Proposta de balanceamento de carga para a redução do tempo de execução em ambientes multiprocessados *

Vinicius R. S. dos Santos [1], Ana Karina M. Machado [1], Edson L. Padoin[1,2]

[1] Universidade Reg. do Noroeste do Estado do Rio G. do Sul (UNIJUI) – Ijui, RS – Brazil
[2] Universidade Federal do Rio Grande do Sul (UFRGS) – Porto Alegre, RS – Brazil
{vinicius.ribas, ana.morales, padoin}@unijui.edu.br

Resumo

Este artigo apresenta uma proposta de um novo balanceador 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 CHARM++. *Os resultados preliminares apresentaram reduções significativas no número de tarefas migradas assim como no tempo total de execução.*

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.

A partir da evolução do *hardware* do processador, as limitações foram-se sendo ultrapassadas, proporcionando a execução simultâea de diversas aplicações. Isso habilitou à programação pralela a divisão das aplicações em partes inseridas nos núcleos das unidades de processamento [6].

A modelagem de um problema complexo pode culminar em um desbalanceamento de carga e excessiva comunicação entre tarefas [5]. 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 [4].

Soluções que utilizam estratégias para o emprego adequado dos recursos disponíveis vem sendo desenvolvidas. Balanceadores de Carga(BC), nome dado à estratégia, tem por objetivo detectar e corrigir o desbalanceamento de carga, aprimorando a utilização dos recursos disponíveis a aplicação [3].

O restante deste artigo está assim organizado. A Seção 2 apresenta os trabalhos relacionados. A Seção 4 apresenta a proposta do balanceador. A Seção 3 descreve a metodologia que será empregada na implementação do novo balanceador de carga. Por fim, são discutidos na Seção 6, algumas conclusões e perspectivas de trabalhos futuros.

2. Trabalhos Relacionados

Várias plataformas possuem suporte à programação paralela com memória compartilhada e distribuída. Porém, aplicações que foram desenvolvidas utilizando linguagens procedurais, baseadas em um paradigma de troca de mensagem paralelo, apresentam padrões de carga dinâmico, o que tornaria necessária a mudança na estrutura da aplicação [1] para haver um balanceamento de carga.

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 atualmente novas abordagem hierárquica vem sendo propostas. Nestas novas abordagens, os núcleos de processamento são divididos em grupos independentes e organizados de 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 con-

* This work was supported by CNPq, CAPES, FAPERGS and FINEP. This research has received funding from the European Community's Seventh Framework Programme (FP7-PEOPLE) under grant agreement number 295217, funding from the EU H2020 Programme and from MCTI/RNP-Brazil under the HPC4E Project, grant agreement number 689772 and STIC-AmSud/CAPES scientific-technological cooperation program under EnergySFE research project grant 99999.007556/2015-02.

segue *speedup* de 6 com 2.048 *cores* e 145 com 8.192 *cores* [7].

Outros balanceadores abordados no artigo são o RefineLB e o GreedyLB. Ambos são disponibilizados pelo ambiente de programação CHARM++. O BC RefineLB move objetos dos processadores mais sobrecarregados para os menos carregados almejando atingir uma média, sendo limitado o número do objetos migrados [2]. Já o BC GreedyLB possui um algoritmo de abordagem gulosa. Esse paradigma é frequentemente utilizado na teoria e na prática de otimização combinatória. Seu algoritmo objetiva migrar objetos pesados para o processador com menor carga. Isso se repete até que a carga de todos os processadores alcance uma proximidade com a carga média.

3. Metodologia

Para validação da proposta, foi utilizado um equipamento com um processador Intel modelo i7-6500U. Este processador possui 4 núcleos com 2 SMT/núcleo, totalizando 8 núcleos. Para os testes, utilizou-se o sistema operacional Linux Ubuntu 16.04 com kernel versão $4.4.33 - 1$. A versão do CHARM++ utilizada foi a 6.5.1 e do compilador g++ a versão 5.4.1.

Os balanceadores de carga foram submetidos a simulações utilizado o *benchmark* LB_Test. Esse *benchmark* foi escolhido por ser facilmente configurável para apresentar diferentes níveis de desbalanceamento de carga, permitindo que a carga computacional de cada tarefa seja configurada em diferentes padrões de carga, tanto de irregularidade quanto de comunicação, além de ser disponibilizado pelo próprio ambiente de programação.

Testes foram realizados com 50 e 100 tarefas, sendo estas com cargas computacionais que variam entre $1500ms$ e 150000ms. As sincronizações para chamada do balanceador de carga foi definidas a cada 10 iterações.

Os resultados alcançados, tempo total de execução e a quantidade total de objetos migrados, foram comparados com os balanceadores de carga REFINELB e GREEDYLB.

4. SmartLB

A implementação do balanceador de cargas SMARTLB foi realizada sob o modelo de programação CHARM++, que é um paradigma paralelo voltado à produtividade. Este *framework* de balanceamento de carga foi escolhido uma vez que permite tanto a criação de novos BC quanto a utilização dos BCs disponibilizados pelo ambiente para comparações de resultados.

O CHARM++ é usado para o desenvolvimento de grandes aplicações científicas, possibilitando a execução de programas paralelos com balanceamento de carga automático e *checkpointing* [6].

A estratégia utilizada para implementação do balanceamento de carga proposto, constitui-se de melhorias nas estratégias utilizadas nos algoritmos GREEDYLB e 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.

A partir de informações fornecidas pelo CHARM++, o algoritmo SMARTLB busca atingir balanceamento levando em consideração a diferença de carga entre o núcleo mais carregado e o menos carregado. Migrando tarefas do processador mais carregado para o processador com menor carga, equilibrando a carga total do sistema, reduzindo o número de migrações e o tempo total de execução da aplicação.

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

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

Tabela 1. Principais parâmetros utilizados no algoritmo SMARTLB

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(getProcessadorAtual(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 }
```

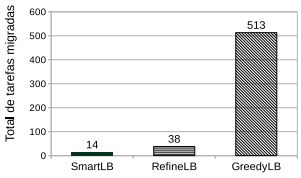
Assim, quando o balanceado é aplicado, ele primeiro analisa a diferença de carga entre o processador mais e menos carregado. Caso essa diferença for maior que o *threshold* o balanceador busca tarefas do processador mais carregado testando se a carga da tarefa é menor ou igual ao

desbalanceamento. Caso seja, ele realiza a migração desta tarefa do processador mais carregado para o menos carregado e encontra o novo processador mais carregado e o novo processador menos carregado, como demonstrado no Algoritmo 1.

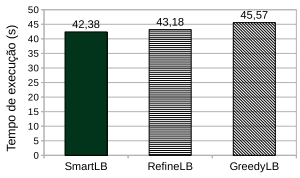
Desta forma, consegue-se um balanceamento de carga mais preciso, evitando migrações desnecessárias. Após a execução, quando não existem mais processadores a serem mapeados e as cargas de todas as unidades de processamento possuem um valor próximo um do outro, o balanceamento é encerrado.

5. Resultados

Durante os testes, foram levados em consideração o tempo total de execução e a quantidade total de tarefas migradas.



(a) Total de tarefas migradas

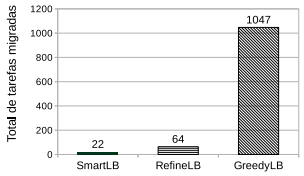


(b) Tempo total de execução em segundos

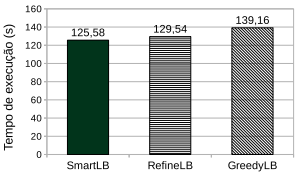
Figura 1. Resultados dos testes com 50 tarefas

Analisando a Figura 1 Com base nos tempos totais de execução mensurados, observa-se que quando executado os testes com 50 tarefas, o menor tempo de execução é alcançado pelo algoritmo do SMARTLB. Apresentando um tempo 42,34 segundos, tendo uma diferença de $6,94\%$ menor que o algoritmo do GREEDYLB que por sua vez obteve o maior tempo, com 45,54 segundos.

Observando a quantidade total de tarefas migradas, nota-se quando realizados testes com com 50 tarefas, que o SMARTLB foi o BC com menor migrações de tarefas, com apenas 14 migrações enquanto o REFINELB e o GREEDYLB migraram 38 e 513 tarefas respectivamente.



(a) Total de otarefas migradas



(b) Tempo total de execução em segundos

Figura 2. Resultados dos testes com 100 tarefas

Podemos notar na Figura 2, que dobrando a quantidade de tarefas para 100, mais uma vez o SMARTLB alcançou o menor tempo de execução, bem como também foi o BC que apresentou a menor quantidade de tarefas migradas.

Baseado no tempo total de execução aferido nos gráficos da Figura 2, nota-se que, quando realizados os testes com 100 processos, o algoritmo do SMARTLB obtém um tempo de execução $3,06\%$ menor que algoritmo do REFINELB. Esse, por sua vez, obteve um tempo $6,91\%$ menor que o BC GREEDYLB, que atingiu o maior tempo, executando seu algoritmo em 139,16 segundos.

Nota-se que aumentando-se o número de tarefas para 100, o algoritmo do GREEDYLB migra 1047 tarefas, enquanto os algoritmos do SMARTLB e do REFINELB migram apenas 22 e 64 tarefas respectivamente. A disparidade na quantidade de tarefas migradas pelos balanceadores SMARTLB e REFINELB com o GREEDYLB é consequência da sua estratégia gulosa que migra tarefas do processador mais carregado para o menos carregado.

Os algoritmos do RefineLB e do SMARTLB fazem o cálculos para determinar quais tarefas devem ser migrados, evitando desperdícios e, consequentemente, diminuindo a quantidade de tarefas migradas.

6. Conclusão

Este artigo apresentou uma proposta de balanceamento de carga para a redução do tempo de execução de aplicações paralelas executadas em ambientes multiprocessados. Os resultados obtidos foram comparados com por outros dois balanceadores de carga.

O balanceador de carga proposto SMARTLB apresentou melhor desempenho nos testes realizados com o benchmark lb_test, tanto na quantidade de tarefas migradas quanto no tempo total de execução quando comparação com os balanceadores de carga REFINELB e GREEDYLB. Estes resultados positivos são alcançados devido ao maior controle na migração das tarefas, evitando assim migrações desnecessárias.

Como futuros trabalhos, pretende-se realizar melhorias no algoritmo SMARTLB almejando gerenciar ainda mais o controle de migrações de tarefas. Pretende-se também realizar testes em sistemas paralelos utilizando problemas reais de computação científica e comparar com outros balanceadores de carga do estado da arte.

Referências

[1] M. Bhandarkar, L. V. Kalé, E. de Sturler, and J. Hoeflinger. Adaptive load balancing for mpi programs. In *International Conference on Computational Science*, pages 108–117. Springer, 2001.

[2] G. Freytag, G. Arruda, R. S. Martins, and E. L. Padoin. Análise de desempenho da paralelizaç ao do problema de caixeiro viajante. 2015.

[3] B. S. Padilha and E. L. Padoin. Análise de desempenho da aplicação de balanceamento de carga em benchmark sintéticos. *Salão do Conhecimento*, 2(2), 2016.

[4] E. L. Padoin, M. Castro, L. L. Pilla, P. O. Navaux, and J.-F. Méhaut. Saving energy by exploiting residual imbalances on iterative applications. In *High Performance Computing (HiPC), 2014 21st International Conference on*, pages 1–10. IEEE, 2014.

[5] E. L. Padoin, M. B. Castro, L. L. Pilla, T. C. Bozzetti, P. O. A. Navaux, and J.-F. Méhaut. 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.

[6] L. L. Pilla and E. Meneses. teste. *XV Escola Regional de Alto Desempenho do Estado do Rio Grande do Sul, Gramado, RS, Brasil*, 2015.

[7] G. Zheng, E. Meneses, A. Bhatele, and L. V. Kale. Hierarchical load balancing for charm++ applications on large supercomputers. In *Parallel Processing Workshops (ICPPW), 2010 39th International Conference on*, pages 436–444. IEEE, 2010.