Qun Lou 862325101 qlou005@ucr.edu

using 2 grace days

1

Cores: 2*12=24
HyperThread: 48*2=96
L1d cache: 32K
L1i cache: 32K
L2 cache: 1024K
L3 cache: 16896K

```
[qlou005@xe-01 homework0-ultramar1ne]$ lscpu
Architecture:          x86_64
CPU op-mode(s):        32-bit, 64-bit
Byte Order:            Little Endian
CPU(s):                48
On-line CPU(s) list:   0-47
Thread(s) per core:    2
Core(s) per socket:    12
Socket(s):             2
NUMA node(s):          2
Vendor ID:             GenuineIntel
CPU family:            6
Model:                 85
Model name:            Intel(R) Xeon(R) Silver 4214R CPU @ 2.40GHz
Stepping:              7
CPU MHz:               3500.000
CPU max MHz:           3500.0000
CPU min MHz:           1000.0000
BogoMIPS:              4800.00
Virtualization:        VT-x
L1d cache:             32K
L1i cache:             32K
L2 cache:              1024K
L3 cache:              16896K
NUMA node0 CPU(s):     0-11,24-35
NUMA node1 CPU(s):     12-23,36-47
```

2

I little changed the program as `Usage: ./new_reduce [num_elements] [num_thread]` using `__cilkrts_set_param("nworkers",argv[2])`

(a)

51sec for 1 thread vs 0.06 second in sequential computation.

```
Singularity> ./new_reduce 1000000000 1
time for thread num 1 : 51.1878
499999999500000000
time for sequential computation: 0.060709
499999999500000000
```

(b)

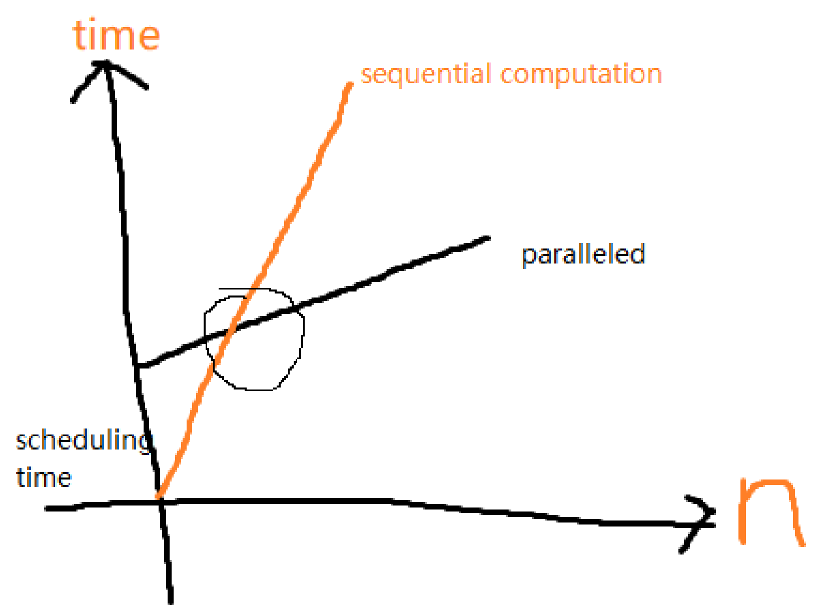
(repeat for 10 times and get the average)

As thread num increases, the time cost changes roughly inversely in proportion.

```
time for thread num 1 : 51.1878
time for thread num 2 : 25.1264
time for thread num 4 : 13.0387
time for thread num 8 : 6.79833
time for thread num 12 : 4.62333
time for thread num 24 : 2.84901
time for thread num 48 : 1.66931
```

3

I want to use **binary search** to find the threshold. Because in my opinion, the paralled algorithm and the sequential one shold perform as the graph below:



But unluckily, the truth is that it seems under no circumstance can the paralleled ones beat the sequential algorithm.

I also close the optimization option in makefile and the situation keeps the same.

```
Singularity> ./new_reduce 1000000000 48
time for thread num 48 : 1.66931
499999999500000000
time for sequential computation: 0.060827
499999999500000000
```

As a result, my threshold is that adding one by one is the best choice under all circumstances.

code

change thread num

```
#include <iostream>
#include <cstdio>
#include <stdlib.h>
#include <cilk/cilk.h>
#include <cilk/cilk_api.h>
#include "get_time.h"
using namespace std;

size_t reduce(size_t* A, int n) {
        if (n == 1) return A[0];
        size_t L, R;
        L = cilk_spawn reduce(A, n/2);
        R = reduce(A+n/2, n-n/2);
        cilk_sync;
        return L+R;
}

long long add(int n){
    if (n==1) return 0;
    long long res = 0;
    for (int i=0; i<n; i++){
        res+=i;
    }
    return res;
}

int main(int argc, char** argv) {
        if (argc != 3) {
                cout << "Usage: ./new_reduce [num_elements] [num_thread]" <<
endl;
                return 0;
        }
        int n = atoi(argv[1]);
        timer t;
        __cilkrts_set_param("nworkers",argv[2]);
        size_t* A = new size_t[n];
        cilk_for (int i = 0; i < n; i++) A[i] = i;
        t.start(); double t_start=t.get_total();
        size_t x = reduce(A, n);
        t.stop();
        cout << "time for thread num "<<  __cilkrts_get_nworkers() <<" : " <<
t.get_total()-t_start << endl;
        cout << x << endl;
    timer t2; t2.start();
    long long res = add(n);
    t2.stop();
    cout << "time for sequential computation: " << t2.get_total() << endl;
    cout << res << endl;
        return 0;
}
```

find threshold

```
while (abs(t_para-t_seq) > 0.01*min(t_para,t_seq) ){
    if t_para>t_seq:
        n= (n_mid+n_max)/2
        n_min = n_mid
        n_mid = (n_min + n_max)/2
    else:
         n= (n_min+n_mid)/2
         n_max = n_mid
         n_mid = (n_min + n_max)/2
    t_para=para_calculate(n)
    t_seq=seq_cal(n)
}
```