Homework3 Writeup

1. Programming Models

“for i" loop only

Read-only: N, data_array

Read/write non-conflicting: data_gridY, data_gridX

Read-write conflicting: i, j, product, sum, measurement

“for j" loop only

Read-only: i, N, sum

Read/write non-conflicting: data_gridX, data_gridY

Read-write conflicting: j, product, measurement

2. Code Analysis for Parallel Task Identification

Dependences:

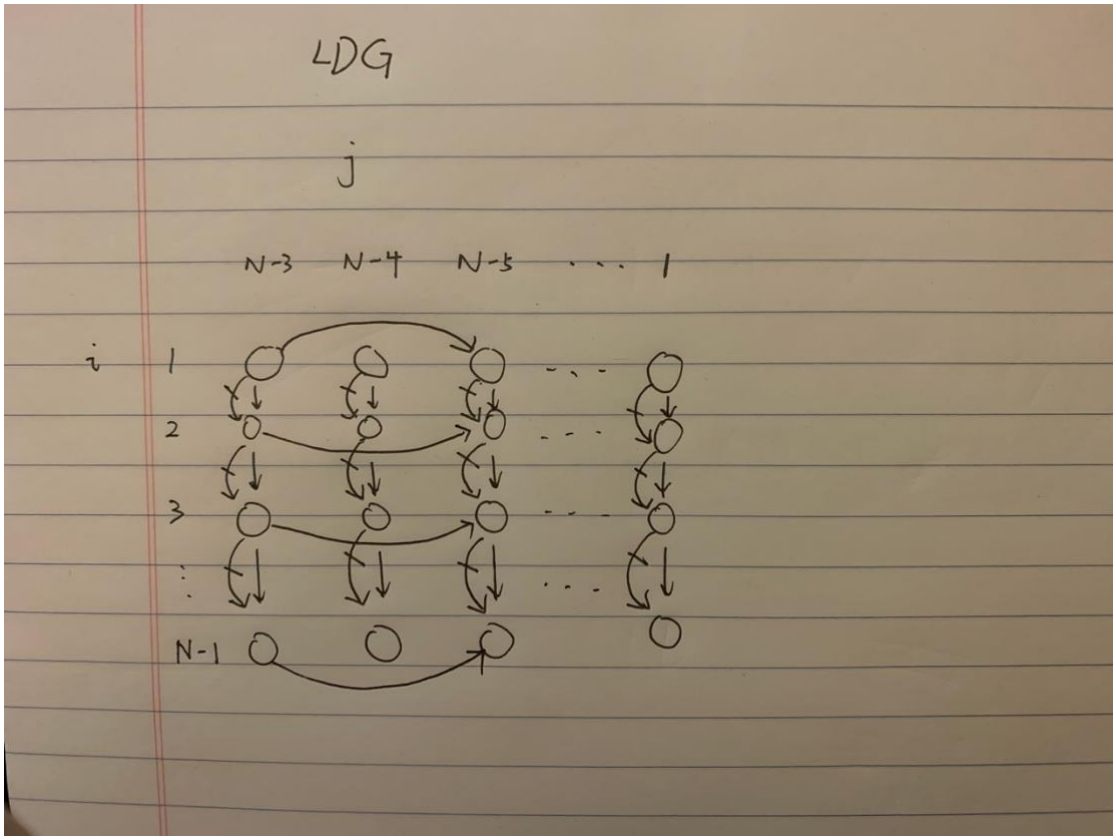
S2[i,j] => T S2[i+1,j] (loop carried)

S2[i-1,j] => A S2[i,j] (loop carried)

S1[i,j] => A S2[i,j] (loop independent)

S1[i,j] => T S1[i,j-2] (loop carried)

LDG:



(a)

No, because there are two loop carried dependences:
S2[i,j] => T S2[i+1,j] (loop carried)
S2[i-1,j] => A S2[i,j] (loop carried)

(b)

No, because there is a loop carried dependence:
S1[i,j] => T S1[i,j-2] (loop carried)

(c)

According to the LDG, (Here my LDG need to be Mirrored left and right to make sure the smallest j is on the left)
Update each node along a diagonal is an independent parallel task.
Update each node along an anti-diagonal is not an independent parallel task.

(d)

For a fixed i, the j loop can be divided onto two parallel tasks: all the iterations with odd j can be paralleled with all the iterations with even j.

3. Code Profiling & Performance Counters.

(a)

Without -fno-inline:

Function 1

1) the function name

void miniFE::cg_solve<miniFE::CSRMatrix<double, int, int>, miniFE::Vector<double, int, int>, miniFE::matvec_std<miniFE::CSRMatrix<double, int, int>, miniFE::Vector<double, \
int, int> > >(miniFE::CSRMatrix<double, int, int>&, miniFE::Vector<double, int, int> const&, miniFE::Vector<double, int, int>&, miniFE::matvec_std<miniFE::CSRMatrix<double, int, int>, miniFE::Vector<double, int, int> >, miniFE\
::CSRMatrix<double, int, int>::LocalOrdinalType, miniFE::TypeTraits<miniFE::CSRMatrix<double, int, int>::ScalarType>::magnitude_type&, miniFE::CSRMatrix<double, int, int>::LocalOrdinalType&, miniFE::TypeTraits<miniFE::CSRMatri\
x<double, int, int>::ScalarType>::magnitude_type&, double*)

2) the number of calls

1

3) the percentage of execution time for all calls to the function

64.31%

Function 2

1) the function name

void miniFE::impose_dirichlet<miniFE::CSRMatrix<double, int, int>, miniFE::Vector<double, int, int> >(miniFE::CSRMatrix<double, int, int>::ScalarType, miniFE::CSRMatrix<dou\
ble, int, int>&, miniFE::Vector<double, int, int>&, int, int, int, std::set<miniFE::CSRMatrix<double, int, int>::GlobalOrdinalType, std::less<miniFE::CSRMatrix<double, int, int>::GlobalOrdinalType>, std::allocator<miniFE::CSRM\
atrix<double, int, int>::GlobalOrdinalType> > const&)

2) the number of calls

2

3) the percentage of execution time for all calls to the function

7.50%

Function 3

1) the function name

void miniFE::Hex8::diffusionMatrix_symm<double>(double const*, double const*, double*)

2) the number of calls

512000

3) the percentage of execution time for all calls to the function

6.67%

Function 4

1) the function name

void miniFE::sum_in_symm_elem_matrix<miniFE::CSRMatrix<double, int, int> >(unsigned long, miniFE::CSRMatrix<double, int, int>::GlobalOrdinalType const*, miniFE::CSRMatrix<d\
ouble, int, int>::ScalarType const*, miniFE::CSRMatrix<double, int, int>&)

2) the number of calls

512000

3) the percentage of execution time for all calls to the function

5.56%

Function 5

1) the function name

void miniFE::Hex8::gradients_and_invJ_and_detJ<double>(double const*, double const*, double*, double&)

2) the number of calls

4096000

3) the percentage of execution time for all calls to the function

3.89%

Function 6

1) the function name

void miniFE::Hex8::gradients_and_detJ<double>(double const*, double const*, double&)

2) the number of calls

4096000

3) the percentage of execution time for all calls to the function

2.50%

Function 7

1) the function name

int miniFE::generate_matrix_structure<miniFE::CSRMatrix<double, int, int> >(miniFE::simple_mesh_description<miniFE::CSRMatrix<double, int, int>::GlobalOrdinalType> const&, \
miniFE::CSRMatrix<double, int, int>&)

2) the number of calls

1

3) the percentage of execution time for all calls to the function

2.36%

Function 8

1) the function name

void miniFE::init_matrix<miniFE::CSRMatrix<double, int, int> >(miniFE::CSRMatrix<double, int, int>&, std::vector<miniFE::CSRMatrix<double, int, int>::GlobalOrdinalType, std\
::allocator<miniFE::CSRMatrix<double, int, int>::GlobalOrdinalType> > const&, std::vector<miniFE::CSRMatrix<double, int, int>::LocalOrdinalType, std::allocator<miniFE::CSRMatrix<double, int, int>::LocalOrdinalType> > const&, s\
td::vector<int, std::allocator<int> > const&, int, int, int, miniFE::CSRMatrix<double, int, int>::GlobalOrdinalType, miniFE::simple_mesh_description<miniFE::CSRMatrix<double, int, int>::GlobalOrdinalType> const&)

2) the number of calls

1

3) the percentage of execution time for all calls to the function

1.39%

With -fno-inline:(This version is not correct due to the large amount of time that frame_dummy consumes)

Function 1

1) the function name

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>&)

2) the number of calls

201

3) the percentage of execution time for all calls to the function

32.24%

Function 2

1) the function name

frame_dummy

2) the number of calls

1622833938

3) the percentage of execution time for all calls to the function

13.22%

Function 3

1) the function name

std::_Rb_tree<int, int, std::_Identity<int>, std::less<int>, std::allocator<int> >::_S_key(std::_Rb_tree_node<int> const*)

2) the number of calls

57598102

3) the percentage of execution time for all calls to the function

6.85%

Function 4

1) the function name

std::_Rb_tree<int, int, std::_Identity<int>, std::less<int>, std::allocator<int> >::_S_value(std::_Rb_tree_node<int> const*)

2) the number of calls

435792686

3) the percentage of execution time for all calls to the function

4.67%

Function 5

1) the function name

std::_Rb_tree_node<int>::_M_valptr()

2) the number of calls

435928532

3) the percentage of execution time for all calls to the function

4.51%

Function 6

1) the function name

void miniFE::Hex8::diffusionMatrix_symm<double>(double const*, double const*, double*)

2) the number of calls

512000

3) the percentage of execution time for all calls to the function

3.71%

Function 7

1) the function name

std::pair<int const, int>* std::__addressof<std::pair<int const, int> >(std::pair<int const, int>&)

2) the number of calls

510695430

3) the percentage of execution time for all calls to the function

2.98%

Function 8

1) the function name

int* std::lower_bound<int*, unsigned long>(int*, int*, unsigned long const&)

2) the number of calls

32768000

3) the percentage of execution time for all calls to the function

2.98%

(b)

For the correct version: without the -fno-inline

The top function's percentage is 64.31%

So speedup = 1/(1-64.31% + 64.31%/5) =~ 2.06

(c)

Instructions: 39728442650

CPU cycles (and also show IPC, instructions per cycle): 18035237785 **IPC=~2.2**

Branch instructions: 6086907303

Branches misses (mispredictions) 12513472

Cache references 334524113

L1 data cache load misses 739973717

L1 instruction cache load misses 859152

LLC (last level cache) loads 630678387

LLC (last level cache) load misses 339503180

Data TLB load misses 12156972

4. Performance Counters

Note: using ./testIJK <1/2/3> to run the code.

1:IJK 2:JKI 3:IKJ

```
yz558@leviathan:~/ECE565/ECE565_Parallel/hw2/hw2Q4$ ./testIJK 1
IJK_Time=21.132577 Seconds
yz558@leviathan:~/ECE565/ECE565_Parallel/hw2/hw2Q4$ ./testIJK 2
JKI_Time=28.102554 Seconds
yz558@leviathan:~/ECE565/ECE565_Parallel/hw2/hw2Q4$ ./testIJK 3
IKJ_Time=16.421212 Seconds
```

Loop type	Time(second)
IJK	21.132577
JKI	28.102554
IKJ	16.421212

Perf Results:

IJK

```
yz558@leviathan:~/ECE565/ECE565_Parallel/hw2/hw2Q4$ perf stat -e L1-dcache-load-miss ./testIJK 1
IJK_Time=20.854039 Seconds

 Performance counter stats for './testIJK 1':

       1887558535      L1-dcache-load-miss

     20.882189334 seconds time elapsed
```

JKI:

```
yz558@leviathan:~/ECE565/ECE565_Parallel/hw2/hw2Q4$ perf stat -e L1-dcache-load-miss ./testIJK 2
JKI_Time=26.576884 Seconds

 Performance counter stats for './testIJK 2':

       3507912705      L1-dcache-load-miss

     26.594020160 seconds time elapsed
```

IKJ:

```
yz558@leviathan:~/ECE565/ECE565_Parallel/hw2/hw2Q4$ perf stat -e L1-dcache-load-miss ./testIJK 3
IKJ_Time=16.382889 Seconds

 Performance counter stats for './testIJK 3':

        138893850      L1-dcache-load-miss

      16.408541739 seconds time elapsed
```

Loop type	L1-dcache-load-miss counts	L1-dcache-load-miss counts/(1024^3)
IJK	1887558535	1.75
JKI	3507912705	3.26
IKJ	138893850	0.129

This ratio matches the class results. As theoretically misses per iteration for the 3 cases are

ijk:1.125

jki:2

ikj:0.25

This match the ratio of the experiment result that jki is the slowest and ikj is the fastest.

For IJK AND JKI it is little larger, I think it might be caused by other data already in the cache. For IKJ it is smaller I think it is caused by the prefetch.