UNIVERSIDADE REGIONAL DO NOROESTE DO ESTADO DO RIO GRANDE DO SUL - UNIJUI

VINICIUS RIBAS SAMUEL DOS SANTOS

SMARTLB: PROPOSTA DE BALANCEAMENTO DE CARGA PARA REDUÇÃO DE TEMPO DE EXECUÇÃO EM AMBIENTES MULTIPROGRAMADOS

Ijuí/RS

2017

VINICIUS RIBAS SAMUEL DOS SANTOS

SMARTLB: PROPOSTA DE BALANCEAMENTO DE CARGA PARA REDUÇÃO DE TEMPO DE EXECUÇÃO EM AMBIENTES MULTIPROGRAMADOS

Trabalho de Conclusão de Curso apresentado ao Colegiado de Coordenação do Curso de Ciência da Computação da Universidade Regional do Noroeste do Estado do Rio Grande do Sul (UNIJUÍ), como requisito parcial para obtenção do Título de Bacharel em Ciência da Computação.

Orientador: Professor DSc. Edson Luiz Padoin

Ijuí/RS

2017

Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet.

AGRADECIMENTOS

Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet.

Duis autem vel eum iriure dolor in hendrerit in vulputate velit esse molestie consequat, vel illum dolore eu feugiat nulla facilisis at vero eros et accumsan et iusto odio dignissim qui blandit praesent luptatum zzril delenit augue duis dolore te feugait nulla facilisi. Lorem ipsum dolor sit amet, consectetuer adipiscing elit, sed diam nonummy nibh euismod tincidunt ut laoreet dolore magna aliquam erat volutpat.

Ut wisi enim ad minim veniam, quis nostrud exerci tation ullamcorper suscipit lobortis nisl ut aliquip ex ea commodo consequat. Duis autem vel eum iriure dolor in hendrerit in vulputate velit esse molestie consequat, vel illum dolore eu feugiat nulla facilisis at vero eros et accumsan et iusto odio dignissim qui blandit praesent luptatum zzril delenit augue duis dolore te feugait nulla facilisi.

"A imaginação é mais importante que o conhecimento. O conheicmento é limitado. A imaginação circunda o mundo."

(Albert Einstein)

RESUMO

Este artigo apresenta uma proposta de balanceamento de carga para a redução do tempo de execução de aplicações paralelas executadas em ambientes multiprocessados. O algoritmo do balanceador coleta informações do sistema e da aplicação em tempo real e as utiliza na tomada de decisões de balanceamento de carga dinamicamente, visando reduzir o número de migrações de tarefas enquanto reduzindo o tempo total de execução. Para implementação foi utilizado o modelo de programação paralela $\textsc{Charm++}$. Os resultados preliminares apresentaram reduções de 3 à 137 vezes na quantidade de migrações de tarefas e reduções de $3,5\%$ à $19,8\%$ no tempo total de execução.

AINDA NÃO ESTÁ PRONTO, SÓ COPIEI DE UM ARTIGO MEU.

ABSTRACT

This paper presents a load balancer proposal to reduce the execution time of parallel applications when they run on multiprocessors environments. The load balancer algorithm collects system and application information in real time and uses them to make load balancing decisions dynamically, aiming to reduce the rate of migration and to decrease the execution time. CHARM++was used for implementation due to its compatibility with the desired environments. The results show that our load balancer improves performance by reducing task migration from 3 up to 137 times and reductions from 3.5% to 19.8% on total execution time.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

BC	Balanceador de Carga
E/S	Entrada e Saída
FIFO	First In First Out
HPC	High Performance Computing
LIFO	Last In First Out
OOPP	Object Orientation Parallel Programming
PE	Processing Elemnents
RTS	Runtime System

SUMÁRIO

1 INTRODUÇÃO

A necessidade de alto desempenho, proveniente de um cresimento da produção de software deu abertura para o desenvolvimento dos sistemas computacionais, surgidos para suprir essa demanda (SANTOS; PADOIN, 2017).

A medida que novas simulações computacionais são desenvolvidas, aumenta a demanda por processamento dos sistemas de HPC. Um dos componentes de hardware que apresentou uma enorme evolução foi o processador, que passou a proporcionar a execução simultanea de aplicações. Assim, a programação paralela passa ser importante, possibilitando que aplicações sejam divididas em partes e executadas em paralelo nas unidades de processamento (PILLA; MENESES, 2015a).

A maioria das aplicações paralelas envolve comportamentos dinâmicos ou cálculos baseados em diversas fórmulas complexas. Por conta disso, empresas e instituições buscam adquirir uma infraestrutura suficiente para suportar tais aplicações (ARRUDA, 2015). O grande problema por trás disso é que na maioria das vezes não há uma preocupação com o desbalanceamento de carga gerado por estas aplicações, impedindo que as máquinas paralelas aproveitem todo o seu potencial (PADOIN et al., 2014b). Diante deste problema é que balanceadores de carga são desenvolvidos almejando um equilíbrio de cargas nos processadores.

1.1 Problema a ser discutido

A demanda por sistemas de alto desempenho cresce cada vez mais, à medida que novas aplicações simulam sistemas cada vez mais complexos. As evoluções na forma de concepção dos sistemas e na fabricação dos processadores tem permitido ultrapassar barreiras de desempenho, uma vez que aplicações complexas são divididas em tarefas menores e executadas simultaneamente em nos núcleos das unidades de processamento.

No entanto, tais aplicações geralmente apresentam cargas computacionais diferentes, o que dificulta uma eficiente utilização dos sistemas computacionais. A modelagem deste problema é complexa fazendo com que muitas aplicações sejam executadas com desbalanceamento de carga e excessiva comunicação entre tarefas (PILLA; MENESES, 2015a; PADOIN; NAVAUX; MéHAUT, 2017). Essa é uma preocupação que surge devido ao seu caráter impeditivo quanto ao alcance de uma boa eficiência na utilização dos recursos dos sistemas paralelos (PADOIN et al., 2014a).

Nesse contexto, soluções que empregam estratégias para aumentar a eficiência dos recursos paralelos disponíveis vem sendo cada vez mais desenvolvidas e utilizadas. Balanceadores de Carga (BC), almejam detectar e corrigir dinamicamente tais desbalanceamentos, melhorando a utilização dos recursos disponíveis (SANTOS et al., 2017). Deste modo, este trabalho

apresenta um balanceador de carga denominado SmartLB que almeja a redução do tempo de execução de aplicações paralelas executadas em ambientes multiprocessados.

1.2 Objetivo

Este artigo tem por objetivo mostrar o desenvolvimento do Balanceador de Carga(BC) SmartLB juntamente com seus resultados, que visam a redução de migrações de processos, uma vez que o número de migrações de processos impacata no tempo total de execução da aplicação. Desta forma optimizando a execução de grandes aplicações computacionais.

1.3 Organização do trabalho

Este trabalho está organizado em cinco capítulos, distribuídos da seguinte maneira:

No Capítulo 1 apresenta-se a introdução do trabalho, onde ocorre a contextualização referente ao tema de pesquisa, bem como é exposto o objetivo deste trabalho.

No Capítulo 2, realiza-se um apanhado geral sobre o estado da arte das tecnologias a serem utilizadas, abordando os principais conceitos sobre HPC e E/S, bem como informações das arquiteturas dos dispositivos e softwares, dentre outros itens que irão compor o escopo deste trabalho.

No Capítulo 3, é apresentado a metodologia e o ambiente onde os testes foram realizados, descrevendo os computadores utilizados em nossos testes, processador, sistema operacional, informações da ferramenta de benchmark utilizada, dentre outras configurações.

No Capítulo 4 é mensurados os resultados obtidos, apresentando as diferenças percebidas e os principais pontos observados nas execuções e arquiteturas.

Por fim, no Capítulo 5 serão apresentadas as considerações finais acerca dos resultados alcançados, bem como a sugestão para trabalhos futuros.

2 ESTADO DA ARTE E TRABALHOS RELACIONADOS

2.1 Trabalhos Relacionados

Diferentes abordagens tem alcançado resultados positivos quando empregado balanceamento de carga para redução do tempo de execução. Dentre elas destacam-se as estratégias centralizadas e distribuídas, sendo que atualmente novas abordagem hierárquica vem sendo propostas. Nestas novas abordagens, os núcleos de processamento são divididos em grupos independentes e organizados em uma árvore onde cada nível da árvore é composto por grupos de núcleos. Deste modo, quanto mais núcleos são adicionados aos grupos, menor é o uso da memória pelo BC. Usando esta abordagem, Zheng apresenta um BC hierárquica denominado HybridLB e consegue *speedup* de 6 com 2.048 *núcleos* e 145 com 8.192 *núcleos* em relação a versão sequencial (ZHENG et al., 2010).

Por outro lado, estratégias centralizadas efetuam decisões de balanceamento de carga em um único processador. Para tanto, os dados de carga e comunicação de todas as tarefas são acumulados em um processador específico, o qual executa um processo de decisão com base nessas informações. Neste tipo de estratégia pode-se citar os balanceadores GreedyLB e RefineLB. O primeiro, adota uma abordagem de agendamento agressivo, empregando uma heurística gulosa para tomada de decisões. Seu algoritmo objetiva migrar objetos pesados para o núcleo com menor carga, repetindo até que a carga de todos os processadores alcance uma proximidade com a carga média. Já o segundo, toma suas decisões considerando a distribuição de carga atual dos núcleos utilizados. A proposta é mover tarefas dos núcleos mais sobrecarregados para os menos carregados almejando atingir uma média, limitando o número do tarefas migradas (ZHENG et al., 2011).

Outras ainda, chamadas de estratégias distribuídas visam melhorar o desempenho de sistemas de grande escala. Nessas estratégias, os processadores trocam informações apenas entre os seus vizinhos, como forma de descentralizar o processo de balanceamento de carga e apresentar menor sobrecarga de balanceamento de carga do que estratégias centralizadas (KALÉ; KRISHNAN, 1993). Neste tipo de balanceador pode-se citar os balanceadores GrapeVineLB e GrapePlusLB. Este algoritmos realizam em paralelo, o cálculo da carga média de cada processador, sendo este valor médio usado para definir o estado global do sistema (MENON; KALÉ, 2013).

2.2 Balanceamento de Carga

O balanceamento de carga é uma técnica de distribuição de carga computacional e de comunicação uniformemente em todos os processadores de uma máquina paralela, para que nenhum processador seja sobrecarregado. As estratégias de balanceamento de carga

podem ser divididas em duas categorias. Para aplicativos onde novas tarefas são criadas e programado durante a execução e aqueles para aplicações iterativas com padrões de carga persistentes (ZHENG et al., 2010).

As estratégias de balanceamento de carga podem ser usadas durante a execução do aplicativo para melhorar a distribuição das tarefas. Essas estratégias tentam encontrar uma nova distribuição de tarefas que maximize o uso do núcleo, levando em consideração os tempos de execução das tarefas. Ainda assim, esta distribuição de trabalho pode não proporcionar um ótimo desempenho devido à comunicação de despesas gerais a partir do projeto multi-core de sistemas atuais (PILLA et al., 2014).

De acordo com (HENDRICKSON; DEVINE, 2000), muitas aplicações paralelas estão se movendo em direção a clusters de computadores de memória compartilhada distribuída e sistemas de computação heterogêneos. O balanceamento de carga nesses sistemas está se tornando uma área de pesquisa ativa. Uma abordagem para o balanceamento de carga nesse tipo de sistema é simplesmente mudar a atribuição de trabalho para processadores. O espaço de endereço global das máquinas pode então localizar dados quando necessário pelo aplicativo, ignorando a migração de dados complicada. No entanto, uma vez que as referências de memória aplicativo são caras, é vantajoso mover realmente os dados atribuídos para a própria memória do processador. Por razões de desempenho, então, o problema dinâmico de balanceamento de carga em sistemas de memória compartilhada compartilhada parece quase idêntico ao dos computadores MIMD.

2.3 Ambientes de Programação Paralela

Atualmente existem ambientes de programação paralela com capacidade de resolver os problemas provenientes do desbalanceamento de carga, nesta section será mostrado alguns dos principais ambientes que se encaixam nesta categoria.

2.3.1 POb C++

O POb C++ (PINHO; JUNIOR, 2010) é uma extensão paralela da linguagem de programação C++ que introduz um novo estilo de OOPP(Object Oriented Parallel Programming), que pode ser facilmente aplicável a qualquer linguagem orientada a objetos. A decisão de suportar C++ vem da ampla aceitação da linguagem em alguns nichos HPC. A premissa principal que guiou o estilo do POb C++ foi a preservação dos princípios básicos de orientação de objeto ao mesmo tempo que introduziu um estilo de programação muito próximo ao MPI. Portanto, propomos uma idéia em que os objetos intrinsecamente paralelos distinguem a interação do processo de passagem de mensagem (intra-objeto), usando a comunicação baseada em canal, a partir da passagem de mensagem de coordenação de objeto (interobject), geralmente feita por chamada de método.

2.3.2 JavaParty

De acordo com (PINHO, 2012), JavaParty é uma extensão das capacidades da linguagem Java para trabalhar com computação distribuída. Classes em JavaParty podem ser declaradas como remotas. Assim, objetos remotos são acessíveis, no ambiente JavaParty, em qualquer parte do sistema. O principal ganho ao usa-la no lugar da linguagem Java é a possibilidade de trabalhar transparentemente com objetos remotos.

Um programa Java multi-threaded pode facilmente ser transformado em um programa distribuído JavaParty, identificando as classes e threads que devem ser espalhados pelo ambiente distribuído. O programador indica isso por um modificador de classe introduzido recentemente. O novo modificador é a única extensão do Java. Como os tópicos de Java são objetos de uma classe de thread, os tópicos remotos podem ser criados como objetos de uma classe de thread remota. Não há necessidade de reescrever ou reorganizar significativamente um determinado programa Java. (PHILIPPSEN; ZENGER, 1997).

2.3.3 AMPI

O Adaptative MPI (HUANG; LAWLOR; KALE, 2003) foi desenvolvido em CHARM ++ e usa suas facilidades de comunicação, estratégias de balanceamento de carga e modelo de threading. O CHARM ++ usa um modelo baseado em objetos: os programas consistem em uma coleção de objetos orientados por mensagem, mapeados para processadores físicos pelo sistema de tempo de execução CHARM ++. Os objetos se comunicam com outros objetos invocando um método de entrada assíncrona no objeto remoto. Em cada uma dessas conexões assíncronas, uma mensagem é gerada e enviada para o processador de destino onde o objeto remoto reside. Adaptive MPI implementa seus processadores MPI como CHARM++ threads "nível de usuário"vinculados a CHARM ++ comunicando objetos. A passagem de mensagens entre os processadores virtuais AMPI é implementada como comunicação entre esses objetos CHARM ++ e as mensagens subjacentes, que são tratadas pelo sistema de tempo de execução. Mesmo com a migração de objetos, o CHARM ++ suporta roteamento e encaminhamento eficientes das mensagens.

2.3.4 CHARM++

O Charm ++ (KUNZMAN et al., 2006) é um paradigma de passagem de mensagens assíncronas. O programa é dividido em objetos chamados de chares. Cada chare, individualmente, faz uma parte da computação geral. Eles passam mensagens entre si para coordenar e executar a computação inteira. Cada chare tem um ou mais métodos de entrada. Basicamente, os métodos de entrada são funções de membros que atuam como pontos de recebimento de mensagens. Quando um chare envia uma mensagem para outro chare, ele especifica tanto a mensagem quanto o método de entrada que receberá a mensagem, como se apenas estivesse fazendo uma chamada de função de membro normal no objeto chare de recebimento. Os próprios chares

estão espalhados por todos os processadores durante a execução de um programa Charm ++. Normalmente, existem muitos chares por processador. Cada processador possui um sistema de tempo de execução Charm ++ que controla a execução, balanceamento de carga, envio e recebimento de mensagens entre outras funções, para todos os chares localizados nesse processador.

2.4 Ambiente de Programação CHARM++

De acordo com os ambientes de programação citados na seção 2.3, o ambiente escolhido para este trabalho é o Charm++. Além das informações descritas sobre ele na seçao 2.3.4, a principal razão para sua escolha é seu framework de balanceamento de carga, que permite tanto criar um novo Balanceador de Carga(BC) quanto utilizar um disponibilizado pelo próprio ambiente de programação.

O Charm++ foi desenvolvido pelo Laboratório de Programação Paralela da Universidade de Illinois, em 1993. Trata-se de uma extensão da linguagem C++, proporcionando um ambiente para programação paralela orientada a objetos.

De acordo (KALE; KRISHNAN, 1993) Charm++ é basicamente C++ sem suas variaveis globais, com algumas extenções que suportam execução paralela. Operações e manipulações de chares são restritos, em comparação com objetos sequenciais, para se adequear aos requisitos da execução paralela.

2.4.1 Chares

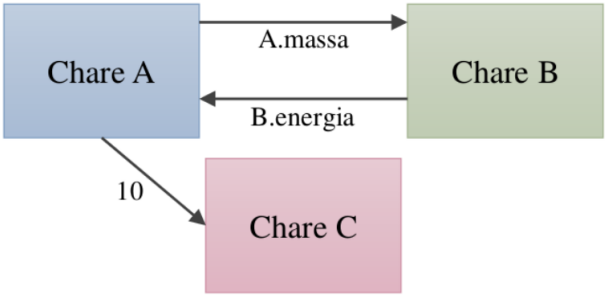
Para (PILLA; MENESES, 2015b, p.27), em programas escritos em CHARM++, toda a computação é realizada por objetos especiais chamados de objetos chare. Nesse contexto, objetos chare são similares a threads em OPENMP e processos em MPI, executando de forma concorrente ou paralela em uma plataforma computacional. O conjunto de todos os objetos chare que compõem uma aplicação é chamado de espaço global de objetos. O espaço global de objetos pode mudar durante a execução da aplicação através da criação de novos objetos chare, assim como a deleção dos mesmos.

Os Chares são criados dinamicamente e permitem expressar cálculos onde a criação de tarefas dinâmicas é necessária. Eles podem ser automaticamente mapeados e agendados, proporcionando assim ao usuário capacidades de alto nível. Um chare está agendado para execução apenas quando há uma mensagem disponível para ele. Ao contrário dos processos, os chares não estão executando perpetuamente, nem seu tempo de execução é cortado em processadores. Um chare está sempre pronto para executar qualquer mensagem disponível dirigida a ele (KALE et al., 1995, p.3).

Cada chare conta com seus próprios dados e estado privados, o que implica que outros objetos na aplicação não podem acessar essas informações diretamente. Todo o compartilha-

mento de dados acontece através da comunicação direta entre chares por troca de mensagens. O modelo de programação permite que um chare se comunique com qualquer outro chare que ele conheça no espaço global de objetos (PILLA; MENESES, 2015b, p.27). A Figura 1 ilustra a interação entre chares citada anteriormente.

Figura 1 – Comunicação entre os chares A, B e C através de troca de mensagens



Fonte: (PILLA; MENESES, 2015b, p.27)

Todo programa deve incluir um chare principal que deve ter um ponto de entrada chamado CharmInit. A execução do programa começa pela criação de uma instância do chare principal e a execução de seu ponto de entrada CharmInit (KALE, 1993, p.3).

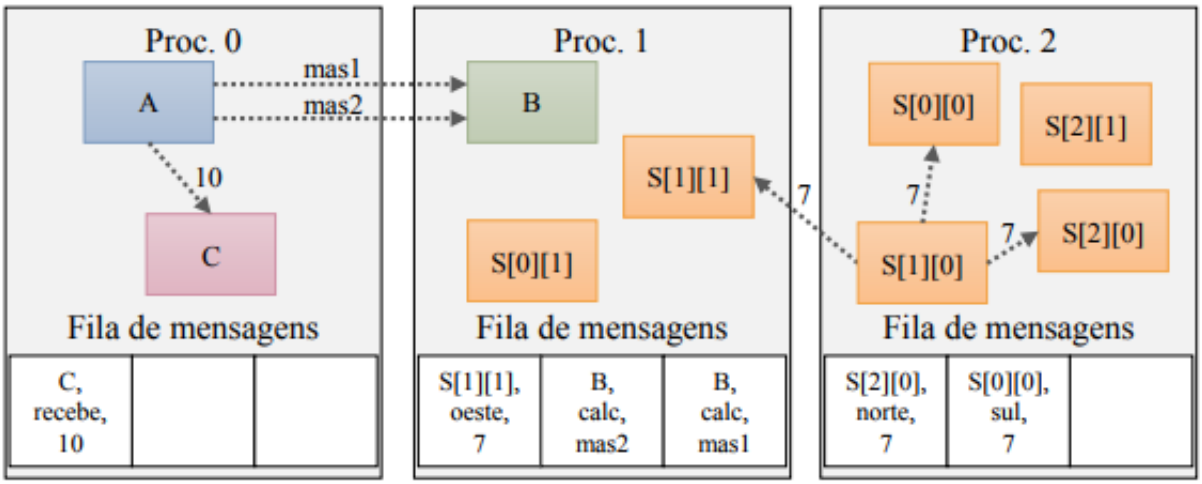
2.4.2 Trocas de mensagens

A troca de mensagens em CHARM++ é feita de forma assíncrona. A troca de mensagens assíncrona faz com que a invocação de um método de entrada em outro chare tenha retorno imediato. Em outras palavras, o chare que envia uma mensagem continu sua execução sem esperar por uma resposta. Adicionalmente, o chare cujo método é chamado pode não começar a executá-lo imediatamente. Esse mecanismo é usado para guiar a execução da aplicação (PILLA; MENESES, 2015b, p.28).

Uma mensagem é uma estrutura constituída por vários campos de dados e é definida de forma semelhante à estrutura de definição em C. As mensagens chegadas são programadas de acordo com uma estratégia de agendamento. A estratégia de agendamento e a estratégia dinâmica de balanceamento de carga são componentes separáveis de modo modular do sistema Charm Runtime, chamado Chare Kernel. O sistema fornece FIFO, LIFO e estratégias de agendamento baseadas em prioridade, com níveis ilimitados de prioridades. Da mesma forma, fornece uma variedade de estratégias dinâmicas de balanceamento de carga desenvolvidas ao longo dos anos. (KALE, 1993).

Todas as mensagens enviadas para chares em uma unidade de processamento são colocadas em sua fila de mensagens. Cada entrada na fila de mensagens inclui informações sobre qual chare é o receptor, o método invocado e os dados enviados na mensagem (PILLA; MENESES, 2015b, p.32). Como podemos ver na Figura 2.

Figura 2 – Distribuição de chares em unidades de processamento e suas filas de mensagens.



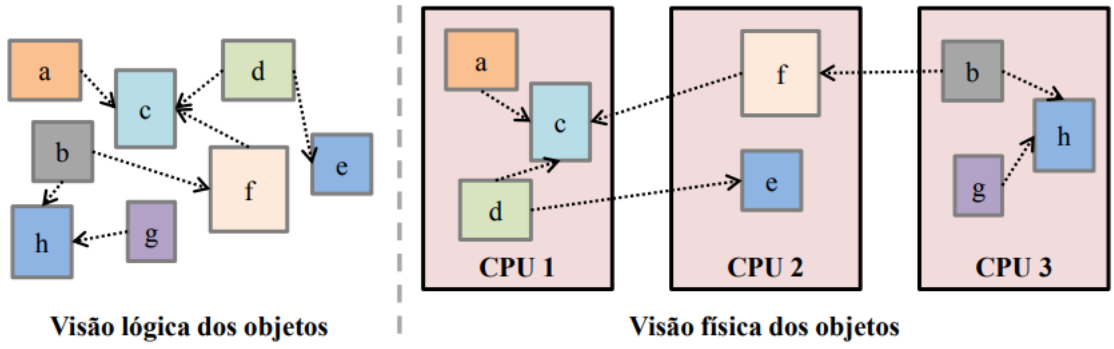
Fonte: (PILLA; MENESES, 2015b, p.32)

De acordo com (PILLA; MENESES, 2015b, p.30) O uso de um modelo de comunicação assíncrona é benéfico pois remove a obrigação de uma sincronização entre o emissor e o receptor de uma mensagem. Isso remove tanto a espera do emissor por uma resposta escondendo parte da latência de comunicação da plataforma quanto o bloqueio da unidade de processamento do receptor enquanto espera pela recepção de uma mensagem.

2.4.3 Modelo de execução

O processamento nas aplicações em Charm++ é decomposto em objetos chamados chares. O programador implementa as computações e comunicações descrevendo como esses objetos vao interagir e o ambiente de Charm++ gerencia as mensagens geradas por essas interações. Os objetos se comunicam através de chamadas remotas de métodos. Ainda, o ambiente é responsável pelo gerenciamento dos recursos arquiteturais (PILLA et al., 2011). Na Figura 3 podemos ver uma idéa de abstração de aplicação.

Figura 3 – Abstração de aplicação na plataforma Charm++

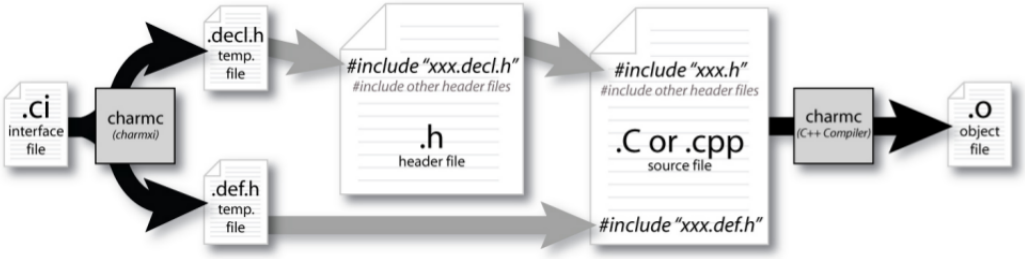


Fonte: (PILLA et al., 2011)

No CHARM++, os usuários definem suas aplicações em termos de objetos C++ especiais chamados de chares e coleções indexadas de chares chamados de arrays de chare. Para a maioria, os conjuntos de chares e chare são definidos exatamente como classes padrão e arrays C++, permitindo que o programador aproveite o encapsulamento e abstração de dados. Normalmente, há muitos mais chares do que PEs(processing elements), permitindo que o RTS(Runtime System) aproveite a composição. Como o CHARM ++ usa um paradigma baseado em objetos, o programador também pode implementar facilmente diferentes tipos de unidades de trabalho (ACUN et al., 2014).

No momento em que um projeto do Charm++ é compilado além dos arquivos de cabeçalho com extensão .h e os códigos padrões com extensões .C por exemplo, o compilador do Charm++ denominado CHARMC utiliza um tradutor que faz a leitura do arquivo de interface. Após a leitura do arquivo de interface são gerados 2 arquivos com extensões .def.h e .decl.h. Após essa etapa o compilador é utilizado novamente para gerar o arquivo de saída com extensão .o. Para rodar a aplicação é necessário utilizar um arquivo de execução do próprio Charm denominado CHARMRUN. Este Fluxo pode ser visto na Figura 4 a seguir:

Figura 4 – Processo de compilação de um projeto utilizando CHARM++



Fonte: (COMPONENTS. . . , 2017)

2.4.4 Balanceamento de carga

Muitas estratégias de balanceamento de carga em CHARM++ são baseadas em uma heurística conhecida como princípio da persistência. Ele postula que, empiricamente, para certas classes de aplicações científicas e de engenharia, quando elas são expressadas em um termo de objetos naturais, as cargas computacionais e padrões de comunicação tende a persistir ao longo do tempo, mesmo em cálculos com evolução dinâmica (**??**).

De acordo com (PILLA; MENESES, 2015b), O balanceamento de carga em CHARM++ é baseado na medição do tempo de atividade dos chares e das unidades de processamento. Durante um certo período de execução de uma aplicação, o sistema coleta uma serie de informações , esses dados são organizados pelo ambiente em um vetor de informações sobre chares e um vetor de informações sobre unidades de processamento, os quais são encaminhados a um algoritmo de balanceamento de carga. O algoritmo é responsável por avaliar as informações atuais recebidas e prover um novo mapeamento de chares para unidades de processamento.

Para habilitar o balanceamento de carga em uma aplicação em CHARM++, o programador é responsável apenas por implementar métodos de serialização para os objetos, por inserir uma chamada ao método bloqueante AtSync() em algum momento da execução de todos os chares e por definir um método de entrada ResumeFromSync(), o qual será chamado após o término do balanceamento de carga. Quando todos os chares alçarem a chamada AtSync, um algoritmo de balanceamento de carga externo, o qual foi compilado em anexo à aplicação e escolhido no momento de sua execução, computará um novo mapeamento.

2.5 Considerações do Capítulo

Neste capítulo foi apresentado os conceitos e definições das diferentes tecnologias relacionadas à computação de alto desempenho. Foi explanado as arquiteturas e plataformas que se encaixam neste contexto, além de apresentar alguns estudos e trabalhos relacionados.

No Capítulo 3, será apresentado o ambiente de trabalho onde os testes foram realizados, especificando as configurações dos dispositivos de processamento, armazenamento e benchmark que foram utilizados para a implementação dos testes.

3 SMARTLB

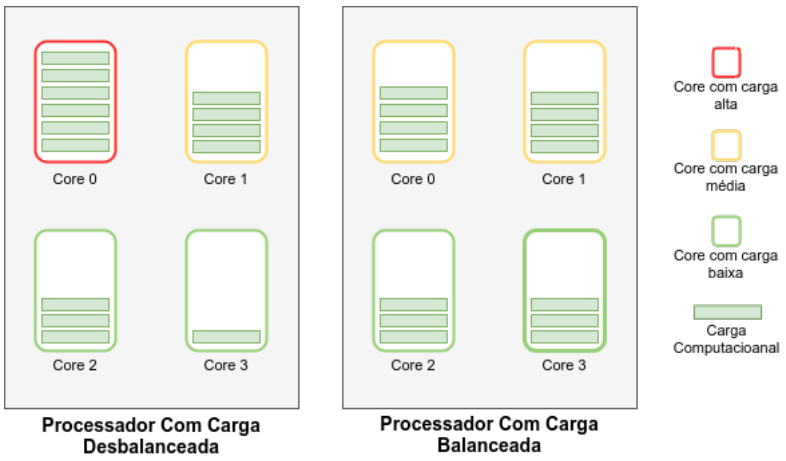
3.1 Proposta

A grande motivação para o desenvolvimento deste balanceador de carga proveio do desígnio da melhoria e correção de alguns problemas de um BC que utilizando uma abordagem gulosa na composição de seu algoritmo, onde trabalhava com a idéia de calcular a média aritmética do processador e com base nesse calculo então decidir qual tarefa iria ser migrada de fato. O Almejo de um ganho maior de desempenho na realização de testes com grandes cargas computacionais, dimanou na criação de um novo balanceador de carga que também utiliza uma abordagam centralizada, com uma estratégia de tomada de decisão um pouco diferente. Culminando em um ganho significativo de desempenho em relação a este BC específico e também a outros balanceadores que utilizam diferentes estratégias de tomada de decisão. O balanceador de carga em questão levou o nome de SmartLB .

3.2 Metodologias de Implementação

Atualmente muitas aplicações são dinâmicas ou realizam grandes cálculos computacionais, demandando cada vez de mais capacidade computacional. De acordo com (PADOIN et al., 2014a), o grande problema por trás disso é que na maioria das vezes não há uma preocupação com o desbalanceamento de carga gerado por estas aplicações, impedindo que as máquinas paralelas aproveitem todo o seu potencial. Na figura 5 podemos notar a diferença entre um processador balanceado e um que não está balanceado.

Figura 5 – Diferença Entre um Processador com Carga Desbalanceada e Balanceada



Também baseado nesse problema é que o SmartLB foi desenvolvido buscando atingir o estado de equilíbrio entre os processadores rapidamente, sem partir diretamente para os chares mais carregados evitando migrações desnecessárias. O BC proposto possui uma abordagem centralizada, então o processo de tomada de decisão e a estrutura de cargas e

comunicação da máquina ficam armazenadas em um único computador. Em virtude de realizar um balanceamento de carga mais preciso e trabalhar muito bem com escalabilidade é que foi escolhido a abordagem centralizada.

Durante o processo de execução, o BC coleta dados dos processadores e da aplicação e os armazena em um banco de dados de balanceamento de cargas. Dentre estas informações, destacam-se o número total de objetos, a carga total de cada processador, a carga de cada objeto e o número total de processadores.Estes dados serão utilizados na hora de decidir quais chares devem ser migrados e qual valor cada processador deve ter para estar em equilíbrio.

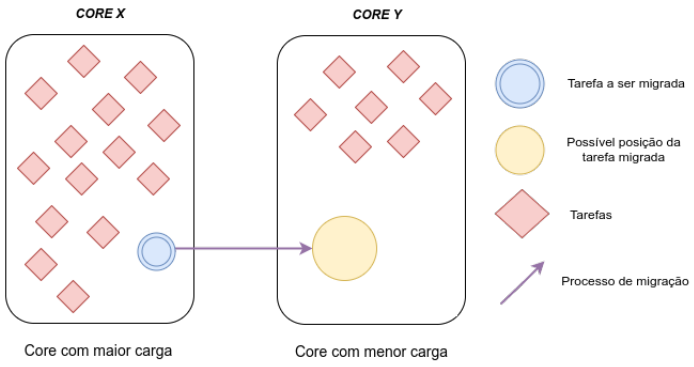
O SmartLB foi desenvolvido utilizando o framework de balanceamento de cargas disponibilizado pelo Charm++. Este framework de balanceamento de carga foi escolhido uma vez que permite tanto a criação de um novo BC quanto a utilização dos BCs disponibilizados pelo ambiente para comparações de resultados.

O Charm++adota uma metodologia baseada na medição das cargas das tarefas que executam em cada core. Para isso, o framework coleta automaticamente estatísticas da carga computacional e da comunicação destes objetos e armazena tais informações em uma base de dados que pode ser utilizada pelos BC para a tomada de decisões (JYOTHI; LAWLOR; KALÉ, 2004).

3.3 Algoritmo

Na Figura 6 é exemplificado o funcionamento da estratégia de tomada de decisão do SmartLB . A partir de informações fornecidas pelo Charm++, o algoritmo proposto busca atingir ba lanceamento levando em consideração a diferença de carga entre o core mais carregado e o menos carregado. Desta forma, migra tarefas do core mais carregado para o core com menor carga, buscando equilibrar a carga total do sistema, reduzir o número de migrações e reduzir o tempo total de execução da aplicação.

Figura 6 – Balanceamento de carga executado pelo SmartLB



A estratégia utilizada para implementação do balanceamento de carga proposto constitui-se de melhorias nas estrategias utilizadas nos algoritmos GreedyLBe RefineLB. Nossas

melhorias buscam equilibrar as cargas entre os processadores reduzindo o número de migrações, adotando um threshold para definir o desbalanceamento de carga aceitável.

Na Tabela 1 são apresentados os principais parâmetros utilizados na implementação do algoritmo proposto.

Tabela 1 – Principais parâmetros utilizados no algoritmo SmartLB

Parâmetro	**Definição**
PM	Core com maior carga
Pm	Core com menor carga
D	Desbalanceamento entre PM e Pm
ct	Carga da tarefa P
$nTarefas$	Número de tarefas
$getCoreAtual(i)$	Retorna o processador atual da tarefa
$getCargaTarefa()$	Retorna a carga da tarefa
$getCargaMaior()$	Retorna a carga do core mais carregado
$getCargaMenor()$	Retorna a carga do core menos carregado
$migrarProcesso(i, PM, Pm)$	Migrar tarefa i de PM para Pm

Quando o balanceador é aplicado primeira mente verifica as cargas do core mais carregado e o core menos carregado, a partir dessa informação ele analisa a diferença de carga entre o core mais e menos carregado. Caso essa diferença for maior que o *threshold* definido o balanceador busca tarefas do processador mais carregado testando se a carga da tarefa é menor ou igual ao desbalanceamento. Caso seja, ele realiza a migração desta tarefa do core mais carregado para o menos carregado e encontra o novo core mais carregado e o novo processador menos carregado, como demonstrado no Algoritmo 1.

Algoritmo 1: Implementação do SmartLB

```
1  PM = getCargaMaior();
2  Pm = getCargaMenor();
3  if((Pm/PM) > Threshold) {
4      for(i = 1; i <= nTarefas; i++) {
5          if(getCoreAtual(i) == PM) {
6              ct = getCargaTarefa(i);
7              D = PM − Pm;
8              if(ct <= D) {
9                  migrarProcesso(i, PM, Pm);
10                 PM = getCargaMaior();
11                 Pm = getCargaMenor();
12             }
13         }
14     }
15 }
```

Desta forma, consegue-se um balanceamento de carga mais preciso, pois caso o desbalanceamento entre o core mais e menos carregado muito pequeno o algoritmo não irá realizar nenhuma ação, e quando esse desbalanceamento for grande ele migra somente tarefas que carga menor a diferença. evitando muitas migrações desnecessárias. Após a execução, quando não existem mais cores a serem mapeados e as cargas de todas as unidades de processamento possuem um valor próximo um do outro, o balanceamento é encerrado.

4 METODOLOGIA

4.1 Balanceadores de Carga

Para analisar os resultados alcançados, o BC proposto foi comparado com outros três balanceadores de carga, dois deles disponíveis no CHARM++ e outro cujo foi base para a criação do SMARTLB . Estes balanceadores são:

- **RefineLB**: Disponibilizado juntamente com o CHARM++, com abordagem centralizada, o REFINELB é baseado em refinamento. O BC Move objetos dos core mais sobrecarregados para os menos carregados até atingir uma média, que é definida através de um método específico, limitando o número do objetos migrados (ARRUDA, 2015).
- **GreedyLB**: É um algoritmo de balanceamento de carga guloso, o qual remove todas as tarefas de seus núcleos e as mapeia em ordem decrescente de carga entre os núcleos com as menores cargas(FREITAS; PILLA, 2016). Em outras palavras a ideia central do GREEDYLB consiste na migração de objetos mais pesados para o processador coma menor carga, até que a carga de todos os processadores esteja próxima à carga média.
- **AverageLB**: De acordo com (ARRUDA et al., 2015), a estratégia utilizada para balanceamento de carga no algoritmo AVERAGELB constitui-se de uma melhoria da estratégia gulosa utilizada no algoritmo GREEDYLB o número de migrações, aumentando assim o desempenho dos sistemas de alto desempenho. A implementação utiliza uma abordagem centralizada e busca atingir balanceamento levando em consideração a média aritmética das cargas de cada processador. Foi através da necessidade de melhorias no desempenho do algoritmo do BC em questão que surgiu o SMARTLB .

4.2 Hardware Utilizado

Para validação da proposta, foi utilizado um equipamento com um processador Intel modelo i7- 6500U. Este processador possui 4 núcleos com 2 Simultaneous multithreading(SMT)/núcleo, totalizando 8 núcleos. Para os testes, utilizou-se o sistema operacional Linux Ubuntu 16.04 com kernel versão 4.4.33. A versão do CHARM++ utilizada foi a 6.5.1 e do compilador g++ a versão 5.4.1.

4.3 Benchmarks

Para gerar os resultados os balanceadores de carga foram submetidos a simulações utilizando 3 benchmarks, disponibilizados pelo próprio ambiente de programação CHARM++.

Os benchmarks escolhidos foram o LB_test, KNeighbor e o Stencil3D, que foram configurados da seguinte forma:

- **LB_Test**: O benchmark está configurado para realizar 150 iterações por tarefa, com sincronização a cada 10 iterações e uma carga computacional variante entre 1000ms e 150000ms.
- **KNeighbor**: O benchmark foi configurado com para realizar testes com 15000 mensagens, 150 iterações por tarefa sendo que o benchmark será sincronizado a cada 10 iterações.
- **Stencil3D**: O benchmark foi configurado para executar com 20 blocos, realizando 150 iterações por tarefas com sincronização do benchmark a cada 10 iterações.

Em ambos os benchmarks foram realizados testes alternando entre 50, 100 e 200 tarefas.

4.4 Considerações do Capítulo

5 RESULTADOS

5.1 Considerações do Capítulo

6 CONCLUSÃO

6.1 Trabalhos Futuros

REFERÊNCIAS

ACUN, B. et al. Parallel programming with migratable objects: Charm++ in practice. In: IEEE PRESS. **Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis**. [S.l.], 2014. p. 647–658.

ARRUDA, G. et al. Proposta de balanceamento de carga para redução de migração de processos em ambientes multiprogramados. In: **XVI Simpósio de Sistemas Computacionais (WSCAD-WIC)**. Florianópolis, SC: [s.n.], 2015. p. 1–8.

ARRUDA, G. H. S. Balanceamento de carga em sistemas multiprocessadores utilizando o modelo de programação charm++. 2015.

COMPONENTS of a Charm++ Program. 2017. <http://charmplusplus.org/tutorial/CharmComponents.html>. Acessado em: 23/11/2017.

FREITAS, V. M. C. T. de; PILLA, L. L. Um protótipo de algoritmo de balanceamento de carga guiado pelos próprios núcleos. **ERAD-RS**, 2016.

HENDRICKSON, B.; DEVINE, K. **Dynamic load balancing in computational mechanics**. **Computer methods in applied mechanics and engineering**, Elsevier, v. 184, n. 2, p. 485–500, 2000.

HUANG, C.; LAWLOR, O.; KALE, L. V. Adaptive mpi. In: SPRINGER. **International workshop on languages and compilers for parallel computing**. [S.l.], 2003. p. 306–322.

JYOTHI, R.; LAWLOR, O. S.; KALÉ, L. V. Debugging support for charm++. In: IEEE. **Parallel and Distributed Processing Symposium, 2004. 18th International**. [S.l.], 2004. p. 264.

KALE, L. Parallel programming with charm: An overview. **Dept. of Computer Science, University of Illinois at Urbana-Champaign, Tech. Rep**, p. 93–8, 1993.

KALE, L. et al. **The Charm Parallel Programming Language and System: Part I - Description of Language Features**. 1995.

KALÉ, L. V.; KRISHNAN, S. CHARM++: A portable concurrent object oriented system based on C++. In: ANNUAL CONFERENCE ON OBJECT-ORIENTED PROGRAMMING SYSTEMS, LANGUAGES, AND APPLICATIONS (OOPSLA). **Proceedings...** [S.l.]: ACM, 1993. p. 91–108. ISBN 0-89791-587-9.

KALE, L. V.; KRISHNAN, S. Charm++: a portable concurrent object oriented system based on c++. In: ACM. **ACM Sigplan Notices**. [S.l.], 1993. v. 28, n. 10, p. 91–108.

KUNZMAN, D. et al. Charm++, offload api, and the cell processor. **Urbana**, v. 51, p. 61801, 2006.

MENON, H.; KALÉ, L. A distributed dynamic load balancer for iterative applications. In: INTERNATIONAL CONFERENCE FOR HIGH PERFORMANCE COMPUTING, NETWORKING, STORAGE AND ANALYSIS (SC), Denver, CO. **Proceedings...** [S.l.]: ACM, 2013. p. 15.

PADOIN, E. L. et al. Saving energy by exploiting residual imbalances on iterative applications. In: IEEE. **High Performance Computing (HiPC), 2014 21st International Conference on**. [S.l.], 2014. p. 1–10.

PADOIN, E. L. et al. Balanceamento de carga visando reduçao do consumo de energia para o modelo de programaçao charm++. **XIV Escola Regional de Alto Desempenho do Estado do Rio Grande do Sul, Alegrete, RS, Brasil**, 2014.

PADOIN, E. L.; NAVAUX, P. O. A.; MéHAUT, J.-F. Using power demand and residual load imbalance in the load balancing to save energy of parallel systems. In: **International Conference on Computational Science (ICCS)**. Zurich, Switzerland: [s.n.], 2017. p. 1–8.

PHILIPPSEN, M.; ZENGER, M. Javaparty - transparent remote objects in java. **Concurrency Practice and Experience**, Chichester, Sussex: J. Wiley, c1989-c2000., v. 9, n. 11, p. 1225–1242, 1997.

PILLA, L. L.; MENESES, E. Analise de desempenho da paralelização do problema de caixeiro viajante. **XV Escola Regional de Alto Desempenho do Estado do Rio Grande do Sul, Gramado, RS, Brasil**, 2015.

PILLA, L. L.; MENESES, E. Programação paralela em charm++. **XV Escola Regional de Alto Desempenho do Estado do Rio Grande do Sul, Gramado, RS, Brasil**, 2015.

PILLA, L. L. et al. A topology-aware load balancing algorithm for clustered hierarchical multi-core machines. **Future Generation Computer Systems**, Elsevier, v. 30, p. 191–201, 2014.

PILLA, L. L. et al. Avaliaç ao da adequaç ao da plataforma charm++ para arquiteturas multicore com memória hierárquica. **X Workshop em Desempenho de Sistemas Computacionais e de Comunicaç ao, Natal, RN, Brasil**, 2011.

PINHO, E. G. **Uma linguagem de programação paralela orientada a objetos para arquiteturas distribuídas**. Tese (Doutorado), 2012.

PINHO, E. G.; JUNIOR, F. H. de C. A language for object-oriented parallel programming targeted at cluster computing platforms. **14th Brazilian Symposium on Programming Languages**, SBC, 2010.

SANTOS, V. R. S. D. et al. Proposta de balanceamento de carga para redução do tempo de execução de aplicações em ambientes multiprocessados. **XVIII Simpósio em Sistemas Computacionais de Alto Desempenho (WSCAD-WIC)**, 2017.

SANTOS, V. R. S. D.; PADOIN, E. L. Análise de desempenho da aplicação de balanceamento de carga em sistemas multiprocessadores. **Salão do Conhecimento 2017 - Ijuí RS**, v. 3, n. 3, 2017.

ZHENG, G. et al. Periodic hierarchical load balancing for large supercomputers. **International Journal of High Performance Computing Applications**, SAGE Publications, v. 25, n. 4, p. 371–385, 2011.

ZHENG, G. et al. Hierarchical load balancing for charm++ applications on large supercomputers. In: IEEE. **Parallel Processing Workshops (ICPPW), 2010 39th International Conference on**. [S.l.], 2010. p. 436–444.