Uma Abordadagem de Co-Escalonamento Adaptativa para Ambientes de Computação Oportunista de Configurações Multiprocessadas

R.P. Mendonça, M.M. Piffer, D.R. Viegas, and M.A.R. Dantas

Universidade Federal de Santa Catarina (UFSC)
Departamento de Informática e Estatística (INE)
Florianópolis - SC, Brasil
{rodrigop,m_piffer,diogo,mario}@inf.ufsc.br

Resumo Configurações multi-cores e de multi-processadores vêm sendo utilizadas por muitas organizações com o propósito de aumentar o poder computacional para as suas aplicações. Contudo, o aumento no uso dessas configurações traz um novo desafio para a comunidade científica, que pode ser traduzido como: o limitado número de aplicações desenvolvidas visando aproveitar toda a capacidade dessas configurações. Sistemas de computação oportunista, os quais visam usar a capacidade ociosa dos recursos disponíveis, também precisam se adaptar a essas novas configurações. Caso contrário, não é esperado que esses sistemas utilizem de forma eficiente os recursos multiprocessados. Neste artigo, é apresentada uma abordagem de co-escalonamento adaptativo para ser agregada a um sistema de computação oportunista, buscando aproveitar toda a capacidade ociosa de recursos multi-cores e de multi-processadores. Essa proposta adota o uso de múltiplas *threads* para alcançar um melhor aproveitamento de cada unidade de processamento disponível. Resultados empíricos indicam que o esforço obteve sucesso em garantir o uso eficiente dos recursos multiprocessados e também em reduzir o tempo total de uma aplicação utilizada para testes.

1 Introdução

Vem sendo verificada uma tendência no uso de configurações multi-core e de multi-processadores fracamente acoplados, tendo em vista melhorar o desempenho de aplicações nas organizações. No entanto, como [1] observa, essas configurações sozinhas não representam a resposta para um grande número de aplicações. Em outras palavras, um *middleware*, adotando alguns parâmetros, é necessário para auxiliar na submissão de aplicações para esses ambientes.

Sistemas de computação oportunista (e.g. [2], [3], [4] e [5]), podem ser classificados como *middlewares* que permitem a execução de aplicações complexas em grandes configurações de sistemas distribuídos. Esses sistemas, também chamados de *metacomputing systems*, são usualmente projetados para reunir os recursos ociosos disponíveis para executar aplicações que requerem um uso intenso de processadores. Alguns exemplos dessas aplicações são: seqüenciamento molecular [6], otimização combinatória [7] e pesquisa de números primos [8].

XXXIV Conferencia Latinoamericana de Informática (CLEI 2008), pp. 803-812, 2008.

Neste trabalho de pesquisa é apresentado um estudo empírico do uso de um Co-Escalonador de tarefas para o sistema de processamento oportunista ATHA [5]. O objetivo principal é reunir toda a capacidade ociosa de um ambiente de configurações multi-core e de multi-processadores fracamente acoplados, para ser eficientemente usado por uma aplicação demonstrativa.

Este artigo está organizado como segue. A seção 2 ilustra alguns trabalhos relacionados com o presente artigo. Na seção 3 são explicadas algumas características do ambiente de processamento oportunista ATHA, um ambiente desenvolvido pelo nosso grupo de pesquisa e estendido neste trabalho. A motivação e a contribuição deste trabalho são apresentadas na seção 4. O ambiente experimental adotado na pesquisa e os resultados empíricos são descritos na seção 5. Finalmente na seção 6, algumas conclusões e trabalhos futuros são apontados.

2 Trabalhos Correlatos

Várias pesquisas têm sido focadas no uso eficiente de configurações multi-cores e de multi-processadores. Em [9] é apresentada uma investigação dos efeitos de particionamento, alocação e granularidade de tarefas em configurações com múltiplas unidades de processamento. Alguns aspectos sobre o uso de aplicações MPI em ambientes de configurações multi-core e de multi-processadores fracamente acoplados são apresentados em [10]. O trabalho desenvolvido em [11], explora algumas políticas de co-escalonamento em máquinas multiprocessadas.

Na literatura é possível observar alguns projetos de sistemas de processamento oportunista, visando resolver o problema do uso de ciclos ociosos no processamento de aplicações de granularidade grossa. Exemplos clássicos desses sistemas são: BOINC [2], POPCORN [3] e Charlotte [4].

Com o objetivo de observar como esses projetos lidam com o problema do presente artigo (i.e. investigar o comportamento desses sistemas diante de *hosts* multiprocessados), nós selecionamos o sistema BOINC para executar alguns experimentos preliminares. O BOINC foi escolhido por ser um projeto de pesquisa aberto e possui muitos dados relacionados ao projeto que podem ser encontrados em [2]. O ambiente utilizado foi formado por uma máquina Pentium Dual-Core, com 1 GByte de RAM, rodando o sistema operacional Windows XP. Com o auxílio do Gerenciados de Tarefas do Windows, foi possível observar que o BOINC já possui um mecanismo que submete mais do que uma tarefa para os recursos dotados de múltiplas unidades de processamento. Dessa forma, esse sistema já é capaz de se adaptar às diversas configurações disponíveis, quanto ao número de unidades de processamento. Em [12], o autor descreve algumas políticas utilizadas no escalonamento local do BOINC.

3 Sistema de Computação Oportunista ATHA

O sistema de computação oportunista ATHA [5][13], foi desenvolvido pelo nosso grupo de pesquisa [14], com o objetivo de reunir a capacidade ociosa dos recursos de um ambiente (Intranet ou Internet) para a execução aplicações complexas.

O ATHA possui uma abordagem similar ao BOINC, roubando ciclos ociosos de máquinas em uma rede. O sistema foi desenvolvido na linguagem de programação Java, tendo em vista a diversidade dos recursos encontrados em um grande sistema distribuído. O sistema usa o paradigma mestre-escravo para executar as tarefas. Sendo assim, um módulo mestre foi desenvolvido, onde uma classe Java é submetida, para ser executada de modo paralelo e distribuído. Nesse momento, também é necessário informar como parâmetro, o intervalo de dados que será processado pela aplicação escolhida. O ATHA possui também o módulo escravo, o qual é responsável por executar remotamente nos *hosts* escravos algumas partes da aplicação principal.

Uma vez submetida a classe no módulo mestre do ATHA, esse será responsável por dividir os dados recebidos como argumentos em pedaços menores e iguais. Esses novos intervalos serão submetidos para os *hosts* disponíveis, sendo que na versão original do ATHA, cada *host* recebe e executa um pedaço do problema por vez. Quando cada *host* conclui a sua tarefa, ele retorna o resultado parcial para o *host* mestre, que mais uma vez irá submeter um novo intervalo do problema para ser processado remotamente, enquanto o resultado final da aplicação não for alcançado.

4 Motivação e Contribuição

A principal motivação do presente estudo foi realizar um teste empírico da proposta apresentada em [15], usando um ambiente real. Mais especificamente, o objetivo dessa pesquisa pode ser entendido, como uma investigação dos resultados alcançados ao agregar uma estratégia de co-escalonamento adaptativo em um sistema de processamento oportunista, quando consideramos um ambiente distribuído de configurações multi-core e de multi-processadores fracamente acoplados.

O ATHA foi escolhido como o sistema de processamento oportunista pelas seguintes razões: foi desenvolvido pelo nosso grupo de pesquisa [14]; possui o código fonte desenvolvido em Java, o que é uma propriedade relevante quando consideramos a diversidade de máquinas disponíveis na Internet; e finalmente por ter o código aberto, o que é essencial para estudos futuros e melhorias no projeto.

Considerando a versão original do ATHA, foram necessárias modificações nos módulos mestre e escravo. Isso porque quando um recurso multiprocessado é identificado no ambiente, o módulo mestre deve ser capaz de enviar novos intervalos de dados para esses recursos, o que não é verdade na versão original do ATHA, como já comentado. Uma vez enviado novos intervalos, estes serão processados nos *hosts* escravos através de múltiplas *threads* da aplicação principal. Assim, espera-se que todo o processamento ocioso desses recursos seja usado, independente da diversidade de configurações encontradas no ambiente. Em outras palavras, a estratégia considera que as máquinas podem ter múltiplas unidades de processamento e a aplicação submetida não foi desenvolvida com a preocupação de aproveitar as arquiteturas multiprocessadas disponíveis.

5 Ambiente e Resultados Experimentais

5.1 Ambiente Experimental

O algoritmo RC5 [16] foi selecionado como a aplicação teste para ser executada no sistema oportunista ATHA. Esse algoritmo foi escolhido pois representa uma aplicação *CPU-Bound*, ou seja, faz uso intenso de CPU. Uma outra razão pela escolha dessa aplicação nos experimentos, se deu porque esse algoritmo foi extremamente testado no sistema ATHA na sua versão original [5].

Os recursos utilizados em nossos testes empíricos são parte do ambiente de uma companhia brasileira de desenvolvimento de *software*. As máquinas dessa organização usualmente executam tarefas como: compilação de programas; edição de textos e planilhas eletrônicas; processamento de aplicações CRM; e-mail pessoal e VOIP.

A tabela 1 mostra alguns detalhes das máquinas utilizadas em nossos testes. Ademais, esses *hosts* estão interconectados através de Switches Gigabit Ethernet da 3Com, modelo SuperStack 3. Placas de rede da Broadcom NetXtreme Gigabit Ethernet foram utilizadas para a conexão de cada *host*.

Tabela 1. Configuração do Ambiente Experimental

Máquina	Arquitetura	RAM (MByte)	S.O.
Mestre	Intel Pentium 4 HT CPU 2.80 GHz	512	Windows XP 5.1
(A)	Intel Xeon 4 CPUs 3.2 GHz	6144	Linux Suse 8
(B)	Intel Pentium 4 HT CPU 2.80 GHz	512	Windows XP 5.1
(C)	Intel Celeron CPU 2.8 GHz	512	Windows XP 5.1
(D)	Intel Pentium Dual Core CPU 1.6 Ghz	1024	Windows XP 5.1

Visando um maior controle dos testes e um melhor acompanhamento dos resultados, foi decidido inicialmente usar apenas cinco máquinas. Uma outra razão pela decisão de usar menos máquinas, se deu pelo fato de que com poucas máquinas foi possível analisar, com mais detalhes, o comportamento de cada recurso diante da estratégia de co-escalonamento proposta. Das máquinas selecionadas, apenas uma foi utilizada como mestre, enquanto que as outras quatro foram as escravas. Vale ressaltar o cuidado tomado na escolha das quatro máquinas escravas, visto que cada uma delas representa um tipo diferente das configurações disponíveis na organização.

Com o propósito de verificar a importância de uma estratégia de co-escalonamento em um ambiente de computação oportunista de recursos multi-core e de multi-processadores, foi utilizado o sistema ATHA na sua versão original, sendo que nessas condições o sistema não foi capaz de reunir toda a capacidade ociosa dos recursos para processamento da aplicação teste. A figura 1 ilustra a tela do modulo mestre do ATHA, quando este submete apenas uma tarefa para ser executada na máquina escrava de IP 10.15.10.1. Vale ressaltar que esse IP refere-se a máquina escrava (A), ou seja, uma máquina dotada de quatro processadores.

A ferramenta TOP [17] do Unix foi utilizada em nossos testes como forma de monitorar a utilização das CPUs da máquina escrava (A). Como resultado desse primeiro monitoramento, foi possível verificar com a figura 2, que apenas um processo Java foi iniciado apesar dos quatro processadores disponíveis. Além disso, essa figura também mostra que o processo usa 97,3% de uma unidade de processamento, deixando as CPUs 0, 1 e 3 com uma ociosadade de 57,8%, 46,6% e 57,5%, respectivamente.

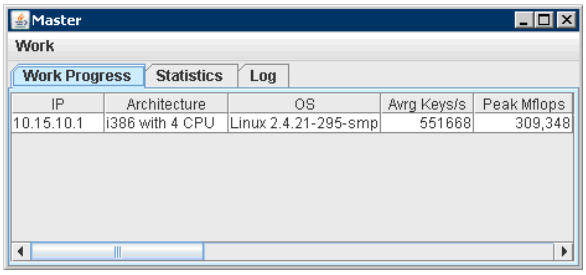

Figura 1. ATHA sem Co-Escalonamento para máquina de quatro processadores.

```
top - 23:14:34 up 5 days, 21:00,  9 users,  load average: 2.97, 2.37, 2.07
Tasks: 566 total,   2 running, 564 sleeping,   0 stopped,   0 zombie
 Cpu0 :  14.4% user,  27.8% system,   0.0% nice,  57.8% idle
 Cpu1 :  13.4% user,  40.1% system,   0.0% nice,  46.6% idle
 Cpu2 : 100.0% user,   0.0% system,   0.0% nice,   0.0% idle
 Cpu3 :  19.4% user,  23.1% system,   0.0% nice,  57.5% idle
Mem:   6210920k total,  6163400k used,    47520k free,    54164k buffers
Swap:  2097136k total,    33112k used,  2064024k free,  3460960k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
16370 rodrigom  25   0 71148  69m 5716 R 97.3  1.1   0:06.36 java
18095 rodrigom  25   0  1168 1168  668 R 34.4  0.0   1:25.41 top
```

Figura 2. Top na máquina escrava de quatro processadores sem Co-Escalonamento.

Após esse teste, realizamos o mesmo experimento usando a versão do ATHA com a abordagem de co-escalonamento proposta. Isso, com o objetivo de acompanhar o comportamento do ambiente iniciando múltiplas *threads* nos recursos escravos de configurações multiprocessadas. Sendo assim, usamos a mesma aplicação teste com as mesmas máquinas mestre e escrava do teste anterior.

A figura 3 ilustra quatro tarefas iniciadas para a máquina escrava de IP 10.15.10.1. Ademais, na figura 4 podem ser verificados quatro processos Java iniciados na máquina escrava. Ainda na figura 4 pode ser verificado que esses processos estão consumindo uma fatia relevante da capacidade total de processamento da máquina escrava. Com isso, a ociosidade das quatro unidades de processamento disponíveis ficou em 0%. Dessa forma, esse experimento demonstra que quando utilizado uma estratégia de co-escalonamento, foi alcançado uma maior eficiência no uso do recurso multiprocessado disponível.

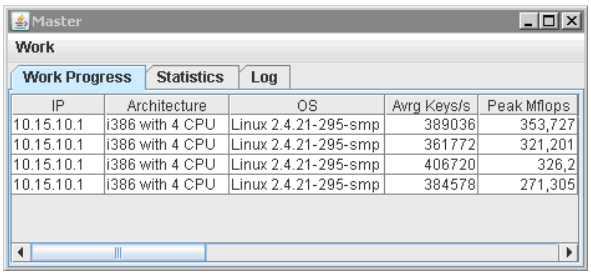

Figura 3. ATHA com Co-Escalonamento para máquina de quatro processadores.

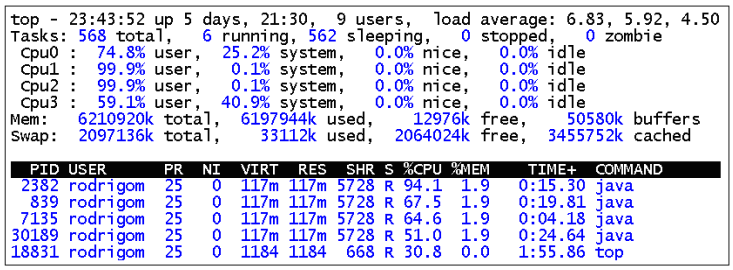

Figura 4. Top na máquina escrava de quatro processadores com Co-Escalonamento.

5.2 Resultados Empíricos

Após essa primeira análise, podemos inferir que o sistema ATHA quando executado em um ambiente de recursos com múltiplas unidades de processamento não utiliza de forma eficiente toda a capacidade de processamento disponível.

Novos experimentos comparativos foram necessários para analisar com detalhes os benefícios de usar o conceito de múltiplas *threads* em um ambiente de computação oportunista de recursos com múltiplas unidades de processamento.

O sistema ATHA possui relatórios de processamento que apresentam a capacidade de processamento em MFlops, de cada *host*, através do *benchmark* Linkpack [13]. O ATHA também captura o número de chaves testadas por segundo em cada *host* escravo. Para analisar cada unidade de processamento de um recurso multiprocessado, foi necessário adaptar o ATHA, para considerar as informações dos relatórios em nível de unidade de processamento.

5.3 Performance

A figura 5 apresenta um gráfico da capacidade de processamento de cada *host* escravo. Com essa figura pode-se verificar que o ambiente experimental utilizado é formado por duas máquinas com mais capacidade de processamento (A e D), uma máquina intermediária (B) e uma mais limitada (C). Outro ponto observado nesse gráfico é a variação de performance entre as máquinas, visto que, podemos

observar uma grande oscilação da linha que representa a máquina (A). Esse fato ocorre, porque essa máquina é um servidor que executa um grande número de tarefas concomitantes da empresa. Por outro lado, as linhas que representam os outros *hosts* (B, C e D) apresentam um comportamento mais estável.

Complementando a análise da figura 5, foi gerada a figura 6 que representa a somatória da performance das máquinas escravas. Pode-se observar que no melhor momento, a performance global do ambiente chega a 1.000 MFlops e aos 22 minutos, com o resultado do algoritmo alcançado, a linha do gráfico finaliza.

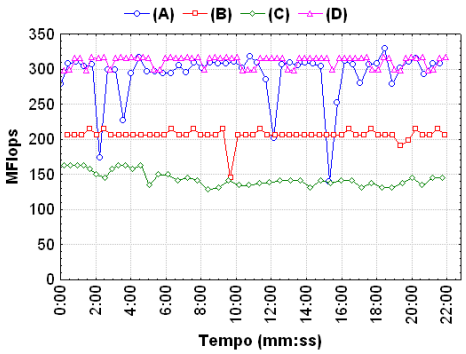

Figura 5. Performance individual dos recursos no ATHA sem Co-Escalonamento.

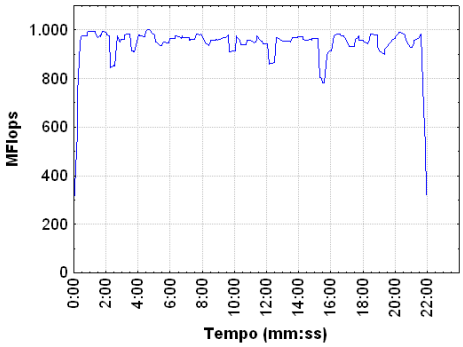

Figura 6. Performance Total do ATHA sem Co-Escalonamento.

Depois da análise da capacidade de processamento com a versão original do ATHA, foram realizados os mesmos experimentos, porém dessa vez utilizando a abordagem de co-escalonamento proposta para melhorar o sistema ATHA.

O gráfico da figura 7, adotando a estratégia de co-escalonamento proposta, apresenta a variação de performance de cada unidade de processamento disponível. As unidades de processamento são identificadas pelo número que segue a letra que representa cada máquina, por exemplo, o identificador (A0) trata-se da unidade de processamento 0 da máquina (A). Nesse gráfico, a oscilação das linhas que representam as unidades de processamento da máquina escrava (A) é mais expressiva, quando comparamos com a mesma linha da figura 5. Já para as outras máquinas disponíveis, as linhas apresentam-se semelhantes quando comparamos com a figura 5. Isso se deve ao fato de que a máquina (A), como citado anteriormente, trata-se de um servidor de aplicações constantemente requisitado. Assim, quando é submetida apenas uma tarefa, o servidor consegue gerenciar essa tarefa para ser processada por uma unidade de processamento mais ociosa. No entanto, quando são submetidas quatro tarefas, o servidor distribui uma tarefa para cada unidade de processamento, o que gera uma notável concorrência com as outras tarefas submetidas à esse servidor de aplicações.

Ademais, com o gráfico da figura 7 é possível verificar que a linha que representa a unidade de processamento da máquina (C), se mostra superior do que as linhas da máquina (B), já no gráfico 5 a linha da máquina (C) se mo-

stra inferior à da máquina (B). Nós acreditamos que isso se deve ao fato de que a máquina (B) não possui, de fato, múltiplas unidades de processamento, mas utiliza a tecnologia *Hyper-Threading Technology*[18]. Essa tecnologia provê um paralelismo em nível de *threads*, buscando tornar mais eficiente o processamento de tarefas concomitantes. Essa diferença não é observada para as máquinas (A) e (D), que possuem dois cores e quatro processadores, respectivamente.

Finalmente, com a figura 7, observamos que o tempo total de processamento da aplicação diminuiu para 12 minutos e 30 segundos, quando utilizada a estratégia de co-escalonamento proposta, enquanto que sem o co-escalonamento esse tempo fica em 22 minutos como visto nas figuras 5 e 6.

Adicionando à análise de performance individual, foi gerada a figura 8 que apresenta a somatória da performance dos recursos (cada unidade de processamento) disponíveis no ambiente, quando utilizado a estratégia de co-escalonamento proposta. É possível observar que a performance varia em torno de 2.000 MFlops no decorrer do tempo, enquanto que na figura 6 (sem co-escalonamento) a performance atinge no máximo 1.000 MFlops em alguns momentos.

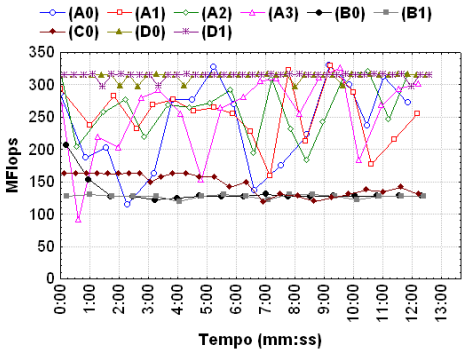

Figura 7. Performance individual dos recursos no ATHA sem Co-Escalonamento.

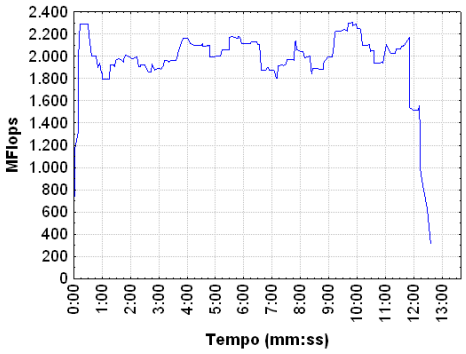

Figura 8. Performance Total do ATHA sem Co-Escalonamento.

5.4 Capacidade de Processamento

Como citado anteriormente, o sistema ATHA também gera relatórios do total de chaves testadas por segundo em cada *host*. Assim, foi possível gerar um gráfico da capacidade de processamento do ambiente nas duas situações testadas anteriormente. As figuras 9 e 10 apresentam a somatória das chaves testadas por segundo no ambiente, sem e com a estratégia de co-escalonamento proposta, respectivamente.

No gráfico da figura 9 pode-se observar que inicialmente a linha alcança 3.000.000 chaves testadas por segundo. Em seguida, existe uma suave redução na quantidade de chaves testadas, estabilizando-se em aproximadamente 2.750.000. Também é possível observar que aos 21 minutos a linha sofre uma queda, onde inferimos que o resultado da aplicação foi alcançado e aos 22 minutos a linha finaliza, indicando que todas as tarefas das máquinas escravas encerraram.

A figura 10 apresenta a somatória das chaves testadas por segundo no ambiente com o sistema ATHA, considerando o uso da estratégia de co-escalonamento proposta. Pode ser visto que o número de chaves testadas por segundo inicialmente ultrapassa 5.000.000, estabilizando-se em seguida um pouco abaixo desse valor. No entanto, esse número ainda é 60% superior ao apresentado na figura 9. Esse fato enfatiza a relevância do Co-Escalonador adaptativo de tarefas para um ambiente de recursos multiprocessados.

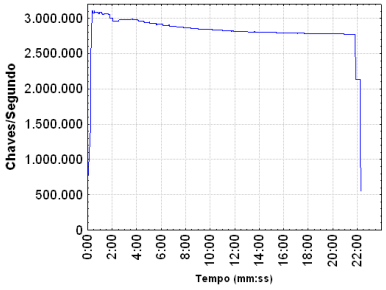

Figura 9. Capacidade Total do ATHA sem Co-Escalonamento.

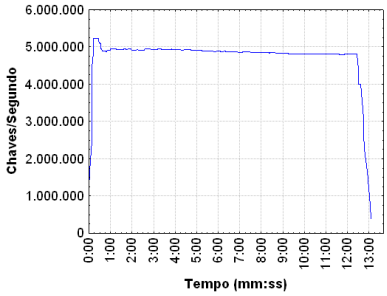

Figura 10. Capacidade Total do ATHA com Co-Escalonamento.

6 Conclusão e Trabalhos Futuros

Essa pesquisa apresenta alguns testes empíricos, da utilização de uma estratégia de co-escalonamento adaptativo em um sistema de processamento oportunista, que será executado em um ambiente de configurações multi-cores e multiprocessadores fracamente acoplados. Para isso, um sistema denominado ATHA, previamente desenvolvido pelo nosso grupo de pesquisa [14], foi adaptado para tratar múltiplas *threads* de uma aplicação complexa, que serão executadas nas máquinas escravas dotadas de múltiplas unidades de processamento.

Resultados experimentais indicam, como esperado, que uma aplicação se não for preparada adequadamente, não utilizará de forma eficiente os recursos multiprocessados disponíveis. Um exemplo disso, foi a submissão do algoritmo RC5 [16], utilizado como aplicação teste, no sistema ATHA processando em uma máquina escrava de quatro processadores. Para esse caso foi verificado que apesar da característica do algoritmo, *CPU-Bound*, o recurso ainda ficou com processadores ociosos. Contudo, quando usada a estratégia de co-escalonamento adaptativo proposta, observou-se que todas as unidades de processamento do recurso escravo, foram utilizadas para executar partes da aplicação.

O mesmo experimento, quando usado um ambiente com quatro máquinas escravas, o número de chaves testadas por segundo aumentou em 60%, quando usamos a estratégia de co-escalonamento proposta. Isso indica que o uso de uma técnica de co-escalonamento adaptativo, que submete o número de tarefas correspondente ao número de unidades de processamento disponível, garante um melhor uso dos recursos, e conseqüentemente, um menor tempo no processamento da aplicação.

Como trabalhos futuros, planejamos realizar os mesmos experimentos do presente artigo, no entanto em um ambiente com aproximadamente 200 máquinas disponíveis na companhia usada como ambiente experimental. Mais uma pesquisa planejada é utilizar outra aplicação que requer um intenso uso de CPU, como por exemplo, seqüenciamento molecular [6], otimização combinatória [7] e pesquisa de números primos [8], para analisar as vantagens e desvantagens no uso da estratégia de co-escalonamento proposta.

Referências

1. Fortes, J.: HCW panel: programming heterogeneous systems - less pain! better-performance! Parallel and Distributed Processing Symposium, IPDPS. (2006)
2. BOINC: BOINC: Berkeley Open Infrastructure for Network Computing. (2008)
3. Nisan, N., London, S., Regev, O., Camiel, N.: Globally distributed computation over the internet-the POPCORN project. Distributed Computing Systems, 1998. Proceedings. 18th International Conference on (1998) pages 592–601
4. Baratloo, A., Karaul, M., Kedem, Z.M., Wyckoff, P.: Charlotte: Metacomputing on the web. (1996)
5. Hosken, A., Dantas, M.A.R.: The ATHA environment: Experience with a user friendly environment for opportunistic computing. 18th International Symposium on High Performance Computing Systems and Applications - HPCS (2004)
6. Strumpen, V.: Parallel molecular sequence analysis on workstations in the internet. (1993)
7. Papadimitriou, C.H., Steiglitz, K.: Combinatorial optimization: algorithms and complexity. Prentice-Hall, Inc., Upper Saddle River, NJ, USA (1982)
8. GIMPS: GIMPS: The Great Internet Mersenne Prime Search. (Maio 2008)
9. Cvetanovic, Z.: The effects of problem partitioning, allocation, and granularity on the performance of multiple-processor systems. IEEE Transactions on Computers **36**(4) (1987) 421–432
10. Pourreza, H., Graham, P.: On the programming impact of multi-core,multi-processor nodes in MPI clusters. HPCS '07: Proceedings of the 21st International Symposium on High Performance Computing Systems and Applications (2007)
11. El-Moursy, A., Garg, R., Albonesi, D., Dwarkadas, S.: Compatible phase co-scheduling on a CMP of multi-threaded processors. IPDPS (2006) page 119
12. Anderson, D.: Local scheduling for volunteer computing. Parallel and Distributed Processing Symposium, IPDPS. IEEE International (2007) pages 1–8
13. Hosken, A.: A parallel processing environment for opportunist on the internet. Master's thesis, UnB - Brazil (2003)
14. LAPESD: Research Laboratory in Distributed Systems, UFSC-CTC-INE. (Maio 2008)
15. Mendonça, R.P., Dantas, M.A.R.: A co-scheduling approach for an opportunistic software environment. Artigo submetido no 14th International European Conference on Parallel and Distributed Computing, Euro-Par, 2008.
16. Rivest, R.L.: The RC5 encryption algorithm. In B. Preneel, editor, Fast Software Encryption, volume 1008 of Lecture Notes in Computer Science, pages 86–96, 1995. Springer Verlag.
17. Unix: Top. (Maio 2008)
18. Intel: HTT: Hyper-Threading Technology. (Maio 2008)