ECE 565
Fall 2021

Assignment #3

Submit instructions: Submit a file named **hw3_writeup.pdf**.

1. **Programming Models.** For the following code, identify variables that are read-only, read/write non-conflicting, and read-write conflicting, if we consider parallelizing the "for i" loop only or if we consider parallelizing the "for j" loop only. In the code, the function 'ceil(arg)' returns the integer ceiling of the passed argument.

```
float product;
int i, j, N, sum;
int data_array[N];
int data_gridX[N][N], data_gridY[N][N];
float measurement[N];

for (i=1; i<N; i++) {
   product = 1;
   for (j=1; j<N; j++) {
        product = product * sum * measurement[j]
        measurement[ceil(product)]++;
        if ((i-j) >= 0)
             data_gridX[i-j][j] = data_gridX[i-j][j] * 2;
        if ((j-i) >= 0)
             data_gridY[i][j-i] = data_gridY[i][j-i] / 4;
   }
   sum = sum + data_array[i] + product;
}
```

1) Considering parallelizing i-loop only:
   read-only: ***N, data_array***
   read/write non-conflicting: ***data_gridX, data_gridY***
   read-write conflicting: ***product, sum, i,j, measurement***

2) Considering parallelizing j-loop only:
   read-only: ***i, data_array, N, sum***
   read/write non-conflicting: ***data_gridX, data_gridY***
   read-write conflicting: ***product, measurement, j***

2)

Sol:

Dependencies:

Loop-independent:

S1[i,j] -> A S2[i,j] (1)

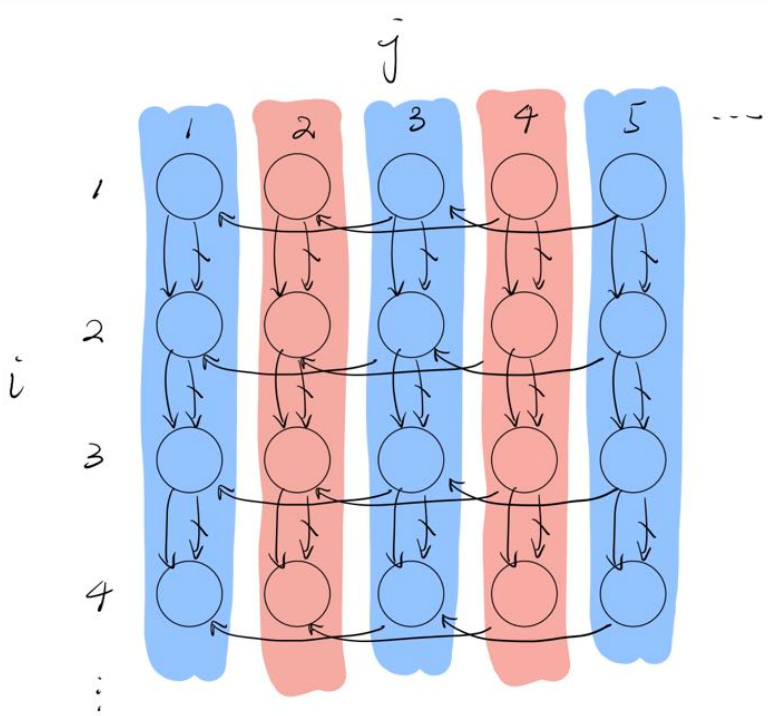
Loop-carried:

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] (4)

LDG:



a)

No. It's not, since we have loop-carried dependencies (3) and (4).

b)

No. It's not, since we have loop-carried dependency (2).

c)

Diagonal is independent while anti-diagonal is not.

d)

Yes. As shown in the LDG, the odd j and even j can be grouped as two groups of tasks. And the two groups are independent to each other.

3.

a)

There was a frame dummy function, costing a lot of time. But after looking up some documentations, it seems to be a linux feature and not really what we want to study here so I omitted it.

Function Name	# of calls	% of excution time
MatvecOp<miniFE::CSRMatrix<double, int, int, SerialComputeNode> >::operator()(int)	51	9.15
std::_Rb_tree<int, int, std::_Identity<int>, std::less<int>, std::allocator<int> >::_M_lower_bound (std::_Rb_tree_node<int> const*, std::_Rb_tree_node_base const*, int const&) const	28316792	7.15
NoOpMemoryModel::~NoOpMemoryModel()	364141551	5.35
void std::advance<int*, long>(int*&, long)	32768000	5.35
void miniFE::Hex8::diffusionMatrix_symm<double> (double const*, double const*, double*)	512000	5.35
std::_Rb_tree<int, int, std::_Identity<int>, std::less<int>, std::allocator<int> >::_S_key (std::_Rb_tree_node<int> const*)	435792686	5.21
__gnu_cxx::__aligned_membuf<int>::_M_addr() const	435837968	4.33
std::_Rb_tree_node<int>::_M_valptr() const	435837968	4.13

b)

The top function's percentage is 9.15%.
The speed up could be 1/(90.85%+9.15%/5) = 1.08x

c)

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

  376,988,763,112      instructions              #    1.54  insn per cycle          (50.00%)
  244,018,144,509      cycles                                                       (50.00%)
   70,655,773,639      branches                                                     (50.00%)
    2,159,582,700      branch-misses             #    3.06% of all branches         (50.00%)
       54,445,256      cache-references                                             (50.00%)
      213,571,171      L1-dcache-load-misses                                        (40.00%)
       17,229,192      L1-icache-load-misses                                        (40.00%)
       89,187,119      LLC-loads                                                    (40.00%)
       50,079,839      LLC-load-misses           #   56.15% of all LL-cache accesses  (40.00%)
        4,584,019      dTLB-load-misses                                             (40.00%)

    104.513483417 seconds time elapsed

    104.376476000 seconds user
      0.079954000 seconds sys
```

4)

The new run results are shown below:

	i-j-k	k-i-j	j-k-i
Time elapsed (s)	12.014252	0.982263	26.374790

Using perf to see L1-dcache-load-misses:

```
xl350@leviathan:~/hw2/problem4$ perf stat -e L1-dcache-load-misses ./matrix 1
Time = 12.429502s

 Performance counter stats for './matrix 1':

     1,625,341,489      L1-dcache-load-misses

      12.451678299 seconds time elapsed

      12.440566000 seconds user
       0.003998000 seconds sys


xl350@leviathan:~/hw2/problem4$ perf stat -e L1-dcache-load-misses ./matrix 2
Time = 0.988752s

 Performance counter stats for './matrix 2':

       137,619,351      L1-dcache-load-misses

       1.010864582 seconds time elapsed

       0.993943000 seconds user
       0.016031000 seconds sys


xl350@leviathan:~/hw2/problem4$ perf stat -e L1-dcache-load-misses ./matrix 3
Time = 26.362303s

 Performance counter stats for './matrix 3':

     2,974,224,054      L1-dcache-load-misses

      26.384277687 seconds time elapsed

      26.357337000 seconds user
       0.011993000 seconds sys
```

It can be easily seen that the run time is proportional to the count of L1-dcache-load-misses and that makes sense because that's where most of the time spent.