An Adventure in Race Conditions

Prepared for ACCU 2019

Felix Petriconi
felix@petriconi.net

2019-04-10

Felix Petriconi

Felix Petriconi
felix@petriconi.net

- Started with C++ 1994
- Programmer and development manager since 2003 at MeVis Medical Solutions AG, Bremen, Germany
  - Development of medical devices in the area of mammography and breast cancer therapy (C++, Ruby)
- Programming activities:
  - Blog editor of ISO C++ website
  - Active member of C++ User Group Bremen
  - Contributor to stlab's concurrency library
  - Member of ACCU conference committee
- Married with Nicole, having three children, living near Bremen, Germany
- Other interests: Classic film scores, composition

Being wrong isn't a bad thing like they teach you in school.
It is an opportunity to learn something.

Richard Feynman

Felix Petriconi
felix@petriconi.net

Why I am here?

- I like being a programmer
- I like sharing my experience
- I like to learn from you

Felix Petriconi
felix@petriconi.net

Felix Petriconi
felix@petriconi.net

Why are you here?

My Domain

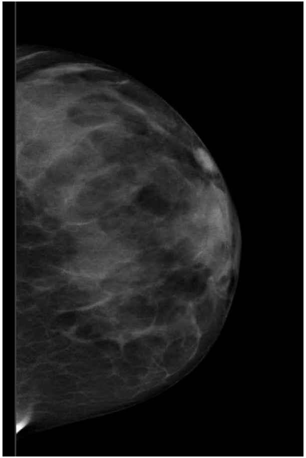
Felix Petriconi
felix@petriconi.net

Display of radiological images for breast cancer detection and diagnosis

Problem from my domain

Felix Petriconi
felix@petriconi.net

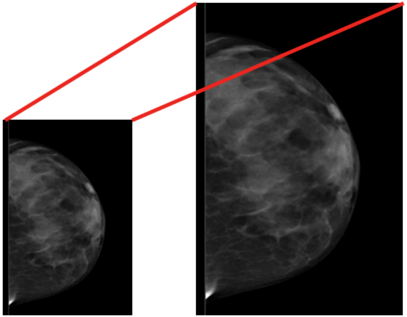
- 3D Mammography 16bit grayscale images of 2048*2560, 50-90 slices
- Display up to 30fps cine mode on 5MP displays
- Only lossless compression is allowed
- JPEG 2000 decompression is too slow while decompression for display
- Re-compression into proprietary format
- Initial approach used 2 user threads
- 620'000 slices / day $\equiv\sim 9h$ processing time



[1] Mammography image from http://www.dclunie.com/

Problem became recently more challenging

- 3D Mammography 16bit grayscale images of 3328*4096, 50-90 slices
- Users expected same performance for display
- Some improvements were necessary...

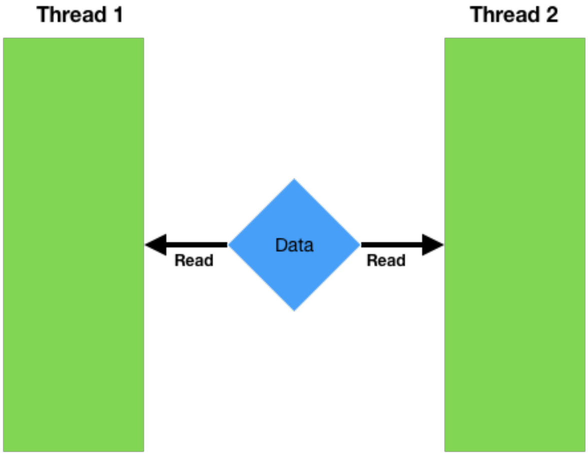


Felix Petriconi
felix@petriconi.net

Parallel Data Access - Basics

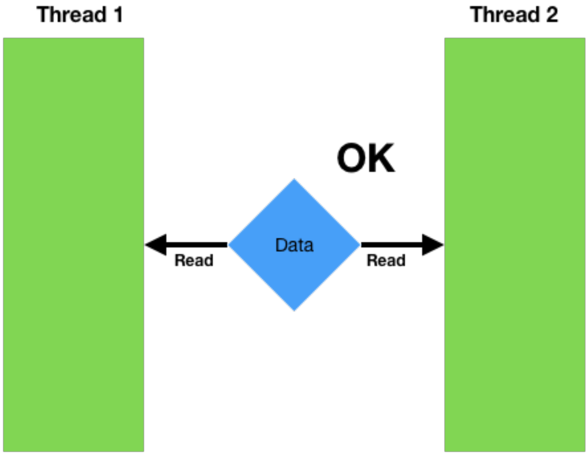
Felix Petriconi
felix@petriconi.net

Parallel Data Read-Only Access



Felix Petriconi
felix@petriconi.net

Parallel Data Read-Only Access



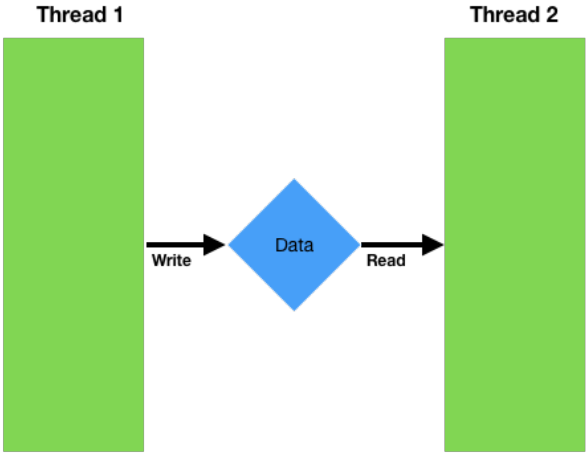
Felix Petriconi
felix@petriconi.net

Synchronisation Primitives

- Atomic
- Mutex
- Semaphore
- Memory Fence
- Transactional Memory

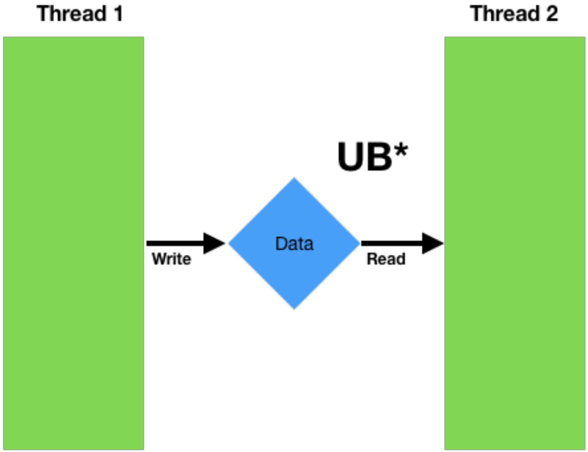
Felix Petriconi
felix@petriconi.net

Parallel Data Read/Write Access



Felix Petriconi
felix@petriconi.net

Parallel Data Read/Write Access



* Undefined Behavior

triconi
fe iconi.net

Code Example - Undefined Behaviour I

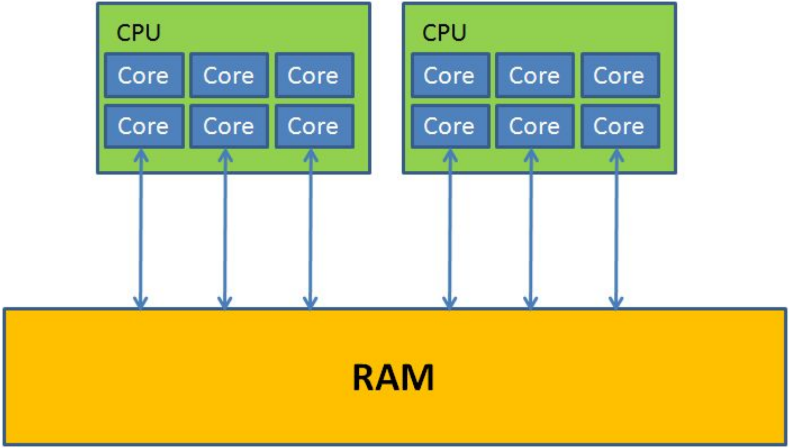
Felix Petriconi
felix@petriconi.net


```cpp
#include <iostream>
#include <thread>

using namespace std;

int main() {
  int value = 42;

  auto t1 = thread{ [&value]{ ++value; } };
  auto t2 = thread{ [&value]{ value *= 2; } };

  t1.join();
  t2.join();

  cout << value << endl;
}
```

Output

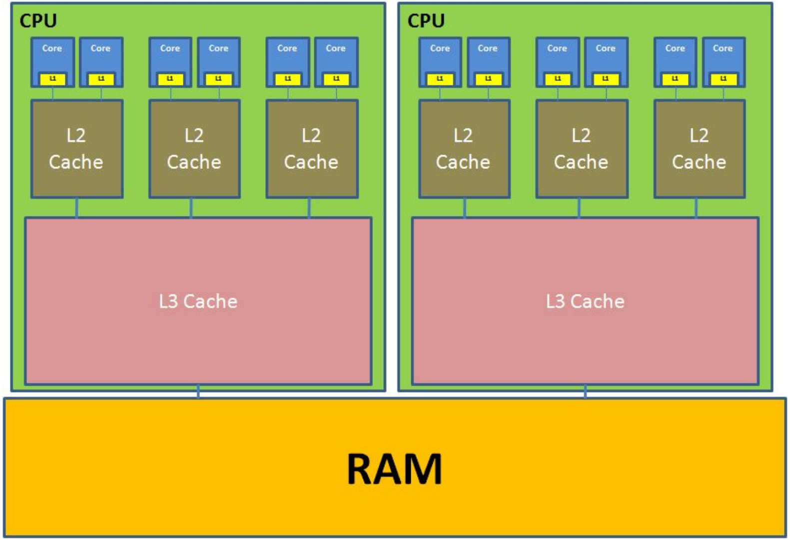
Possible results on my machine: 43, 84, 85, 86

View on a CPU



Felix Petriconi
felix@petriconi.net

More Accurate View of a CPU



Felix Petriconi
felix@petriconi.net

Code Example - Using atomic

```
#include <atomic>
#include <iostream>
#include <thread>

using namespace std;

int main() {
  atomic_int value{42};

  auto t1 = thread{ [&value]{ ++value; } };
  auto t2 = thread{ [&value]{ value = value * 2; } };

  t1.join();
  t2.join();

  cout << value << endl;
}
```

Output

Possible results on my machine: 84, 85, 86

Felix Petriconi
felix@petriconi.net

Code Example - Using mutex

```
int main() {
  int value{42};
  mutex m;
  auto t1 = thread{ [&]{
    unique_lock block{m};
    ++value;
  } };
  auto t2 = thread{ [&]{
    unique_lock block{m};
    value *= 2;
  }};

  t1.join(); t2.join();

  cout << value << endl;
}
```

Output

```
Possible results: 85, 86
```


Felix Petriconi
felix@petriconi.net

Beware of your compiler - Undefined Behaviour II

Felix Petriconi
felix@petriconi.net


```cpp
#include <thread>
using namespace std;

int x = 0, y = 1;
int test = x;

void setTest() {
  test = y;
}

int main() {
  thread run(setTest);

  while (test == 0) { }

  run.join();
  return 0;
}
```

https://godbolt.org/z/g7ZEXL

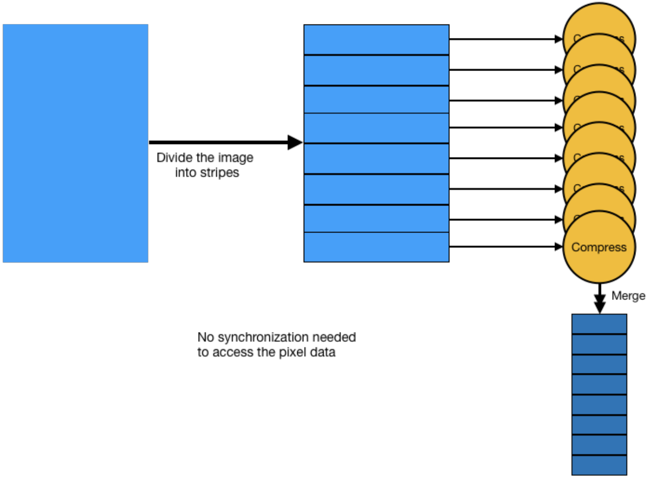
Beware of your compiler - Undefined Behaviour II

Felix Petriconi
felix@petriconi.net


```cpp
#include <thread>
using namespace std;

int x = 0, y = 1;
int test = x;      // test is not an atomic

void setTest() {
  test = y;        // access to test is not synchronized
}

int main() {
  thread run(setTest);

  while (test == 0) { } // Since the access to test is not synchronized,
                        // the compiler can assume, that test is only
  run.join();           // changed by this thread, so it optimizes it
  return 0;             // away and the program never terminates.
}
```

https://godbolt.org/z/g7ZEXL

Image Compression Strategy



Felix Petriconi
felix@petriconi.net

Initial Approach

```cpp
struct CompressContext{} ctx;        // Holds source and target pixel
void compress(CompressContext&) {} // compresses a single stripe
void merge(CompressContext&) {}    // merges all compressed stripes

int main() {
  const int ThreadNumber = 2;
  vector<thread> threads{ThreadNumber};

  for (auto& item : threads)
    item = thread{ []{ compress(ctx); } };

  for (auto& item : threads)
    item.join();

  merge(ctx);
}
```


Felix Petriconi
felix@petriconi.net

Problems of User Threads

- User threads are very expensive
- We have seen that it could take in seldom cases up to 1s to start a user thread
- Starting of more user threads than available cores leads to oversubscription. This leads to expensive context switches.

Felix Petriconi
felix@petriconi.net

Usage of ThreadPool

```
  const int TaskNumber{16};
  atomic_int to_do{TaskNumber};



  for (int i = 0; i < TaskNumber; ++i)
    stlab::default_executor(  // thread pool from stlab/concurrency
      [&]() {
        compress(data);
        --to_do;

      });


  while (to_do != 0)


  merge(data);
}
```

Usage of ThreadPool

```
  const int TaskNumber{16};
  atomic_int to_do{TaskNumber};
  mutex block;
  condition_variable cv;

  for (int i = 0; i < TaskNumber; ++i)
    stlab::default_executor(  // thread pool from stlab/concurrency
      [&]() {
        compress(ctx);
        --to_do;
        cv.notify_one();
      });

  unique_lock lock{block};
  while (to_do != 0)
    cv.wait(lock);

  merge(ctx);
}
```


Felix Petriconi
felix@petriconi.net

The Adventure Begins

1st Race

```
const int TaskNumber{16};
atomic_int to_do{TaskNumber};
mutex block;
condition_variable cv;

for (int i = 0; i < TaskNumber; ++i)
  stlab::default_executor(  // thread pool from stlab/concurrency
    [&]() {
      compress(ctx);
      --to_do;
      cv.notify_one();
    });

unique_lock lock{block};
while (to_do != 0)
  cv.wait(lock);

merge(ctx);
}
```


Felix Petriconi
fel iconi.net

1st Race

```
const int TaskNumber{16};
atomic_int to_do{TaskNumber};
mutex block;
condition_variable cv;

for (int i = 0; i < TaskNumber; ++i)
  stlab::default_executor(  // thread pool from stlab/concurrency
    [&]() {
      compress(ctx);
      --to_do;                // Even if the shared variable is atomic,
      cv.notify_one();        // it must be modified under the mutex in
    });                       // order to correctly publish the
                              // modification to the waiting thread.
unique_lock lock{block};
while (to_do != 0)
  cv.wait(lock);

merge(ctx);
}
```

https://en.cppreference.com/w/cpp/thread/condition_variable

Felix Petriconi
felix@petriconi.net

1st Correction

```cpp
  const int TaskNumber{16};
  atomic_int to_do{TaskNumber};
  mutex block;
  condition_variable cv;

  for (int i = 0; i < TaskNumber; ++i)
    stlab::default_executor(  // thread pool from stlab/concurrency
      [&]() {
        compress(ctx);
        --to_do;
        cv.notify_one();
      });

  unique_lock lock{block};
  while (to_do != 0)
    cv.wait(lock);

  merge(ctx);
}
```


Felix Petriconi
felix@petriconi.net

1st Correction

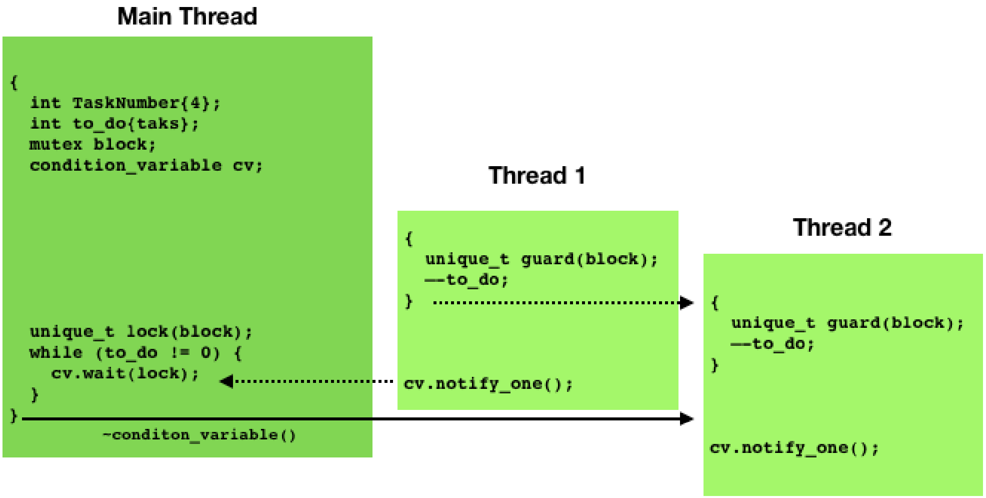
```
  const int TaskNumber{16};
  atomic_int to_do{TaskNumber};
  mutex block;
  condition_variable cv;

  for (int i = 0; i < TaskNumber; ++i)
    stlab::default_executor(
      [&]() {
        compress(ctx);
        {
          unique_lock guard{block};
          --to_do;
        }
        cv.notify_one();
      });

  unique_lock lock{block};
  while (to_do != 0)
    cv.wait(lock);

  merge(ctx);
}
```


Felix Petriconi
felix@petriconi.net

2nd Race

```
  const int TaskNumber{16};
  atomic_int to_do{TaskNumber};
  mutex block;
  condition_variable cv;

  for (int i = 0; i < TaskNumber; ++i)
    stlab::default_executor(
      [&]() {
        compress(ctx);
        {
          unique_lock guard{block};
          --to_do;
        }
        cv.notify_one();
      });

  unique_lock lock{block};
  while (to_do != 0)
    cv.wait(lock);

  merge(ctx);
}
```

2nd Race



Main Thread
{
int TaskNumber{4};
int to_do{taks};
mutex block;
condition_variable cv;
unique_t lock(block);
while (to_do != 0) {
cv.wait(lock);
}
}
~conditon_variable()
Thread 1
{
unique_t guard(block);
--to_do;
}
cv.notify_one();
Thread 2
{
unique_t guard(block);
--to_do;
}
cv.notify_one();

triconi
fe iconi.net

2nd Correction

```
  const int TaskNumber{16};
  atomic_int to_do{TaskNumber};
  mutex block;
  condition_variable cv;

  for (int i = 0; i < TaskNumber; ++i)
    stlab::default_executor(
      [&]() {
        compress(ctx);
        {
          unique_lock guard{block};
          --to_do;
        }
        cv.notify_one();
      });

  unique_lock lock{block};
  while (to_do != 0)
    cv.wait(lock);

  merge(ctx);
}
```

2nd Correction

```
  const int TaskNumber{16};
  atomic_int to_do{TaskNumber};
  mutex block;
  condition_variable cv;

  for (int i = 0; i < TaskNumber; ++i)
    stlab::default_executor(
      [&]() {
        compress(ctx);
        {
          unique_lock guard{block};
          --to_do;
          cv.notify_one();
        }
      });

  unique_lock lock{block};
  while (to_do != 0)
    cv.wait(lock);

  merge(ctx);
}
```


Felix Petriconi
felix@petriconi.net

Adding Error Handling

```
struct CompressContext{} ctx;
bool compress(CompressContext&) // true when OK, false when failed
void merge(CompressContext&) {}

int main() {
  const int TaskNumber{16};
  int to_do{TaskNumber};
  atomic_bool abort{false};
  mutex block;
  condition_variable cv;

  for (int i = 0; i < TaskNumber; ++i)
    stlab::default_executor(
      [&]() {
        if (abort) return;
        auto do_abort = !compress(ctx);
        {
          unique_lock guard{block};
          --to_do;
          abort = do_abort || abort;
          cv.notify_one();
        }
      });
```


Felix Petriconi
felix@petriconi.net

Adding Error Handling

```
int to_do{TaskNumber};
atomic_bool abort{false};
mutex block;
condition_variable cv;

for (int i = 0; i < TaskNumber; ++i)
  stlab::default_executor(
    [&]() {
      if (abort) return;
      auto do_abort = !compress(ctx);
      {
        unique_lock guard{block};
        --to_do;
        abort = do_abort || abort;
        cv.notify_one();
      }
    });

unique_lock lock{block};
while (to_do != 0 && !abort)
  cv.wait(lock);

merge(ctx);
```

3rd Race

```
const int TaskNumber{16};
int to_do{TaskNumber};
atomic_bool abort{false};
mutex block;
condition_variable cv;

for (int i = 0; i < TaskNumber; ++i)
  stlab::default_executor(
    [&]() {
      if (abort) return;
      auto do_abort = !compress(ctx);
      {
        unique_lock guard{block};
        --to_do;
        abort = do_abort || abort;
        cv.notify_one();
      }
    });

unique_lock lock{block};
while (to_do != 0 && !abort)
  cv.wait(lock);
```


Felix Petriconi
felix@petriconi.net

3rd Race

```
const int TaskNumber{16};        // None of these variables exist
int to_do{TaskNumber};           // when there is an early exit
atomic_bool abort{false};        // with abort is been set. So the
mutex block;                     // other running tasks must not
condition_variable cv;           // use it further.

for (int i = 0; i < TaskNumber; ++i)
  stlab::default_executor(
    [&]() {
      if (abort) return;
      auto do_abort = !compress(ctx);
      {
        unique_lock guard{block};
        --to_do;
        abort = do_abort || abort;
        cv.notify_one();
      }
    });

unique_lock lock{block};
while (to_do != 0 && !abort)
  cv.wait(lock);
```


Felix Petriconi
felix@petriconi.net

3rd Correction

Felix Petriconi
felix@petriconi.net

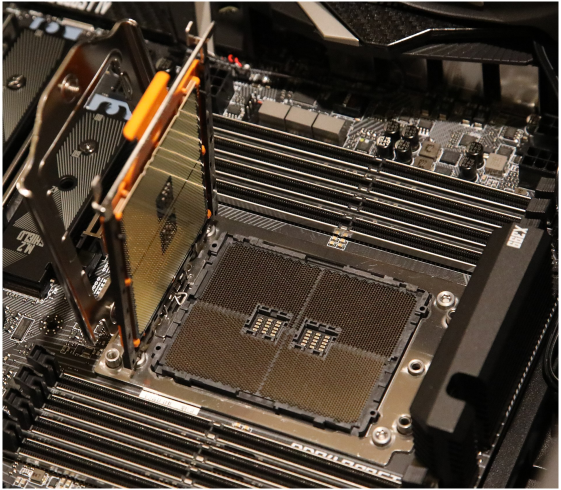

```
struct CompressContext{} ctx;
bool compress(CompressContext&)
{ return true; }
void merge(CompressContext&) {}

struct ProcessContext
{
  mutex block;
  condition_variable cv;
  int to_do = 0;
  atomic_bool abort{false};
};
```

3rd Correction

```
const int TaskNumber{16};
auto pctx = make_shared<ProcessContext>();
pctx->to_do = TaskNumber;
for (int i = 0; i < TaskNumber; ++i)
  stlab::default_executor(
    [_weakContext = weak_ptr<ProcessContext>(pctx)] {
      auto p = _weakContext.lock();
      if (!p || p->abort)
        return;
      auto do_abort = !compress(ctx);
      {
        unique_lock guard{p->block};
        --p->to_do;
        p->abort = do_abort || p->abort;
        p->cv.notify_one();
      }
    });

unique_lock lock{pctx->block};
while (pctx->to_do != 0 && !pctx->abort)
  pctx->cv.wait(lock);

merge(ctx);
```


Felix Petriconi
felix@petriconi.net

Conclusion



[2]Geni - photo by user:geni, CC BY-SA 4.0
https://commons.wikimedia.org/w/index.php?curid=71925797

Felix Petriconi
felix@petriconi.net

Conclusion

- It is easy to get a CPU with more cores.
- It is hard to write concurrent code correct.
- It is even harder to use low synchronization primitives correctly.

Try to use high level abstractions like

- Future
- Channel
- Actor
- ...

Felix Petriconi
felix@petriconi.net

Example with boost futures

```
#include <vector>
#define BOOST_THREAD_PROVIDES_FUTURE
#define BOOST_THREAD_PROVIDES_FUTURE_CONTINUATION
#define BOOST_THREAD_PROVIDES_FUTURE_WHEN_ALL_WHEN_ANY
#include <boost/thread/future.hpp>

using std::vector;

struct CompressContext{} ctx;
bool compress(CompressContext&)
{ return true; }
void merge(CompressContext&) {}

int main() {
    vector<boost::future<void>> tasks{16};

    for (auto& f : tasks)
        f = boost::async( []{ compress(ctx); } );

    auto done = boost::when_all(tasks.begin(), tasks.end())
      .then([](auto) { merge(ctx); });
}
```

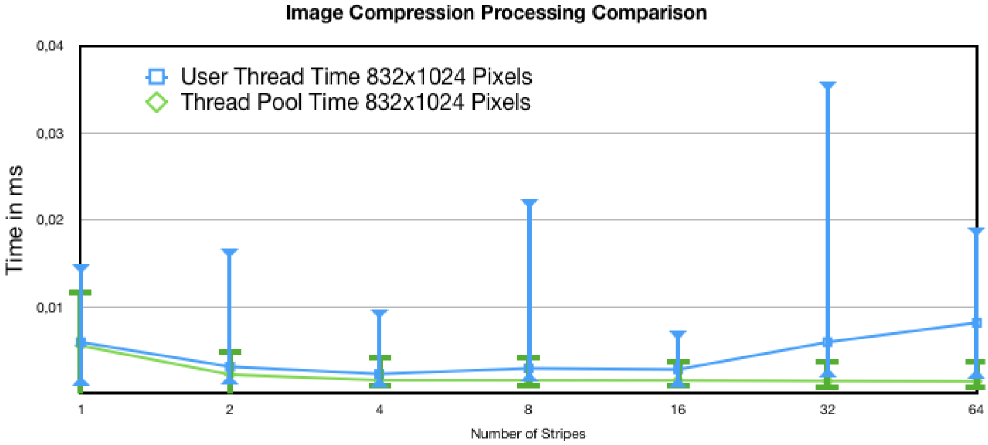

Felix Petriconi
felix@petriconi.net

Example with stlab futures

```cpp
struct CompressContext{} ctx;
bool compress(CompressContext&)
{ return true; }
void merge(CompressContext&) {}

int main() {
  size_t TaskNumber{16};

  vector<stlab::future<void>> tasks{TaskNumber};

  for (auto& task : tasks)
    task = stlab::async(stlab::default_executor,
      [] { compress(ctx); });

  auto done = stlab::when_all(stlab::default_executor,
    []{ merge(ctx); }, make_pair(tasks.begin(), tasks.end()) );

  stlab::blocking_get(done);
```


Felix Petriconi
felix@petriconi.net

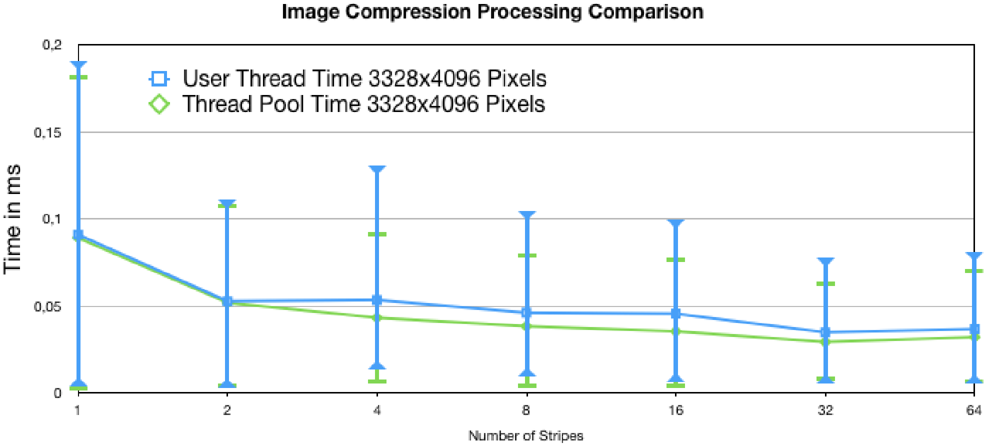
Performance Comparison I



Felix Petriconi
fel[illegible]iconi.net

Performance Comparison II

triconi
fe iconi.net



Strategies

Felix Petriconi
felix@petriconi.net

- Try to break down your problem into small parts that can be solved without any synchronization.
- Whenever it is possible prefer high level abstractions over low level synchronization primitives.
- Try to think in parallel. Have in mind that
  - any operation can be interrupted at any time,
  - e.g. between any lines or even within a single line.
  - This is true for hidden code, e.g. destructors too.

Acknowledgement

- My family, who supports me in my work on the concurrency library and this conference.
- Sean Parent, who taught me over time lots about concurrency and abstraction. He gave me the permission to use whatever I needed from his presentations for my own.
- My company MeVis Medical Solutions AG, who give me the possibility to be here.
- The C++ UserGroup in Bremen, where I can test my sessions.
- All contributors to the stlab library.

Felix Petriconi
felix@petriconi.net

Reference

- Concurrency library `https://github.com/stlab/libraries`
- Documentation `http://stlab.cc/libraries`
- Communicating Sequential Processes by C. A. R. Hoare `http://usingcsp.com/cspbook.pdf`
- The Theory and Practice of Concurrency by A.W. Roscoe `http://www.cs.ox.ac.uk/people/bill.roscoe/publications/68b.pdf`
- Towards a Good Future, C++ Standard Proposal by Felix Petriconi, David Sankel and Sean Parent `http://open-std.org/JTC1/SC22/WG21/docs/papers/2017/p0676r0.pdf`
- A Unified Futures Proposal for C++ by Bryce Adelstein Lelbach, et al `http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2018/p1054r0.html`

Felix Petriconi
felix@petriconi.net

Further reading I

Software Principles and Algorithms

- Elements of Programming by Alexander Stepanov, Paul McJones, Addison Wesley
- From Mathematics to Generic Programming by Alexander Stepanov, Daniel Rose, Addison Wesley

Felix Petriconi
felix@petriconi.net

Further reading II

Felix Petriconi
felix@petriconi.net

Concurrency and Parallelism

- HPX `http://stellar-group.org/libraries/hpx/`
- C++CSP `https://www.cs.kent.ac.uk/projects/ofa/c++csp`
- CAF_C++ Actor Framework `http://actor-framework.org/`
- C++ Concurrency In Action by Anthony Williams, Manning, 2nd Edition

Further listening and viewing

- Goals for better code by Sean Parent: `http://sean-parent.stlab.cc/papers-and-presentations`
- Goals for better code by Sean Parent: Concurrency: `https://youtu.be/au0xX4h8SCI?t=16354`
- Future Ruminations by Sean Parent `http://sean-parent.stlab.cc/2017/07/10/future-ruminations.html`
- CppCast with Sean Parent `http://cppcast.com/2015/06/sean-parent/`
- Thinking Outside the Synchronization Quadrant by Kevlin Henney: `https://vimeo.com/205806162`
- Inside Windows 8 Thread Pool `https://channel9.msdn.com/Shows/Going+Deep/Inside-Windows-8-Pedro-Teixeira-Thread-pool`

Felix Petriconi
felix@petriconi.net

stlab Futures

Felix Petriconi
felix@petriconi.net

stlab::future

Source: `https://github.com/stlab/libraries`

Documentation: `https://www.stlab.cc/libraries`

Thank's for your attention!

- Mail: felix@petriconi.net
- GitHub: `https://github.com/FelixPetriconi`
- Web: `https://petriconi.net`
- Twitter: @FelixPetriconi

Q & A

- Mail: felix@petriconi.net
- GitHub: `https://github.com/FelixPetriconi`
- Web: `https://petriconi.net`
- Twitter: @FelixPetriconi

Feedback is always welcome!