

# Charm++

## Seminar: Languages for Scientific Computing

Florian Sehl

Rheinisch-Westfälische Technische Hochschule Aachen

30.01.2014

# table of contents

- 1 Charm++ Concept
  - Overview
- 2 Basic Charm++ Programming
- 3 Advanced Charm++ Programming
- 4 summary

# History

- Parallel Programming Laboratory at the University of Illinois
- first prototype Chare Kernel(1.0) late 1980s

# History

- Parallel Programming Laboratory at the University of Illinois
- first prototype Chare Kernel(1.0) late 1980s
- 1993: CHARM(4.0) included Charm++

# History

- Parallel Programming Laboratory at the University of Illinois
- first prototype Chare Kernel(1.0) late 1980s
- 1993: CHARM(4.0) included Charm++
- current version 6.5

# History

- Parallel Programming Laboratory at the University of Illinois
- first prototype Chare Kernel(1.0) late 1980s
- 1993: CHARM(4.0) included Charm++
- current version 6.5

# What is Charm++?

- object-oriented asynchronous message passing parallel programming paradigm
  - programming paradigm: Charm++ is a programming model

# What is Charm++?

- object-oriented asynchronous message passing parallel programming paradigm
  - programming paradigm: Charm++ is a programming model
  - parallel: for writing parallel applications

# What is Charm++?

- object-oriented asynchronous message passing parallel programming paradigm
  - programming paradigm: Charm++ is a programming model
  - parallel: for writing parallel applications
  - object-orientated: uses C++ programming language (Sequential code also in C, Fortran, ...)

# What is Charm++?

- object-oriented asynchronous message passing parallel programming paradigm
  - programming paradigm: Charm++ is a programming model
  - parallel: for writing parallel applications
  - object-orientated: uses C++ programming language (Sequential code also in C, Fortran, ...)
  - asynchronous message passing: chare objects communicate communicate by messages

# What is Charm++?

- object-oriented asynchronous message passing parallel programming paradigm
  - programming paradigm: Charm++ is a programming model
  - parallel: for writing parallel applications
  - object-orientated: uses C++ programming language (Sequential code also in C, Fortran, ...)
  - asynchronous message passing: chare objects communicate communicate by messages
- single workstation, a network of workstations, BlueGene/L, BlueGene/P, BlueGene/Q, Cray XT, XE and XK series

# What is Charm++?

- object-oriented asynchronous message passing parallel programming paradigm
  - programming paradigm: Charm++ is a programming model
  - parallel: for writing parallel applications
  - object-orientated: uses C++ programming language (Sequential code also in C, Fortran, ...)
  - asynchronous message passing: chare objects communicate communicate by messages
- single workstation, a network of workstations, BlueGene/L, BlueGene/P, BlueGene/Q, Cray XT, XE and XK series

# Program Structure

- Char is a process, an actor
- Application uses over-decomposition

# Program Structure

- Char is a process, an actor
- Application uses over-decomposition
- Chars are C++ Objects

# Program Structure

- Char is a process, an actor
- Application uses over-decomposition
- Chars are C++ Objects
- chare classes and entry methods in interface file „.ci“

# Program Structure

- Char is a process, an actor
- Application uses over-decomposition
- Chars are C++ Objects
- chare classes and entry methods in interface file „.ci“
- Charm++ runtime system (Charm RTS) maps the Chars to processing elements

# Program Structure

- Char is a process, an actor
- Application uses over-decomposition
- Chars are C++ Objects
- chare classes and entry methods in interface file „.ci“
- Charm++ runtime system (Charm RTS) maps the Chars to processing elements

# Program Structure

- Char is a process, an actor
- Application uses over-decomposition
- Chars are C++ Objects
- chare classes and entry methods in interface file „.ci“
- Charm++ runtime system (Charm RTS) maps the Chars to processing elements

# Charm++ Runtime System

- Mapping Chare Objects to Physical Processors
- Load-Balancing Chare Objects

# Charm++ Runtime System

- Mapping Chare Objects to Physical Processors
- Load-Balancing Chare Objects
- Routing of Message

# Charm++ Runtime System

- Mapping Chare Objects to Physical Processors
- Load-Balancing Chare Objects
- Routing of Message
- Fault-Tolerance

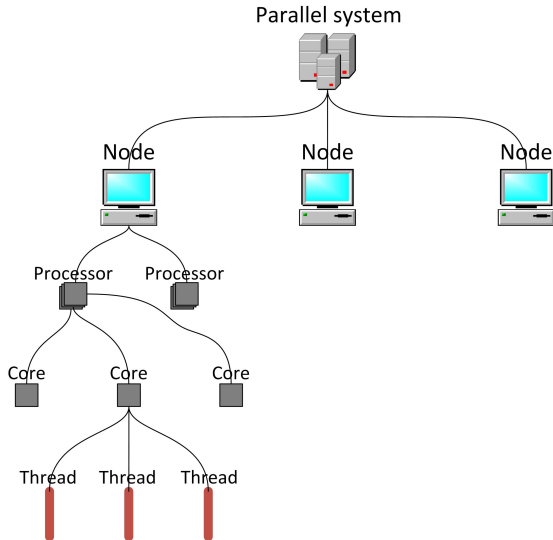
# Charm++ Runtime System

- Mapping Chare Objects to Physical Processors
- Load-Balancing Chare Objects
- Routing of Message
- Fault-Tolerance
- Dynamic Re-Allocation of Physical Resources

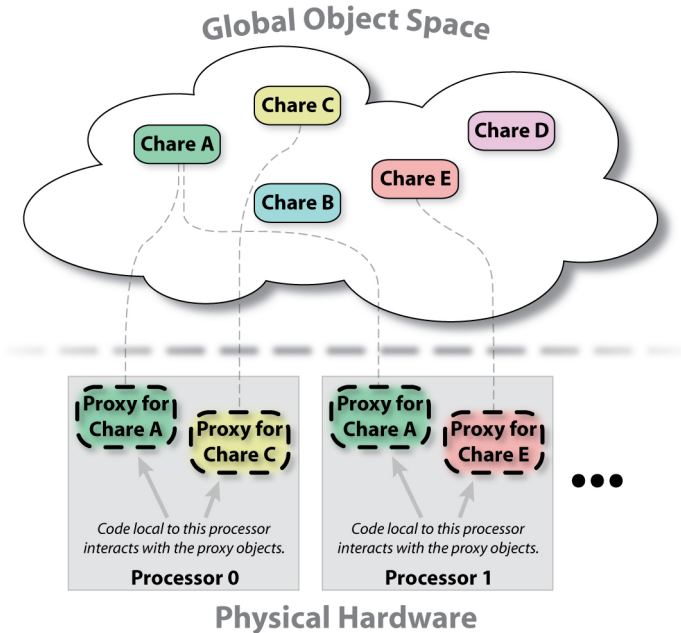
# Charm++ Runtime System

- Mapping Chare Objects to Physical Processors
- Load-Balancing Chare Objects
- Routing of Message
- Fault-Tolerance
- Dynamic Re-Allocation of Physical Resources

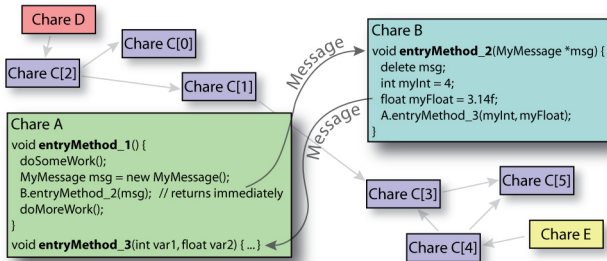
# node & processing elements



# char mapping



# chars



# Fibonacci

.ci:

```
mainmodule fib {  
    mainchare Main {  
        entry Main(CkArgMsg* m);  
    };  
};
```

.cc:

```
struct Main : public CBase_Main {  
    Main(CkArgMsg* m) { CProxy_Fib::ckNew(atoi(m->a  
};
```

# Fibonacci

.ci:

```
chare Fib {
    entry Fib(int n, bool isRoot, CProxy_Fib parent) {
    entry void calc(int n) {
        atomic {
            CProxy_Fib::ckNew(n - 1, false, thisProxy);
            CProxy_Fib::ckNew(n - 2, false, thisProxy);
        }
        when response(int val)
            when response(int val2)
                atomic { respond(val + val2); }
    };
    entry void response(int val);
};
```

# Fibonacci

.cc:

```
struct Fib : public CBase_Fib {  
    Fib_SDAG_CODE  
    CProxy_Fib parent; bool isRoot;  
  
    Fib(int n, bool isRoot_, CProxy_Fib parent_)  
        : parent(parent_), isRoot(isRoot_) {  
        calc(n);  
    }  
};
```

# Fibonacci

```
void respond(int val) {  
    if (!isRoot) {  
        parent.response(val);  
        delete this;  
    } else {  
        CkPrintf("Fibonacci_number_is: %d\n", val);  
        CkExit();  
    }  
}
```

# Hello Array

.ci:

```
array [1D] Hello {  
    entry Hello(void);  
    entry void SayHi(int hiNo);  
};
```

# Hello Array

.C:

```
class Hello : public CBase_Hello {
public: Hello(){
    CkPrintf("Hello_%d_created\n",thisIndex);
}
Hello(CkMigrateMessage *m) {}
void SayHi(int hiNo){
    CkPrintf("Hi [%d]_from_element_%d\n",hiNo,thisIndex);
    if (thisIndex < nElements-1)
        thisProxy[thisIndex+1].SayHi(hiNo+1);
    else
        mainProxy.done();
}
};
```

# Charm++ Runtime System

- Serialization using Pack and UnPack (PUP) Framework.
- Load Balancing

# Charm++ Runtime System

- Serialization using Pack and UnPack (PUP) Framework.
- Load Balancing
- Intercepting Messages via Delegation

# Charm++ Runtime System

- Serialization using Pack and UnPack (PUP) Framework.
- Load Balancing
- Intercepting Messages via Delegation
- Meta Programming with Templates

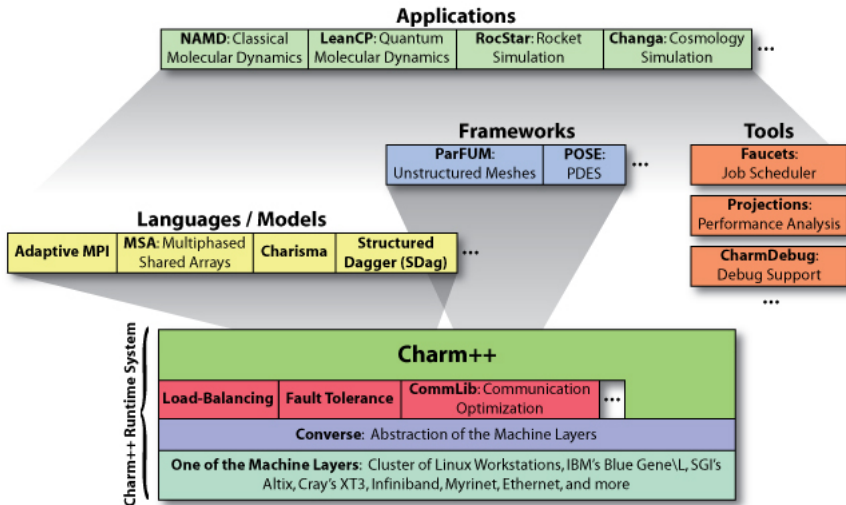
# Charm++ Runtime System

- Serialization using Pack and UnPack (PUP) Framework.
- Load Balancing
- Intercepting Messages via Delegation
- Meta Programming with Templates
- Dynamic Code Injection

# Charm++ Runtime System

- Serialization using Pack and UnPack (PUP) Framework.
- Load Balancing
- Intercepting Messages via Delegation
- Meta Programming with Templates
- Dynamic Code Injection

# Charm Hierarchy



# Zusammenfassung und Ausblick

- powerful extention to C++
- easy to get in

# Zusammenfassung und Ausblick

- powerful extention to C++
- easy to get in
- no benchmarks :-)

# Zusammenfassung und Ausblick

- powerful extention to C++
- easy to get in
- no benchmarks :-(
- Still improved and used (University of Illinois)

# Zusammenfassung und Ausblick

- powerful extention to C++
- easy to get in
- no benchmarks :-(
- Still improved and used (University of Illinois)