Summary of last lecture

• Shared-memory programming and OpenMP
• Fork-join parallelism
• OpenMP vs MPI: ease of programming, performance
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++: Global view

**Global Object Space**

**User View**

**System View**

**CPU A**

**CPU B**

**CPU C**
Key Principles

• Programmer decomposes data and work into objects
  • 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()

• Message-driven execution
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
Over-decomposition and virtualization

- Create lots of “small” objects per physical core
  - Objects grouped into arrays: 1D, 2D, …
- System assigns objects to processors and can migrate objects between physical resources
- Facilitates automatic load balancing
Questions

The Charm++ Programming Model

- What are some of its limitations?
- Could we talk through an example where using the structured dagger would be relevant?
- Can you still have bottlenecks with message passing? What would an example of this look like?
Questions
Parallel Programming with Migratable Objects: Charm++ in Practice

- Is there an alternative to the checkpointing discussed? Some way to add redundancy to the parallel computation so that it is fault tolerant? Or is this generally not worth doing? Is there anything complex about the implementation of the checkpointing process?

- Are the examples (e.g., Barnes-Hut simulation) standard benchmarks in HPC literature, or selected specially for making Charm++ look good? If these are standard, why are they standard? Just due to their popularity?
Questions?

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