HPC Home Assignment Exercise 1

Yuxuan Jing 1454901

**1.1**

base = n / p;

remainder = n % p;

for (any core){

    nthread = get\_thread\_num();

    if (nthread < reminder){

        my\_first\_i = nthread \* (base + 1);

        my\_last\_i = (nthread + 1) \* (base + 1);

    }

    else{

        my\_first\_i = nthread \* base + remainder;

        my\_last\_i = (nthread + 1) \* base + remainder;

    }

}

**1.2**

//solution 1

n\_tot = (1+n)\*n/2;

each\_core\_work = n\_tot/p;

my\_first\_i[p];

my\_last\_i[p];

sum = 0

my\_first\_i.append(0)

for (size\_t i = 0; i < p; i++){

    sum += i + 1;

    if (sum > each\_core\_work){

        my\_last\_i.append(i-1)

    }

    sum = i + 1;

    my\_first\_i.append(i)

}

my\_last\_i.append(p-1);

for(each\_core){

    nthread = get\_thread\_num();

    my\_first\_i = my\_first\_i[nthread]

    my\_last\_i = my\_last\_i[nthread]

}

//solution 2

n\_each = n/p + (n%p != 0);

assignment[p][n\_each];

for (size\_t i = 0; i < p; i++){

    if (i%(2\*p) < p){

        assignment[i%p].append(i);

    }

    else{

        assignment[p - i%p - 1].append(i);

    }

}

//solution 3

n\_each = n/p + (n%p != 0);

assignment[p][n\_each];

for (size\_t i = 0; i < p; i++){

    if (i%(2\*p) < p){

        assignment[i%p].append(n-i);

    }

    else{

        assignment[p - i%p - 1].append(n-i);

    }

}

**1.3**

int divisor = 1;

int n\_thread = get\_thread\_total();

while (divisor < n\_thread){

    for (size\_t i = 0; i < n\_thread, i += 2 \* divisor){

        if (i % (2\*divisor) == 0){

            sum(core i) += sum(core i + divisor);

        }

        // else{

        //     send sum to core i - divisor;

        // }

    }

    divisor \*= 2;

}

**1.4**

bit = 0;

int n\_thread = get\_thread\_total();

while ( (1 << bit) < n\_thread){

    bitmask = 1 << bit;

    for (size\_t i = 0; i < n\_thread, i += bitmask){

        if ( (i & bitmask) == 0){

            sum(core i) += sum(core i + bitmask);

        }

        // else{

        //     send sum to core i - bitmask;

        // }

    }

    bit += 1;

}

**1.5**

int divisor = 1;

int n\_thread = get\_thread\_total();

while (divisor < n\_thread){

    for (size\_t i = 0; i < n\_thread, i += 2 \* divisor){

        if (i % (2\*divisor) == 0){

            if (i + divisor < n\_thread){  // adding for this problem

                sum(core i) += sum(core i + divisor);

            }

        }

    }

    divisor \*= 2;

}

bit = 0;

int n\_thread = get\_thread\_total();

while ( (1 << bit) < n\_thread){

    bitmask = 1 << bit;

    for (size\_t i = 0; i < n\_thread, i += bitmask){

        if ( (i & bitmask) == 0){

            if (i + bitmask < n\_thread){  // adding for this problem

                sum(core i) += sum(core i + bitmask);

            }

        }

    }

    bit += 1;

}

**1.6**

p = n\_thread\_total();

n1 = p - 1;

n2 = log2(p) + !((double)(int)log2() == log());

p       Original    Tree

2^1     2^1-1       1

2^2     2^2-1       2

2^3     2^3-1       3

2^4     2^4-1       4

2^5     2^5-1       5

2^6     2^6-1       6

2^7     2^7-1       7

2^8     2^8-1       8

2^9     2^9-1       9

2^10    2^10-1      10

**1.7**

combination of task- and data- parallelism.

task-parallelism: send sum and receive sum, add sum at the same time.

data-parallelism: receive nodes are doing the same kinds of task: adding two sums.

**1.8**

a.

cleaning the house

set the room

prepare food

b.

5 people clean 5 parts of the department at the same time.

c.

5 people abcde

clean 5 parts in the department

then

ab prepare the food while cde set the rooms

        a       b       c       d       e

time1   clean1  clean2  clean3  clean4  clean5

time2   food1   food2   set1    set2    set3

time3   food1   food2   set1    set2    set3

time4   food1   food2   set1    set2    set3