Introdução a Charm++

Dennis José da Silva (9176517)

Instituto de Matemática e Estatística
da
Universidade de São Paulo

Professor: Prof. Dr. Alfredo Goldman
Monitor: Ms. Eng. Marcos Amarís González

São Paulo, Julho de 2015

Resumo

Silva, D. J. **Introdução a Charm++**. 2015. Monografia (Mestrado) para disciplina MAC5742 Programação Paralela e Distribuída - Instituto de Matemática e Estatística, Universidade de São Paulo, São Paulo, 2015.

Charm++ é uma ferramenta de programação para computação de alto desempenho desenvolvida pela universidade de Illinois que busca aumentar a produtividade de desenvolvedores que trabalham com esse tipo de aplicação. Charm++ utiliza fortemente conceitos como *Over-Decomposition*, *Assynchronous Message-Driven Execution* e *migratability* que com ajuda de um sistema RTS (*Run-Time System* fornece diversas funcionalidades para seus desenvolvedores como balanceamento de carga, tolerância a falha, reuso e modularidade. Todas essas funcionalidades podem fazer uma aplicação melhorar seu desempenho de aplicações em relação a aplicações que utilizam ferramentas já conhecidas como MPI. Este trabalho tem como objetivo introduzir os principais conceitos utilizados no design de Charm++ realizando uma comparação histórica das feramentas e suas características e como elas se relacionam com as decisões de design do Charm++, demonstrar os componentes do RTS e como eles trabalham juntos para fornecer as funcionalidades a todo o ecosistema do Charm++ mostrar como uma aplicação é desenvolvida passo a passo, demonstrando o sistema de compilação os arquivos que são gerados e como liga-los para montar uma aplicação Charm++ por meio de duas aplicações uma aplicação "Hello World"que mostrar os componentes básicos de qualquer aplicação Charm++ e uma que calcula a multiplicação de duas matrizes quadradas. Também serão analisados duas aplicações desenvolvidas pela equipe de desenvolvimento do Charm++ que obtiveram bom resultados, serão mostrados gráficos de dados considerados importantes e Também serão levantados as características dos aplicativos que levaram o mesmo a obter um bom resultado. Por fim uma breve conclusão que avalia as características do Charm++ e a concluir as vantagens de utiliza-lo e qual situação que essas vantagens são evidentes.

Palavras-chave: Charm++, Sistemas de alto desempenho, MIMD.

Abstract

Silva, D. J. **Introduction to Charm++**. 2015. Document (Master) to MAC5742 Parallel and Distribution System subject - Institute of Mathematics and Statistic, University of São Paulo, São Paulo, 2015.

Charm++ is a tool to high performace computing programming developed by Illinois University with the goal of increase developer wich works with HPC produtivity. Charm++ strongly uses Over-Decomposition, Assynchronous Message-Driven Execution and migrability concept and the RTS (RunTime System) to provide functionalities like load balance, fault tolerance, reuse and modularity to Charm++ application developers. All these functionalities may increase application performace over same applications wich uses well known tools like MPI. This document goal is introduce the main concepts used on the charm++ design trough a historical comparation of HPC programming tools and its features with Charm++ design decision, it shows the RTS components and how they work together to provide the funcinalities to all Charm++ system. It also show how to develop Charm++ application step-by-step introduce the charm++ compilation system and the files it generates and how these files are join together to make the Charm++ application, these funcinalities are shown trough a "Hello World"application which introduce all Charm++ application components and a square matrix multiplication application wich shows a message passing from Charm++ objects and broadcasting. Then the document shows Charm++ applications developed by the Charm++ team which got good results and the Charm++ features used in these applications that make them improved the performace all these will be shown by graphics of specific attributes that got improved. Finally this document bring a general Charm++ conclusion which evaluates the Charm++ features and attributes and when Charm++ should be used.
Keywords: Charm++, High Performace Computing, MIMD.

Sumário

Lista de Figuras v

1 Introdução 1
1.1 Objetivos . . . 1
1.2 Organização do Trabalho . . . 2

2 Fundamentos 3
2.1 O que é Charm++ ? . . . 3
2.2 Contexto histórico . . . 3
2.3 características e atributos . . . 5
2.4 RTS - RunTime System . . . 6

3 Programação 9
3.1 Componentes de uma aplicação Charm++ . . . 9
3.2 Hello World . . . 10
3.3 Multiplicação de Matrizes . . . 12

4 Aplicações 19
4.1 LeanMD . . . 19
4.2 LULESH . . . 20

5 Conclusões 22

Referências Bibliográficas 23

Lista de Figuras

2.1 Linha do tempo das principais tecnologias que podem ser relacionadas ao Charm++. . . . 4
2.2 Visão do desenvolvedor e do sistema e sua relação com o RTS (imagem modificada de PPL (2015g)). 7
2.3 Visão simplificada das partes integrantes do Charm++. Figura retirada de PPL (2015g) 8

3.1 Processo de compilação de um *chare*. Figura retirada de PPL (2015b) 9

4.1 comportamento do aplicativo LeanMD em relação ao balanceamento carga adaptativo. Imagem retirada de Acun *et al.* (2014) 20
4.2 Desempenho das aplicação LULESH comparado com suas versões que utilizam AMPI. Figura retirada de Acun *et al.* (2014) 21

Capítulo 1

Introdução

Charm++ é uma ferramenta para programação de alto desempenho desenvolvido por um grupo de pesquisadores do laboratório PPL (*Parallel Programming Laboratory*) da universidade de Illinois. Charm++ foi desenvolvido em cima da linguagem de programação C++ utilizando seus conceitos de programação orientada a objetos tendo como objetivo fornecer Portabilidade eficiente, tolerância a latência, Balanceamento de carga dinâmico, Reuso e Modularidade (**?**).

Charm++ busca fornecer essas características por meio de troca de mensagens entre objetos chamados *chare*, esses objetos são implementados pelo desenvolvedor e depois gerenciados por um sistema RTS (*Runtime System*) que é responsável por controlar a aplicação do usuário, distribuindo esses objetos nas PE (*Processing Element* - unidades de processamentos) disponíveis, além disso o RTS fornece diversas funcionalidades como tratar da comunicação dos *chares*, balanceamento de carga, Re-alocação de recursos dinamicamente, controle de tolerância a falha e ponto de restauração (*checkpoint*). Esse modelo então abstrai código que está sendo executado em processadores na mesma máquina (memória local) e o processamento que está ocorrendo em outras máquinas (memória compartilhada), tirando assim a responsabilidade do programador de criar mecanismo de controle para comunicação entre máquinas e código rondando em um modelo *multithreading* e passa para o RTS que trabalha com o balanceamento de carga para distribuir as tarefas entre os PEs.

Uma das caracteristicas importantes que o Charm++ possuí é a migrabilidade de seus objetos, cada *chare* é alocado em uma PE pelo RTS mas pode ser re-alocado de acordo com as informações coletadas pelo RTS, isso permite uma melhor distribuição de tarefas assim como identificação de falhas (se um *chare* parar de responder suas requisições) e a recuperação do sistema. Essa distribuição além de poder alocar melhor as tarefas dentro dos recursos disponível melhorando o desempenho também ajuda a controlar melhor a energia necessária para rodar aplicação em determinado intervalo de tempo, conforme mostrado no artigo escrito por Acun (Acun *et al.* (2014)), apesar da migrabilidade causar um queda repentina na execução do programa a melhor distribuição dos recursos geralmente melhora o tempo final de execução da aplicação.

Foram desenvolvidas algumas aplicações utilizando o Charm++ com sucesso, muitas vezes melhorando o desempenho de uma versão anterior, dentre elas LULESH utilizada para cálculos de hidrodinâmica e o *LeanMD* que simula dinâmica molecular (Acun *et al.* (2014)).

1.1 Objetivos

Essa monografia tem como objetivo introduzir programação em Charm++ assim como apresentar o design da ferramenta e seu contexto histórico. Portanto não serão abordados tópicos avançados, mas sim as principais características da ferramenta e como ela agregou as tecnologias que foram sendo desenvolvidas ao decorrer do tempo na área de computação de alto desempenho.

1.2 Organização do Trabalho

O trabalho foi divido em 4 capítulos, no capitulo 2 será apresentado uma visão histórica das tecnologias para programação concorrente e distribuída e suas relações com o desenvolvimento Charm++, também será apresentado apresentado características e atributos do Charm++.

No capitulo 3 será apresentado um tutorial como 2 exemplos de programas escritos em Charm++, um *Hello world* com o mínimo necessário para o desenvolvimento de uma aplicação em Charm++, e um exemplo de multiplicação de matrizes que demostra a utilização de comunicação entre processos e o *broadcasting* em Cham++.

No capitulo 4 serão apresentados 2 aplicações de alto desempenho usadas como *case* do Charm++ pelo seus desenvolvedores.

No capitulo 5 será apresentado as reflexões feitas apos o estudo do Charm++.

Capítulo 2

Fundamentos

Neste capitulo será apresentado uma visão geral de Cham++ assim como um perspectiva histórica das tecnologias que a influenciaram ou tiveram impacto na computação de alto desempenho. também serão discutidos as características e atributo do Charm++ assim como foi descrita por seus criadores e será discutido a arquitetura básica que o Charm++ utiliza para executar seus programas.

2.1 O que é Charm++ ?

Em resumo Charm++ é uma ferramenta para o desenvolvimento de aplicações de alto desempenho desenvolvida pelo PPL (*Parallel Programing Laboratory*) da universidade de Illinois nos Estados Unidos. Não foi encontrado a data da primeira publicação do Charm++, a referência mais antiga encontrada foi um artigo publicado em 1993 por Laxmikant V. Kale e Sanjeev Krishman (Kale e Krishnan (1993)), Kale continua contribuindo para o desenvolvimento da ferramenta até hoje, pelo projeto do Charm++ ter sido desenvolvida há mais de 22 anos sua definição acabou sendo modificada a decorrer do tempo.

No artigo do Kale de 1993 a definição dada para Charm++ foi uma linguagem de programação paralela portável e orientada a objetos, de fato a abordagem do Charm++ foi desenvolver um sistema de programação de alto desempenho onde houvesse troca de mensagens entre objetos que não estavam sendo executados em um mesmo processo, mas a definição de Charm++ ser uma linguagem pode causar confusão, por que Charm++ é programado utilizando uma mistura de C++ como uma linguagem declarativa onde são declaradas as interfaces dos objetos que poderão serem executados em processos diferentes (*Chares*). Em um artigo de 2014 (Acun *et al.* (2014)) cujo qual Kale é um dos co-autores Charm++ é descrito como um sistema de programação paralela que se apresenta como uma melhor definição já que Charm++ não é uma nova linguagem mais sim um ambiente para o desenvolvimento de aplicações de alto desempenho mas que no fundo são desenvolvido em linguagem C++, por outro lado Charm++ não pode ser considerado apenas um Framework ou biblioteca para C++ por que oferece todo um ecossistema para o desenvolvimento das aplicações, incluindo mas não limitando a uma linguagem declarativa para interface de objetos que podem ser executados em outros processos, um sistema de tempo de execução (*RunTime System - RTS*) que controla esses objetos e local onde são executados e bibliotecas de funções para C++ manipularem esses objetos.

2.2 Contexto histórico

Charm++ foi desenvolvido ha mais de 22 anos, portanto é necessário conhecer o seu contexto histórico para que melhor entender o seu design a figura 2.1 abaixo apresenta uma linha do tempo com as principais tecnologias e seu ano de lançamento.

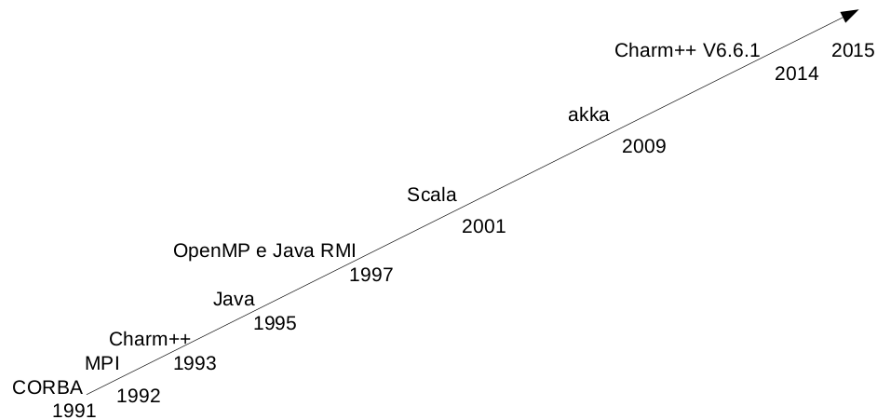

Figura 2.1: *Linha do tempo das principais tecnologias que podem ser relacionadas ao Charm++.*

Em 1993, já estavam sendo utilizadas linguagens orientadas a objetos assim como o processamento paralelo estava se destacando pelos avanços na área na década de 80, existindo inclusive máquinas paralelas comercialmente disponível, por exemplo, o iPSC /860 da Intel e o Paragon. Ainda nesta época existia diversos problemas de portabilidade por que os programas desenvolvidos para um computador paralelo não funciona em outro. Charm++ foi desenvolvido como uma tentativa de resolver esses problemas de compatibilidade e aproveitar o uso dos métodos de abstração da orientação objetos e aplica-los em um contexto de programa paralelo. (Kale e Krishnan (1993))

Observa-se que CORBA teve sua primeira publicação em 1991, CORBA é uma arquitetura padrão desenvolvida pelo Object Management Group para troca de mensagens entre sistemas heterogêneos e distribuído, CORBA introduziu a linguagem IDL (*Interface Definition language*) utilizada para integrar sistemas heterogêneos (OMG (2015)). A IDL pode ser relacionado ao fato do Charm++ adotar uma linguagem para descrever a interface de seus objetos já que os sistemas das duas ferramentas são bem semelhantes por que é desenvolvido um código fonte de descrição da interface que é então compilado para alguma outra linguagem (em C++ no caso do Charm++, CORBA pode utilizar diversas linguagens de programação).

MPI (*Message Passing Interface*) é um padrão para o desenvolvimento de aplicações que são executadas concorrentemente em um ambiente de memória distribuída lançada em 1992, MPI consiste de um conjunto de funções que descreve como deve ser feito a troca de mensagens entre dois processos de um mesmo programa (Meglicki (2015)). O Charm++ também utiliza o conceito de troca de mensagens, mas diferentemente do MPI, Charm++ tem uma abstração diferente em vez da troca de mensagens entre processos Charm++ possui o conceito de troca de mensagens entre objetos portanto ao invocar um método de um objeto *chare* o RTS (*RunTime System*) do Charm++ é encarregado de encontrar esse objeto e passar a mensagem que pode ser inclusive via rede.

Em 1995 Java foi lançado como uma linguagem orientada a objetos para uso geral. Java contribui com a popularização da orientação a objetos e a faze-lo um dos principais métodos de programação que é amplamente utilizado até os dias atuais em 1997 é lançado Java RMI (*Remote Method Invocation*) que possibilitou que Java tivesse uma arquitetura de objetos distribuídos (Oracle (2015)), ou seja, Poderia ser desenvolvido uma aplicação Java que tivesse comunicação de objetos em diferentes processos que poderia estar em diferentes máquinas, em um sistema semelhante aos objetos *chare* do Charm++.

Scala é uma linguagem de programação que junta os paradigmas de orientação a objetos e programação funcional, lançada em 2001 (ScalaLang (2015)) ganhou destaque ao popularizar a junção do paradigma funcional e orientado a objetos a programação web também conhecida por suportar a ferramenta akka lançada em 2009 que tem como base a implementação do modelo de atores na linguagem Java e Scala (Typesafe (2015)), apesar do modelo de atores ser conhecido hoje no mundo corporativo ele foi desenvolvido na década de 80 implementada na linguagem de programação Erlang. O modelo de atores apesar de não ser adotado em Charm++ o seu modelo de troca de mensagens entre atores foi uma forte influência as primeiras versões de Charm++

(Kale e Krishnan (1993)).

A última versão estável do Charm++ é a 6.6.1 e foi lançada em Dezembro de 2014 e o projeto continua tendo contribuições de diversas pessoas (PPL (2015c)).

2.3 características e atributos

Charm++ mantém sendo desenvolvido com base em 4 características que seus desenvolvedores buscam manter:

- Portabilidade eficiente
- Tolerância a latência
- Balanceamento de carga dinâmico
- Reuso e Modularidade

A portabilidade eficiente foi muito importante no inicio do desenvolvimento do Charm++ pela variedade e diferenças significativas nas maquinas paralelas da época, atualmente essa característica se mantêm muito importante inclusive impactando diretamente no reuso (outra característica importante do Charm+), a equipe do Charm++ se compromete em manter programas em C++ funcionando em qualquer máquina MIMD (*Multiple Instructions Multiple Data* sem a necessidade de alterar o código fonte do programa.

A Latência dentro do contexto de comunicação de sistemas é quantidade de tempo que uma mensagem enviada demora ser recebida por seu receptor, Charm++ busca minimizar o impacto de uma comunicação lenta (o que normalmente ocorre em sistemas distribuídos onde a comunicação é feita via rede). para minimizar a latência, Charm++ utiliza uma execução baseada em mensagem, ou seja, um processador é alocado para executar um processo somente quando a mensagem para o processo é recebida enquanto isso não ocorre o processador pode atender outros processos, esse mecanismo também cuida de processos que devem receber diversas mensagens e acaba ficando bloqueado, o sistema do Charm++ irá desbloquear esse processo assim que receber a qualquer uma das mensagens que esse processo deve receber, esses mecanismos ajudam Charm++ a esconder ou diminuir grandes latências para o sistema.

Balanceamento de Carga dinâmico é uma característica importante do Charm++ que o ajuda a melhorar o uso dos processadores disponível movendo certa quantidade de trabalho entre os processadores de acordo com a disponibilidade. Atualmente Charm++ possui estrategias de balanceamento de carga dinâmico e estático.

Charm++ fornece uma construção de módulos (*module*) e outras construções que facilitam a reutilização de módulos diferentes espalhados por diversos núcleos de processamento que podem trocar mensagens de uma maneira distribuída. (PPL (2015e))

Para manter essas características os designers do Charm++ desenvolveram funcionalidades em 3 atributos que cobrem as características propostas:

- *Over-decomposition*
- Execução baseada mensagens assíncronas
- *Migratability*

O atributo de *over-decomposition* refere-se a capacidade do software de ser devidos em um grande número de unidades de trabalho e dado, esse número é geralmente maior que o número de elementos de processamento em que o software ira ser executado trazendo assim uma abstração de elemento de processamento. Esse atributo auxilia o programador a desenvolver aplicações abstraindo o número de elementos de processamento em que essa aplicação vai ser executada, aumento assim sua produtividade, deixando a complexidade de atribuir a quantidade certa de trabalho ao

RTS (*RunTime System*) que possuí algoritmos para mapear essas unidades aos processadores disponíveis. Essa separação ainda pode melhor a utilização de cache já que as unidades representa um subdomínio do problema pequeno. Para isso Charm++ possuí objetos chamados *Chare* que representam essa unidade de trabalho e dados, um *chare* é uma classe C++ qualquer para cria-lo o desenvolvedor deve criar um arquivo de interface compila-lo e então realizar um “link” entre a classe C++ que terá o código que o *chare* executará com os arquivos que o compilador do Charm++ gera a partir da interface. Pelo *chare* ser uma classe qualquer ela pode ser estendida por meio de heranças e tem a habilidade de encapsulamento.

A abordagem de execução baseada em mensagem é desenvolvida em cima do fato que o envio e recebimento de mensagens geralmente são operações lentas, então a politica utiliza é fazer que um *chare* só pode ocupar o processador se ele já estiver recebido todas as mensagens que foram envidas-lhe, isso implica que quem envia a mensagem não seja bloqueado por uma mensagem de resposta de imediato, qualquer resposta será tratada como uma mensagem qualquer ao elemento que envio a primeira mensagem todo esse processo é tratado pelo RTS, esse sistema em cenário dinâmico onde um elemento que envia a mensagem precisaria esperar a resposta pode ser bem eficiente, usando melhor os elementos de processamento e ajudando a esconder a latência. Essa funcionalidade é fornecida por dois recursos no Charm++, *entry* e proxy, quando um *chare* é criado em uma aplicação Charm++, o RTS fornece um objeto especial chamado proxy que possibilita recuperar um objeto remoto qualquer no espaço global de *chares*, um *chare* é definido por meio de um arquivo de interface onde todos os métodos que podem ser invocados remotamente devem ser marcados como *entry* possibilitando assim que o compilador CHARMC gere os arquivos fonte necessários para a chamada remota desses métodos.

Migratability refere-se a habilidade de mover as unidades de trabalho e dados entre os elementos de processamento no Charm++ esse recurso é fornecido pelo RTS que inclusive pode mover essas unidades dinamicamente durante a execução de uma aplicação, esse recursos implica em diversas vantagens dentre balanceamento de carga dinâmico, recuperação do estado atual da aplicação, recuperação de algumas falhas fáceis e difíceis, controle de temperatura e redistribuição consciente de energia e maleabilidade no agendamento de tarefas. Essa funcionalidade é fornecida através do conjunto entre o PUP framework e o RTS, o PUP framework fornece a funcionalidade onde o programador pode implementar a serialização e deserialização de *chares* em bytes que o RTS usa para move-los entres os elementos de processamento. (Acun *et al.* (2014)).

2.4 RTS - RunTime System

O desenvolvimento de uma aplicação Charm++ consiste na criação de objetos denominados *chares* e na comunicação entre esses objeto, na arquitetura do Charm++ não é responsabilidade do desenvolvedor definir tipos de processador, tipos de conexão entre outros detalhes essa visão de uma aplicação em Charm++ é conhecida como visão do usuário, Por outro lado quando uma aplicação é compilada é definido o tipo de processador, a interconexão e outros detalhes de baixo nível ainda mais quando uma aplicação é executada diversos recursos são disponibilizados para a aplicação todo esse controle e gerenciamento desses recursos é feito pelo RTS e dentro do contexto do Charm++ esses procedimentos são chamados de visão do sistema e o RTS é um dos principais elementos, abaixo segue a imagem 2.2 que mostra esse relacionamento entre as visões e RTS.

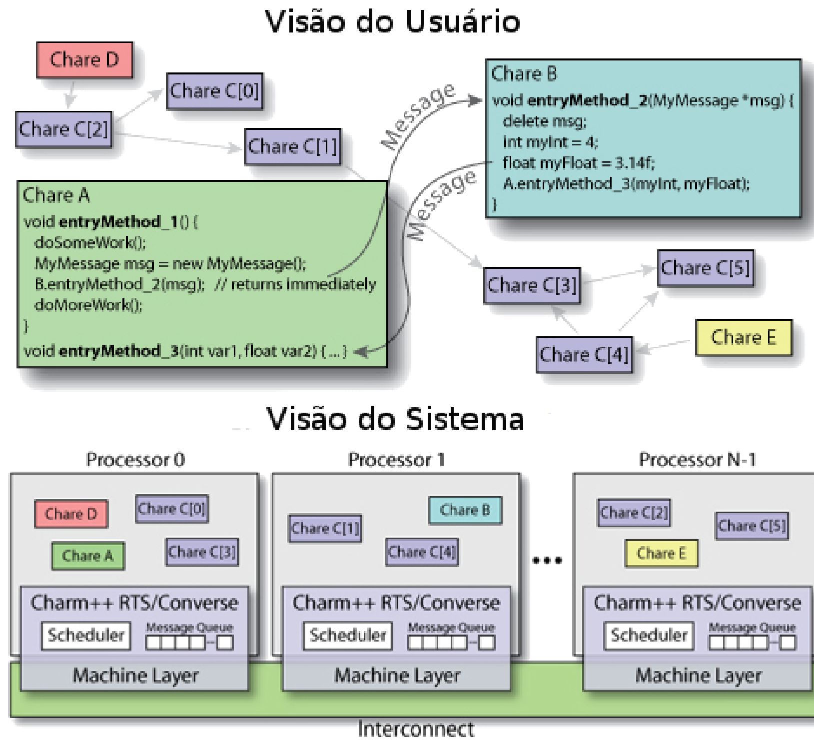

Figura 2.2: *Visão do desenvolvedor e do sistema e sua relação com o RTS (imagem modificada de PPL (2015g)).*

A imagem mostra que para o usuário ou desenvolvedor todos os *chares* estão disponível em meio mais abstrato, em uma especie de nuvem onde cada *chare* pode ser requisitado independentemente de máquina, conexão, se está no mesmo processador ou em outra máquina, por outro lado a visão do sistema conhece todos os recursos e distribui os *chares* de acordo com esses recursos. Toda a visão do sistema é gerenciado pelo RTS, todo esse gerenciamento do RTS implica em tomadas de decisões de funcionalidades do sistemas por exemplo:

- Mapeamento de *chares* em elementos de processamento físicos
- Balanceamento de carga dos objeto *chare*
- Routerização de mensagens
- Tolerância a falhas
- Realocação dinâmica de recursos físicos

Como exibido na imagem cada elemento de processamento tem seu próprio RTS em execução, cada RTS é responsável por realizar as operações locais, assim como operações em grupo onde o RTSs trocam mensagens entre si, por exemplo, para restauração, balanceamento de carga e recuperação a falhas.

O RTS no fundo significa o agrupamento de 4 componentes: Machine Layer, Converse, Message Queue e o Scheduler:

- **Machine Layer**: Código de baixo nível utilizado pelos componentes do RTS em alto nível, o Machine Layer fornece uma abstração dessas funcionalidades de baixo nível tal que o os códigos de nível superior podem ser desenvolvidos sem a necessidade de saber em qual hardwares ele serão executados

- **Converse**: É uma camada acima da componente Machine Layer que é responsável por abstrai-la e fornecer uma visão ainda mais abstrata para camadas acima, além disso Converse pode selecionar implementações da Machine Layer, por exemplo, casos o sistema esteja sendo executado em um computador que possui uma otimização para *broadcasting* a implementação da Machine Layer que usa essa otimização será escolhida.

- **Message Queue**: A fila de mensagens serve como um repositório de mensagens que armazena as mensagens conforme a mesma vai chegando e então as repassa para os *chare* que as foram endereçadas, cada elemento de processamento tem sua fila e os *chares* que estão nesse elemento de processamento tem acesso a fila. A fila pode ser simples funcionando como as fila FIFO ou podem ser filas mais complexas que levam em consideração prioridades definidas pelo usuário.

- **Scheduler**: O escalonador é um elemento do do componente Converse que é responsável por selecionar uma mensagem da fila (Message Queue) e coloca-la para executar, uma vez que uma mensagem começa a ser executada a mesma só sai do elemento de processamento ápos o seu termino com ou com a chamada manual de *ckSuspend*. Essa abordagem evita o uso de locks e controles de acessos por que existe apenas uma mensagem sendo executada por vez em um elemento de processamento.

A imagem 2.3 mostra uma versão simplificada, mas completa das partes integrantes do Charm++:

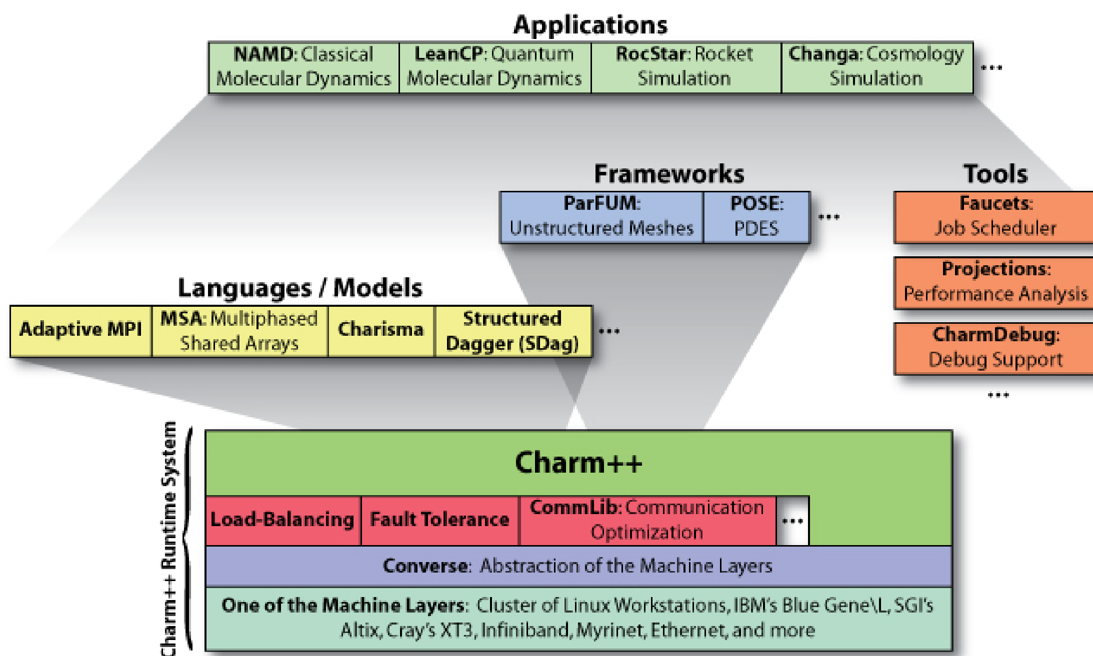

Figura 2.3: *Visão simplificada das partes integrantes do Charm++. Figura retirada de PPL (2015g)*

Charm++ constitui de diversas partes para formar o RTS além disso existem diversas ferramentas, frameworks e modelos construídos em cima do ecossistema do Charm++ mas essas ferramentas estão fora do escopo desse trabalho (PPL (2015g)).

Capítulo 3

Programação

Neste capitulo apresentará como desenvolver uma aplicação utilizando Charm++, explicará em linhas gerais os componentes de uma aplicação Charm++ e a interação desses componentes e então serão desenvolvidos 2 programas que apresentam algumas funcionalidades básicas do Charm++. O primeiro é uma versão básica que utiliza apenas um *chare*, o segundo é uma implementação de multiplicação de matrizes que utiliza *broadcasting* e por último será realizada uma comparação simples com uma versão do programa feito usando MPI com mensagens assíncronas.

3.1 Componentes de uma aplicação Charm++

Uma aplicação Charm++ é formada por objetos chamados *chare*, esses objetos são objetos C++ que podem ter métodos chamados *entry* que podem ser invocados em diferentes elementos de processamento. Para realizar a implementação de um *chare* o desenvolver precisa desenvolver um arquivo de interface (um arquivo de extensão .ci) que deve ser compilado pelo charmc (compilador do Charm++) que gerará arquivos de header que possui a declaração e a implementação das funcionalidades da invocação dos métodos remotos.

O arquivos de interface (.ci) são os arquivos onde o desenvolvedor declara as funcionalidades paralelas de seus *chares* por meio de uma linguagem declarativa própria do Charm++, essa linguagem é semelhante a C++ com algumas palavras chaves (*keywords*) a mais, inclusive é possível escrever trechos de código sequenciais em C++ em algumas construções desse arquivo de interface (PPL (2015f)). A compilação desse arquivo gerará dois arquivos com mesmo nome do arquivo de interface com a extensão ".decl.h"e ".def.h", o arquivo ".decl.h"é o arquivo de declaração que expõe as funcionalidades, ou seja, ele é equivalente ao arquivo ".h"das classes em C++, nele são declaradas as classes de proxy e a classe base do *chare*, portanto essa classe deve ser incluída no inicio do arquivo de implementação do *chare* para que o mesmo possa herdar o código base de sua implementação, o arquivo ".def.h"é o arquivo que tem a implementação das funções declaradas no arquivo ".h.decl"e o mesmo deve estar incluso no fim da classe de implementação do *chare* conforme mostrada na figura 3.1

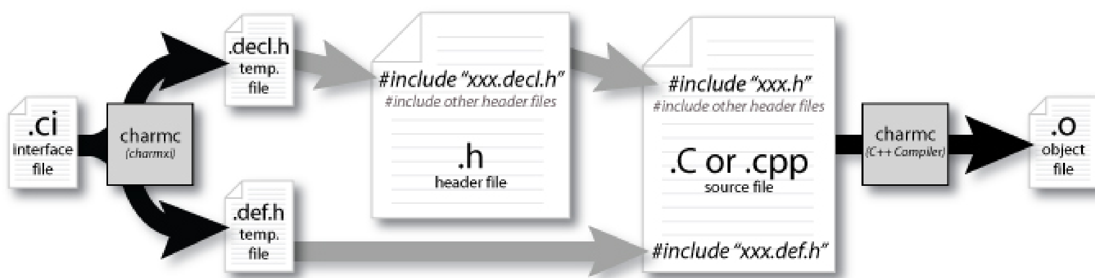

Figura 3.1: *Processo de compilação de um chare. Figura retirada de PPL (2015b)*

A imagem mostra o arquivo de interface ".ci"sendo processado pelo compilado charmc (compi-

lador do Charm++) que gerou as os arquivos "xxx.decl.h"e "xxx.def.h", o arquivo "xxx.decl.h"foi inserido no inicio do arquivo de cabeçalho da classe do *chare* e o arquivo "xxx.def.h"foi incluído no final do arquivo .C (extensão normalmente utilizada para indicar que o arquivo se refere a um *chare*) ou .cpp para que a implementação das declarações do arquivo "xxx.decl.h"sejam encontradas no arquivo de código fonte. Esse arquivo ".C"ou ".cpp"então é compilado pelo charmc que utiliza um compilador C++ que estiver configurado (por exemplo, g++) para compilar o código para o arquivo de objeto ".o"(PPL (2015b)).

3.2 Hello World

O primeiro exemplo, foi a implementação do tutorial da página de tutorial do Charm++ (PPL (2015d)). Esse exemplo apenas demonstrar o mínimo de código necessário para o desenvolvimento de uma aplicação em Charm++, ele é um exemplo de HelloWorld onde a mensagem "Hello World"será impressa pelo único *chare* que a aplicação terá. Abaixo segue a implementação do arquivo de interface do *chare*, neste arquivo é onde é indicado os métodos que serão usados para enviar mensagens para o *chare*, assim como a sua estrutura e outros detalhes, no caso da aplicação HelloWorld existirá apenas o *chare* principal é um único método neste caso o construtor da classe Main.

```
1 mainmodule main
2 {
3    mainchare Main
4    {
5       entry Main(CkArgMsg* msg);
6    };
7 };
```

Neste arquivo é definido o modulo principal (modulo é uma unidade de desenvolvimento do Charm++ semelhante a namespace em C++ ou package em Java, está fora do escopo deste trabalho) e dentro deste modulo existe a declaração do *chare* Main que é declarado como o *chare* principal, ou seja, ele contém o ponto de entrada para aplicação Charm++. O método Main é um método neste caso o construtor que pode ser chamado remotamente, note que ele foi declarado com a palavra chave entry indicado que ele pode ser chamado remotamente.

O arquivo abaixo refere-se a classe de cabeçalho do *chare* onde é declarado a classe C++ que irá receber a implementação do *chare* note que essa classe é uma subclasse da classe "CBase_Main"que irá ser gerada pelo compilador Charm++ a partir do arquivo de interface main.ci.

```
1  #ifndef __MAIN_H__
2  #define __MAIN_H__
3
4  class Main : public CBase_Main
5  {
6  public:
7     Main(CkArgMsg * msg);
8     Main(CkMigrateMessage* msg);
9  };
10
11 #endif
```

O arquivo de header de um *chare* é o mesmo de um arquivo de header de uma classe C++ comum, note que o *chare* Main possuí dois construtores, o construtor da linha 7 é o mesmo declarado no arquivo de interface "main.ci"o mesmo pode ser chamado remotamente e como essa classe é um *chare* principal então esse método será chamado pelo RTS assim que aplicação ser iniciada e passará os argumentos pela classe "CkArgMsg"que encapsulas os mesmo da entrada pelo usuário. O construtor da linha 8 é utilizado internamente pelo RTS mesmo que ele não seja utilizado é necessário que ele seja declarado.

Assim como qualquer classe C++ também é necessário que um *chare* tenha um arquivo de implementação onde será escrito todo o código C++ que será executado. O arquivo main.C abaixo é a implementação do arquivo de header main.h acima.

```
1  #include "main.decl.h"
2  #include "main.h"
3
4  Main::Main(CkArgMsg* msg)
5  {
6     CkPrintf("Hello World!\n");
7     CkExit();
8  }
9
10 Main::Main(CkMigrateMessage* msg)
11 {}
12
13 #include "main.def.h"
```

A primeira linha inclui a declaração do *chare* gerada pelo arquivo de interface compilado, note que esse inclusão deve ser feita antes de incluir o header da própria classe (main.h) para que ele conheça as declarações feitas é possa utiliza-la, note que é incluído o arquivo "main.def.h"no final do arquivo esse arquivo é necessário por que ele contém o código de implementação necessário para para implementar as funcionalidades do arquivo "main.decl.h". O construtor da linha 10 é um construtor utilizado pelo RTS e não será implementado para essa aplicação em especifico, mesmo não sendo utilizado pela aplicação é necessário que ele seja declarado para que o RTS possa utiliza-lo. O construtor da linha 4 é equivalente a função "main"do C++ o programa será executado a partir de sua chamada, a classe "CkArgMsg"é fornecida pelo Charm++ e ela serve como um encapsulamento dos argumentos passados para aplicação equivalente aos parâmetros "int argc"e "char* argv[]"de uma aplicação C++. A função CkPrintf é uma função fornecida pela API do Charm++ que é utilizada pelo RTS para enviar o texto para o nó que o usuário possa visualizar e ckExit() é chamado para finalizar a aplicação ele é necessário para que o RTS possa finalizar todos os seus nós.

O relacionando do arquivo de interfaces e os arquivos de header pode ser visualizado pelo arquivo de makefile da aplicação:

```
1  CHARMDIR = DIRETORIO ONDE ESTÁ O CHARM++
2  CHARMC = $(CHARMDIR)/bin/charmc $(OPTS)
3
4  default: all
5  all: hello
6
7  hello : main.o
8     $(CHARMC) -language charm++ -o hello main.o
9
10 main.o : main.C main.h main.decl.h main.def.h
11    $(CHARMC) -o main.o main.C
12
13 main.decl.h main.def.h : main.ci
14    $(CHARMC) main.ci
15
16 clean:
17    rm -f main.decl.h main.def.h main.o hello charmrun
```

charmc é o compilado do Charm++ ele encapsula um compilador C++ e adiciona algumas funcionalidades a mais, por exemplo, a compilação de arquivos de interface ".ci"e a geração do arquivo executável do charm++ "charmrun". A linha 13 é passado o arquivo de interface para que o compilador gere os arquivos necessários para a implementação do *chare*, note que para compilação do arquivo objeto (.obj) necessita dos arquivos gerados do arquivo de interface pelo charmc na linha 10. O comando para gerar o arquivo de executável é utilizado passando o arquivo de objeto para o compilador charmc e passando como parâmetro a linguagem charm++ como está feito na linha 8.

Para executar esse programa basta utiliza-lo como uma aplicação em C++ ./hello já que como ele possuí apenas um *chare* uma chamada a "charmrun"não teria efeito (PPL (2015d)), o próximo exemplo de multiplicação de matrizes terá mais de um *chare* é utilizará os recursos de paralelismo

do Charm++.

3.3 Multiplicação de Matrizes

Esse é um exemplo mais completo que busca demonstrar como é feito um *broadcasting* em Charm++ e consequentemente como é feita a comunicação entre os *chares*, mesmo sendo uma aplicação simples é um exemplo que resolve um problema real.

Para multiplicação de matrizes será utilizado dois tipos de *chares* um que é o *chare* principal onde o programa será iniciado e onde será exibido o resultado final e *chare* MatrixCellMultiplicationChare que será responsável por calcular uma célula da matriz resultante, abaixo o arquivo de interface do MatrixCellMultiplicationChare.

```
1 module MatrixCellMultiplicationChare
2 {
3    array [1D] MatrixCellMultiplicationChare
4    {
5       entry MatrixCellMultiplicationChare();
6       entry void multiplicate(long a[n*n], long b[n*n], int n);
7    };
8 };
```

Para esse programa será criado um modulo chamado MatrixCellMultiplicationChare assim como um vetor de *chares* note que é um vetor de apenas uma dimensão assim como está anotado na linha 3. O *chare* de multiplicação de matrizes terá dois métodos que podem ser chamados remotamente o construtor e um método de multiplicação, o construtor é o construtor padrão e não recebe nenhum argumento, o método de multiplicação recebe as duas matrizes de tamanha NxN e o tamanho de uma dimensão (tamanho N), note que a multiplicação é feita somente para matrizes quadradas, na declaração dos parâmetros dentro dos colchetes que denotam que o parâmetro é um vetor pode ser inserido uma expressão C++ qualquer, como cada *chare* recebe as matrizes que ele deve calcular (para usar linha e coluna) por isso cada vetor que o método recebe tem dimensão NxN.

Abaixo o arquivo de cabeçalho que implementará o *chare* descrito no arquivo de interface.

```
1  #ifndef __MATRIX_CELL_MULTIPLICATION_CHARE_H__
2  #define __MATRIX_CELL_MULTIPLICATION_CHARE_H__
3
4  class MatrixCellMultiplicationChare : public CBase_MatrixCellMultiplicationChare
5  {
6  public:
7     MatrixCellMultiplicationChare();
8     MatrixCellMultiplicationChare(CkMigrateMessage *msg);
9
10    void multiplicate(long* a, long* b, int matrix_size);
11 };
12
13 #endif
```

Assim como no programa HelloWorld a classe que implementa o *chare* é uma classe C++, os métodos declaradas na interface devem ser declarados nesse arquivo também note que os vetores no parâmetro do método "multiplicate"se transformaram em ponteiros, esse arquivo também incluir o construtor que recebe um objeto do tipo "CkMigrateMessage"de uso interno do RTS. A classe "CBase_MatrixCellMultiplicationChare"dever ser a super classe do *chare* a mesma foi gerada pelo compilador charmc ao compilar o arquivo de interface.

A classe de header deve ter ser arquivo de implementação no caso do *chare* "MatrixCellMultiplicationChare"esse arquivo deve implementar o método de multiplicação que realiza a multiplicação e envia o resultado de volta para o *chare* principal, note que como os *chares* iram ser executados em diferentes processos não é possível que um método do tipo entry retorne valor, essa comunicação deve ser feita por meio de troca de mensagens (para métodos entry multithread é permito retornar valor, mas esse tipo de método está fora de escopo desse trabalho).

```
1  #include "MatrixCellMultiplicationChare.decl.h"
2  #include "MatrixCellMultiplicationChare.h"
3  #include "main.decl.h"
4
5  extern CProxy_Main mainProxy;
6
7  MatrixCellMultiplicationChare::MatrixCellMultiplicationChare()
8  {}
9
10 MatrixCellMultiplicationChare::MatrixCellMultiplicationChare(CkMigrateMessage *msg)
11 {}
12
13 void MatrixCellMultiplicationChare::multiplicate(long* a, long* b, int matrix_size)
14 {
15    int i = thisIndex / matrix_size;
16    int j = thisIndex % matrix_size;
17    long result = 0;
18
19    for (int k = 0; k < matrix_size; ++k)
20       result += a[i * matrix_size + k] * b[k * matrix_size + j];
21
22    mainProxy.setMatrixCResult(thisIndex, result);
23    mainProxy.done();
24 }
25
26 #include "MatrixCellMultiplicationChare.def.h"
```

O arquivo de implementação começa com os arquivos de headers necessários para sua implementação, então é declarado um proxy para a *Main* já que o *chare* de multiplicação de matrizes terá que comunicar o valor obtido é quando ele terminou de executar os cálculos (linhas 22 e 23). Além dos construtores a classe MatrixCellMultiplicationChare tem o método de multiplicação que pode ser chamado remotamente, ele é um método de C++ comum, ele utiliza a variável thisIndex que foi introduzida pela classe "CBase_MatrixCellMultiplicationChare", esse membro ajuda a identificar o *chare* dentro do array de *chares* e ao final do método ele se comunica com o *chare* principal para informar o resultado da multiplicação e avisa-lo que o calculo daquela célula foi encerrado.

Além do *chare* de multiplicar uma célula da matriz esse programa possuí o *chare* principal que vai servir como ponto de entrada do programa, cria as matrizes utilizadas no calculo e envia-las para multiplicação. Para isso é criado um arquivo de interface para esse *chare*.

```
1  mainmodule main
2  {
3     readonly CProxy_Main mainProxy;
4
5     extern module MatrixCellMultiplicationChare;
6
7     mainchare Main
8     {
9        entry Main(CkArgMsg *msg);
10       entry void setMatrixCResult(int index, long value);
11       entry void done();
12    };
13 };
```

Esse arquivo começa declarando que essa interface terá um proxy global de acesso (que é usado pelo *chare* de multiplicação de matrizes) e têm o modulo de multiplicação de matrizes como dependência, esse modulo foi criado no arquivo de interface anterior onde foi declarado o *chare* "MatrixCellMultiplicationChare". Ele também define o *chare* principal e três métodos entry, o construtor que é o ponto de entrada do programa, o método que recebe o valor da multiplicação e o método que indica que a célula foi calculada como vista no trecho de código anterior esses métodos são utilizados pelo *chare* da multiplicação de matrizes.

O arquivo de header desse *chare* é mostrado abaixo:

```
#ifndef __MAIN_H__
#define __MAIN_H__

#define mul 5

class Main : public CBase_Main
{
private:
   int matrix_size;
   long *matrix_a, *matrix_b, *matrix_c;

   int doneCount;
   int numElements;
public:
   Main(CkArgMsg* msg);
   Main(CkMigrateMessage* msg);

   void done();
   void setMatrixCResult(int index, long value);

private:
   void generateMatrix(long *a, long *b, long *c, int matrix_size);
   void checkAnswer();
   void finish();
};

#endif
```

O arquivo de header desse *chare* além de conter as declarações dos métodos de entry também possui a declaração de métodos privados, por exemplo, os métodos que gera as matrizes, que verifica se o calculo da matriz foi realizado corretamente com "asserts"e o método de finalização, que são locais ao *chare* que pode ser chamado apenas localmente. Além disso esse arquivo define a constate "mul"que será utilizado para gerar as matrizes, também são declarados os atributos do *chare* que são as matrizes a serem calculadas o tamanho delas (são matrizes quadradas, por isso apenas um atributo para o tamanho é o suficiente), e variáveis de controle, por exemplo, o número total de elementos e a quantidade de resultados já calculadas.

O arquivo de implementação será quebrados em vários trechos de códigos, por que ele é extenso. O primeiro trecho mostra os headers inclusos ao arquivo de implementação da classe que representa o *chare* principal.

```
#include "main.decl.h"
#include "main.h"
#include "MatrixCellMultiplicationChare.decl.h"
#include <assert.h>
```

Esse arquivo vai precisar se ligar com os arquivos de de cabeçalho geradas de sua interface e o arquivo de declaração do *chare* de multiplicação de matrizes por que é o arquivo que expõe os métodos que podem ser chamados (métodos entry) também é incluso o header para o uso de asserções.

Os trechos a seguir mostram como foi implementado os métodos auxiliarem que são utilizados pelo construtor (ponto de entrada) para realizar o calculo, os métodos geram os valores para as matrizes, realiza a verificação do resultado e finaliza a programa.

```
void Main::generateMatrix(long *a, long* b, long* c, int matrix_size)
{
   for (int i = 0; i < matrix_size; ++i)
   {
      for (int j = 0; j < matrix_size; ++j)
      {
         a[i * matrix_size + j] = i*mul;
         b[i * matrix_size + j] = i;
         c[i * matrix_size + j] = 0;
      }
   }
}

void Main::finish()
{
   checkAnswer();

   delete matrix_a;
   delete matrix_b;
   delete matrix_c;

   CkExit();
}

void Main::checkAnswer()
{
   long col_sum = matrix_size * (matrix_size-1) / 2;
   for (int i=0; i<matrix_size; i++)
   {
      for (int j=0; j<matrix_size; j++)
      {
         assert (matrix_c[i * matrix_size + j] == i*mul * col_sum);
      }
   }
}

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

Esse trecho de código é o final do arquivo, o arquivo de definição do *chare* principal foi inserido na última linha do arquivo. A função que gera as matrizes colocam valores por meio de um cálculo utilizando a constante "mult"definida no header de tal forma que depois é possível verificar o resultado com poucas contas. O método de finalização chama o método de verificar que verifica se a resposta está correta utilizando um calculo em que utiliza as dimensões das matrizes e a constante "mul", após a verificação as matrizes são desalocadas e liberadas da memoria então a aplicação é finalizada.

O *chare* principal desta aplicação pode ser divido em métodos que são chamados pelo *chare* da multiplicação de matrizes e o método principal que chama o *chare* da multiplicação de matriz e é o método de entrada da aplicação, o trecho abaixo descreve os métodos que são chamados pelos *chares* de multiplicação de matrizes.

```
void Main::done()
{
   doneCount++;
   if (doneCount >= numElements)
      finish();
}

void Main::setMatrixCResult(int index, long value)
{
   matrix_c[index] = value;
}
```

O método "done"apenas incrementa o número de multiplicações realizadas e se todas as células da matriz resultante estiver pronta então ele encerra a aplicação e o método "setMatrixCResult"é utilizado para o *chare* de multiplicação de matrizes matrizes enviar o resultado da multiplicação para o *chare* principal.

O trecho a seguir descreve o método principal (construtor do *chare* principal), onde é realizado a comunicação entre os *chares* que realizarão a multiplicação da matriz, além disso esse método aloca e inicializa as matrizes.

```
1  CProxy_Main mainProxy;
2
3  Main::Main(CkArgMsg *msg)
4  {
5     doneCount = 0;
6     matrix_size = 64;
7
8     if (msg->argc > 1)
9        matrix_size = atoi(msg->argv[1]);
10
11
12    numElements = matrix_size * matrix_size;
13
14    matrix_a = new long[numElements];
15    matrix_b = new long[numElements];
16    matrix_c = new long[numElements];
17
18    generateMatrix(matrix_a, matrix_b, matrix_c, matrix_size);
19
20
21    delete msg;
22
23    mainProxy = thisProxy;
24
25    CProxy_MatrixCellMultiplicationChare cellMultArray =
26       CProxy_MatrixCellMultiplicationChare::ckNew(numElements);
27
28    cellMultArray.multiplicate(matrix_a, matrix_b, matrix_size);
29 }
30
31 Main::Main(CkMigrateMessage * msg)
32 {}
```

Esse trecho começa com a declaração do proxy da própria classe que é inicializado na linha 23 para que possa ser utilizado pelo *chare* da multiplicação das matrizes. Então o construtor começa lendo o tamanho da matriz por meio dos parâmetros lidos da linha de comando e utiliza-lo para calcular as dimensões da matrizes. As matrizes são alocadas e inicializadas, então é criado um array de *chares* onde cada posição é um *chare* de multiplicação de matrizes que multiplica uma célula da matriz final, por isso são criados NxN *chares* de multiplicação de matrizes, então é feito um *broadcasting* para todos os *chares* (para comunicar com um *chare* especifico utiliza-se o array de *chare* indexado, por exemplo, cellMultArray[0].multiplicate(matrix_a, matrix_b, matrix_size) então seria enviado uma mensagem para *chare* da posição 0), note na linha 28 que mesmo "cellMultArray"sendo um array o método de multiplicação é chamado diretamente no mesmo objeto do array, não é necessário chamar o método para cada *chare* manualmente.

Como as trocas de mensagens do Charm++ são sempre assíncronas então é necessário que o método aguarde todas os *chares* terminarem de multiplicar as matrizes para o programa finalizar, esse é o motivo da função "done".

Para compilar esse programa é necessário compilar os arquivos de interface primeiro e então utilizar os arquivos gerados pela compilação para compilar o códigos dos *chares*. No caso desse programa de multiplicação de matrizes foram definidos dois *chares* que comunicam entre si, portanto existe uma dependência entre os dois *chares* isso pode ser resolvido compilando um arquivo de interface e depois outro, conforme mostrado no trecho do arquivo de makefile abaixo:

```
CHARMDIR = DIRETORIO ONDE ESTÁ O CHARM++
CHARMC = $(CHARMDIR)/bin/charmc $(OPTS)

default: all
all: multmatrix

multmatrix: main.o MatrixCellMultiplicationChare.o
   $(CHARMC) -language charm++ -omultmatrix main.o MatrixCellMultiplicationChare.o

main.o: main.C main.h main.decl.h main.def.h MatrixCellMultiplicationChare.decl.h
   $(CHARMC) -o main.o main.C

main.decl.h main.def.h: main.ci
   $(CHARMC) main.ci

#Diretivas para compilar MatrixCellMultiplication Comprimidas.

clean:
   rm -f main.decl.h main.def.h main.o
   rm -f MatrixCellMultiplicationChare.decl.h MatrixCellMultiplicationChare.def.h
   rm -f MatrixCellMultiplicationChare.o
   rm -f multmatrix charmrun
```

O arquivo de makefile possui as seções “default”, “all” e “clean” que são utilizadas para definir a comportamento padrão ao chamar o makefile a definição de compilar o aplicativo inteiro e a remoção dos arquivos gerado ápos a compilação respectivamente. Neste trecho do arquivo as linhas que compilam o *chare* da multiplicação foram suprimidos por que as linhas são muitos longas e não seria visualmente bom, porém essas linhas podem ser inferidas a partir da compilação do *chare* principal. Note que a arquivo de objeto possuí o arquivo “MatrixCellMultiplicationChare.decl.h” como dependência por que o mesmo é usado para fazer a comunicação entre o *chares*, semelhante a linha suprimida que compila o arquivo objeto da classe “MatrixCellMultiplicationChare” depende do arquivo de declaração “main.decl.h” , note que não existe uma dependência cíclica para essa compilação por que esses arquivos são gerados pela compilação dos arquivos de interface que não possuem dependência.

Para executar esse aplicativo é necessário utilizar o programa charmrun utilitário gerado pelo Charm++ que inicia o RTS e aloca os *chares* nos elementos de processamento definidos pelo arquivo *nodelist* esse arquivo defini as máquinas que serão executados o programa (esse arquivo está fora de escopo desse trabalho, mas o mesmo é semelhante o arquivo utilizado em MPI para definição das máquinas que serão executado os processos), como esse experimento foi realizado em apenas um computador esse não foi necessário definir esse arquivo. Um exemplo de execução desse programa para 4 processos (PPL (2015a)) seria:

user@ComputerName$ ./charmrun +p4 ./multmatrix 512

O parâmetro +p4 indica que serão usados 4 processos (o número seguido de p indica a quantidade de processos a serem utilizados) cada um será um elemento de processamento que o RTS terá de gerenciar, esses 4 processos iram se comunicar entre si para calcular a multiplicação de matrizes entre os 512x512 *chares* neste caso, o último parâmetro (512) é um parâmetro da aplicação que é lido pelo *chare* principal conforme mostrado no trecho de código de sua implementação e para esse caso é o lado de uma matriz quadrada portanto o programa calculará a multiplicação de duas matrizes de dimensão 512X512.

O objetivo do programa de multiplicação de matrizes neste exemplo foi mostrar a funcionalidades de comunicação e *broadcasting*, portanto ele não foi otimizado, provavelmente é possível fazer uma versão melhorada que faz a comunicação entre *chares* que passam como mensagem apenas a linha e coluna que será realizado a multiplicação, mas para fins de comparação e analise de *overhead* do RTS o mesmo foi comparado com uma aplicação de multiplicação de matrizes escrita utilizando MPI

(código de ambas aplicações podem ser baixadas pelo repositório desse trabalho https://bitbucket.org/dennis_ime/mac5742-charm-project/) e o resultado foi que o MPI foi quase 3.5 vezes mais rápido com tempo médio depois de 10 execuções de 0.45 segundos comparado com as mesmas 10 execuções do aplicativo Charm++ que gastou em média 1.55 segundos para matrizes de 512x512 posições, o computador utilizado foi um Intel(R) Core(TM) i5 CPU M 430 @2.27GHz com 4 GB de memoria RAM e sistema operacional Ubuntu 14.04 LTS.

Esse resultado é importante para notar o *overhead* do RTS do Charm++, para que possa se fazer um bom uso dele é necessário que a aplicação seja distribuída e suas funcionalidades sejam utilizadas, por exemplo, balanceamento de carga. Uma aplicação que foi portada para AMPI (ferramenta que facilita o porte de uma aplicação MPI para o uso de Charm++) é mostrada no capítulo 4, onde a aplicação que usa Charm++ foi mais eficiente que a aplicação que sua versão MPI.

Capítulo 4

Aplicações

Este capítulo mostrará duas aplicações desenvolvidas em Charm++, a LeanMD que foi escrita em cima de outra aplicação NAMD que ganhou o prêmio Gordon Bell de 2002 e a aplicação LULESH que faz um porte de uma aplicação que utiliza MPI para uma versão escrita em AMPI que por sua vez utiliza os recursos do Charm++. Ambas as aplicações estão descritas em Acun *et al.* (2014)

4.1 LeanMD

LeanMD é uma aplicação de simulação de dinâmica celular escrita em Charm++ que simula o comportamento de átomos baseados na formula de potencial de Lennard-Jones. O LeanMD foi escrito em cima de outra aplicação chamada NAMD também escrita em Charm++ que ganhou o prêmio Gordon Bell de 2002.

Segundo os autores do artigo os seguintes atributos foram importantes para o ganho de desempenho do aplicativo:

- ***Over-Decomposition***: O espaço de átomos utilizados na simulação foram melhor representados pelo um array de *chares* denso com dimensão 6 esse nível de decomposição permitiu uma comunicação mais eficiente realizando sobreposições e um melhor balanceamento de carga.
- **Balanceamento de Carga Adaptativo**: O balanceamento de carga realizando pelo RTS foi importante para a detecção de um desbalanceamento e então a sua correção, esse balanceamento de carga junto com o alto nível de decomposição foi importante para que a aplicação fosse escalável e passa-se de uma máquina de 1K cores para outra com 32K com uma bom desempenho nessa transição.
- **Ponto de restauração e reinicio em memoria (com simulação de falha**: O sistema de tolerância a falhas se mostrou muito bem no caso de queda de um nó da aplicação, o sistema de barreira foi importante para recuperação de um nó em caso de falha.

O aplicativo LeanMD foi utilizado para testar a habilidade de diminuir e aumentar cores do computador Stampede da TACC. Essa funcionalidade é importante por que uma aplicação pode diminuir o uso de energia quando não precisar de alto poder de processamento ou diminuir o tempo de execução quando os resultados precisarem ser calculados mais rapidamente além disso se a aplicação estiver sendo executado em um computador que também executa outras aplicação essa funcionalidade pode ajudar o computador a melhor utilizar o seus recursos de acordo com a prioridade das aplicações, esse controle de número de cores é feito por meio de envio de comando para aplicação o resultado é demonstrado na imagem 4.1.

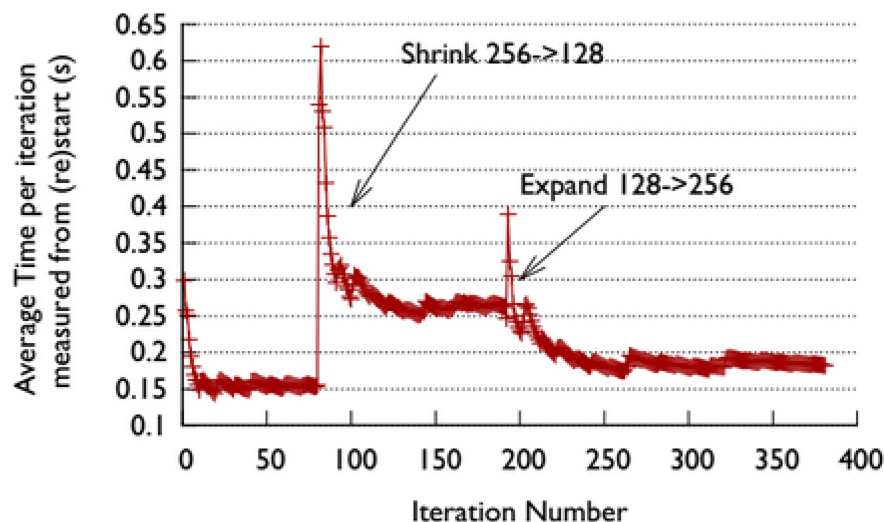

Figura 4.1: *comportamento do aplicativo LeanMD em relação ao balanceamento carga adaptativo. Imagem retirada de Acun et al. (2014)*

O aplicativo começou a ser executado com 256 cores então foi executado um comando de redução para 128 cores note que existe um pico de tempo em que a aplicação demora um tempo para estabilizar e migrar os *chares* para os cores que se mantiveram disponível e então estabiliza o tempo de execução. Então é executado um comando de expansão para 256 cores novamente ocorre um pico até o estabelecimento do tempo de execução. Note que a aplicação perdeu muito pouco poder de processamento com essas operações.

4.2 LULESH

LULESH é a sigla para *Livermore Unstructured Lagrangian Explicit Shock Hydrodinamics*, que é um aplicativo desenvolvido com a utilização de MPI para imitar o comportamento de um grande número de classes de aplicações de ciência e engenharia que utiliza modelos de hidrodinâmica. A equipe desenvolvimento utilizou AMPI (*Adaptativo MPI*) para adaptar essa aplicação para a utilização das funcionalidades do Charm++, de acordo com os autores do artigo as principais características dessa aplicação em comparação a aplicação original que utiliza MPI são:

- **Porte do aplicativo para AMPI**: De acordo com os autores o porte do aplicativo exigiu pouco esforço necessitando apenas de algumas variáveis estáticas e a adição de um compilador apropriado e flags de ligação. também foi desenvolvidas rotinas do pup framework mesmo existindo essa funcionalidade automática, o objetivo era melhorar a portabilidade.

- **melhora na utilização de Cache**: De acordo com os autores o aplicativo LULESH possuí um acesso complexo a memoria que dificultaria a utilização de cache em modelo de programação MPI, o Charm++ melhorou isso introduzindo virtualização passando a impressão que existisse mais elementos de processamento do que realmente tinha fisicamente.

- **Balanceamento de carga automático**: Mesmo com pouco desbalanceamento de carga a aplicação melhorou seu desempenho pela inclusão da funcionalidade de balanceamento de carga automático do Charm++

- **Execução em vários números de Cores**:A aplicação LULESH necessita de um número cúbico de processadores para executar sua tarefa com a introdução da virtualização essa limitação pode ser superada incluindo um número cúbico de processadores virtuais.

Com a porte da aplicação para o uso do Charm++ a equipe conseguiu melhorar seu desempenho conforme mostrada na figura 4.2

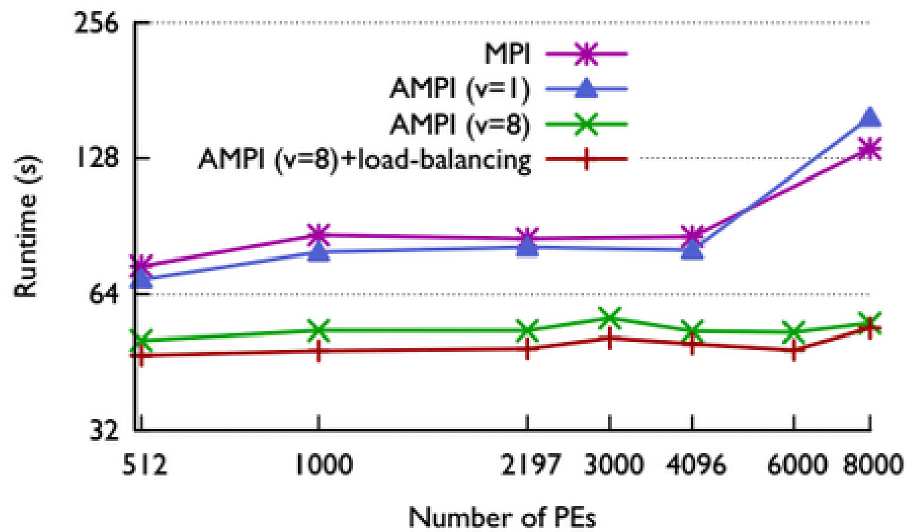

Figura 4.2: *Desempenho das aplicação LULESH comparado com suas versões que utilizam AMPI. Figura retirada de Acun et al. (2014)*

O gráfico mostra que utilizando AMPI aplicação escalou melhor sendo que em todas as versões o tempo de execução da aplicação foi melhor quando utilizado AMPI e 8000 PEs. Note que AMPI v8 foi melhor que MPI em todos os cenários e esse desempenho foi ainda melhor quando incluído balanceamento de carga.

Capítulo 5

Conclusões

Charm++ se apresentou uma boa opção para aplicativos que precisem de alto desempenho que possuem características MIMD, ele possuí gerenciamento de carga, um design que permite alta decomposição de códigos em objetos, pode mover objetos entre processadores e outras características que contribuí para um aplicativo melhorar o desempenho em diversas máquinas. Charm++ possuí grande portabilidade abstraindo diversas funções de baixo nível para que o aplicativo desenvolvido funcione em diversas máquinas com nenhuma ou poucas alterações para isso ele utiliza RTS (*Run-Time System*) que é responsável por fornecer diversas funcionalidades para aplicação abstraindo rotinas de baixo além de analisar o sistema para melhor o uso dos recursos físicos.

Foram desenvolvidos duas aplicações de exemplo com o objetivo de introduzir a programação para Charm++, ele utiliza diversos conceitos de computação paralela e distribuída como objetos distribuídos, passagem de mensagens e linguagem de interface, nesses exemplos demonstrou que é necessário gerar um arquivo de interface que passa por um processo de compilação para geração de arquivos de header de declaração e definição esse processo apesar de computacionalmente eficiente do ponto de vista de manutenibilidade não pode não ser ideal já que alterações na interface requer alteração em diversos arquivos e pode quebrar o programa inteiro, além de proliferação de arquivos que são utilizados pelos *chare* o uso de herança poderia melhorar esse quesito mas prejudicaria o desempenho. O aplicativo de multiplicação de matrizes mostrou que o Charm++ precisa de uma infraestrutura e que o problema a ser resolvido tenha complexidade para que o sistema de RTS possa otimizar os recursos computacionais e seu *overhead*, seja pequeno em relação aos benefícios que o sistema fornece.

Os aplicativos mostrados no capítulo 4 mostraram que Charm++ pode melhorar o uso dos recursos computacionais fornece controle sobre os mesmo para um melhor uso dos processadores tendo em vista a necessidade do usuário para dado aplicativo e ambiente. Charm++ pode gerenciar o número de processadores utilizados Charm++ fornece um maior controle sobre os recursos permitindo a flexibilidade entre consumo de energia e poder de processamento assim como prioridade da aplicação. O aplicativo LULESH mostrou que em algumas situações o uso de Charm++ é melhor que MPI se for balanceado a produtividade de desenvolvimento e a escalabilidade do sistema, por que com o seu porte a aplicativo em Charm++ teve melhor desempenho que um aplicativo em MPI.

Referências Bibliográficas

Acun *et al.* (2014) Bilge Acun, Abhishek Gupta, Nikhil Jain, Akhil Langer, Harshitha Menon, Eric Mikida, Xiang Ni, Michael Robson, Yanhua Sun, Ehsan Totoni, Lukasz Wesolowski e Laxmikant Kale. Parallel programming with migratable objects: Charm++ in practice. Em *SC14: International Conference for High Performance Computing, Networking, Storage and Analysis*, páginas 647–658. Citado na pág. v, 1, 3, 6, 19, 20, 21

Kale e Krishnan (1993) Laxmikant Kale e Sanjeev Krishnan. Charm++: A portable concurrent object oriented system based in c++. Relatório técnico, University of Illinois, Urbana-Champaign, IL, USA. Citado na pág. 3, 4, 5

Meglicki (2015) Zdzislaw Meglicki. History of mpi. http://beige.ucs.indiana.edu/I590/node54.html, 2015. Último acesso em 28/6/2015. Citado na pág. 4

OMG (2015) OMG. History of corba. http://www.omg.org/gettingstarted/history_of_corba.htm, 2015. Último acesso em 28/6/2015. Citado na pág. 4

Oracle (2015) Oracle. The history of java technology. http://www.oracle.com/technetwork/java/javase/overview/javahistory-index-198355.html, 2015. Último acesso em 28/6/2015. Citado na pág. 4

PPL (2015a) Illinois University PPL. Broadcast hello world program: A parallel hello world program. http://charm.cs.illinois.edu/tutorial/BroadcastHelloWorld.htm, 2015a. Último acesso em 28/6/2015. Citado na pág. 17

PPL (2015b) Illinois University PPL. Components of a charm++ program. http://charm.cs.illinois.edu/tutorial/BasicHelloWorld.htm, 2015b. Último acesso em 28/6/2015. Citado na pág. v, 9, 10

PPL (2015c) Illinois University PPL. Charm.cs code review. http://charm.cs.illinois.edu/gerrit/gitweb?p=charm.git, 2015c. Último acesso em 28/6/2015. Citado na pág. 5

PPL (2015d) Illinois University PPL. Basic hello world program. http://charm.cs.illinois.edu/tutorial/BasicHelloWorld.htm, 2015d. Último acesso em 28/6/2015. Citado na pág. 10, 11

PPL (2015e) Illinois University PPL. Charm++ parallel programming with migratable objects. http://ppl.cs.illinois.edu/research/charm, 2015e. Último acesso em 28/6/2015. Citado na pág. 5

PPL (2015f) Illinois University PPL. The charm++ parallel programming system manual. http://charm.cs.illinois.edu/manuals/pdf/charm++.pdf, 2015f. Último acesso em 28/6/2015. Citado na pág. 9

PPL (2015g) Illinois University PPL. Introduction to the charm++ runtime system. http://charm.cs.illinois.edu/tutorial/CharmRuntimeSystem.htm, 2015g. Último acesso em 28/6/2015. Citado na pág. v, 7, 8

ScalaLang (2015) ScalaLang. Scala language specification. http://www.scala-lang.org/files/archive/spec/2.11/, 2015. Último acesso em 28/6/2015. Citado na pág. 4

Typesafe (2015) Typesafe. Java concurrency and scalability platform akka celebrates fifth anniversary. https://www.typesafe.com/company/news/java-concurrency-and-scalability-platform-akka-celebrates-fifth-anniversary, 2015. Último acesso em 28/6/2015. Citado na pág. 4