Lecture 7: Task-based Models and Charm++

Abhinav Bhavele, Department of Computer Science
Announcements

• Group project grading

• Peer evaluation: you are given $100 that you will allocate as a performance bonus to your group members based on your assessment of their contributions to the project (you cannot keep any money for yourself but you cannot donate to charity)
Announcements

• Interim and final report
  • Provide details about the project: serial algorithm, parallel algorithm, languages being used
  • Deliverables and metrics for success
  • Contributions of individual group members

• Final presentation
  • Introduce your project so that it is understandable by a CS audience
  • Present what you are implementing or evaluating (serial / parallel algorithms)
  • Progress so far, and results (performance / performance analysis)
Summary of last lecture

- Performance analysis
  - Identify performance bottlenecks, anomalies
  - Measurement, analysis, visualization tools
- Tracing and profiling
- Calling context trees, graphs
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
Charm++: Key principles

- Programmer decomposes data and work into objects (called *char*es)
  - Decoupled from number of processes or cores
- 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()`
- Asynchronous message-driven execution
Hello World in Charm++

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

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

```cpp
mainmodule 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](http://charmplusplus.org/tutorial/ArrayHelloWorld.html)
Hello World in Charm++

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

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

Main::Main(CkArgMsg* msg) {
    numObjects = 5; // number of objects
    CProxy_Hello helloArray = CProxy_Hello::ckNew(numObjects);
    helloArray.sayHi();
}
```

Charm++ Tutorial: [http://charmplusplus.org/tutorial/ArrayHelloWorld.html](http://charmplusplus.org/tutorial/ArrayHelloWorld.html)
Compiling a charm program

• Charm translator for .ci file
  • Generates charm_hello.decl.h and charm_hello.def.h

```bash
charmc hello.ci
```

• C++ code:

```bash
charmc -c hello.C
charmc -o hello hello.o
```
Chare arrays

- User can create indexed collection of data-driven objects

  \[
  \text{CProxy\_Hello \ helloArray = CProxy\_Hello::ckNew(numElements);}
  \]

- Different kinds: 1D, 2D, 3D, …

- Mapping of array elements (objects) to hardware resources handled by the runtime system (RTS)
Object-based virtualization and over-decomposition

- User programs in terms of chares or objects
Charm scheduler and message queue

- An object is scheduled by the runtime scheduler only when a message for it is received
- Facilitates adaptive overlap of computation and communication
Cost of creating more objects?

- Context switch overhead
- Cache performance
- Memory overhead
- Fine-grained messages
mainmodule hello {
    readonly CProxy_MyMain myMainProxy;
    readonly int numChares;

    mainchare MyMain {
        entry MyMain(CkArgMsg *msg);
        entry void done(void);
    };

    array [1D] Hello {
        entry Hello(void);
        entry void sayHi(int);
    };
};
Hello world: MyMain class

/*readonly*/ CProxy_MyMain myMainProxy;
/*readonly*/ int numChars;

class MyMain: public CBase_MyMain {
    public:
    MyMain(CkArgMsg* msg) {
        numChars = atoi(msg->argv[1]); // number of elements

        myMainProxy = thisProxy;
        CProxy_Hello helArrProxy = CProxy_Hello::ckNew(numChars);

        helArrProxy[0].sayHi(20);
    }

    void done(void) {
        ckout << "All done" << endl;
        CkExit();
    }
};
#include "hello.decl.h"
extern /*readonly*/ CProxy_MyMain myMainProxy;

class Hello: public CBase_Hello {
  public:
    Hello(void) { }

    void sayHi(int num) {
      cout << "Chare " << thisIndex << " says Hi!" << num << endl;

      if(thisIndex < numChars-1)
        thisProxy[thisIndex+1].sayHi(num+1);
      else
        myMainProxy.done();
    }
};

#include "hello.def.h"
Proxy class

- Runtime needs to pack/unpack data and also figure out where the chare is
- Proxy class generated for each chare class
  - Proxy objects know where the real object is
  - Methods invoked on these proxy objects lead to messages being sent to the destination processor
Broadcast, barrier, and reduction

- Entry method called on a chare proxy without subscript is essentially a broadcast:
  ```
  charProxy.entryMethod()
  ```

- Barrier: reduction without arguments:
  ```
  contribute();
  ```

- Reduction with arguments:
  ```
  void contribute(int bytes, const void *data, CkReduction::reducerType type);
  ```
Callback for reduction

• Where does the output of the reduction go?

• Use a callback object known as a reduction client

CkCallback* cb = new CkCallback(CkIndex_myType::myReductionFunction(NULL), thisProxy);
contribute(bytes, data, reducerType, cb);

• Use the reduction data in the callback:

```cpp
void myType::myReductionFunction(CkReductionMsg *msg) {
    int size = msg->getSize() / sizeof(type);
    type *output = (type *) msg->getData();

    ...
}
```

Questions?

Abhinav Bhatele
5218 Brendan Iribe Center (IRB) / College Park, MD 20742
phone: 301.405.4507 / e-mail: bhatele@cs.umd.edu