Homework 3

Yuanrui Zhang (yz545)

Problem 1. Programming Models

(1) "for i" loop only
- Read-only: N, data_array
- Read/write non-conflicting: data_gridX, data_gridY
- Read/write conflicting: i, j, sum, product, measurement

(2) "for j" loop only
- Read-only: N, data_array, sum, i
- Read/write non-conflicting: None
- Read/write conflicting: j, data_gridX, data_gridY, product, measurement.

Problem 2. Code Analysis for Parallel Task Identification

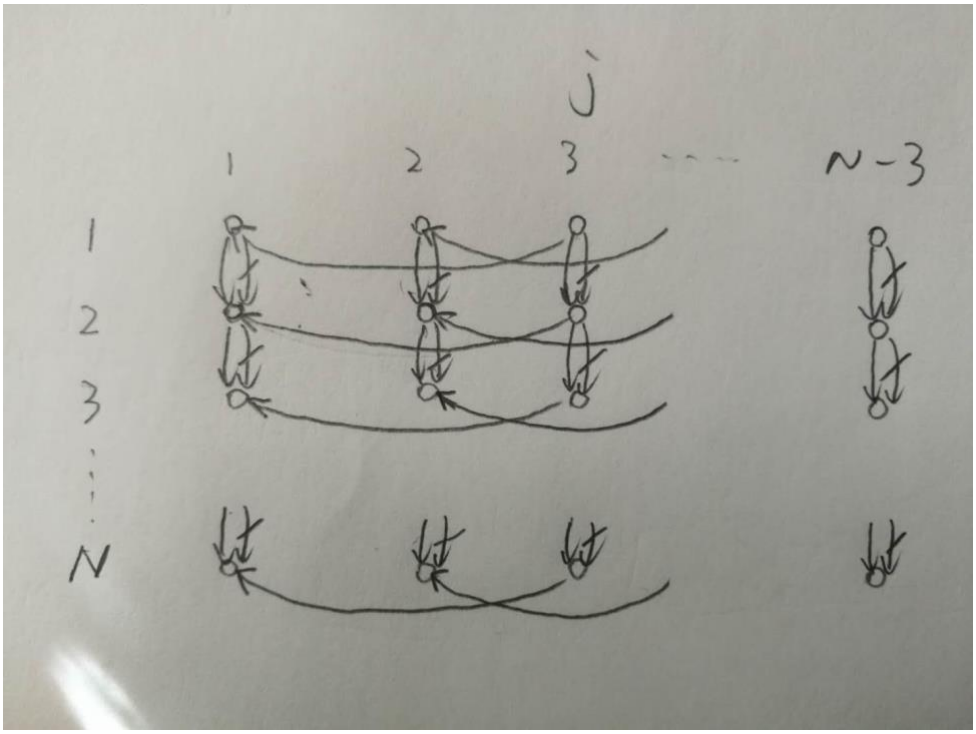
Loop-carried dependency:

(1) S1[i, j] → T S1[i, j - 2]

(2) S2[i, j] → T S2[i + 1, j]

(3) S2[i, j] → A S2[i + 1, j]

LDG:



(a) No, it isn't. Because of the loop carried dependency (2) and (3) list above, there can't be DOALL parallelism.

(b) No, the same reason as (a) – there is a loop carried dependency (1) between for j loop iterations.

(c) The update of each node along a diagonal is an independent parallel task because there is chain of dependency in this pattern, e.g., [1, 3]→[1, 1]→[2,1]→[3, 1], but along anti-diagonal is not.

(d) Yes. We can also parallelize for j loop with odd indices and with even indices. Because according to the LDG, the loop-carried dependency only exists among odd columns or among even columns. That's said, no such dependency exists across an even and an odd column. Thus, we can effectively divide it into 2 parallel tasks.

Problem 3. Code Profiling & Performance Counters

(a) Performance profiling

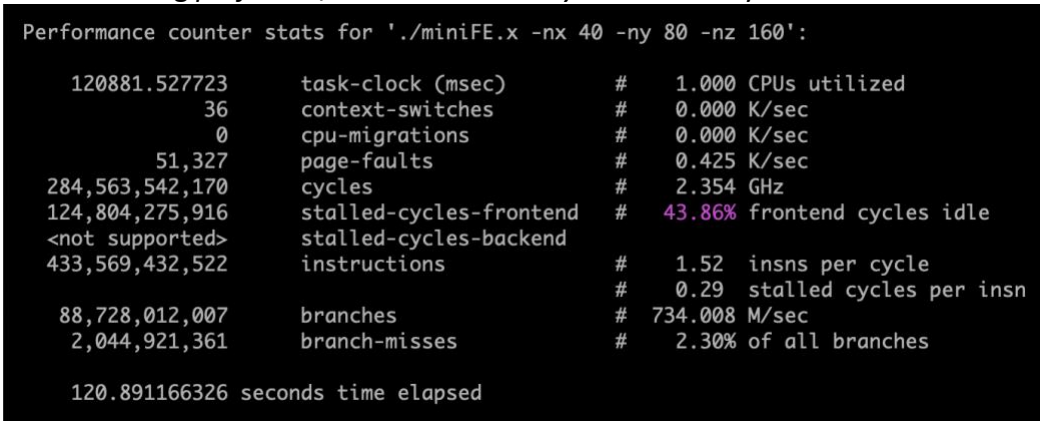
Function index	Function name	The number of calls	percentage of execution time
1	miniFE::matvec_std<miniFE::CSRMatrix<double, int, int>, miniFE::Vector<double, int, int> >::operator()(miniFE::CSRMatrix<double, int, int>&, miniFE::Vector<double, int, int>&, miniFE::Vector<double, int, int>&)	201	30.09%
2	frame_dummy	1597918831	11.08%
3	std::_Rb_tree<int, int, std::_Identity<int>, std::less<int>, std::allocator<int> >::_S_key(std::_Rb_tree_node<int> const*)	57598102	6.52%
4	std::_Rb_tree<int, int, std::_Identity<int>, std::less<int>, std::allocator<int> >::_S_value(std::_Rb_tree_node<int> const*)	435792686	4.48%
5	int* std::lower_bound<int*, unsigned long>(int*, int*, unsigned long const&)	32768000	3.69%
6	__gnu_cxx::__aligned_membuf<int>::_M_ptr()	435883250	3.22%
7	std::_Rb_tree_node<int>::_M_valptr()	435883250	2.99%
8	miniFE::decide_how_to_shrink(Box const&, Box const&)	719	2.99%

(b) Amdahl's Law

Speedup = 1/(1 - 0.3009 + 0.3009 / 5) = 1.32

(c) Performance Counters

(1) When running *perf stat ./miniFE.x -nx 40 -ny 80 -nz 160* by default:



(2) Measure Instructions; CPU cycles (and also show IPC, instructions per cycle); Branch instructions; Branches misses (mispredictions)

```
Performance counter stats for './miniFE.x -nx 40 -ny 80 -nz 160':

  281,371,002,033      cpu-cycles
  433,564,078,692      instructions              #    1.54  insns per cycle
   88,726,944,642      branches
    1,872,806,757      branch-misses             #    2.11% of all branches

    119.743417490 seconds time elapsed
```

(3) Measure Cache references; L1 data cache load misses; L1 instruction cache load misses; LLC (last level cache) loads; LLC (last level cache) load misses; Data TLB load misses

```
Performance counter stats for './miniFE.x -nx 40 -ny 80 -nz 160':

      338,645,856      cache-references                                              (66.67%)
      767,371,473      L1-dcache-load-misses                                         (66.67%)
       11,115,297      L1-icache-load-misses                                         (66.67%)
      655,282,132      LLC-loads                                                     (66.67%)
      350,068,628      LLC-load-misses           #   53.42% of all LL-cache hits     (66.67%)
       11,679,454      dTLB-load-misses                                              (66.66%)

    120.129659673 seconds time elapsed
```

Problem 4. Performance Counters

1. Re-run my program across different loop nest orderings on the machine where I am using ‘perf’

Loop nest orderings	Time
I-J-K	14.641327 s
I-K-J	0.716340 s
J-K-I	27.432645 s

2. Use ‘perf’ to see performance counters

(1) I-J-K

```
********************I-J-K********************
Time = 14.508862 s


 Performance counter stats for './matrix 1':

       182,402,594      cache-references                                              (66.65%)
     2,161,328,049      L1-dcache-load-misses                                         (66.68%)
           625,832      L1-icache-load-misses                                         (66.69%)
       181,109,333      LLC-loads                                                     (66.69%)
         4,702,837      LLC-load-misses           #    2.60% of all LL-cache hits     (66.68%)
     1,080,018,666      dTLB-load-misses                                              (66.65%)

      14.577152333 seconds time elapsed
```

(2) I-K-J

```
********************I-K-J********************
Time = 0.749057 s


 Performance counter stats for './matrix 2':

        11,425,959      cache-references                                              (66.62%)
       139,369,377      L1-dcache-load-misses                                         (66.62%)
            91,449      L1-icache-load-misses                                         (66.62%)
        11,042,391      LLC-loads                                                     (66.98%)
           245,795      LLC-load-misses           #    2.23% of all LL-cache hits     (67.02%)
         3,313,330      dTLB-load-misses                                              (66.66%)

       0.815300188 seconds time elapsed
```

(3) J-K-I

```
********************J-K-I********************
Time = 27.226312 s


 Performance counter stats for './matrix 3':

       534,973,081      cache-references                                              (66.65%)
     3,228,790,274      L1-dcache-load-misses                                         (66.67%)
           957,443      L1-icache-load-misses                                         (66.68%)
       534,981,252      LLC-loads                                                     (66.68%)
           788,012      LLC-load-misses           #    0.15% of all LL-cache hits     (66.67%)
     1,892,331,671      dTLB-load-misses                                              (66.65%)

      27.300640638 seconds time elapsed
```

From the three screenshots above, the first column of performance counter stats represents the raw counter numbers. We are easy to tell from those numbers that I-K-J has the least

number of all kinds of cache misses, while J-K-I has the most and I-J-K stays in between. In summary, performance counter results explain why these three patterns have different performance.