ParallelPrefix

March 19, 2019

1 Parallel Prefix

Let's try to write first the sequential version of the parallel prefix. Initially we will just consider a short vector, than we'll assemble the actual main code with larger vectors. First we define the standard pre-amble

```
In [1]: #include <iostream>
        #include <vector>

        using namespace std;
```

We define the sample vector to compute

```
In [2]: vector<int> y = {1,2,3,4,5,6,7,8};
```

and the sequential code, needed to take into account the "quality" of what we are going to parallelize . . .

```
In [3]: int oplus(int x, int y) {
            auto temp = x+y;
            // possibly we will fill here some delay/longer computation
            return temp;
        }
```

```
In [4]: void prefix(vector<int> &x, function<int(int,int)> oplus) {
            for(int i=1; i<x.size(); i++)
                x[i]= oplus(x[i],x[i-1]);
            return;
        }
```

```
In [6]: void printv(vector<int> x){
            for(int i=0; i<x.size(); i++)
                cout << x[i] << " ";
            cout << endl;
            return;
        }
```

```
input_line_13:1:6: error: redefinition of 'printv'
void printv(vector<int> x){
     ^
input_line_12:1:6: note: previous definition is here
void printv(vector<int> x){
     ^
```

```
In [7]: printv(y);
```

```
1 2 3 4 5 6 7 8
```

Now we can compute the prefix and see if the vector has changed . . .

```
In [8]: prefix(y,oplus);
```

```
In [9]: printv(y);
```

```
1 3 6 10 15 21 28 36
```

which is ok. In order to have an idea of what's going on here, let's try to measure how much time we spend computing the prefix. We first define the usual timer:

```
In [10]: #include <chrono>
         #include <thread>


         class utimer {
           std::chrono::system_clock::time_point start;
           std::chrono::system_clock::time_point stop;
           std::string message;
           using usecs = std::chrono::microseconds;
           using msecs = std::chrono::milliseconds;

         public:

           utimer(const std::string m) : message(m) {
             start = std::chrono::system_clock::now();
           }

           ~utimer() {
             stop =
               std::chrono::system_clock::now();
             std::chrono::duration<double> elapsed =
               stop - start;
             auto musec =
               std::chrono::duration_cast<std::chrono::microseconds>(elapsed).count();
```

```
                std::cout << message << " computed in " << musec << " usec "
                          << std::endl;

              }
            };
```

```
In [11]: vector<int> x = {1,2,3,4,5,6,7,8};

         {
             utimer t("prefix8");
             prefix(x, oplus);
         }
```

```
prefix8 computed in 8 usec
```

which shows a fairly small amount of time involved, at about 1usec per add operation. We would like to expect that a longer vector takes a proportionally longer time:

```
In [12]: vector<int> longx(1024);
         for(auto &i : longx) i = 1;

In [13]: printv(longx);
```

```
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
```

```
In [14]: {
             utimer t2("longx");
             prefix(longx,oplus);
         }
```

```
longx computed in 1537 usec
```

```
In [15]: printv(longx);
```

```
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35
```

let's try to make it more significant, we add some delay in oplus, such that it will take longer …

```
In [16]: void active_udelay(int usecs) {
           // read current time
           auto start = std::chrono::high_resolution_clock::now();
           auto end   = false;
           while(!end) {
             auto elapsed = std::chrono::high_resolution_clock::now() - start;
```

```
              auto usec = std::chrono::duration_cast<std::chrono::microseconds>(elapsed).count();
              if(usec>usecs)
                end = true;
            }
            return;
          }
```

```
In [17]: int oplus2(int x, int y) {
              auto temp = x + y;
              active_udelay(1000);
              return temp;
          }
```

```
In [ ]: cout << oplus(1,1) << endl;
```

```
In [18]: vector<int> w = {1,1,1,1,1,1,1,1};
```

```
In [19]: prefix(w,oplus2);
```

```
In [20]: printv(w);
```

```
1 2 3 4 5 6 7 8
```

```
In [21]: {
              utimer t3("now");
              prefix(w,oplus2);
          }
```

```
now computed in 7044 usec
```

```
In [ ]: const int init = 1;
        vector<int> d(1024);
```

```
In [ ]: for(auto &i: d) i=2;
```

```
In [ ]: printv(d);
```

```
In [ ]: {
             utimer t("longy");
             prefix(d, oplus2);
        }
```

```
In [ ]: // printv(d);
```

2 Parallel version

We now define the parallel version. We assume the lenght of the vector is a multiple of the parallelism degree. We compute the parallel prefix by: 1) computing the local prefix on each partition 2) computing the prefix of the vector made of thre last item computed in each partition 3) adding in partition i (excluding the first one) the sum of current item with the i-1 position of the prefix vector just computed

```
In [ ]: printv(w);
```

In the first phase, we have to compute the local prefixes:

```
In [22]: void phase1(vector<int> &x,       // the input/output vector
                          vector<int> &p, // the prefix vector
                          int i,          // the partition index
                          function<int(int,int)> oplus, // the combiner function
                          int nw          // the parallelism degree
                          ) {

             auto m = x.size();
             auto delta = m / nw;
             // phase one, compute prefix of the assigned partition
             for(int j=i*delta+1; j<(i+1)*delta; j++)
                 x[j] = oplus(x[j-1], x[j]);
             // and assign final item to the prefix vector
             p[i] = x[(i+1)*delta-1];
             // cout << "OK";
             return;
         }
```

Assuming we have 2 workers, we compute the vector p of prefixes:

```
In [23]: printv(w);

1 3 6 10 15 21 28 36


In [24]: vector<int> p(2);

In [25]: vector<int> q = {1,2,3,4,5,6,7,8};

In [27]: phase1(q,p,0,oplus,2);

In [28]: printv(q);

1 3 6 10 5 6 7 8


In [29]: printv(p);
```

```
10 0
```

```
In [30]: phase1(q,p,1,oplus,2);

In [31]: printv(p);
```

```
10 26
```

```
In [32]: printv(q);
```

```
1 3 6 10 5 11 18 26
```

Then we have to compute the prefix of the prefixes

```
In [33]: prefix(p,oplus);

In [34]: printv(p);
```

```
10 36
```

```
In [35]: printv(q);
```

```
1 3 6 10 5 11 18 26
```

and eventually we have to use the vector to update the partitions

```
In [36]: void phase2(vector<int> &x, vector<int> &p, int i, function<int(int,int)> oplus, int nw
             auto m = x.size();
             auto delta = m /nw;
             for(int j=i*delta; j<(i+1)*delta; j++)
                 x[j] = oplus(x[j], p[i-1]);
             return;
         }
```

the first partition does not need to be updated . . .

```
In [ ]: // phase2(w,p,0,oplus,2);
```

the rest of the partitions need to be updated

```
In [38]: phase2(q,p,1,oplus,2);

In [39]: printv(q);
```

```
1 3 6 10 15 21 28 36
```

Now we can arrange things in parallel. The naif solution creates and join threads for both phases:

```
In [ ]: auto f =  [] (int x) { cout << x << endl; return; };
```

```
In [ ]: void foo()
        {
            // simulate expensive operation
            std::this_thread::sleep_for(std::chrono::seconds(1));
            cout << "ciao" << endl;
        }
```

```
In [ ]: #include <thread>
        std::thread helper2(foo);
```

```
In [ ]: helper2.join();
```

```
In [40]: void compute_prefix(vector<int> &x, function<int(int,int)> oplus, int nw) {

             vector<thread*> tid(nw);
             vector<int> p(nw);

             // first create threads for phase 1
             for(int i=0; i<nw; i++)
                 tid[i] = new thread(phase1, ref(x), ref(p), i, oplus, nw);
             for(int i=0; i<nw; i++)
                 tid[i]->join();

             // then compute prefix of prefixes
             // prefix(p,ref(oplus));
             for(int i=1; i<nw; i++)
                 p[i] = oplus(p[i-1],p[i]);

             // then threads for phase 2
             for(int i=0; i<nw; i++)
                 tid[i] = new thread(phase2, ref(x), ref(p), i, oplus, nw);
             for(int i=0; i<nw; i++)
                 tid[i]->join();
             return;
          }
```

```
In [41]: vector<int> a(8,1);
```

```
In [42]: printv(a);
```

```
1 1 1 1 1 1 1 1
```

```
In [43]: compute_prefix(a,oplus,2);
In [44]: printv(a);
```

```
1 2 3 4 5 6 7 8
```

this works, let's add some timing . . .

```
In [45]: void compute_prefix_timed(vector<int> &x, function<int(int,int)> oplus, int nw) {

             vector<thread*> tid(nw);
             vector<int> p(nw);

             {
                 utimer t0("all");

                 // first create threads for phase 1
                 {
                     utimer t1("phase1");
                     for(int i=0; i<nw; i++)
                         tid[i] = new thread(phase1, ref(x), ref(p), i, oplus, nw);
                     for(int i=0; i<nw; i++)
                         tid[i]->join();
                 }

             // then compute prefix of prefixes
             // prefix(p,ref(oplus));
                 {
                     utimer t1("phase2");
                     for(int i=1; i<nw; i++)
                         p[i] = oplus(p[i-1],p[i]);
                 }

             // then threads for phase 2
                 {
                     utimer t1("Phase3");
                     for(int i=0; i<nw; i++)
                         tid[i] = new thread(phase2, ref(x), ref(p), i, oplus, nw);
                     for(int i=0; i<nw; i++)
                         tid[i]->join();
                 }
             }
             return;
         }
```

```
In [46]: vector<int> b(1024,1);
```

```
In [47]: printv(b);
```

```
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
```

```
In [48]: compute_prefix_timed(b,oplus,4);

phase1 computed in 718 usec
phase2 computed in 5 usec
Phase3 computed in 1341 usec
all computed in 2479 usec


In [ ]: vector<int> c(1024,1);

In [ ]: compute_prefix_timed(b,oplus,1);

In [ ]: vector<int> a1(1024,1);

In [ ]: vector<int> a2(1024,1);

In [ ]: compute_prefix_timed(a1,oplus2,4);

In [ ]: compute_prefix_timed(a2,oplus2,1);

In [ ]: {
            utimer("seq");

            vector<int> a1(1024,1);
            for(int i=1; i<1024; i++)
                a1[i] = oplus2(x[i-1],x[i]);

        }

In [ ]: cout <<
            oplus2(1,2);
```