Advanced Tools For Parallel Programming

Marwan Burelle

Threading Building Blocks

Looping in parallel

Pipeline

Containers

Other tools

Advanced Tools For Parallel Programming

Marwan Burelle

marwan.burelle@lse.epita.fr
http://wiki-prog.kh405.net

Outline

1. Threading Building Blocks
2. Looping in parallel
   - Ranges
   - Data Partitioner
   - Parallel For
   - Parallel Reduce
   - Other kinds of loops
3. Pipeline
4. Containers
   - Concurrent Hash Map
   - Concurrent Queue
5. Other tools

Advanced Tools For Parallel Programming

Marwan Burelle

Threading Building Blocks

Looping in parallel

Pipeline

Containers

Other tools

Threading Building Blocks

TBB ?

- TBB is a librairy designed to provide high level constructions for parallel programming.
- Initialy TBB was an Intel project to demonstrate gains of multi-core processors.
- It provides algorithms (such as *parallel for*), containers, tools and underlying framework for parallel computing.
- It is a pure C++ librairy with templates classes. No additional support is required from the compiler (unlike OpenMP.)

Contents

- Bounded Parallel Iterators: *parallel for* and *parallel reduce*
- Dynamic Parallel Loop: *parallel do*
- Pipeline
- Spawing and continuation based tasks system
- Containers: *queues*, *vectors* and *hash tables*
- Atomic Types
- Various locks
- Threading API *compatible* with requirements for C++11
- Various utilities: *wall clock*, *memory allocators* . . .

Advanced Tools For Parallel Programming

Marwan Burelle

Threading Building Blocks

Looping in parallel

Ranges

Data Partitioner

Parallel For

Parallel Reduce

Other kinds of loops

Pipeline

Containers

Other tools

Looping in parallel

Parallel for and reduce

Advanced Tools For Parallel Programming

Marwan Burelle

Threading Building Blocks

Looping in parallel

Ranges

Data Partitioner

Parallel For

Parallel Reduce

Other kinds of loops

Pipeline

Containers

Other tools

- Parallel for and reduce provides the simplest but probably the most usefull tools of TBB.
- They provide a simple way to achieve a good data-driven decomposition.
- Optimal data decomposition can be achieved almost *automatically* through the concept of range and *partitioner*.
- Using C++11's lambda (anonymous functions) writing a simple parallel for can be as simple as writing a for loop.

Advanced Tools For Parallel Programming

Marwan Burelle

Threading Building Blocks

Looping in parallel

Ranges

Data Partitioner

Parallel For

Parallel Reduce

Other kinds of loops

Pipeline

Containers

Other tools

Ranges

What are ranges ?

- A range a is a concept derived from iterator;
- It is used by parallel loops to *describe* inner step of linear blocks in the parallel iterations.
- It is also used to define partitions in data-set.
- The way it divides data (and the minimal grain of its division) drives the scheduling of the parallel iterations.

Advanced Tools For Parallel Programming
Marwan Burelle
Threading Building Blocks
Looping in parallel
Ranges
Data Partitioner
Parallel For
Parallel Reduce
Other kinds of loops
Pipeline
Containers
Other tools

What a range provides ?

- As an iterator, a range provides the usual iteration operations:
  - `begin()` and `end()` (returning *const iterator*)
  - `size()`
- In order to divide the data-set, the range provides the following opertations:
  - `grainsize()` (minimal size of a sub-range)
  - `is_divisible()` (whether we can split the actual range or not)
  - And *split* constructor (almost like a copy constructor but with a dummy split argument to differenciate it from a copy constructor.)

Advanced Tools For Parallel Programming
Marwan Burelle
Threading Building Blocks
Looping in parallel
Ranges
Data Partitioner
Parallel For
Parallel Reduce
Other kinds of loops
Pipeline
Containers
Other tools

A Basic Integer Range

```
struct IntRange {
  int lower; int upper;
  bool empty() const {return lower==upper;}

  bool is_divisible() const
  { return upper>lower+1; }

  IntRange(IntRange& r, split) {
    int m = (r.lower+r.upper)/2;
    lower = m;
    upper = r.upper;
    r.upper = m;
  }
};
```


Threading Building Blocks

Looping in parallel

Ranges

Data Partitioner

Parallel For

Parallel Reduce

Other kinds of loops

Pipeline

Containers

Other tools

Provided Ranges

- TBB provides *blocked range* templates useful for most cases.
- The templates can be used with any integral type convertible to `size_t`
- *Blocked ranges* comes in 3 flavors:
  - `blocked_range` for half-open interval;
  - `blocked_range2d` for two dimensional ranges;
  - `blocked_range3d` for three dimesional ranges.

Advanced Tools For Parallel Programming
Marwan Burelle
Threading Building Blocks
Looping in parallel
Ranges
Data Partitioner
Parallel For
Parallel Reduce
Other kinds of loops
Pipeline
Containers
Other tools

Advanced Tools For Parallel Programming

Marwan Burelle

Threading Building Blocks

Looping in parallel

Ranges

Data Partitioner

Parallel For

Parallel Reduce

Other kinds of loops

Pipeline

Containers

Other tools

Data Partitioner

Partitioner concept

- A partitioner specifies how a parallel loop should partition its works among threads.
- Parallel loops try to recursively split a range in order to keep processors busy.
- Partitioners control the *split politics*
- TBB provides three partitioners:
  - `auto_partitioner`: the default behavior
  - `affinity_partitioner`: similar to auto but tries to be nice with cache
  - `simple_partitioner`: divides ranges until `is_divisible` return false.

Advanced Tools For Parallel Programming
Marwan Burelle
Threading Building Blocks
Looping in parallel
Ranges
Data Partitioner
Parallel For
Parallel Reduce
Other kinds of loops
Pipeline
Containers
Other tools

Choosing a partitioner

- In most cases, the automatic partitioner provides the best behavior.
- The affinity partitioner is used when data-set fits in cache and loop may be executed on the same data.
- The affinity partitioner may improve performances in some cases but need careful adjustement.
- The simple partitioner give you the full control over partitioning.
- You should use simple partitioner only if you have a clear idea on how data should be split.

Advanced Tools For Parallel Programming
Marwan Burelle
Threading Building Blocks
Looping in parallel
Ranges
Data Partitioner
Parallel For
Parallel Reduce
Other kinds of loops
Pipeline
Containers
Other tools

Advanced Tools For Parallel Programming

Marwan Burelle

Threading Building Blocks

Looping in parallel

Ranges

Data Partitioner

Parallel For

Parallel Reduce

Other kinds of loops

Pipeline

Containers

Other tools

Parallel For

Bounded Parallel Iterations

Advanced Tools For Parallel Programming

Marwan Burelle

Threading Building Blocks

Looping in parallel

Ranges

Data Partitioner

Parallel For

Parallel Reduce

Other kinds of loops

Pipeline

Containers

Other tools

- A *parallel for* loop is a traditional *for* loop that TBB will execute in parallel.
- The loop iterate on a range and use the *splittable* concept of range to divide tasks on physical threads.
- A *parallel for* loop performs independent tasks on a set of data with result recollection at the end.
- The template provides by TBB can be used with functor objects or C++11's lambdas.

Using A Parallel For

Advanced Tools For Parallel Programming

Marwan Burelle

Threading Building Blocks

Looping in parallel

Ranges

Data Partitioner

Parallel For

Parallel Reduce

Other kinds of loops

Pipeline

Containers

Other tools

We will suppose we have an operation `F(e)` operating on a single data (`double` in our example.) Our data-set is an array. The linear version will look like:

```
void Serial(double data[], size_t n)
{
  for (size_t i=0; i != n; ++i)
    F(data[i]);
}
```

Using A Parallel For (simple functor)

```
class Parallel {
  double *const my_data;
public:
  void operator()
    (const blocked_range<size_t>& r) const
  {
    double *a = my_data; // local copy
    for (size_t i=r.begin();i!=r.end();++i)
      F(a[i]);
  }
  Parallel (double a[]) : my_data(a) {}
};

void ParallelRun(double data[], size_t n) {
  parallel_for(blocked_range<size_t>(0,n),
               Parallel(data));
}
```


Advanced Tools For Parallel Programming

Marwan Burelle

Threading Building Blocks

Looping in parallel

Ranges

Data Partitioner

Parallel For

Parallel Reduce

Other kinds of loops

Pipeline

Containers

Other tools

More possibilities

- You can use a lambda rather than a functor (more *compact* solution)
- If you only plans to iterate on an integer range you can just provide the range bound rather than a range object (when using lambda.)
- You can also provide a partitioner.
- You can use parallel for to iterate on STL containers quite easily.

Advanced Tools For Parallel Programming
Marwan Burelle
Threading Building Blocks
Looping in parallel
Ranges
Data Partitioner
Parallel For
Parallel Reduce
Other kinds of loops
Pipeline
Containers
Other tools

How parallel for works

- The parallel for template algorithm relies on a smart scheduling of sub-ranges.
- When starting the parallel for, the initial range is divided among threads.
- Each thread will then split the work depending on partitioner politics
- When a thread has finished its own sub-range, it can steal work from other threads.
- Depending on the choosen partitioner and the time spent in each blocks, TBB will try to keep each available threads occupied.

Advanced Tools For Parallel Programming

Marwan Burelle

Threading Building Blocks

Looping in parallel

Ranges

Data Partitioner

Parallel For

Parallel Reduce

Other kinds of loops

Pipeline

Containers

Other tools

Parallel Reduce

Parallel Reduce

Advanced Tools For Parallel Programming

Marwan Burelle

Threading Building Blocks

Looping in parallel

Ranges

Data Partitioner

Parallel For

Parallel Reduce

Other kinds of loops

Pipeline

Containers

Other tools

- *Parallel reduce* is a variation of the *parallel for* used when data recollection is needed.
- Globaly it provides an efficient way to share a kind of accumulator to the loop.
- As for *parallel for*, operations on data should be reflexive.
- The way *parallel reduce* works avoid the need of a locked and shared accumulator.
- The functor provided to the loop must offer a split constructor and join operations.

Using Parallel Reduce

Advanced Tools For Parallel Programming

Marwan Burelle

Threading Building Blocks

Looping in parallel

Ranges

Data Partitioner

Parallel For

Parallel Reduce

Other kinds of loops

Pipeline

Containers

Other tools

In this example we will compute the sum of a vector, the sequential version look like:

```
double Sum(double data[], size_t n)
{
  double res = 0;
  for (size_t i=0; i != n; ++i)
    res += data[i];
  return res;
}
```

Using Parallel Reduce

```
class SumWorker {
  double* my_data;
public:
  double* my_sum;
  void operator() (const blocked_range<size_t>& r) {
    double *a = my_data; double sum = my_sum;
    for (size_t i=r.begin(); i!=r.end; ++i)
      sum += a[i];
    my_sum = sum;
  }
  void join(const SumWorker& y)
  { my_sum += y.my_sum; }
  SumWorker(SumWorker x, split) :
    my_data(x.my_data), my_sum(0) {}
  SumWorker(double data[]) :
    my_data(data), my_sum(0) {}
};
```


Advanced Tools For Parallel Programming

Marwan Burelle

Threading Building Blocks

Looping in parallel

Ranges

Data Partitioner

Parallel For

Parallel Reduce

Other kinds of loops

Pipeline

Containers

Other tools

Using Parallel Reduce

```
double ParallelSum(double data[], size_t n)
{
  SumWorker sw(data);
  parallel_reduce(
    blocked_range<size_t>(0,n),sw
  );
  return sw.my_sum;
}
```

Advanced Tools For Parallel Programming
Marwan Burelle
Threading Building Blocks
Looping in parallel
Ranges
Data Partitioner
Parallel For
Parallel Reduce
Other kinds of loops
Pipeline
Containers
Other tools

Parallel Reduce Hints

Advanced Tools For Parallel Programming

Marwan Burelle

Threading Building Blocks

Looping in parallel

Ranges

Data Partitioner

Parallel For

Parallel Reduce

Other kinds of loops

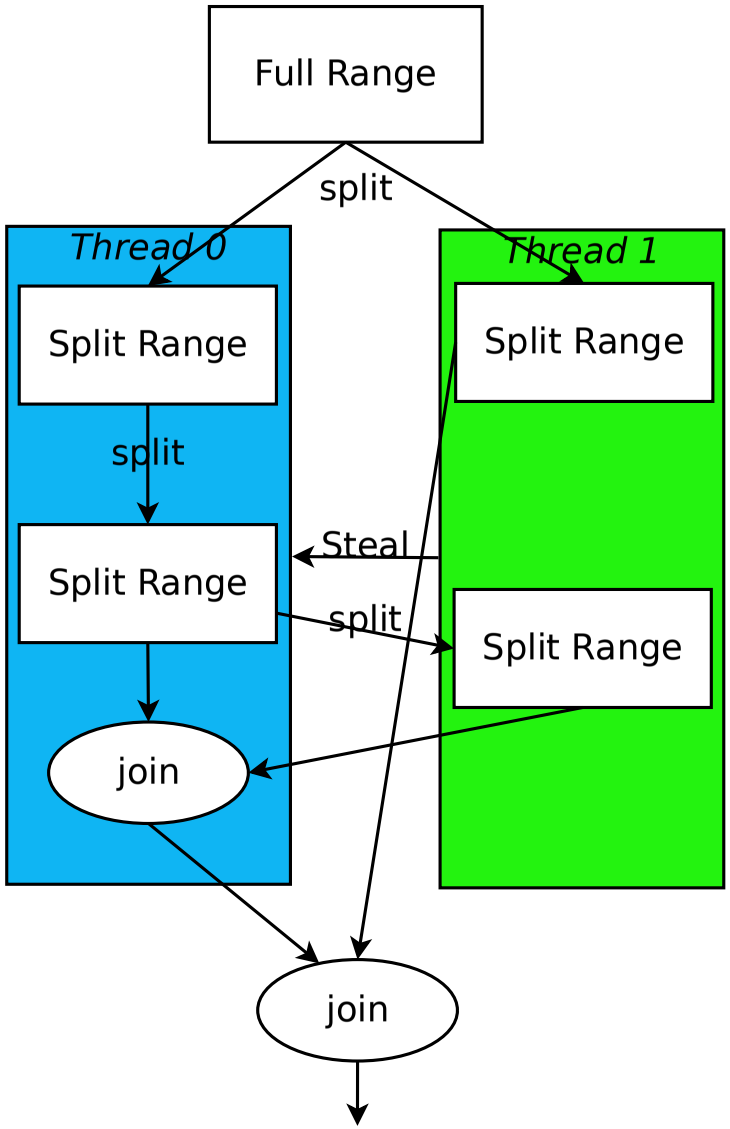
Pipeline

Containers

Other tools

- You have no control over operations order, thus operations must be reflexive.
- The `join` operation is only performed when a range was split in order to transfer works to an other thread.
- Using local copy of variable is strongly advised: local variables will probably be moved to register will object attributes should require indirect access and address computations.
- Like the *parallel for*, you can control data-partitioning with range definitions and partitioners.

How Parallel Reduce Works



Advanced Tools For Parallel Programming
Marwan Burelle
Threading Building Blocks
Looping in parallel
Ranges
Data Partitioner
Parallel For
Parallel Reduce
Other kinds of loops
Pipeline
Containers
Other tools

Advanced Tools For Parallel Programming

Marwan Burelle

Threading Building Blocks

Looping in parallel

Ranges

Data Partitioner

Parallel For

Parallel Reduce

Other kinds of loops

Pipeline

Containers

Other tools

Other kinds of loops

More loops

- `parallel_do`: unbounded loops, where new items can be added dynamically during iterations.
- `parallel_scan`: a complex for of loop with inner sequential dependencies.

Advanced Tools For Parallel Programming
Marwan Burelle
Threading Building Blocks
Looping in parallel
Ranges
Data Partitioner
Parallel For
Parallel Reduce
Other kinds of loops
Pipeline
Containers
Other tools

Advanced Tools For Parallel Programming

Marwan Burelle

Threading Building Blocks

Looping in parallel

Pipeline

Containers

Other tools

Pipeline

The assembly line

- TBB provides a framework to build pipeline of tasks.
- A pipeline is composed by a set of sequential tasks and flow of data that traverse these tasks.
- Each task will receive chunks of data from the previous task and send the resulting chunks to the next task.
- When a task receive several chunks, it can process them in parallel or sequentially.
- When a task process chunks in parallel, results can be emitted in FIFO order or *asynchronously* as processing ends.
- Pipeline provides a simple and efficient way of doing data-flow driven partitioning.

Filters

Advanced Tools For Parallel Programming

Marwan Burelle

Threading Building Blocks

Looping in parallel

Pipeline

Containers

Other tools

- Filters are tasks that can be added to a pipeline
- A filter is basically a functor providing information for the pipeline organisation (sequential or parallel computation, FIFO outputs . . .)
- TBB provides a template to build a filter out of simple functor.
- Filers can build and added in the pipeline rather at pipeline creation (using `operator&` to concatenate filters) or added afterward by using the `add_filter` method.

Pipeline

Advanced Tools For Parallel Programming

Marwan Burelle

Threading Building Blocks

Looping in parallel

Pipeline

Containers

Other tools

- Once filters are designed, you can construct your pipeline and add filters in order.
- When using parallel filter, you must take care of the number of *tokens* sends to filter: each token triggers invocation of an instance of the filters, if the next filter in line is serial, the parallel filter may start to much tasks. Constructor for pipeline provide a control over the maximum of living tokens in the pipeline.
- A pipeline does not provides a non-linear structures: each filter as exactly one input (or none if first) and one ouput (or none if last.)

Building A Pipeline

1. Build your filters by derivating class `filter`;
2. Override `operator()` to perform operation on item. You must take a pointer to the current item and return a pointer for the next filter. Last filter's output is ignored;
3. The first filter is a special case: it generates the stream for the chain and returns `NULL` to indicate the end of the stream;
4. Create an instance of class `pipeline`;
5. Create instances of your filters and add them to the pipline in order from first to last. Each instance can be added at most once to a pipline and should never be a member of more than one pipline at a time.
6. Call method `pipeline::run` and set carefully the maximum live tokens to avoid too much memory usage.

Skeleton Example (Simple Interface)

```
struct StartFilter {
public:
  void* operator()(InType* x)
  { /* do the job here */ }
};
struct MiddleFilter {
public:
  Type2* operator()(Type1* x)
  { /* do the job here */ }
};
struct LastFilter {
public:
  void* operator()(Type2* x)
  { /* do the job here */ }
};
```

Skeleton Example

```
void MyPipeline(int maxToken, /* other data */)
{
  tbb::t_filter<void,Type1>
    fInput(tbb::serial_in_order,StartFilter());
  tbb::t_filter<Type1,Type2>
    fMiddle(tbb::parallel,MiddleFilter());
  tbb::t_filter<Type2,void>
    fOutput(tbb::serial_in_order,LastFilter());
  tbb::parallel_pipeline(maxToken,
       fInput & fMiddle & fOutput
  );
}
```

Advanced Tools For Parallel Programming

Marwan Burelle

Threading Building Blocks

Looping in parallel

Pipeline

Containers

Concurrent Hash Map

Concurrent Queue

Other tools

Containers

Concurrent Containers

Advanced Tools For Parallel Programming

Marwan Burelle

Threading Building Blocks

Looping in parallel

Pipeline

Containers

Concurrent Hash Map

Concurrent Queue

Other tools

- TBB provides some containers *parallel friendly*
- All containers are safe to be used in multi-threaded context (even when using system threads.)
- All TBB's containers are not only safe, but also aime to have finest grain locking or (when possible) non-blocking policy.
- Latest TBB version include containers similar to C++11 concurrent containers.

Advanced Tools For Parallel Programming

Marwan Burelle

Threading Building Blocks

Looping in parallel

Pipeline

Containers

Concurrent Hash Map

Concurrent Queue

Other tools

Concurrent Hash Map

Concurrent Hash Map

- The template class `concurrent_hash_map` provides associative maps (backed with hash table.)
- Concurrent Hash Map provides a *Readers/Writer* policy at element level
- Operations can be performed concurrently (even element removal)
- The map stores `std::pair<const Key,T>` and use a notion of *accessor* to indicate access for reading or for writing.
- The `Key` template parameter must respect the `HashCompare` concept.

HashCompare

Advanced Tools For Parallel Programming

Marwan Burelle

Threading Building Blocks

Looping in parallel

Pipeline

Containers

Concurrent Hash Map

Concurrent Queue

Other tools

- `HashCompare` concept defines in the same object the hash function and equality predicate.
- It must satisfies usual constraints:
  - Two equals key must have the same hash code
  - Hash code of keys must not change when keys are in used.
- You should also verify that hash function and equality predicate must not raise exceptions.

Usage

- When retrieve a pair you should provide an accessor.
- The item is locked (for reading or writing) until the accessor is deleted.
- The type of the accessor indicate the kind of operation you want (reading or writing): `accessor` or `const_accessor`

```
{
  MyTable::accessor a; // lock for writing
  table.insert(a,key); // add or find if key exists
  a->second = new_val;
  // when leaving the block, a is destroyed
}
```

Advanced Tools For Parallel Programming
Marwan Burelle
Threading Building Blocks
Looping in parallel
Pipeline
Containers
Concurrent Hash Map
Concurrent Queue
Other tools

Advanced Tools For Parallel Programming

Marwan Burelle

Threading Building Blocks

Looping in parallel

Pipeline

Containers

Concurrent Hash Map

Concurrent Queue

Other tools

Concurrent Queue

Concurrent Queues

Advanced Tools For Parallel Programming

Marwan Burelle

Threading Building Blocks

Looping in parallel

Pipeline

Containers

Concurrent Hash Map

Concurrent Queue

Other tools

- TBB provide non-blocking queues (lock free) and blocking queues (locked and possibly bounded.)
- The class template `concurrent_queue<T,Alloc>` provides unbounded lock-free concurrent FIFO queues of elements of type `T`
- The class template `concurrent_bounded_queue<T,Alloc>` provides blocking FIFO queues, possibly bounded.
- All queues behave like STL queues.

Advanced Tools For Parallel Programming

Marwan Burelle

Threading Building Blocks

Looping in parallel

Pipeline

Containers

Other tools

Other tools

Locks

Advanced Tools For Parallel Programming

Marwan Burelle

Threading Building Blocks

Looping in parallel

Pipeline

Containers

Other tools

- TBB provides a wide variety of locks
- Locks in TBB are C++ friendly:
  - You instantiate a local special object (from the shared lock)
  - Once you acquire the lock, it will remains until the object exists
  - The lock is release when you destroy the object

Atomic Types

- TBB provides template types for atomic values
- Atomic types in TBB is very similar to C++11 atomic types
- Atomic types provides a set of operations that are safe to use concurrently
- Since atomic types are bounded to integer like types, TBB override usual operators:
  - ++ and --
  - += and other *op-assign* operators
  - = assign operations

Wall clock

Advanced Tools For Parallel Programming

Marwan Burelle

Threading Building Blocks

Looping in parallel

Pipeline

Containers

Other tools

New C++11 features in `std::chrono`

- TBB provides a working wall clock for performances measures
- Wall clocks are safe to use in concurrent environments and offer some garantee on the time information they deliver.
- Precision is as accurate as system clock (often few milliseconds.)

Tasks Based Programming

Advanced Tools For Parallel Programming

Marwan Burelle

Threading Building Blocks

Looping in parallel

Pipeline

Containers

Other tools

- TBB uses as its core a tasks system.
- You can use it directly.
- Task are single small computation.
- Each task can spawn new tasks and wait (or not) for it.
- You can also describe your tasks in continuation style: each task can return a new task to be performed.
- Building a simple tasks based program:
  - You describe your tasks with class derived from the class `task`
  - Your first task will spawn other tasks
  - You launch it (using `task::spawn_root_and_wait`)
  - That's (almost) all !

Task Example

```
class FibTask: public task {
public:
 const long n; long* const sum;
 FibTask( long n_, long* sum_ ) :
   n(n_), sum(sum_) {}
 task* execute() {
   long x, y;
   FibTask& a =
     *new(allocate_child()) FibTask(n-1,&x);
   FibTask& b =
     *new(allocate_child()) FibTask(n-2,&y);
   set_ref_count(3);
   spawn( b );
   spawn_and_wait_for_all(a);
   *sum = x+y;
   return NULL;
 }
};
```

Task Example

```
long ParallelFib( long n ) {
  long sum;
  FibTask& a =
    *new(task::allocate_root()) FibTask(n,&sum);
  task::spawn_root_and_wait(a);
  return sum;
}
```

More . . .

Advanced Tools For Parallel Programming

Marwan Burelle

Threading Building Blocks

Looping in parallel

Pipeline

Containers

Other tools

- TBB provides efficient allocator to be used instead of standard allocators.
- Exceptions and correct cancellation operations.
- Read the docs !