

Pragmatic Introduction to Intel's Threading Building Blocks

Example 1

A simple cout of a for loop!

Listing 1: **Serial**

```
1 for(int i = 0; i < nMax; i++){  
2     std::cout << "iteration_" << i << std::endl;  
3 }
```

Listing 2: **Parallel**

```
1 struct NumPrinter  
2 {  
3     void operator()(const tbb::blocked_range<size_t>& r) const {  
4         tbb::blocked_range<size_t>::const_iterator rangeItr = r.begin();  
5         tbb::blocked_range<size_t>::const_iterator rangeEnd = r.end();  
6  
7         for (; rangeItr != rangeEnd; ++rangeItr)  
8         {  
9             std::cout << "iteration_" << rangeItr << std::endl;  
10        }  
11    }  
12 };  
13  
14 int main(int argc, char* argv[])  
15 {  
16  
17     //...  
18     int grainsize = nIterations/nThreads;  
19     tbb::parallel_for(tbb::blocked_range<size_t>(0,  
20                                     nIterations,  
21                                     grainsize),  
22                     NumPrinter()  
23                     );  
24     //...  
25 }
```

Example 2

A custom random number generator!

Listing 3: **Calling Serial and Parallel Code**

```
1 int main(int argc, char* argv[])
2 {
3     //...
4     if(nThreads==1){
5         std::cout << "serial_version_" << std::endl;
6         serialRandom(data);
7         std::cout << "sum_is_" << serialSum(data) << std::endl;
8     }
9     else{
10        ////////////////////////////////////
11        //DO PARALLEL FOR
12        int grainsize = (nIterations/nThreads);
13        std::cout << "parallel_version_with_" << nThreads
14                    << "_threads,_grainsize_"<< grainsize << std::endl;
15
16        Random rndWorker(data);
17        tbb::parallel_for(tbb::blocked_range<size_t>(0,
18                                                    nIterations,
19                                                    grainsize),
20                        rndWorker);
21
22        ////////////////////////////////////
23        //DO PARALLEL REDUCE
24        Sum sumWorker(data);
25        tbb::parallel_reduce(tbb::blocked_range<size_t>(0,nIterations,
26                                                    grainsize),
27                            sumWorker);
28        std::cout << "sum_is_" << sumWorker.result << std::endl;
29    }
30    //...
31 }
```

Listing 4: **Serial**

```

1 void serialRandom(tbb::concurrent_vector<int>* _container){
2
3     boost::random::mt19937_64 rng(42);
4     boost::random::uniform_int_distribution< > uni_dist(1,6);
5     boost::variate_generator<boost::random::mt19937_64&,
6                             boost::random::uniform_int_distribution< >
7                             > six(rng, uni_dist);
8
9     for(int index = 0; index<_container->size(); index++)
10         _container->at(index) = six();
11
12 }
13
14 int serialSum(tbb::concurrent_vector<int>* _container){
15     tbb::concurrent_vector<int>::const_iterator contItr = _container->begin();
16     tbb::concurrent_vector<int>::const_iterator contEnd = _container->end();
17
18     int value = 0;
19
20     for(; contItr!=contEnd;++contItr)
21         value += *contItr;
22
23     return value;
24 }

```

Listing 5: Random Class

```

1  class Random
2  {
3      tbb::concurrent_vector<int> * const inputData;
4
5
6  public:
7      //constructor
8      Random( tbb::concurrent_vector<int>* _data ) :
9          inputData(_data)
10     {}
11
12     void operator()( const tbb::blocked_range<size_t>& r) const {
13
14         tbb::blocked_range<size_t>::const_iterator rangeItr = r.begin();
15         tbb::blocked_range<size_t>::const_iterator rangeEnd = r.end();
16
17         boost::random::mt19937_64 rng(42);
18         boost::random::uniform_int_distribution<> uni_dist(1,6);
19         boost::variate_generator<boost::random::mt19937_64&,
20                                 boost::random::uniform_int_distribution<>
21                                 > six(rng, uni_dist);
22
23
24         for (;rangeItr!=rangeEnd; ++rangeItr)
25         {
26             inputData->at(rangeItr) = six();
27         }
28     }
29 }
30
31 };

```

Listing 6: **Sum Class**

```

1 class Sum
2 {
3     tbb::concurrent_vector<int>* inputData;
4
5 public:
6     //constructor
7     Sum( tbb::concurrent_vector<int>* _data ) :
8         inputData(_data),
9         result(0.)
10    {}
11
12    //copy constructor
13    Sum( const Sum& _rhs ) :
14        inputData(_rhs.inputData),
15        result(0.)
16    {}
17
18    //special Copy constructor
19    Sum( const Sum& _rhs, tbb::split ) :
20        inputData(_rhs.inputData),
21        result(0.)
22    {}
23
24    void join(const Sum& _other){
25        result += _other.result;
26    }
27
28    void operator()(const tbb::blocked_range<size_t>& r) {
29
30        int thisSum = result;
31
32        tbb::blocked_range<size_t>::const_iterator rangeItr = r.begin();
33        tbb::blocked_range<size_t>::const_iterator rangeEnd = r.end();
34        for (; rangeItr!=rangeEnd; ++rangeItr)
35        {
36            thisSum+=inputData->at( rangeItr );
37        }
38
39        result = thisSum;
40    }
41
42    int result;
43
44 };

```