Pragmatic Introduction to Intel's Threading Building Blocks

Example 1

A simple cout of a for loop!

Listing 1: **Serial**

```
for(int i = 0;i<nMax;i++){
  std::cout << "iteration "<< i << std::endl;
}
```

Listing 2: **Parallel**

```
struct NumPrinter
{
   void operator()(const tbb::blocked_range<size_t>& r) const {
    tbb::blocked_range<size_t >::const_iterator rangeItr = r.begin();
    tbb::blocked_range<size_t >::const_iterator rangeEnd = r.end();

    for (;rangeItr!=rangeEnd; ++rangeItr)
    {
      std::cout << "iteration "<< rangeItr << std::endl;
    }
  }
};

int main(int argc, char* argv[])
{

  //...
  int grainsize = nIterations/nThreads;
  tbb::parallel_for(tbb::blocked_range<size_t >(0,
                                                   nIterations,
                                                   grainsize),
                    NumPrinter()
                    );
  //...
}
```

Example 2

A custom random number generator!

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

```
int main(int argc, char* argv[])
{
  //...
  if(nThreads==1){
      std::cout << "serial version " << std::endl;
      serialRandom(data);
      std::cout << "sum is " << serialSum(data) << std::endl;
  }
  else{
    ////////////////////////////////////////////////////////////////
    //DO PARALLEL FOR
    int grainsize = (nIterations/nThreads);
    std::cout << "parallel version with " << nThreads
              << " threads, grainsize "<< grainsize << std::endl;

    Random rndWorker(data);
    tbb::parallel_for(tbb::blocked_range<size_t>(0,
                                                  nIterations,
                                                  grainsize),
                      rndWorker);

    ////////////////////////////////////////////////////////////////////
    //DO PARALLEL REDUCE
    Sum sumWorker(data);
    tbb::parallel_reduce(tbb::blocked_range<size_t>(0,nIterations,
                                                      grainsize),
                         sumWorker);
    std::cout << "sum is " << sumWorker.result << std::endl;
  }
  //...
}
```

Listing 4: **Serial**

```
void serialRandom(tbb::concurrent_vector<int>* _container){

  boost::random::mt19937_64 rng(42);
  boost::random::uniform_int_distribution<> uni_dist(1,6);
  boost::variate_generator<boost::random::mt19937_64&,
                           boost::random::uniform_int_distribution<>
                          > six(rng, uni_dist);

  for(int index = 0;index<_container->size();index++)
    _container->at(index) = six();

}

int serialSum(tbb::concurrent_vector<int>* _container){
  tbb::concurrent_vector<int>::const_iterator contItr = _container->begin();
  tbb::concurrent_vector<int>::const_iterator contEnd = _container->end();

  int value = 0;

  for(;contItr!=contEnd;++contItr)
    value += *contItr;

  return value;
}
```

Listing 5: **Random Class**

```
class Random
{
  tbb::concurrent_vector<int> * const inputData;


public:
    //constructor
  Random( tbb::concurrent_vector<int>* _data ) :
    inputData(_data)
  {}

  void operator()(const tbb::blocked_range<size_t>& r) const {

    tbb::blocked_range<size_t >::const_iterator rangeItr = r.begin();
    tbb::blocked_range<size_t >::const_iterator rangeEnd = r.end();

    boost::random::mt19937_64 rng(42);
    boost::random::uniform_int_distribution<> uni_dist(1,6);
    boost::variate_generator<boost::random::mt19937_64&,
                              boost::random::uniform_int_distribution<>
                             > six(rng, uni_dist);


    for (;rangeItr!=rangeEnd; ++rangeItr)
     {
       inputData->at(rangeItr) = six();
     }

  }

};
```

Listing 6: **Sum Class**

```
class Sum
{
  tbb::concurrent_vector<int>* inputData;

public:
  //constructor
  Sum( tbb::concurrent_vector<int>* _data ) :
    inputData(_data),
    result(0.)
  {}

  //copy constructor
  Sum( const Sum& _rhs ) :
    inputData(_rhs.inputData),
    result(0.)
  {}

  //special Copy constructor
  Sum( const Sum& _rhs, tbb::split ) :
    inputData(_rhs.inputData),
    result(0.)
  {}

  void join(const Sum& _other){
    result += _other.result;
  }

  void operator()(const tbb::blocked_range<size_t>& r)  {

    int thisSum = result;

    tbb::blocked_range<size_t >::const_iterator rangeItr = r.begin();
    tbb::blocked_range<size_t >::const_iterator rangeEnd = r.end();
    for (;rangeItr!=rangeEnd; ++rangeItr)
     {
       thisSum+=inputData->at(rangeItr);
     }

    result = thisSum;
  }

  int result;

};
```