

Math 4997-1

Lecture 13: Futurization of the 1D heat equation

Patrick Diehl 

<https://www.cct.lsu.edu/~pdiehl/teaching/2020/4997/>

This work is licensed under a Creative Commons "Attribution-NonCommercial-NoDerivatives 4.0 International" license.



Reminder

Lecture 12

What you should know from last lecture

- ▶ One-dimensional heat equation
- ▶ Serial implementation of the one-dimensional heat equation

HPX features

A ready future

Some times, we need a future, which is already ready, since there is no computation needed.

```
hpx::lcos::future<double> f  
    = hpx::make_ready_future(1);
```

Example

```
auto f = hpx::make_ready_future(1);  
/*  
 * Since the future is ready the output will happen  
 * and there will be no barrier.  
 */  
std::cout << f.get() << std::endl;
```

Data flow I

```
std::vector<hpx::lcos::future<int>> futures;
futures.push_back(hpx::async(square,10));
futures.push_back(hpx::async(square,100));

// When all returns a future containing the vector
// of futures
hpx::when_all(futures).then([](auto&& f){
    // We need to unwrap this future to get
    // the content of it
    auto futures = f.get();
    int result = 0;
    for(size_t i = 0; i < futures.size();i++)
        result += futures[i].get();
    std::cout << result << std::endl;
});
```

Data flow II

```
hpx::dataflow(hpx::launch::sync, [](auto f){  
    int result = 0;  
    for(size_t i = 0; i < f.size(); i++)  
        result += f[i].get();  
    std::cout << result << std::endl;  
}, futures);
```

Parameters

1. `hpx::launch::async` Or `hpx::launch::sync`
2. The function to call
3. Futures to the arguments to the arguments of the function

Passing futures

```
void sum(int first, int second){  
  
    std::cout << first + second << std::endl;  
  
}  
  
auto f1 = hpx::async(square,10);  
auto f2 = hpx::async(square,100);  
  
// We have to call .get() to pass  
// the values of the future  
sum(f1.get(),f2.get());
```

Unwrapping futures

```
void sum(int first, int second){  
  
    std::cout << first + second << std::endl;  
  
}  
  
// We can unwarpp the function  
auto fp = hpx::util::unwrapping(sum);  
  
// After unwrapping, we can pass the future  
// directly to the function  
hpx::dataflow(hpx::launch::sync, fp, f1, f2);
```

Shared future

```
hpx::lcos::future
```

- ▶ Exclusive ownership model
- ▶ If the future is out of the scope, it will be not available anymore.

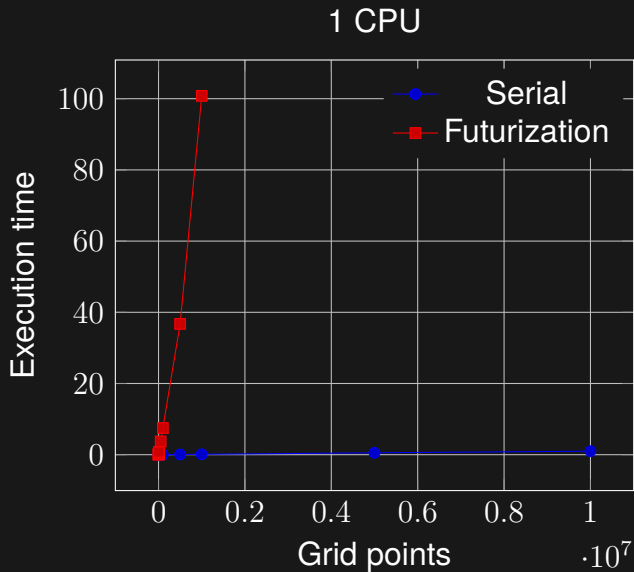
```
hpx::shared_future
```

- ▶ Reference counting ownership model
- ▶ All references to the object are counted and the object is only destroyed if there are zero references.

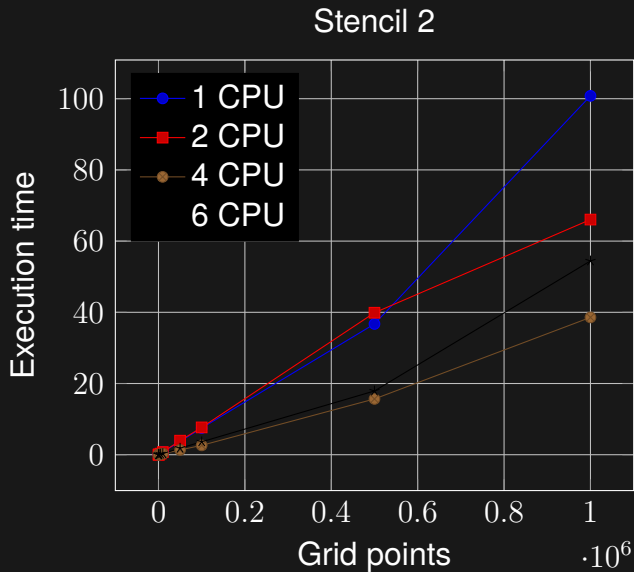
Can be seen to be equal to `std::unique_ptr` and `std::shared_ptr`.

Scaling results

Overhead



Scaling



Summary

Summary

After this lecture, you should know

- ▶ `hpx::make_ready_future`
- ▶ `hpx::dataflow`
- ▶ `hpx::util::unwrapping`
- ▶ `hpx::shared_future`