## Today's Question

How to implement parallel computing structures in C++?

## Serial Workflow

[1]

## Parallel Workflow

[1]

# Pragmatic Introduction to Intel's Threading Building Blocks

Peter Steinbach

Institute for Nuclear and Particle Physics, TU Dresden

June 20th, 2012

# A (bit of) Introduction

## Let's assume ...

- Your software **takes a too long** to complete.
- **You have identified a very CPU intense region in your code!**
- What to do? Let's go **parallel**.

## Multiple Processes

A `process`[2] is an instance of a computer program that is being executed. It contains the program code and its current activity. A process is said to own the following resources: image of the executable machine code, memory (stack, heap), ...

# A (bit of) Introduction

## Let's assume ...

- Your software **takes a too long** to complete.
- **You have identified a very CPU intense region in your code!**
- What to do? Let's go **parallel**.

## Multiple Processes

A `process`[2] is an instance of a computer program that is being executed. It contains the program code and its current activity. A process is said to own the following resources: image of the executable machine code, memory (stack, heap), ...

## Multiple Threads

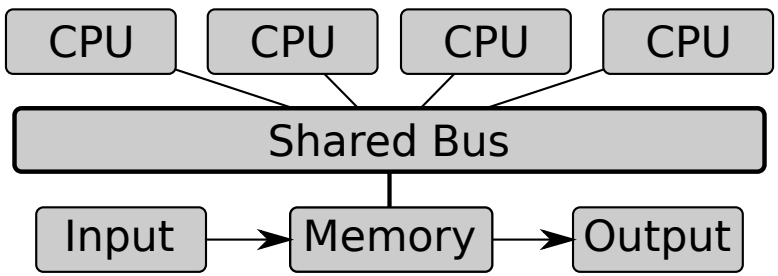
A `thread`[3] of execution is the smallest unit of processing that can be scheduled by an operating system. A thread is a lightweight process. A thread is contained inside a process. Multiple threads can exist within the same process and share resources (memory, ...).

# Parallel $\neq$ Parallel

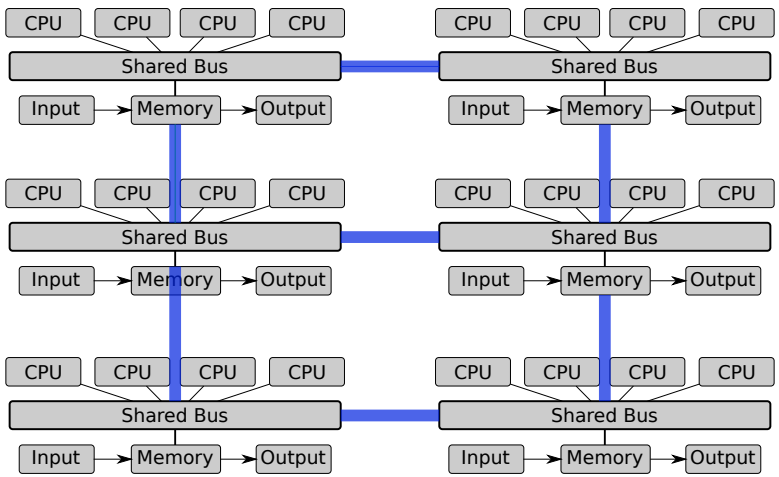
## Does the computer architecture play a role?

For Multi-Threading and Multi-Processing: No.
For your problem: Yes!

*Shared Memory systems*




*Shared/Distributed Memory systems*



# Let's go for Multi-Threading

## I assume ...

- your serial algorithm cannot be optimised further
- problem domain can be decomposed
- speed-up through multi-threading of code is high to justify coding effort

# Let's go for Multi-Threading

## I assume ...

- your serial algorithm cannot be optimised further
- problem domain can be decomposed
- speed-up through multi-threading of code is high to justify coding effort

## Always remember

Implementing concurrency (i.e. multi-threading) is quite an effort!

[4]

# What library to pick?

## Explicit Threading Libraries

Provides low level primitives (such as thread objects etc.)

- C++03 does support threads
  - Boost.Threads
  - POSIX threads
- C++11 does `std::thread`

## Implicit Threading Libraries

Provide higher level API that encapsulate thread control

- OpenMP
- Apple's Grand Central Dispatch
- Intel Threading Building Blocks

## For me as a scientist ...

by first approximation, implicit libraries are my first choice (hide threading details, provide API for most common problems).

# Intel Threading Building Blocks

## Generalities

- Open Source C++ library (GPLv2)
- sponsored by Intel™
- runs on shared memory x86 architectures
- runs with Windows, Mac and Linux (MS Visual++, icc, gcc, ... )
- current version: 4.0

## Design

- generic template library
- design similar to STL (templated algorithms interfaced with templated container types)
- developer friendly (exceptions, compiler checks type safety)
- good documentation (tutorials, Examples, Code reference available online
- some low-level features: custom memory allocators, atomic operation identifiers ...

# A Quick Look Inside

## Containers

- `tbb::concurrent_vector`
- `tbb::concurrent_hash_map`
- `tbb::concurrent_queue`

## Algorithms

- `tbb::parallel_do`
- `tbb::parallel_for` (*)
- `tbb::parallel_sort`
- `tbb::parallel_reduce` (*)
- `tbb::parallel_pipeline`
- ...

# Example 1: A simple logger

## Serial Version

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

# Example 1: A simple logger

## Serial Version

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

## Parallel

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

# Example 2: A throwing random numbers

## Parallel Reduce

```
Sum sumWorker(data);
tbb::parallel_reduce(tbb::blocked_range<size_t>(0,
                                                nIterations,
                                                grainsize),
                     sumWorker);
```

## Difference to Example 1

- `Sum` has no const `operator()`
- `Sum` has dedicated constructor with `tbb::split` macro

# Attention


[1]

## Always ...

Compare Parallel and Serial Performance!
(was/is the effort worthwhile?)

# Summary

## My Conclusion

- Multi-threading is NO SILVER BULLET
- TBB is a well composed template library
- its open source (so sciencists can go for it)
- it offers high-level functionality that encapsulates low level thread management
- it offers an API to approach/solve most problems with parallel solutions
- it has a learning curve

## further reading I recommend

- online TBB documentations
- video lecture series by Clay Breshears

# References

[1] commons.wikimedia.org.
Wikicommons.

[2] Wikipedia: Processes.
Wikipedia.

[3] Wikipedia: Threads.
Wikipedia.

[4] openclipart.org.
OpenClipart library.

## License