More Libraries

Another important library-based solution is the *Message Passing Interface* (MPI) and we shall look at this later when we talk about distributed memory systems.
Another important library-based solution is the *Message Passing Interface* (MPI) and we shall look at this later when we talk about distributed memory systems.

We shall just note here that MPI is an example of one library-based technique that is quite popular: write code that is sequential, or modestly parallel, but call library functions that are parallel—and written by somebody else.
Another important library-based solution is the *Message Passing Interface* (MPI) and we shall look at this later when we talk about distributed memory systems.

We shall just note here that MPI is an example of one library-based technique that is quite popular: write code that is sequential, or modestly parallel, but call library functions that are parallel—and written by somebody else.

Another example, the *Basic Linear Algebra Subprograms* (BLAS)
The BLAS are a (standard for a) collection of functions that implement various algorithms in linear algebra: vector sums; matrix multiplication; vector dot products; etc. for various representations of these datatypes.
More Libraries

The BLAS are a (standard for a) collection of functions that implement various algorithms in linear algebra: vector sums; matrix multiplication; vector dot products; etc. for various representations of these datatypes.

Implementations are written by people who really understand what they are doing in terms of making the best use of hardware: in particular parallel hardware.
More Libraries

The BLAS are a (standard for a) collection of functions that implement various algorithms in linear algebra: vector sums; matrix multiplication; vector dot products; etc. for various representations of these datatypes.

Implementations are written by people who really understand what they are doing in terms of making the best use of hardware: in particular parallel hardware.

If you write your application to use the BLAS your code will be using this expertise.
More Libraries

The BLAS are a (standard for a) collection of functions that implement various algorithms in linear algebra: vector sums; matrix multiplication; vector dot products; etc. for various representations of these datatypes.

Implementations are written by people who really understand what they are doing in terms of making the best use of hardware: in particular parallel hardware.

If you write your application to use the BLAS your code will be using this expertise.

If someone comes out with an improved implementation of the BLAS that goes twice as fast, your code will automatically go twice as fast (in the BLAS bit).
More Libraries

They really can be a factor of two difference on the same hardware
More Libraries

They really can be a factor of two difference on the same hardware.

BLAS libraries are typically tuned to the version of the processor in your machine, taking into account cache sizes; memory speeds and so on.
More Libraries

They really can be a factor of two difference on the same hardware.

BLAS libraries are typically tuned to the version of the processor in your machine, taking into account cache sizes; memory speeds and so on.

The GotoBLAS, written by Goto, are recognised as being particularly good.
More Libraries

They really can be a factor of two difference on the same hardware

BLAS libraries are typically tuned to the version of the processor in your machine, taking into account cache sizes; memory speeds and so on

The GotoBLAS, written by Goto, are recognised as being particularly good

His implementation contains chunks of processor-specific assembler and pays particular attention to the sizes of blocks of data, matching them carefully to cache sizes
More Libraries

Many other libraries exist: for example, the *template* approach
Many other libraries exist: for example, the *template* approach

This is a standard header file with a library of code behind it that introduces a bunch of new classes to aid parallel computation
More Libraries

Many other libraries exist: for example, the *template* approach

This is a standard header file with a library of code behind it that introduces a bunch of new classes to aid parallel computation

For example, C++ AMP (Accelerated Massive Parallelism) from Microsoft defines some parallel container types with methods that act concurrently on them
More Libraries

Many other libraries exist: for example, the *template* approach

This is a standard header file with a library of code behind it that introduces a bunch of new classes to aid parallel computation

For example, C++ AMP (Accelerated Massive Parallelism) from Microsoft defines some parallel container types with methods that act concurrently on them

E.g., `concurrency::parallel_for_each(...)`
More Libraries

Many other libraries exist: for example, the template approach

This is a standard header file with a library of code behind it that introduces a bunch of new classes to aid parallel computation

For example, C++ AMP (Accelerated Massive Parallelism) from Microsoft defines some parallel container types with methods that act concurrently on them

E.g., concurrency::parallel_for_each(...)

The details are hidden from the programmer, who gets a fairly simple API to work with
More Libraries

There are many other template libraries for C++ (a language very suited to this approach):

- Parallel Patterns Library (PPL) from Microsoft
- Thrust from Nividia
- Intel Threading Building Blocks (TBB)
- Boost
- Etc.
We shall look briefly at Threading Building Blocks (TBB) as it contains some interesting ideas
We shall look briefly at Threading Building Blocks (TBB) as it contains some interesting ideas.

It is a standard C++ template library, needing no specific compiler support.
We shall look briefly at Threading Building Blocks (TBB) as it contains some interesting ideas.

It is a standard C++ template library, needing no specific compiler support.

It provides things like concurrent containers and concurrent operations as well as the usual atomics and synchronisations.
TBB Concurrent Operations

These are things like parallel for loops and parallel reductions (see later)

```cpp
#include <tbb/tbb.h>
#include <iostream>

using namespace tbb;
using namespace std;

void hi(int n) {
    cout << "hello: " << n << endl;
}

int main() {
    parallel_for<int>(0, 10, hi);
    return 0;
}
```
Though you quickly realise you should have written

```cpp
std::mutex m;

void hi(int n) {
    m.lock();
    cout << "hello: " << n << endl;
    m.unlock();
}
```
Though you quickly realise you should have written

```cpp
std::mutex m;

void hi(int n) {
    m.lock();
    cout << "hello: " << n << endl;
    m.unlock();
}
```

But not a single `pthread_create` in sight!
TBB Concurrent Containers

Containers are things like vectors, queues and hash tables
Containers are things like vectors, queues and hash tables

You have to take care over concurrent access to these as pushing value to a stack at the same time as another thread is popping a value is an easy route to races
Containers are things like vectors, queues and hash tables

You have to take care over concurrent access to these as pushing value to a stack at the same time as another thread is popping a value is an easy route to races

Thus TBB provides safe datastructures that the details right (we hope!)
TBB Work Stealing

TBB uses the concept of *work stealing* to manage parallelism.
TBB Work Stealing

TBB uses the concept of *work stealing* to manage parallelism.

In something like a `parallel_for` there are a lot of tasks to be scheduled across the available threads.
TBB Work Stealing

TBB uses the concept of work stealing to manage parallelism.

In something like a parallel for there are a lot of tasks to be scheduled across the available threads.

Each thread has a queue of tasks that are ready to be run.
TBB Work Stealing

TBB uses the concept of *work stealing* to manage parallelism.

In something like a `parallel for` there are a lot of tasks to be scheduled across the available threads.

Each thread has a queue of tasks that are ready to be run.

When a new task is spawned it is put onto the end the current thread’s queue.
TBB Work Stealing

TBB uses the concept of *work stealing* to manage parallelism.

In something like a `parallel_for` there are a lot of tasks to be scheduled across the available threads.

Each thread has a queue of tasks that are ready to be run.

When a new task is spawned it is put onto the end of the current thread’s queue.

When a thread becomes idle (after completing its previous task) it pops a task off the end of its queue and runs that.
TBB Work Stealing

If its queue is empty, the thread *steals* a task off the *start* of another thread’s queue and runs that
TBB Work Stealing

If its queue is empty, the thread *steals* a task off the start of another thread’s queue and runs that

Thus keeping all threads busy as long as there are tasks to do
TBB Work Stealing

If its queue is empty, the thread *steals* a task off the start of another thread’s queue and runs that

Thus keeping all threads busy as long as there are tasks to do

Note that pushing and popping a task off your own queue is a relatively cheap operation, so the overhead is kept small
TBB Work Stealing

If its queue is empty, the thread *steals* a task off the *start* of another thread’s queue and runs that

Thus keeping all threads busy as long as there are tasks to do

Note that pushing and popping a task off your own queue is a relatively cheap operation, so the overhead is kept small

In other words, when there is no opportunity for more parallelism as every thread is already busy doing its own tasks, the overhead over simple sequential execution is minimal

TBB Work Stealing

If its queue is empty, the thread *steals* a task off the *start* of another thread’s queue and runs that

Thus keeping all threads busy as long as there are tasks to do

Note that pushing and popping a task off your own queue is a relatively cheap operation, so the overhead is kept small

In other words, when there is no opportunity for more parallelism as every thread is already busy doing its own tasks, the overhead over simple sequential execution is minimal

The overhead of stealing a task is greater, but this only happens when a thread would otherwise be idle
TBB Work Stealing

So: if a thread has work to do it does its most recently created task first, thus preserving locality of execution: the next task executed is “nearest” to one just finished.
TBB Work Stealing

So: if a thread has work to do it does its most recently created task first, thus preserving locality of execution: the next task executed is “nearest” to one just finished.

And if a thread has nothing to do it takes the oldest task off another thread, thus disrupting its locality as little as possible.
So: if a thread has work to do it does its most recently created task first, thus preserving locality of execution: the next task executed is “nearest” to one just finished.

And if a thread has nothing to do it takes the oldest task off another thread, thus disrupting its locality as little as possible.

Exercise. It’s much more complicated than this, of course. Read about the details.

Exercise. Work though how work stealing might execute the parallel for example.
TBB

Benefits of TBB:

• easy-to-write parallelism (for a good C++ programmer)
• is very flexible and extensible (e.g., parallel for works for any type that you can iterate over)
• purely a library, so you can use a standard compiler and is easy to update with new versions of the library
• it provides sophisticated constructs like pipelines and general graph parallelism
• contains a large number of features
Benefits of TBB:

- easy-to-write parallelism (for a good C++ programmer)
Benefits of TBB:

- easy-to-write parallelism (for a good C++ programmer)
- is very flexible and extensible (e.g., `parallel_for` works for any type that you can iterate over)
Benefits of TBB:

- easy-to-write parallelism (for a good C++ programmer)
- is very flexible and extensible (e.g., `parallel_for` works for any type that you can iterate over)
- purely a library, so you can use a standard compiler
Benefits of TBB:

- easy-to-write parallelism (for a good C++ programmer)
- is very flexible and extensible (e.g., `parallel_for` works for any type that you can iterate over)
- purely a library, so you can use a standard compiler
- and is easy to update with new versions of the library
TBB

Benefits of TBB:

- easy-to-write parallelism (for a good C++ programmer)
- is very flexible and extensible (e.g., `parallel_for` works for any type that you can iterate over)
- purely a library, so you can use a standard compiler
- and is easy to update with new versions of the library
- it provides sophisticated constructs like pipelines and general graph parallelism
Benefits of TBB:

- easy-to-write parallelism (for a good C++ programmer)
- is very flexible and extensible (e.g., `parallel_for` works for any type that you can iterate over)
- purely a library, so you can use a standard compiler
- and is easy to update with new versions of the library
- it provides sophisticated constructs like pipelines and general graph parallelism
- contains a large number of features
TBB

Drawbacks:

• the code needs some reasonably advanced C++ constructs (e.g., functors) to get the most benefit
• little checking on the correctness of the use of the constructs: it provides a mechanism but no analysis
• it is tied to C++ and thus not easily interoperable with other languages
• contains a large number of features

Exercise. Read about the large number of other features that TBB provides, particularly ranges for load balancing.
TBB

Drawbacks:

- the code needs some reasonably advanced C++ constructs (e.g., functors) get the most benefit
TBB

Drawbacks:

- the code needs some reasonably advanced C++ constructs (e.g., functors) get the most benefit
- little checking on the correctness of the use of the constructs: it provides *mechanism* but no *analysis*
TBB

Drawbacks:

• the code needs some reasonably advanced C++ constructs (e.g., functors) get the most benefit
• little checking on the correctness of the use of the constructs: it provides *mechanism* but no *analysis*
• it is tied to C++
Drawbacks:

- the code needs some reasonably advanced C++ constructs (e.g., functors) get the most benefit
- little checking on the correctness of the use of the constructs: it provides *mechanism* but no *analysis*
- it is tied to C++
- and thus not easily interoperable with other languages
TBB

Drawbacks:

- the code needs some reasonably advanced C++ constructs (e.g., functors) get the most benefit
- little checking on the correctness of the use of the constructs: it provides mechanism but no analysis
- it is tied to C++
- and thus not easily interoperable with other languages
- contains a large number of features
TBB

Drawbacks:

- the code needs some reasonably advanced C++ constructs (e.g., functors) get the most benefit
- little checking on the correctness of the use of the constructs: it provides *mechanism* but no *analysis*
- it is tied to C++
- and thus not easily interoperable with other languages
- contains a large number of features

Exercise. Read about the large number of other features that TBB provides, particularly ranges for load balancing