Math 4997-3

Lecture 16: Preparation for distributed computing

Patrick Diehl

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

Reminder

Serialization

Components

Summary

References

Reminder

Lecture 15

What you should know from last lecture

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

Serialization

What is this serialization thingy

If we want to send ab object or a bunch of objects over the network to another node of the cluster:

- The sender needs to flatten the object(s) to a one-dimensional stream of bits.
- The receiver needs to unflatten the one-dimensional stream of bits and convert it back to the objects(s)

You have to decide whether to serialize

- human-readable (text)
- non-human-readable (binary)

Serialization in HPX [1]

Initialize data

```
size_t size = 5;
double* data = new double[size];
```

Serialization

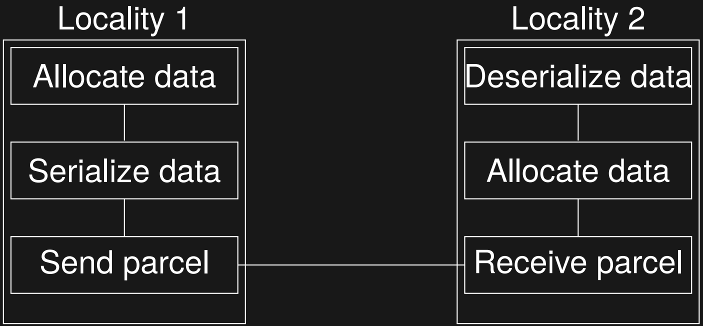
```
using hpx::serialization::serialize_buffer;

serialize_buffer<double> serializable_data(
     data, size,
     serialize_buffer<double>::init_mode::reference);
```

Deserialization

```
double* copied_data = serializable_data.data();
```

Sending data over the network



The communication between the localities (nodes) is handled by the so-called parcel port [2]. HPX uses MPI or libfrabric for communication between nodes.

Components

Components and Actions

For distributed computations within HPX, we need to look into following

1. Server (Component)
   The server represents the global data and is a so-called HPX component which allows to create and access the data remotely through a global address space (AGAS [3]) using `hpx::id_type`.
2. Component action
   Each function of the component needs to be wrapped into a component action to be remotely available.
3. Plain actions
   Allows to wrap global (`static`) functions in an action. So this function can be called remotely on a given locality.

Server (Component)

```
struct data_server
   : hpx::components::component_base<data_server>{

data_server(size_t size)
{
data = std::shared_ptr<double[]>(new double[size]);
}

private:

// This data will be in the global address space
std::shared_ptr<double[]> data;

};
```

Note that we need component actions to call the public functions of the server.

Component actions

```
struct data_server
  : hpx::components::component_base<data_server>{

// This function needs to be wrapped in an action
hpx::serialization::serialize_buffer<double> getData()
{
  typedef hpx::serialization::serialize_buffer<double>
    buffer;

   return buffer(data.get(), size,
      buffer::init_mode::reference);
}
// Define the action for remote access
HPX_DEFINE_COMPONENT_DIRECT_ACTION(data_server,
                     getData, getData_action);

};
```

Component actions

```
// Define the action for remote access
HPX_DEFINE_COMPONENT_DIRECT_ACTION(
                    data_server,
                    getData,
                    getData_action);
```

where

- The first argument is the name of the component
- The second argument is the name of the function
- The third argument is the name of the action

Registering components and actions

```
// Generation of the code, which is needed to call
// the component action remotely

// Registering the component
typedef hpx::components::component<data_server>
    data_server_type;
HPX_REGISTER_COMPONENT(data_server_type,
    data_server);

// Registering the component actions
typedef data_server::getData_action getData_action;
HPX_REGISTER_ACTION(getData_action);
```

The data client I

```
struct data
  : hpx::components::client_base<data,data_server>
{
  // Define client base
  typedef hpx::components::client_base<data,
     data_server> base_type;

  // Constructor to generate the data_server object
  // The where argument specifies where the object is
  // allocated
  data(hpx::naming::id_type where,std::size_t size) :
     base_type(hpx::new_<data_server>(where,size)){}

);
```

The data client II

```
struct data
  : hpx::components::client_base<data,data_server>
{
  // Wrap the actions into futures
  hpx::future<size_t> getSize(){

     return hpx::async(getSize_action(),get_id());
  }

  typedef hpx::serialization::serialize_buffer<double>
     buffer;

  hpx::future<buffer>getData(){

    return hpx::async(getData_action(),get_id());
  }

);
```

Global functions

Function definition

```
static void square(double a ){

std::cout << a * a << std::endl;

}
```

Define the action

```
HPX_PLAIN_ACTION(square, square_action);
```

where

- The first argument is the name of the function
- The second argument is the name of the action

Knowing where you are

Getting the ID of your locality

```
hpx::find_here();
```

Getting the ID of the component’s locality

```
hpx::get_colocation_id(hpx::launch::sync, get_id();
```

Getting the ID of the component’s locality

```
bool is_local
    = (hpx::get_colocation_id(hpx::launch::sync,
          get_id()) == hpx::find_here());
```

Summary

Summary

After this lecture, you should know

- Serialization
- Distributed computing
  - Plain actions
  - Components
  - Components actions

References

References I

[1] John Biddiscombe, Thomas Heller, Anton Bikineev, and Hartmut Kaiser.
Zero Copy Serialization using RMA in the Distributed Task-Based HPX runtime.
In *14th International Conference on Applied Computing*. IADIS, International Association for Development of the Information Society, 2017.

[2] Hartmut Kaiser, Maciek Brodowicz, and Thomas Sterling.
Parallex an advanced parallel execution model for scaling-impaired applications.
In *2009 International Conference on Parallel Processing Workshops*, pages 394–401. IEEE, 2009.

References II

[3] Hartmut Kaiser, Thomas Heller, Bryce Adelstein-Lelbach, Adrian Serio, and Dietmar Fey.
Hpx: A task based programming model in a global address space.
In *Proceedings of the 8th International Conference on Partitioned Global Address Space Programming Models*, page 6. ACM, 2014.