Threading Building Block (TBB)

João Miguel Domingues Pedrosa

June 8, 2016

Introduction

TBB, **T**hreading **B**uilding **B**lock

- Librairie C++ Intel open source
- Programmation parallèle
- Allocateur de mémoire optimisé

Installation

Ubuntu ou Debian Via gestionnaire de package aptitude

- sudo apt-get install libtbb-dev libtbb2

Via archive

- Récupérer archive sur le site
- Extraire l'archive
- Exécuter script tbbvar.sh dans le répertoire bin
- Installation terminer

Compilation

Compilation, ajout de l'option -ltbb afin de linker la librairie:

```
g++ file.cpp -o bin -ltbb
```

Algorithme

Algorithme:

- `parallel_do`
- `parallel_for`
- `parallel_for_each`
- `parallel_reduce`
- `parallel_sort`
- `parallel_invoke`

Parallel for

```
1  template<typename Index, typename Func>
2  Func parallel_for( Index first, Index_type last, const
       Func& f [, partitioner[, task_group_context&
     group]] );
3
4  template<typename Index, typename Func>
5  Func parallel_for( Index first, Index_type last, Index
       step, const Func& f [, partitioner[,
     task_group_context& group]] );
6
7  template<typename Range, typename Body>
8  void parallel_for( const Range& range, const Body&
     body, [, partitioner[, task_group_context& group]]
       );
```

Parallel invoke

```
1 template<typename Func0, typename Func1>
2 void parallel_invoke(const Func0& f0,const Func1& f1);
3
4 template<typename Func0, typename Func1, typename
      Func2>
5 void parallel_invoke(const Func0& f0, const Func1& f1,
       const Func2& f2);
6
7 template<typename Func0, typename Func1, ..., typename
       Func9>
8 void parallel_invoke(const Func0& f0, const Func1& f1,
       ..., const Func9& f9);
```

Blocked range

```
1   template<typename Value>
2   class blocked_range {
3   public:
4       // constructors
5       blocked_range( Value begin, Value end,
6                      size_type grainsize=1 );
7       ...
8       // capacity
9       size_type size() const;
10      bool empty() const;
11      ...
12
13      // iterators
14      const_iterator begin() const;
15      const_iterator end() const;
16  };
```

Container

- Standard container
  - vector
  - hash_map
  - queue
  - ...

Thread local Storage

```
1   template <typename T>
2   class combinable {
3   public:
4       combinable();
5       template <typename FInit>
6       combinable(FInit finit);
7       ...
8       void clear();
9
10      T& local();
11      T& local(bool & exists);
12
13      template<typename FCombine> T combine(FCombine
            fcombine);
14      template<typename Func> void combine_each(Func
            f);
15  };
```

Démo

Conclusion

Question