Otimizando o Uso de Recursos Computacionais através da Multiprogramação

Mariana T. Costa[1], Vinicius S. Silva[1], Fábio D. Rossi[2], Marcelo C. Luizelli[1], Arthur F. Lorenzon[1]

[1]Universidade Federal do Pampa – Campus Alegrete (UNIPAMPA)
Alegrete – RS – Brasil

[2]Instituto Federal Farroupilha - Campus Alegrete – Alegrete – RS – Brasil

{mary.costa201025, viniciusufx}@gmail.com

Resumo. *Um dos principais desafios das grandes empresas está em reduzir os custos para manter os serviços de tecnologia em pleno funcionamento. Neste sentido, diferentes abordagens têm sido propostas para otimizar o uso de recursos computacionais. Neste trabalho, mostramos como a distribuição ideal de recursos entre as aplicações pode reduzir o tempo que o computador ficará processando em até 24%, reduzindo o consumo total de energia*[1].

1. Introdução

Nos últimos anos, grandes empresas de tecnologia têm buscado reduzir os gastos relacionados à conta de energia elétrica para manter operando seus *datacenters* e computadores de alto desempenho. Isto porque, conforme dados da Forbes [Kepes 2015], uma empresa pode chegar a gastar até 80% de seu orçamento apenas com a energia necessária para manter seus serviços de tecnologia funcionando. Neste sentido, diferentes abordagens têm sido empregadas para otimizar o uso de recursos computacionais, reduzindo assim, o custo para mantê-los funcionando.

Uma destas abordagens considera a melhor distribuição e alocação de recursos (e.g., núcleos de processamento e memórias *cache*) entre as aplicações que estão sendo executadas. Devido a muitas aplicações paralelas não escalarem conforme o número de *threads* aumenta, o melhor desempenho para tais aplicações pode ser obtido com um menor número de *threads* [Lorenzon et al. 2019]. Neste sentido, enquanto uma aplicação está executando com um pequeno número de *threads*, pode-se utilizar os recursos que estão em *idle* para executar outra aplicação, com o objetivo de reduzir (*i*) o tempo total que o computador estará processando e (*ii*) os gastos relacionados ao consumo de energia.

Neste sentido, este artigo objetiva estudar diferentes maneiras de alocação de aplicações paralelas com relação ao número de *threads* para reduzir o tempo total para executar um conjunto de aplicações. Nove aplicações foram executadas em um processador com 32 núcleos. Nós mostramos que, através da alocação e distribuição ideal do número de *threads* e aplicações que rodam de maneira concorrente, o tempo de execução de todo o conjunto de aplicações pode ser reduzido em até 24%.

[1]Esse artigo foi parcialmente financiado por FAPERGS nº 19/2551-0001224-1 e 19/2551-0001689-1, CNPq PIBIC e FAPERGS PROBIC

2. Trabalhos Relacionados

Sudarsan e Ribbens [Sudarsan and Ribbens 2016] apresentaram um *framework* que torna dinâmico o dimensionamento das aplicações. Este dimensionamento é baseado em cenários e estratégias onde prioriza-se o menor tempo de execução utilizando todos os recursos disponíveis e em uma outra situação prioriza-se as execuções de alta ou baixa prioridade utilizando os recursos dentro de um limite pré-determinado. Já Coskun et al. [Coskun et al. 2009] apresentam uma solução para gerenciamento do consumo de energia. Os autores criaram um escalonador que permite o gerenciamento da temperatura e o ajuste de frequência do *core* procurando reduzir a temperatura geral do chip com custo de desempenho mínimo ao reduzir proativamente as aplicações menos impactadas.

Por fim, Jorge González-Domínguez e Touriño [Jorge González-Domínguez and Touriño 2012] usam parâmetros do sistema obtidos pelo conjunto de *benchmarks* que fazem parte da *Servet* para selecionar a melhor opção para maximizar o desempenho de aplicações. Diferentemente dos trabalhos citados acima, nosso trabalho faz um estudo sobre diferentes maneiras de alocação de aplicações e *threads*, além de uma busca exaustiva da melhor configuração.

3. Metodologia

Foram utilizados nove *benchmarks* do *NAS parallel benchmark*. O NAS consiste de um conjunto de pequenos programas que têm o intuito de avaliar o desempenho de supercomputadores paralelos. Os *benchmarks* são derivados de dinâmica computacional de fluidos, que consistem de cinco *kernels*: IS, EP, CG, MG e FT. Adicionalmente, ele conta com quatro pseudo-aplicações e uma aplicação para computação não estruturada, sendo eles: BT, SP, LU e UA. Cada aplicação foi executada com a classe "C" de tamanhos e parâmetros predefinidos pelo NPB. Os experimentos foram realizados em uma máquina composta por dois processadores Intel Xeon E5-2650. Cada processador possuí 8 núcleos físicos com suporte a tecnologia *Hyperthreading*, onde cada núcleo físico individual possui duas *threads* totalizando assim 32 *threads*. Cada núcleo físico é equipado com uma cache L1 de dados e instruções, de 32KB cada uma e uma cache L2 de 256KB. Os núcleos de cada processador compartilham uma cache L3 de 20MB. O sistema conta com um total de 64GB de memória principal.

Os seguintes casos de teste foram considerados: **Configuração I**: Nesta configuração, cada aplicação foi executada com o número máximo de *threads*, isto é, 32. Para isto, cada aplicação foi executada uma após a outra, tendo todos os recursos de *hardware* disponíveis para ela. **Configuração II**: Nesta configuração, as aplicações foram executadas em pares, cada uma com 16 *threads*. Para tanto, foi configurado um *script* para executar todas as combinações possíveis de aplicações. Isto deu um total de 2.880 execuções. **Configuração Ideal**: Neste caso, todas as aplicações foram executadas com todas as combinações possíveis de números de *threads*. Isto é, foi realizada uma busca exaustiva pela melhor configuração.

O Sistema Operacional em uso foi o Linux Ubuntu, com *kernel* v.4.15.0, com GCC v.9.2.0 que é um série de compiladores de linguagens de programação e OpenMP 5.0 que é uma interface de programação de aplicações paralelas. Cada aplicação foi compilada com a *flag* de otimização *-O3*. Durante os experimentos, o DVFS foi configurado para o *governor ondemand*, em que a frequência dos núcleos é ajustada de acordo com a carga

de trabalho. Cada configuração foi executada ao menos cinco vezes. Disto, obteve-se a média dos tempos de execuções, onde o desvio padrão foi inferior a 1%.

4. Resultados

A Figura 1 apresenta os resultados para os diferentes casos de testes (descritos na seção anterior). Inicialmente, a Figura 1a mostra o tempo de execução para a Configuração I, onde todas as aplicações foram executadas com 32 *threads*, uma após a outra. Nela, podemos observar que o tempo total de execução para todas as aplicações foi de *558.6* segundos. Estes dados mostram o comportamento das aplicações quando executadas da maneira padrão em um sistema *multicore*. Portanto, estes valores serão utilizados como *baseline* na comparação com os demais resultados.

Podemos observar nas Figuras 1b e 1c, que ao utilizarmos uma divisão igualitária de recursos (16 *threads* para cada aplicação), o tempo total de execução de todo o conjunto de aplicações é reduzido consideravelmente. No melhor dos casos, mostrado na Figura 1c, o tempo de execução é reduzido em 12%. Estes ganhos ocorrem pela seguinte razão: ao executar de maneira concorrente duas aplicações que utilizam diferentes recursos do computador, o impacto do comportamento de uma aplicação na outra é reduzido, e assim, é possível obter melhorias de desempenho. Um exemplo é quando a aplicação BT está sendo executada com 16 *threads* ao mesmo tempo que as aplicações CG, EP, FT e uma parcela da IS. Enquanto que a BT pode ser considerada *Memory-Bound*, as demais aplicações são consideradas *CPU-Bound*.

Threads	32	BT	CG	EP	FT	IS	LU	MG	SP	UA
Tempo (s)		104.5	132.7	159.4	182.5	185.0	249.3	265.0	469.5	558.6

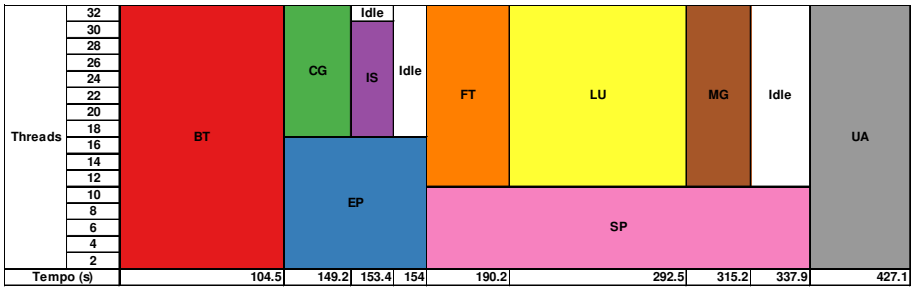
(a) Configuração I. Cada Aplicação executando com 32 threads

Threads	16	UA	LU	CG	IS	EP	MG	
	16	BT	SP			FT	Idle	
Tempo (s)		153.9	169.1	377.7	443.5	449.4	457.3	499.1	508.0	525.6

(b) Configuração II, versão a

Threads	16	CG	EP	FT	IS	MG	SP	
	16	BT			LU		UA	Idle
Tempo (s)		64.0	107.7	152.7	153.9	157.6	184.5	233.1	405.2	493.8

(c) Configuração II, versão b



(d) Configuração Ideal

Figura 1. Tempo de execução das aplicações sob diferentes configurações.

A Figura 1d mostra o resultado para a execução com a configuração mais adequada, que foi obtido através de uma execução exaustiva de todas as possíveis configurações. Podemos observar que apenas duas aplicações tiveram uma contribuição positiva para o resultado global ao executar com o número máximo de *threads*: BT e UA. Por outro lado, as demais aplicações foram melhor executadas com diferentes números de *threads*. Por exemplo, enquanto SP está sendo executada com apenas 8 *threads* (o qual possuí um resultado melhor do que com a execução com 16 e 32 *threads*), é possível utilizar os 22 *cores* que estão em *idle* para executar outras três aplicações (FT, LU, MG), sem perda no desempenho final. Com esta utilização eficiente dos recursos computacionais, o tempo para executar todo o conjunto de aplicações foi reduzido para 427.1 segundos, representando uma melhoria de 24% no desempenho. Os resultados mostram que, quando as aplicações possuem diferentes características e comportamentos com relação ao uso de recursos computacionais, é possível usar da execução concorrente de aplicações paralelas para melhorar o uso dos recursos computacionais e assim reduzir o tempo em que o computador estava processando, e assim, consumindo mais energia.

5. Conclusão

Este trabalho realizou um estudo da execução concorrente de diferentes aplicações paralelas. Através da execução de nove aplicações do conjunto de benchmarks paralelo NAS, nós mostramos que é possível fazer um melhor uso dos recursos computacionais através da divisão dos recursos de hardware entre as diferentes aplicações. Ao executar com uma configuração ideal, mostramos que o tempo de execução pode ser melhorado em até 24% em comparação com a maneira padrão que aplicações paralelas são executadas. Como trabalhos futuros pretendemos explorar o consumo de energia dos sistemas computacionais e o impacto no custo para locação de diferentes sistemas de *cloud computing*.

Referências

Coskun, A., Strong, R., Tullsen, D., and Rosing, T. (2009). Evaluating the impact of job scheduling and power management on processor lifetime for chip multiprocessors. volume 37, pages 169–180.

Jorge González-Domínguez, Guillermo L. Taboada, B. B. F. M. J. M. and Touriño, J. (2012). Automatic mapping of parallel applications on multicore architectures using the servet benchmark suite. *Computers and Eletrical Engineering*, 38(número):258–269.

Kepes, B. (2015). 30% of servers are sitting ”comatose”according to research. Forbes.

Lorenzon, A. F., de Oliveira, C. C., Souza, J. D., and Beck, A. C. S. (2019). Aurora: Seamless optimization of openmp applications. *IEEE Transactions on Parallel and Distributed Systems*, 30(5):1007–1021.

Sudarsan, R. and Ribbens, C. J. (2016). Combining performance and priority for scheduling resizable parallel applications. *Parrallel and Distribuited Computing*, 87(numero):55–66.