

## COL 380 - Assignment 0

**Name:** Bommakanti Venkata Naga Sai Aditya

**Entry No.:** 2019CS50471

### Analysis of Original Code:

**gprof:**


Flat Profile:

Flat profile:

Each sample counts as 0.01 seconds.

time	% cumulative seconds	self seconds	calls	self Ts/call	total Ts/call	name
100.27	2.73	2.73				readRanges(char const*)
0.00	2.73	0.00	3	0.00	0.00	classify(Data&, Ranges const&, unsigned int)
0.00	2.73	0.00	3	0.00	0.00	timedwork(Data&, Ranges const&, unsigned int)
0.00	2.73	0.00	1	0.00	0.00	_GLOBAL__sub_I_Z8classifyR4DataRK6Rangesj
0.00	2.73	0.00	1	0.00	0.00	_GLOBAL__sub_I_Z9timedworkR4DataRK6Rangesj

Call Graph:

 Call graph (explanation follows)

granularity: each sample hit covers 2 byte(s) for 0.37% of 2.73 seconds

index	% time	self	children	called	name
[1]	100.0	2.73	0.00		<spontaneous> readRanges(char const*) [1]
[9]	0.0	0.00	0.00	3/3	timedwork(Data&, Ranges const&, unsigned int) [10] classify(Data&, Ranges const&, unsigned int) [9]
[10]	0.0	0.00	0.00	3/3	repeatrun(unsigned int, Data&, Ranges const&, unsigned int) [14] timedwork(Data&, Ranges const&, unsigned int) [10] classify(Data&, Ranges const&, unsigned int) [9]
[11]	0.0	0.00	0.00	1/1	__libc_csu_init [18] _GLOBAL__sub_I_Z8classifyR4DataRK6Rangesj [11]
[12]	0.0	0.00	0.00	1/1	__libc_csu_init [18] _GLOBAL__sub_I_Z9timedworkR4DataRK6Rangesj [12]

## Valgrind:

### Cachegrind tool report:

```
==11685==
==11685== I   refs:      34,186,177,502
==11685== I1  misses:      3,575
==11685== LLi misses:      3,435
==11685== I1  miss rate:      0.00%
==11685== LLi miss rate:      0.00%
==11685==
==11685== D   refs:      6,443,094,149 (6,343,166,051 rd + 99,928,098 wr)
==11685== D1  misses:      381,834,663 ( 380,362,440 rd + 1,472,223 wr)
==11685== LLd misses:      380,731,515 ( 379,260,366 rd + 1,471,149 wr)
==11685== D1  miss rate:      5.9% (      6.0% +      1.5% )
==11685== LLd miss rate:      5.9% (      6.0% +      1.5% )
==11685==
==11685== LL refs:      381,838,238 ( 380,366,015 rd + 1,472,223 wr)
==11685== LL misses:      380,734,950 ( 379,263,801 rd + 1,471,149 wr)
==11685== LL miss rate:      0.9% (      0.9% +      1.5% )
```

## Performance:

```
aditya@aditya-HP-Laptop-15-da1xxx:~/Documents/A1 Final/A1 Unchanged$ make run
./classify rfile dfile 1009072 4 3
348.959 ms
351.303 ms
361.09 ms
3 iterations of 1009072 items in 1001 ranges with 4 threads: Fastest took 348.95
9 ms, Average was 353.784 ms
```

## Analysis of Modified Code:

### gprof:

#### Flat Profile:

Flat profile:

Each sample counts as 0.01 seconds.

% time	cumulative seconds	self seconds	calls	self Ts/call	total Ts/call	name
100.25	2.33	2.33				readRanges(char const*)
0.00	2.33	0.00	3	0.00	0.00	classify(Data&, Ranges const&, unsigned int)
0.00	2.33	0.00	3	0.00	0.00	timedwork(Data&, Ranges const&, unsigned int)
0.00	2.33	0.00	1	0.00	0.00	_GLOBAL__sub_I_Z8classifyR4DataRK6Rangesj
0.00	2.33	0.00	1	0.00	0.00	_GLOBAL__sub_I_Z9timedworkR4DataRK6Rangesj

## Call graph:



Call graph (explanation follows)

granularity: each sample hit covers 2 byte(s) for 0.43% of 2.33 seconds

index	% time	self	children	called	name
[1]	100.0	2.33	0.00		<spontaneous> readRanges(char const*) [1]
-----					
[9]	0.0	0.00	0.00	3/3	timedwork(Data&, Ranges const&, unsigned int) [10] classify(Data&, Ranges const&, unsigned int) [9]
-----					
[10]	0.0	0.00	0.00	3/3	repeatrun(unsigned int, Data&, Ranges const&, unsigned int) [14] timedwork(Data&, Ranges const&, unsigned int) [10] 0.00 0.00 3/3 classify(Data&, Ranges const&, unsigned int) [9]
-----					
[11]	0.0	0.00	0.00	1/1	__libc_csu_init [18] _GLOBAL__sub_I_Z8classifyR4DataRK6Rangesj [11]
-----					
[12]	0.0	0.00	0.00	1/1	__libc_csu_init [18] _GLOBAL__sub_I_Z9timedworkR4DataRK6Rangesj [12]
-----					

## valgrind:

### Cachegrind tool report:

```
==13005==
==13005== I   refs:      34,208,109,079
==13005== I1 misses:      3,734
==13005== LLi misses:      3,595
==13005== I1 miss rate:      0.00%
==13005== LLi miss rate:      0.00%
==13005==
==13005== D   refs:      6,445,590,660 (6,345,353,636 rd + 100,237,024 wr)
==13005== D1 misses:      382,214,883 ( 380,366,167 rd +   1,848,716 wr)
==13005== LLd misses:      381,319,212 ( 379,471,754 rd +   1,847,458 wr)
==13005== D1 miss rate:      5.9% (      6.0% +      1.8% )
==13005== LLd miss rate:      5.9% (      6.0% +      1.8% )
==13005==
==13005== LL refs:      382,218,617 ( 380,369,901 rd +   1,848,716 wr)
==13005== LL misses:      381,322,807 ( 379,475,349 rd +   1,847,458 wr)
==13005== LL miss rate:      0.9% (      0.9% +      1.8% )
```

## Performance:

```
aditya@aditya-HP-Laptop-15-da1xxx:~/Documents/A1 Final/A1$ make run
./classify rfile dfile 1009072 4 3
298.402 ms
303.208 ms
301.19 ms
3 iterations of 1009072 items in 1001 ranges with 4 threads: Fastest took 298.4
02 ms, Average was 300.933 ms
```

## Modifications:

### 1) Avoiding False sharing:

- a) In classify.cpp, at the time of writing to rangecount and writing to D2, each thread (thread id is tid) is given ranges tid, tid + numt, tid + 2\*numt, .... Here, there is a high probability that the threads are writing to elements which belong to the same cache line. So, I modified the code such that each thread is given ranges  $\text{tid} * (\text{R.num}() / \text{numt})$  to  $(\text{tid} + 1) * (\text{R.num}() / \text{numt})$ . Here, there is less chance that different threads access elements from the same cache line. This will also decrease the data cache miss rate
- b) In Counter class of classify.h, \_counts is an array of size 'num' (no of threads). So, if two threads are writing to this array, it is possible that both threads want to write to an element belonging to the same cache line. So, I added padding to \_counts array. Now, the size of the array is  $16 * \text{num}$  (int is 4 bytes, so  $16 * 4 = 64$  which is the size of the cache line). Now it is guaranteed that each thread writes to different cache lines.
- c) In classify.cpp, counts is an array of size R.num(), where each element of this array is a Counter object. But, if two threads want to modify two counter objects which are near, then there is a chance that they belong to the same cache line. So, I added padding to the counts array. Now, the size of counts array is  $2 * \text{R.num}()$ . So, now there is less chance of false sharing

### 2) Parallelization:

- a) In classify.cpp, the rangecount array is filled sequentially. I parallelized by sharing ranges among different threads.

## Observations:

There is a clear improvement in the performance. The modified code is ~50 ms faster than the original code (for 4 threads).

## Submission:

I have submitted classify.cpp, classify.h, makefile, gprof-analysis-original.txt, gprof-analysis-modified.txt, report.pdf