&P	1000000	2000000	3000000	5000000	10000000
100	11665.7, 11.67	24248.0, 24.25	37148.8, 37.15	61078.4, 61.08	126573.0, 126.58
200	12645.2, 12.65	25763.0, 25.76	40566.4, 40.57	65744.8, 65.74	135319.8, 135.32
250	12696.2, 12.70	26685.0, 26.69	40125.6, 40.13	66077.4, 66.08	132208.6, 132.22

Table 4: table of 17-17 Structure (binary search), unit: clocks, ms.

K&P	1000000	2000000	3000000	5000000	10000000
100	7431.0, 7.43	15892.6, 15.89	23468.0, 23.47	39025.8, 39.02	78923.6, 78.92
200	7306.0, 7.31	15443.6, 15.44	22979.0, 22.98	40431.2, 40.43	80789.2, 80.79
250	7580.8, 7.58	15483.2, 15.48	24064.2, 24.06	39864.8, 39.86	82531.4, 82.53

Table 5: table of 17-17 Structure (non-hardcoded), unit: clocks, ms.



K&P	1000000	2000000	3000000	5000000	10000000
500	14953.4, 14.95	32159.0, 32.16	48432.6, 48.43	81224.6, 81.22	164153.6, 164.15
1000	16988.6, 16.99	36162.2, 36.16	53149.2, 53.15	89969.6, 89.97	177323.6, 177.32
2000	16550.4, 16.55	35140.8, 35.14	54137.8, 54.14	86749.2, 96.75	181872.2, 181.87

Table 6: table of 9-5-5-9 Structure (binary search), unit: clocks, ms.

K&P	1000000	2000000	3000000	5000000	10000000
500	11268.2, 11.26	24498.4, 24.49	36363.0, 36.36	61270.8, 61.27	122880.8, 122.88
1000	11847.6, 11.84	24684.0, 24.68	35973.4, 35.97	60646.4, 60.64	125668.2, 125.67
2000	11421.2, 11.42	23718.0, 23.72	35523.0, 35.52	62742.8, 62.74	126059.2, 126.06

Table 7: table of 9-5-5-9 Structure (non-hardcoded), unit: clocks, ms.