Math 4997-3

Lecture 15: Parallel partition-based 1D heat equation

Patrick Diehl

https://www.cct.lsu.edu/~pdiehl/teaching/2021/4997/

Notes

Notes

Reminder

Notes

Lecture 14

What you should know from last lecture

- Using partitions to control the grain size
- Moving objects with `std::move`
- Semaphores and `hpx::lcos::local::sliding_semaphore`

Notes

Datastructure

Notes

Class for holding the data for each partition I[1]

```
class partition_data
{
private:
    // Since we use a array of doubles,
    // we need to store the size
    std::size_t size_;
    // Instead of a std::vector,
    // we use a array of doubles
    std::unique_ptr<double[]> data_;
};
```

Note that `std::unique_ptr` is a smart pointer containing another object as a pointer and destroy this object when the `std::unique_ptr` goes of of scope.

[1] https://en.cppreference.com/w/cpp/memory/unique_ptr

Notes

Class for holding the data for each partition II

```
class partition_data
{
public:
    partition_data(std::size_t size)
      : data_(new double[size]), size_(size)
    {}

    partition_data(std::size_t size, double init_value)
      : data_(new double[size]),
        size_(size)
    {
        double base_value = double(init_value * size);
        for (std::size_t i = 0; i != size; ++i)
            data_[i] = base_value + double(i);
    }
};
```

Notes

The new expression[2]

```
// Allocation of a single double value
double* p = new double(42);

// Deallocate memory
delete p;

// Allocation of 5 double values
double* p = new double[5];

// Deallocate memory
delete[] p;
```

The `new` expression initializes objects with dynamic storage duration, which means that the lifetime of the object is not limited by the scope of creation.

[2] https://en.cppreference.com/w/cpp/language/new

Notes

Update the types

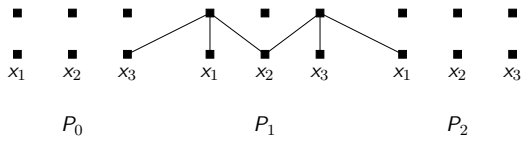
```
// As in the previous example, we use a shared future
// but instead of the double value for the heat, we
// store the partition data there
typedef hpx::shared_future<partition_data> partition;

// The standard vector holds the futures of partitions
// instead of the futures of doubles to control the
// grain size
typedef std::vector<partition> space;
```

Notes

Simulation control

Notes

Accessing neighboring partitions I



Notes

Accessing neighboring partitions II

```
static partition_data heat_part(
    partition_data const& left,
    partition_data const& middle,
    partition_data const& right)
{
    std::size_t size = middle.size();
    partition_data next(size);

    next[0] = heat(left[size-1], middle[0], middle[1]);

    for(std::size_t i = 1; i != size-1; ++i)
     next[i]
         = heat(middle[i-1], middle[i], middle[i+1]);

    next[size-1] = heat(middle[size-2], middle[size-1], right[0]);

    return next;
}
```

Notes

Simulation control I

```
// do all the work on 'np' partitions, 'nx' data points each,
// for 'nt' time steps,
// limit depth of dependency tree to 'nd'
hpx::future<space> do_work(std::size_t np,
    std::size_t nx, std::size_t nt, std::uint64_t nd)
{
// Set the boundary conditions in parallel
auto range = boost::irange(b, np);
using hpx::execution::par;
hpx::parallel::for_each(par, std::begin(range),
    std::end(range),[&U, nx](std::size_t i)
    {
    U[0][i] = hpx::make_ready_future(
            partition_data(nx, double(i)));
    });
}
```

Notes

Simulation control II

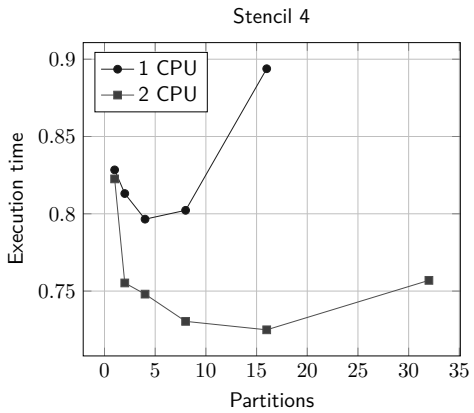
```
hpx::future<space> do_work(std::size_t np,
    std::size_t nx, std::size_t nt, std::uint64_t nd)
{
for (std::size_t t = 0; t != nt; ++t)
{
  for (std::size_t i = 0; i != np; ++i)
  {   // Note that we use async to get a future back
      next[i] = dataflow(
          hpx::launch::async, Op,
          current[idx(i, -1, np)],
          current[i],
          current[idx(i, +1, np)]
          );

   }
}
```

Notes

Scaling results

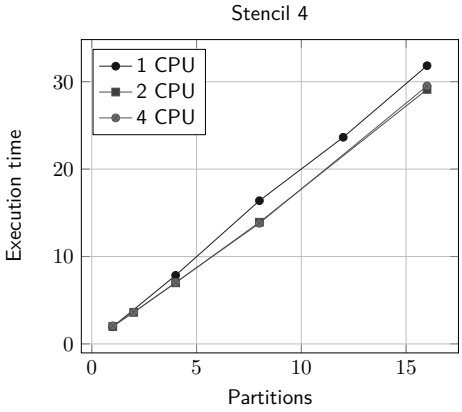
Notes

Scaling for 10000000 discrete mesh points



Notes

Scaling for increasing amount of work



Notes

Summary

Notes

Summary

After this lecture, you should know

- Parallel partition-based implementation
- Allocating and deallocating memory with the `new` and `delete` expression

Notes

Notes