Charm++, what’s that?!
Les Mardis du dev’

François Tessier - Runtime team

October 15, 2013
Outline

1 Introduction

2 Charm++

3 Basic examples

4 Load Balancing

5 Conclusion
## Parallel programming

- **Decomposition**
  - What to do in parallel
- **Mapping**
  - Which processor execute each task
- **Scheduling**
  - The order
- **Machine dependent expression**
  - Express the above decisions for the particular parallel machine
Scalable execution of parallel applications

- Cluster computing
- Number of cores is increasing
- But **memory per core** is decreasing (or increasing slowly)
- Applications need to communicate more and more...

Number of cores and amount of memory per core of the first ranking cluster in the Top500 list since 1993
How to communicate between nodes?

- Message passing paradigm
- Inter-process communication
- MPI, SOAP, Charm++

Figure: Communication from process 1 to process 4 with a message passing interface
Wat? Charm++?

WTF iz that?
Presentation

- Developed at the PPL (Urbana-Champaign, IL)
- High-level abstraction of parallel programming
- C++, Python
Presentation

- Developed at the PPL (Urbana-Champaign, IL)
- High-level abstraction of parallel programming
- C++, Python

How to define Charm++?

- Object-oriented
  - Based on the C++ programming language
  - Objects called chares (contain data, send and receive messages, perform task)
Presentation
- Developed at the PPL (Urbana-Champaign, IL)
- High-level abstraction of parallel programming
- C++, Python

How to define Charm++?
- Object-oriented
  - Based on the C++ programming language
  - Objects called chares (contain data, send and receive messages, perform task)
- Asynchronous message passing
  - Chares exchange messages to communicate
  - Messages sent/received asynchronously to the code execution
Charm++

Presentation
- Developed at the PPL (Urbana-Champaign, IL)
- High-level abstraction of parallel programming
- C++, Python

How to define Charm++?
- Object-oriented
  - Based on the C++ programming language
  - Objects called chares (contain data, send and receive messages, perform task)
- Asynchronous message passing
  - Chares exchange messages to communicate
  - Messages sent/received asynchronously to the code execution
- Parallel
  - Chunks of code
  - Messages
Developed at the PPL (Urbana-Champaign, IL)
High-level abstraction of parallel programming
C++, Python

Object-oriented
- Based on the C++ programming language
- Objects called chares (contain data, send and receive messages, perform task)

Asynchronous message passing
- Chares exchange messages to communicate
- Messages sent/received asynchronously to the code execution

Parallel
- Chunks of code
- Messages

Programming paradigm
- Way of writing program
- Features and structures added on top of C++
What is a Charm++ Program?

- Collection of communicant chare objects, included the "main" chare (global object space)
- No importance about the number of PU or the type of interconnect (RTS)
- Send a message by calling another chare’s entry point function (reception point)
  - returns immediately from the perspective of the calling chare

**Figure:** User’s view of a Charm++ Application
(Credits: http://charm.cs.illinois.edu)
The Charm++ Compilation and the Runtime System

- First strength: it works!
- Application written as a collection of communicating objects
- Compilation, execution: Specific target platform, physical resources (-np)
- RTS manages the details of the physical resources...
- ... and can take some decisions about:
  - Mapping chare objects to physical processors
  - Load-balancing chare objects
  - Checkpointing
  - Fault-tolerance
  - ...

Figure: Compilation Process for a Chare Class (Credits: http://charm.cs.illinois.edu)
1 Introduction
2 Charm++
3 Basic examples
4 Load Balancing
5 Conclusion
Listing 1: Hello.h

```cpp
#ifndef __MAIN_H__
#define __MAIN_H__

class Main : public CBase_Main {

public:
    Main(CkArgMsg* msg);
    Main(CkMigrateMessage* msg);

};

#endif // __MAIN_H__
```

Listing 2: Hello.ci

```charm
mainmodule main {

    mainchare Main {
        entry Main(CkArgMsg* msg);
    }

};
```

Listing 3: Hello.C

```c
#include "main.decl.h"
#include "main.h"

// Entry point of Charm++ application
Main::Main(CkArgMsg* msg) {

    // Print a message for the user
    CkPrintf("Hello World!\n");

    // Exit the application
    CkExit();
}

// Constructor needed for chare object
// migration (ignore for now)
// NOTE: This constructor does not need
// to appear in the ".ci" file
Main::Main(CkMigrateMessage* msg) { }

#include "main.def.h"
```

François Tessier

Charm++ 13 / 25
Listing 4: 1Darray.ci

mainmodule hello {
readonly CProxy_Main mainProxy;
readonly int nElements;

mainchare Main {
   entry Main(CkArgMsg *m);
   entry void done(void);
};
array [1D] Hello {
   entry Hello(void);
   entry void SayHi(int from);
};
}

Listing 5: 1Darray.C

#include <stdio.h>
#include "hello.decl.h"

/* readonly */
CProxy_Main mainProxy;
/* readonly */
int nElements;

/* mainchare */
class Main : public CBase_Main
{
public:
Main(CkArgMsg* m)
{
   if(m->argc >1 )
      nElements=atoi(m->argv[1]);
   delete m;

   CkPrintf("Run on %d processors for %d el.
", CkNumPes(),nElements);
   mainProxy = thisProxy;
   CProxy_Hello arr = CProxy_Hello::ckNew(nElements);
   arr[0].SayHi(-1);
};

void done()
{
   CkPrintf("All done\n");
   CkExit();
};

/* array [1D]*/
class Hello : public CBase_Hello
{
public:
Hello()
{
   CkPrintf("Hello%d created\n",thisIndex);
}
Hello(CkMigrateMessage *m) {}

void SayHi(int from)
{
   CkPrintf("\"Hello\" from Hello chare # %d on %d processor %d \"(told by %d).
", thisIndex, CkMyPe(), from);
   if (thisIndex < nElements-1)
      thisProxy[thisIndex+1].SayHi(thisIndex);
   else
      mainProxy.done(); // Done!
}
$ ./charmrun +p3 ./hello 10
Running "Hello World" with 10 elements using 3 processors.
"Hello" from Hello chare # 0 on processor 0 (told by -1).
"Hello" from Hello chare # 1 on processor 1 (told by 0).
"Hello" from Hello chare # 2 on processor 2 (told by 1).
"Hello" from Hello chare # 3 on processor 0 (told by 2).
"Hello" from Hello chare # 4 on processor 1 (told by 3).
"Hello" from Hello chare # 5 on processor 2 (told by 4).
"Hello" from Hello chare # 6 on processor 0 (told by 5).
"Hello" from Hello chare # 7 on processor 1 (told by 6).
"Hello" from Hello chare # 9 on processor 0 (told by 8).
"Hello" from Hello chare # 8 on processor 2 (told by 7).
Load-balancing

- Balance the load between the processing units
- Goal: Optimize the CPU consumption, temperature, or other metrics...
Charm++ - Load balancing

Initial state

Processor 1

Processor 3

Processor 2

Processor 4
Charm++ - Load balancing

Initial state

<table>
<thead>
<tr>
<th>Processor 1</th>
<th>Processor 3</th>
<th>Processor 2</th>
<th>Processor 4</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1.png" alt="Initial state" /></td>
<td><img src="image2.png" alt="Initial state" /></td>
<td><img src="image3.png" alt="Initial state" /></td>
<td><img src="image4.png" alt="Initial state" /></td>
</tr>
</tbody>
</table>

LB

<table>
<thead>
<tr>
<th>Processor 1</th>
<th>Processor 3</th>
<th>Processor 2</th>
<th>Processor 4</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1.png" alt="LB" /></td>
<td><img src="image2.png" alt="LB" /></td>
<td><img src="image3.png" alt="LB" /></td>
<td><img src="image4.png" alt="LB" /></td>
</tr>
</tbody>
</table>
Listing 6: NewLB.C

CreateLBFunc_Def(NewLB, "Create\_NewLB")

cpy void NewLB::work(BaseLB::LDStats* stats)
{
    CkPrintf("%d\ncharacters\executed\non\%d\nprocs\n", 
        stats->n_obj, stats->nprocs());
    // Processor array
    ProcArray *parr = \new\ ProcArray(stats);
    // Object graph
    ObjGraph *ogr = \new\ ObjGraph(stats);
    std::vector<Vertex>::iterator v_it;
    std::vector<Edge>::iterator e_it;

    for (v_it=ogr->vertices.begin(); v_it!=ogr->vertices.end(); ++v_it) {
        double load = (*v_it).getVertexLoad();
        for (e_it=(*v_it).sendToList.begin(); e_it!=(*v_it).sendToList.end(); ++e_it) {
            int from = (*v_it).getVertexId();
            int to = (*e_it).getNeighborId();
            if (from != to) {
                CkPrintf("%d\nmsgs\sent\nfrom\%d\nto\%d\n",
                    (*e_it).getNumMsgs(), from, to);
            }
        }
    }

    ogr->vertices[12].setNewPe(2);
    ogr->convertDecisions(stats);
}
Load-balancing

- For iterative applications
- `charmrun +p64 ./App +balancer newLB +LBDebug 1 [...]`
- Objects can migrate using a pup function (Pack and Unpack)

Listing 7: PUP function

```c++
class MyClass {
    public:
    int a;
    int *b;

    MyClass() {
        int a = 42;
        int *b = (int *)malloc(a*sizeof(int));
    }

    void pup(PUP::er &p) {
        p|a;
        PUPArray(p, b, a);

        if (p.isUnpacking()) {
            b = (int *)malloc(a*sizeof(int));
        }
    }
};
```
Load-balancing

- For iterative applications
- `charmrun +p64 ./App +balancer newLB +LBDebug 1 [...]`
- Objects can migrate using a pup function (Pack and Unpack)
- Several load balancers
  - GreedyLB
  - RefineLB
  - **Hierarchical load balancers**
  - Thermal aware load balancers...

![Diagram of load balancing]

Groups of chares assigned to cores

CPU Load

0 2 4 6 1 3 5 7
Load-balancing

- For iterative applications
- `charmrun +p64 ./App +balancer newLB +LBDebug 1 [...]`
- Objects can migrate using a `pup` function (Pack and Unpack)
- Several load balancers
  - GreedyLB
  - RefineLB
  - **Hierarchical load balancers**
  - Thermal aware load balancers...
Load-balancing

- For iterative applications
- `charmrun +p64 ./App +balancer newLB +LBDDebug 1 [...]`
- Objects can migrate using a `pup` function (Pack and Unpack)
- Several load balancers
  - GreedyLB
  - RefineLB
  - **Hierarchical load balancers**
  - Thermal aware load balancers...
### Load-balancing

- For iterative applications
- `charmrun +p64 ./App +balancer newLB +LBDebug 1 [...]`
- Objects can migrate using a `pup` function (Pack and Unpack)
- Several load balancers
  - GreedyLB
  - RefineLB
  - **Hierarchical load balancers**
  - Thermal aware load balancers...
kNeighbor

- Benchmarks application designed to simulate intensive communication between processes
- Experiments on PlaFRIM: 8 nodes with 8 cores on each (Intel Xeon 5550)
Outline

1. Introduction
2. Charm++
3. Basic examples
4. Load Balancing
5. Conclusion
To conclude...

- Charm++, message passing since 1984
- High-level abstraction of a parallel program
- Interesting features: load balancing, fault-tolerance, ...
- Useful tools: CharmDebug, Projections
- Used by some important applications: ChaNGa, NAMD, OpenAtom

Want to know more?

- Website: http://charm.cs.uiuc.edu/
- Tutorials: http://charm.cs.illinois.edu/tutorial/TableOfContents.htm
Hey you
Any questions?

Thanks for your attention!