

Charm++

Universidade de Illinois em Urbana Champaign
Laxmikant Kalé & grupo

11 de novembro de 2009



Introdução

- sistema baseado em C++
- criado no final dos anos 80 (em C)
- *chares* (objetos concorrentes) formam unidades de execução
- modelo de execução orientado a eventos
 - chamadas assíncronas de métodos
 - preocupação com interoperabilidade: Converse

Chamadas Assíncronas

- *entry methods* definidos em arquivo de interface
- objeto principal inicia a execução do programa
- chamadas a construtores lançam objetos remotos
 - construtor tem que ser *entry method*

Tradutor de Interfaces

- geração automática de código que registra objetos
- geração de proxies
- suporte a threads (*user-level*) e futuros
 - chamadas síncronas podem ser feitas em threads auxiliares

Modelo de Execução

- objeto principal inicia a execução de programa
- sistema decide onde criar novos *chares*
- Chamada a `CkExit` termina execução do programa
 - chamada em um processador é suficiente

Entidades Charm++

- objetos sequenciais
- mensagens
 - controle de marshalling e unmarshalling, ...
- *chares*
- *chare arrays*
 - número de elementos definidos pelo programa
- *chare groups*
 - um elemento por processador
- *chare nodegroups*
 - um elemento por máquina



Arquivo de Interface

```
mainmodule Hello {  
  readonly CProxy_HelloMain mainProxy;  
  mainchare HelloMain {  
    entry HelloMain(); // implicit CkArgMsg * as argument  
    entry void PrintDone(void);  
  };  
  group HelloGroup {  
    entry HelloGroup(void);  
  };  
};
```

Arquivo .h

```
#include "Hello.decl.h" // Note: not pgm.decl.h
class HelloMain: public Chare {
  public:
    HelloMain(CkArgMsg *);
    void PrintDone(void);
  private:
    int count;
};
class HelloGroup: public Group {
  public:
    HelloGroup(void);
};
```



Arquivo .C

```
#include "pgm.h"
CProxy_HelloMain mainProxy;
HelloMain::HelloMain(CkArgMsg *msg) {
    delete msg;
    count = 0;
    mainProxy=thishandle;
    CProxy_HelloGroup::ckNew(); // Create a new "HelloGroup"
}
void HelloMain::PrintDone(void) {
    count++;
    if (count == CkNumPes()) { // Wait for all group members
        CkExit();
    }
}
HelloGroup::HelloGroup(void) {
    ckout << "Hello World from processor " << CkMyPe() << endl;
    mainProxy.PrintDone();
}
```



Métodos *entry*

- podem ter modificadores em sua declaração
 - threaded
 - sync
 - exclusive
 - expedited
 - python
 - ...

Passagem de Parâmetros

- métodos de marshalling (Pup) podem ser redefinidos pelo programador
- passagem de valores sequenciais é sempre por valor
- mas C++ não reconhece esse tipo de passagem para arrays:
 - cópia na origem
 - se necessário reter valores no destino, aplicação tem que fazer cópia



Mensagens

- programador controla criação de estrutura a ser enviada
- classes pré-definidas para parte do trabalho
- mensagens podem ser escalonadas com diferentes políticas
 - CK_QUEUEING_FIFO
 - CK_QUEUEING_LIFO
 - ... outras que dependem de campo de prioridade na msg

Arrays

- um array de chares é uma coleção

```
array [1D] A { entry A(parameters1);  
    entry void someEntry(parameters2);  
};
```

- identificador CkArrayID descreve array inteiro
- identificador CkArrayIndex descreve elementos individuais
- novos elementos podem ser inseridos dinamicamente



Comunicação com Arrays

- chamada individual de método: `a1[i].someEntry()`
- chamada coletiva de método: `a1.someEntry()`
- chamada a método com atributo `createhere` ou `createhome` causa criação de novo elemento se não existir



Grupos de Chares

- criação de um chare por processador
- chamada individual de método:
`g1[processador].someEntry()`
- chamada coletiva de método: `g1.someEntry()`

Migração

- todos os chares, exceto grupos, podem ser migrados dinamicamente
 - migração ocorre no momento em que chare está passivo



Estratégias

- um conjunto de estratégias de balanceamento está disponível no sistema
- outras estratégias podem ser programadas
- grupo de objetos LBDatabase armazena informação sobre carga em cada processador