HPC Programming Bootcamp

Day 4: Task-based Programming Models

Abhinav Bhatele, Department of Computer Science

UNIVERSITY OF MARYLAND

Task-based programming models

- Describe program / computation in terms of tasks
- Tasks might be short-lived or persistent throughout program execution
- Notable examples: Charm++, StarPU, HPX, Legion
- Attempt at classification: https://link.springer.com/article/10.1007/s11227-018-2238-4

What is Charm++

- Programmer burden is significant when using MPI
- Can we design a programming model/runtime that does a balanced division of labor between the programmer and the system?
- Charm++ is a library for writing parallel programs
  - An alternative to MPI, OpenMP, etc.
  - Can be used for shared and distributed memory
- Includes both the programming model and the runtime

Key Principles

- *Over-decomposition*: Programmer decomposes data and work into objects
  - Decoupled from number of processes or cores
- *Migratability*: Runtime assigns objects to physical resources (cores and nodes)
- Each object can only access its own data
  - Request data from other objects via remote method invocation: foo.get_data()
- *Asynchrony*: Message-driven execution

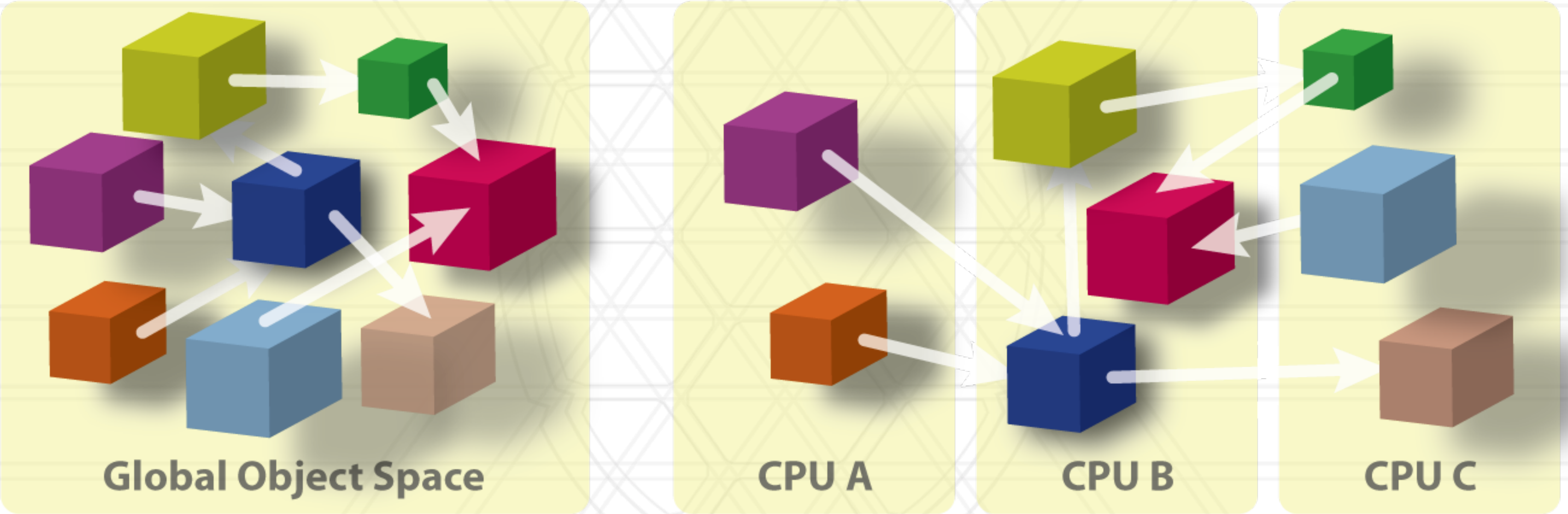
Impact of these design choices

- Over-decomposition: finer-grained data and work units compared to say MPI
- Migratability: communication must be addressed to logical tasks and not physical processors
- Asynchrony: scheduling tasks is more complicated, message-driven

Implementation in Charm++

- Create over-decomposed entities called shares
  - Chares are C++ objects
  - Chares can be organized into indexed collections such as arrays
- Designate certain functions are "entry" methods
  - These can be remotely invoked on one object from another object
  - Communication happens via these entry methods

Charm++: Global view



User View

System View

Hello World in Charm++

```
module hello {

  array [1D] Hello {
    entry Hello();
    entry void sayHi();
  };

};
```

Charm++ Tutorial: http://charmplusplus.org/tutorial/ArrayHelloWorld.html

Hello World in Charm++

```
module hello {

  array [1D] Hello {
    entry Hello();
    entry void sayHi();
  };

};
```

```
void Hello ::sayHi() {
  CkPrintf("Hello from chare %d on processor %d.\n”, thisIndex,
CkMyPe());
}
```

Charm++ Tutorial: http://charmplusplus.org/tutorial/ArrayHelloWorld.html

Hello World in Charm++

```
module hello {

  array [1D] Hello {
    entry Hello();
    entry void sayHi();
  };

};
```

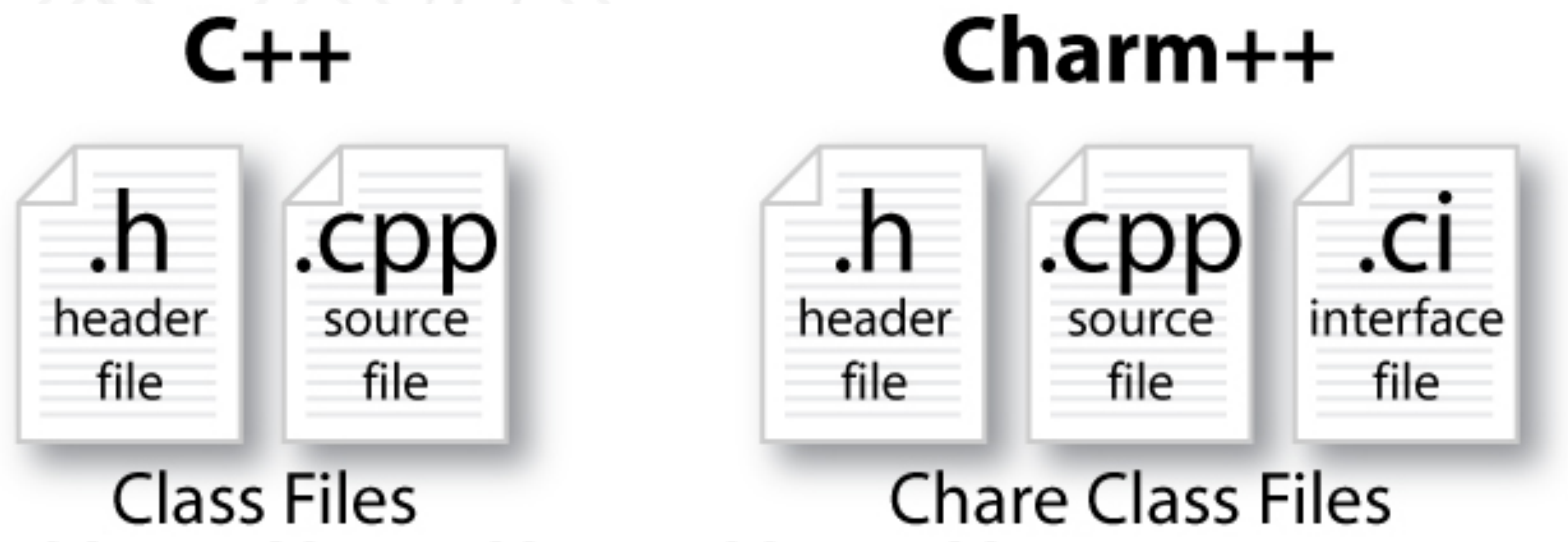
```
Main::Main(CkArgMsg* msg) {
  numElements = 5; // number of elements

  CProxy_Hello helloArray =
          CProxy_Hello::ckNew(numElements);

  helloArray.sayHi();
}
```

```
void Hello ::sayHi() {
  CkPrintf("Hello from chare %d on processor %d.\n", thisIndex,
CkMyPe());
}
```

Charm++ Tutorial: http://charmplusplus.org/tutorial/ArrayHelloWorld.html

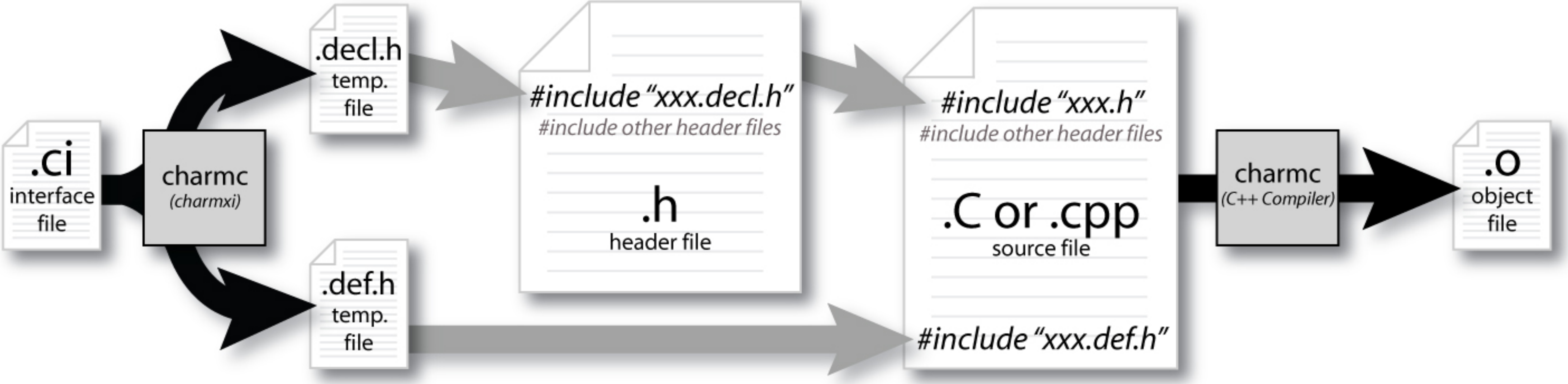
Charm++ file organization

- C++ objects defined in regular .h and .C files
- Share objects and entry methods also defined in a .ci file
  - .ci file is translated to regular C++ code by the charm translator
  - Implemented in the .C file



Compiling a Charm++ program

- Compile .ci file with charms
- Then compile the .C files



Compiling and running Charm++ program

- Compiling the program

```
charmc hello.ci
charmc -c hello.C
charmc -o hello hello.o
```

- Running the program

```
./charmrun +p5 ./hello
```

Charm++ interface file: modules

- Charm++ programs are organized as a collection of modules
- Each module has one or more shares
- The main module contains the mainchare

```
[main]module MyModule {
   //... chare definitions ...
};
```

Charm++ interface file: chares

- Chares are parallel objects managed by the runtime
- Each chare has a set of entry methods that can be invoked remotely

```
[main]chare MyChare {
   //... entry method definitions ...
};
```

- The user extends the generated class CBase_MyChare in the .C file

```
class MyChare : public CBase_MyChare {
   //... entry method implementations ...
};
```

Charm++ interface file: entry methods

- .ci file:

```
entry MyChare(); /* constructor entry method */
entry void foo();
entry void bar(int param);
```

- .C file:

```
MyChare::MyChare() { /*... constructor code ...*/ }
MyChare::foo() { /*... code to execute ...*/ }
MyChare::bar(int param) { /*... code to execute ...*/ }
```

Creating a chare

- A chare can be instantiated by the following call:

```
CProxy_MyChare::ckNew(<args>);
```

- It is best to retain a proxy to the chare to able to communicate with it in the future:

```
CProxy_MyChare proxy = CProxy_MyChare::ckNew(<args>);
```

Asynchronous method invocation

- Entry methods are invoked by calling a C++ method on a chare's proxy:

```
CProxy_MyChare proxy = CProxy_MyChare::ckNew(<args>);

proxy.foo();
proxy.bar(5);
```

- Only one entry method executes on a chare at a time
- No ordering guarantees between entry method invocations

Hello World in Charm++

```
module hello {

  array [1D] Hello {
    entry Hello();
    entry void sayHi();
  };

};
```

Charm++ Tutorial: http://charmplusplus.org/tutorial/ArrayHelloWorld.html

Hello World in Charm++

```
module hello {

  array [1D] Hello {
    entry Hello();
    entry void sayHi();
  };

};
```

```
void Hello ::sayHi() {
  CkPrintf("Hello from chare %d on processor %d.\n”, thisIndex,
CkMyPe());
}
```

Charm++ Tutorial: http://charmplusplus.org/tutorial/ArrayHelloWorld.html

Hello World in Charm++

```
module hello {

  array [1D] Hello {
    entry Hello();
    entry void sayHi();
  };

};
```

```
Main::Main(CkArgMsg* msg) {
  numElements = 5; // number of elements

  CProxy_Hello helloArray =
          CProxy_Hello::ckNew(numElements);

  helloArray.sayHi();
}
```

```
void Hello ::sayHi() {
  CkPrintf("Hello from chare %d on processor %d.\n”, thisIndex,
CkMyPe());
}
```

Charm++ Tutorial: http://charmplusplus.org/tutorial/ArrayHelloWorld.html

Sending data through entry methods

- You can pass basic C++ data types directly
- C++ STL data structures can be passed by including pup_stl.h
- Arrays of basic types can be passed like this:

```
entry void foobar(int length, int data[length]);
```

```
MyChare::foobar(int length, int* data) {
    // ... foobar code ...
}
```

Questions?

UNIVERSITY OF MARYLAND

Abhinav Bhatele

5218 Brendan Iribe Center (IRB) / College Park, MD 20742

phone: 301.405.4507 / e-mail: bhatele@cs.umd.edu