Análise da Execução Concorrente de Aplicações Paralelas em Arquiteturas *Multicore*[*]

Vinicius da Silva[1], Thiarles S. Medeiros[1], Hiago Rocha[3], Marcelo C. Luizelli[1], Fábio Rossi[2], Antonio Carlos S. Beck[3] e Arthur F. Lorenzon[1]

[1]Universidade Federal do Pampa – Alegrete – RS – Brasil

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

[3]Universidade Federal do Rio Grande do Sul – Porto Alegre – RS – Brasil

viniciussouza.aluno@unipampa.edu.br

Abstract. *Thread-level parallelism (TLP) has been widely used to optimize the use of computational resources (e.g., cache memories and functional units from CPU) of high-performance systems. However, as many applications do not scale as the number of threads increase, resources will be wasted when the application is executed with the ideal number of threads. Hence, the concurrent execution of parallel applications can be used to provide a better use of computational resources without impacting the performance and energy consumption of the system. Given that, we have carried out an extensive design space exploration with the execution of twenty-two parallel applications with different characteristics of shared memory accesses, IPC (instructions per cycle) and degree of TLP exploitation in two multicore architectures (Intel and AMD). We show which kind of applications can be concurrently executed and provide better use of computational resources. In the most significant case, the ideal combination of parallel applications running concurrently can optimize the trade-off between performance and energy consumption by up to 49% when compared to the individual execution of each application.*

Resumo. *O paralelismo no nível de threads (TLP) tem sido amplamente utilizado para otimizar o uso de recursos computacionais (e.g., memórias cache e unidades funcionais da CPU) de sistemas de alto desempenho. No entanto, como algumas aplicações não escalam com o número de threads, recursos ficarão ociosos quando a aplicação é executada com o número ideal de threads. Neste sentido, a execução concorrente de aplicações paralelas pode ser utilizada para prover uma melhor utilização dos recursos computacionais sem impactar no desempenho e consumo de energia do sistema como um todo. Dito isto, nós realizamos uma extensa exploração de espaço e projeto com a execução de vinte e duas aplicações paralelas com diferentes características de acesso à memória compartilhada, IPC (instruções por ciclo) e grau de exploração do TLP em duas arquiteturas multicore (Intel e AMD). Nós mostramos quais tipos de aplicações podem ser executadas de maneira concorrente e ainda proporcionar melhor utilização dos recursos computacionais. No caso mais significativo, a combinação ideal de aplicações paralelas executando de maneira concorrente pode otimizar o custo-benefício entre desempenho e consumo de energia em até 49% quando comparado à execução individual de cada aplicação.*

[*]Este trabalho foi parcialmente financiado pela FAPERGS nos projetos 19/2551-0001224-1 e 19/2551-0001689-1

1. Introdução

A exploração do paralelismo no nível de threads (TLP – *thread-level parallelism*) é amplamente utilizada para melhorar o desempenho de aplicações paralelas de diferentes domínios que executam em sistemas computacionais de alto desempenho (HPC – *high-perfomance computing*). No entanto, o consumo de energia tem se tornando uma importante preocupação: enquanto é esperado que o consumo de potência de sistemas HPC aumente significativamente [O'brien et al. 2017], processadores de propósito geral estão tendo seu desempenho limitado pelo TDP (*thermal design power)*. Portanto, ao executar aplicações paralelas, o objetivo não é apenas melhorar o desempenho, mas fazer isto com o mínimo impacto no consumo de energia.

A maneira padrão de executar aplicações paralelas em sistemas multicore é definir o grau de paralelismo igual ao número de núcleos disponíveis na arquitetura alvo. Muito embora esta estratégia seja capaz de proporcionar ganhos de desempenho na grande maioria dos casos, executar todas as aplicações com o número máximo de *threads* nem sempre entregará o melhor custo-benefício entre o consumo de energia e tempo de execução, representado pelo *energy-delay product* (EDP) [Gonzalez and Horowitz 1996, Lorenzon et al. 2017, Lorenzon et al. 2015]. Esta falta de escalabilidade está relacionada a diferentes fatores de *hardware* e *software*, como por exemplo: saturação de unidades funcionais e barramento de comunicação; sincronização de dados; e acessos concorrentes à memória compartilhada [Raasch and Reinhardt 2003, Suleman et al. 2008, Lorenzon et al. 2019].

Portanto, quando uma aplicação paralela que tem sua escalabilidade limitada é executada com o número ideal de threads, que é menor que o número total de núcleos disponíveis no sistema, o mesmo estará sendo sub-utilizado [Lorenzon and Beck 2019, dos Santos Marques et al. 2017]. Isto é, os recursos (i.e., núcleos e memórias *cache*) que não estarão sendo utilizados pela aplicação ficarão ociosos. Neste sentido, a execução de múltiplas aplicações paralelas de forma concorrente vem sendo empregada para otimizar o uso de recursos computacionais . Nela, as aplicações paralelas são executadas ao mesmo tempo com o objetivo de reduzir o tempo total ou consumo de energia da execução de um conjunto de aplicações quando comparada a execução individual de cada aplicação. Logo, enquanto uma aplicação está sendo executada com um pequeno número de *threads*, pode-se utilizar os recursos que estão ociosos para executar outra aplicação. No entanto, embora diferentes trabalhos tenham realizado a otimização do número de *threads*, existe um espaço para realizar uma análise mais completa considerando diferentes métricas de *hardware* quando duas aplicações paralelas podem ser executadas concorrentemente para otimizar os recursos computacionais, como nós fazemos neste artigo.

Em resumo, as contribuições deste artigo são: (*i*) análise da escalabilidade de aplicações paralelas; (*ii*) análise da execução de aplicações paralelas de maneira concorrente em ambientes HPC, considerando diferentes métricas de *hardware*, como por exemplo, acessos à memória compartilhada, instruções por ciclo (IPC) e grau de exploração do TLP; e (*iii*) diretivas para usuários e desenvolvedores de *software* otimizar o uso dos recursos computacionais de acordo com as características de cada aplicação paralela. Através da execução de vinte e duas aplicações paralelas amplamente conhecidas de diferentes domínios em duas arquiteturas multicore (Intel Xeon com 40 núcleos e AMD Ryzen com 16 núcleos), nós mostramos que:

- Quando aplicações com médio grau de TLP e alto número de IPC (instruções por ciclo) são executadas concorrentemente com aplicações que possuem alta taxa de *misses* no último nível da memória cache, o EDP é otimizado em até 39% quando comparado a execução de cada aplicação individualmente;
- Aplicações com baixo grau de TLP e número de acessos à memória apresentam a maior possibilidade de explorar a execução concorrente com as demais aplicações.

Neste caso, a otimização do EDP pode chegar a até 49%;

- Devido a saturação de memória e CPU, aplicações com alto grau de TLP e acessos à memória compartilhada não são indicadas para a execução concorrente com qualquer outro tipo de aplicação. Neste caso,executar aplicações concorrentemente pode degradar o EDP em até 6.8.

O restante deste artigo está organizado como segue. Os trabalhos relacionados são apresentados na Seção 2 onde também discutimos nossas contribuições. A metodologia utilizada é detalhada na Seção 3. Os resultados são discutidos na Seção 4 enquanto as conclusões são destacadas na Seção 5.

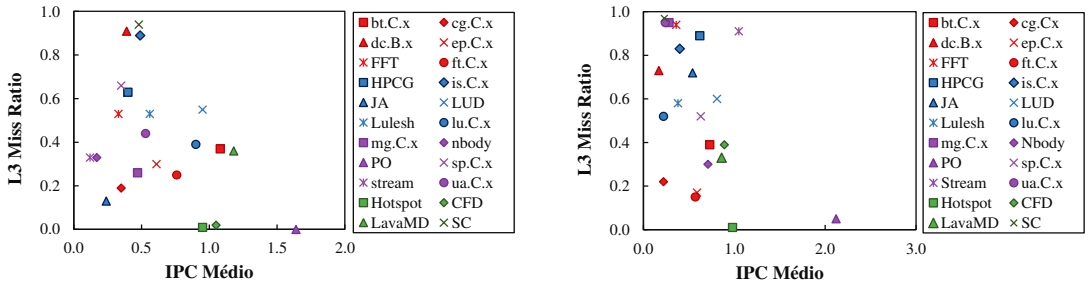
2. Trabalhos Relacionados

Coskun et al. [Coskun et al. 2009] apresentam uma solução para gerenciamento do consumo de energia quando aplicações paralelas estão sendo executadas. Os autores desenvolveram um escalonador que permite o gerenciamento da temperatura e o ajuste de frequência do *core* procurando reduzir a temperatura geral do chip com custo mínimo de desempenho. Jorge González-Domínguez e Touriño [Jorge González-Domínguez and Touriño 2012] apresentam um algoritmo que utiliza parâmetros do sistema para gerar automaticamente uma política eficiente de mapeamento de aplicações em sistemas multicore com o objetivo de maximizar o desempenho.

Timothy Mattausch Creech [Creech et al. 2013] apresenta o sistema *Scheduling and Allocation with Feedback* (*SCAF*). Este sistema é uma solução em tempo de execução onde o objetivo é particionar os contextos de *hardware* disponíveis em processos. O SCAF procura resolver problemas de desempenho relacionados à execução de várias aplicações *multithread* em um sistema *multicore* de memória compartilhada. Ashkan Tousimojarad e Wim Vanderbauwhede [Tousimojarad and Vanderbauwhede 2014] apresentam uma estratégia de mapeamento para fornecer balanceamento de carga de maneira justa entre as aplicações em um ambiente de multiprogramação *multithread*. Harris, et al. [Harris et al. 2014] apresentam Callisto, um sistema de *runtime* para aplicações OpenMP e Domino (um sistema paralelo no nível de tarefas) que atua como uma camada de gerenciamento de recursos.

Georgios Varisteas [Varisteas 2015] propõe uma solução com o objetivo de alcançar uma otimização da taxa de transferência com degradação uniforme do desempenho dos aplicativos que estão executando concorrentemente. Sudarsan e Ribbens [Sudarsan and Ribbens 2016] apresentam 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.

Diferentemente dos trabalhos citados acima, este artigo apresenta uma extensa exploração de espaço e projeto da execução concorrente de aplicações paralelas com diferentes características relacionadas a acessos à memória compartilhada, IPC e grau de exploração de TLP. Além de avaliar a escalabilidade de cada aplicação paralela, nós mostramos quais tipos de aplicações podem ser executadas de maneira concorrente com o objetivo de otimizar o uso dos recursos computacionais. Por fim, nós também fornecemos diretivas para os usuários e desenvolvedores de aplicações paralelas com o objetivo de auxiliar na otimização dos recursos computacionais. Portanto, este artigo pode servir como base para futuros estudos realizados na área de otimização de recursos através da programação paralela.



(a) Intel Xeon 40-core (b) AMD Ryzen 16-core

Figura 1. Características de cada aplicação

3. Metodologia

Nesta Seção, apresentamos as aplicações utilizadas nos experimentos bem como as características de cada aplicação escolhida. Também descrevemos as arquiteturas *multicore* utilizadas e como a avaliação foi realizada.

3.1. Benchmarks

Vinte e duas aplicações paralelas de diferentes suítes de benchmarks foram utilizadas: **Dez kernels e pseudo-aplicações do NAS Parallel Benchmark** [Bailey et al. 1991]: *BT*, *CG*, *DC*, *EP*, *FT*, *IS*, *LU*, *MG*, *SP* e *UA*. **Cinco aplicações da suíte de benchmarks do Rodinia** [Che et al. 2009]: *CFD*, *Hotspot*, **LavaMD**, *LUD* e *Streamcluster*. **Sete aplicações de diferentes domínios**: *FFT*, *HPCCG*, *Método de Jacobi*, *LULESH*, *n-body*, *Poisson* e *Stream*. As aplicações escolhidas possuem diferentes comportamentos com relação ao acesso a memória e utilização do processador. Neste sentido, nós caracterizamos as aplicações de acordo com a taxa de *misses* na cache L3 (*L3 miss ratio*) e o IPC médio, conforme mostrado na Figura 1. Esta caracterização das aplicações foi realizada com o objetivo de auxiliar na análise de quais aplicações podem ser executadas concorrentemente considerando as métricas mostradas na Figura 1. Portanto, as aplicações são representativas para esta análise: aplicações com baixo/médio/alto IPC e aplicações com baixo/médio/alto número de acessos a memória principal, dado pela taxa de *misses* na cache L3. Os dados para esta classificação foram retirados diretamente dos contadores de *hardware* através do Intel *Performance Counter Monitor* e do *AMDuProf*. Para tanto, cada aplicação foi executada com o número de *threads* igual ao número de núcleos disponíveis em cada arquitetura.

3.2. Ambiente de Execução

Os experimentos foram realizados em duas arquiteturas multicores, conforme mostrado na Tabela 1: AMD Ryzen 7 2700 e Intel Xeon E5-2650 v3[1]. Nós usamos o Sistema Operacional Linux Ubuntu com kernel v. 4.19.0. A frequência de cada CPU foi configurada para ajustar de acordo com a carga de trabalho aplicação, através do *governor dynamic voltage and frequency scaling* (DVFS) *ondemand*, que é o *governor* padrão usado na maioria das versões Linux. Cada aplicação foi compilada com GCC/G++ 9.2, usando a flag de otimização $-O3$. A alocação e afinidade de cada *thread* foi configurada através da variável de ambiente do OpenMP: `OMP_PROC_BIND=CLOSE` e `OMP_PLACES=SOCKET` no caso do Intel Xeon. Isto faz com que as *threads* de uma mesma aplicação sejam alocadas próximas uma das outras.

[1]Alguns experimentos deste trabalho utilizaram os recursos da infraestrutura PCAD, http://gppd-hpc.inf.ufrgs.br, no INF/UFRGS

Tabela 1. Arquiteturas Multicore

	AMD Ryzen 7 2700	**Intel Xeon E5-2650 v3**
Microarquitetura	Zen+	Haswell
Número de núcleos	8	20
Número de threads	16	40
Cache L1 I/D	64/32kB	32/32Kb
Cache L2 (total)	4MB	5.1MB
Cache L3 (total)	16MB	50MB
Memória RAM	16GB	128GB

Na avaliação das estratégias, nós consideramos a métrica EDP. Esta métrica é utilizada para estudar o custo-benefício entre o total de energia consumida e o tempo de execução de uma aplicação. Sua fórmula consiste da multiplicação da energia pelo tempo de execução. Ela vem sendo bastante utilizada pois possibilita analisar, em um único valor, a relação entre o consumo de energia e o desempenho. Por exemplo, considerando dois cenários: (*i*) uma aplicação é executada em 100 segundos e consumiu 10 joules de energia; (*ii*) uma aplicação é executada em 50 segundos e consumiu 40 joules de energia; o cenário *i* teria EDP de 1000, enquanto que o cenário *ii* teria EDP de 2000. Isso mostra que, embora o cenário *i* tenha sido duas vezes mais lento, ele possui a melhor relação de consumo de energia e desempenho.

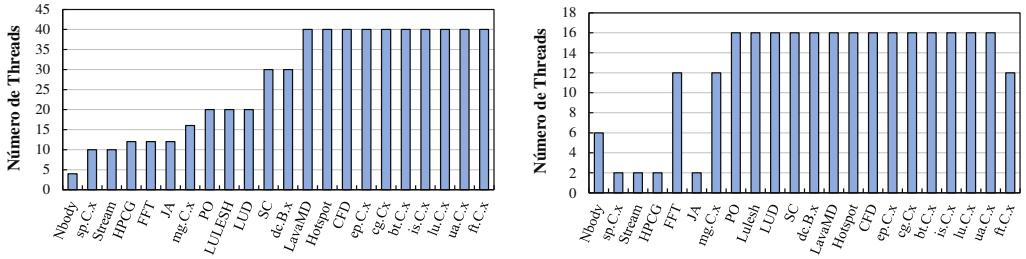
O EDP da execução de cada aplicação foi obtido pela multiplicação do tempo de execução (em segundos) pelo consumo de energia (em Joules). O tempo foi obtido com a função do OpenMP `omp_get_wtime`. Por outro lado, o consumo de energia foi obtido diretamente dos contadores de hardware presentes nos processadores modernos. No caso do processador Intel Xeon, a biblioteca *Running Average Power Limit* (RAPL) foi usada [Hähnel et al. 2012], enquanto que a biblioteca *Application Power Management* foi usada para o processador AMD Ryzen [Hackenberg et al. 2013]. Os resultados apresentados na Sessão 4 são uma média de dez execuções com um desvio padrão menor que 0.5%.

4. Resultados Experimentais

Nesta seção, nós apresentamos e discutimos os resultados obtidos através dos experimentos realizados. Na Seção 4.1, nós mostramos uma análise do comportamento de escalabilidade de cada aplicação conforme o número de *threads* executando a aplicação aumenta. Já na Seção 4.2, nós discutimos os resultados obtidos através dos experimentos da execução concorrente de aplicações paralelas. Por fim, apresentamos as diretivas para os usuários e desenvolvedores de aplicações paralelas na Seção 4.3.

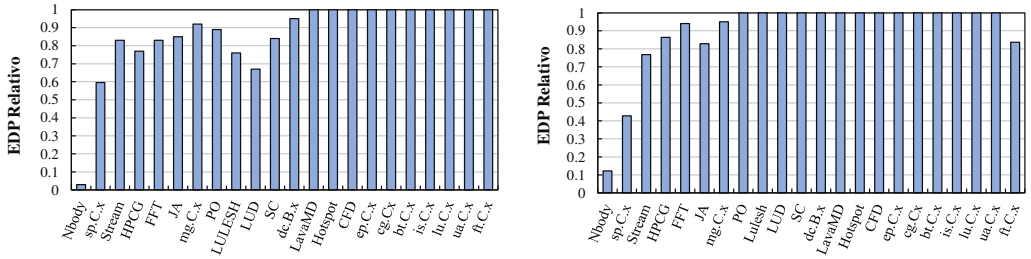
4.1. Análise da Escalabilidade

Para analisar a escalabilidade das aplicações paralelas, isto é, encontrar qual o melhor número de *threads* para executar cada aplicação de maneira individual, nós executamos cada aplicação com diferentes números de *threads*: de 1 até o número de *threads* igual ao número de núcleos. A Figura 2 mostra o número de *threads* que apresentou o melhor resultado de EDP para cada aplicação e arquitetura multicore. Adicionalmente, a Figura 3 mostra o EDP da execução de cada aplicação com o melhor número de *threads* normalizado pela execução com o número de *threads* igual ao número de núcleos, que corresponde a maneira padrão que aplicações paralelas são executadas. Conforme pode-se observar, uma parte significativa das aplicações apresenta melhor EDP quando é executada com um número de *threads* menor que o número de núcleos. Diferentes são as razões para a falta de escalabilidade de tais aplicações: sincronização de dados, acessos concorrentes à memória compartilhada, saturação do barramento *off-chip* e saturação das unidades funcionais [Raasch and Reinhardt 2003, Suleman et al. 2008, Lorenzon et al. 2019].



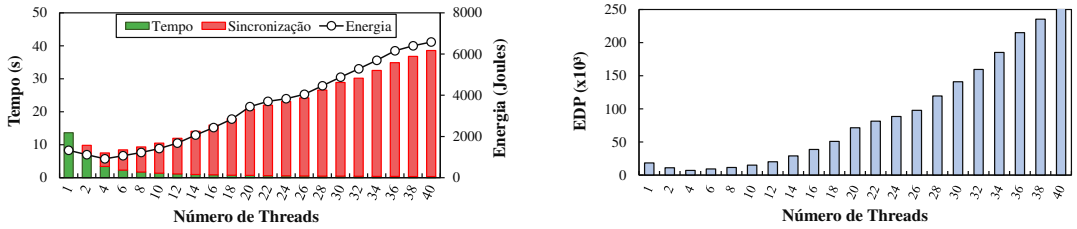
(a) Intel Xeon 40-core (b) AMD Ryzen 16-core

Figura 2. Número de *threads* que apresenta o melhor EDP para cada aplicação



(a) Intel Xeon 40-core (b) AMD Ryzen 16-core

Figura 3. EDP da execução com cada aplicação com o melhor número de *threads* normalizado pela execução padrão de aplicações paralelas.



(a) Tempo de Execução e Consumo de Energia (b) Energy-delay product (EDP)

Figura 4. Comportamento da aplicação *Nbody* no processador Intel Xeon 40-core

Para melhor entender este cenário, vamos considerar o comportamento da aplicação *n-body*, que apresenta nível de escalabilidade baixo em ambos os processadores. O *n-body* é uma aplicação que tem sua escalabilidade limitada pela sincronização de dados. Neste sentido, quanto maior o número de *threads*, maior o tempo de sincronização, o que eventualmente pode aumentar o tempo de execução e consumo de energia, piorando o EDP resultante. Nós mostramos este comportamento na Figura 4 para a execução da aplicação *N-body* no processador Intel Xeon. Ela mostra, na Figura 4.a para cada número de *threads*, o tempo de execução (eixo *y* primário) dividido em duas partes: o tempo para executar a região paralela e para sincronizar (esta aplicação possui apenas uma região paralela); e o consumo de energia (eixo *y* secundário). Conforme pode ser observado, o tempo de execução e consumo de energia reduz quando o número de *threads* aumenta de 2 para 4 *threads*. No entanto, a partir deste ponto, a sincronização leva mais tempo para executar que a própria execução da região paralela. Portanto, como mostrado na Figura 4.b, o EDP piora e não é possível atingir melhores resultados com o aumento no número de *threads*.

Tabela 2. Comportamento de cada aplicação executando com o número ideal de *threads* em cada processador

	Intel Xeon						***AMD Ryzen 7 2700***					
	IPC	***Número de acessos ($x10^6$)***			***Taxa de misses***		***IPC***	***Número de acessos ($x10^6$)***			***Taxa de misses***	
		L2	***L3***	***RAM***	***L2***	***L3***		***L2***	***L3***	***RAM***	***L2***	***L3***
nbody	0.77	113	104	72	0.92	0.69	0.71	307	103	31	0.33	0.3
sp.C.x	1.51	33768	9863	4636	0.29	0.47	0.63	2606955	771116	403169	0.30	0.52
stream	0.44	122891	102500	41000	0.83	0.4	1.05	81275	26008	23667	0.32	0.91
HPCCG	1.23	5602	2713	1248	0.48	0.46	0.62	47349	12081	10755	0.26	0.89
fft	0.76	2547	1402	505	0.55	0.36	0.36	211034	76354	71946	0.36	0.94
já	0.73	15873	9826	2555	0.63	0.26	0.54	45866	15022	10756	0.33	0.72
mg.C.x	1.13	47414	13362	3207	0.28	0.24	0.28	22597	4335	4111	0.19	0.95
PO	3.31	16704	39	39	0.66	0.01	2.12	41622	16432	888	0.39	0.05
LULESH	1.08	112500	27514	5778	0.24	0.21	0.38	3758902	895918	519038	0.24	0.58
LUD	2.41	9900	1478	680	0.15	0.46	0.81	20504	4716	2829	0.23	0.60
SC	0.54	2463	2368	2226	0.96	0.94	0.23	9727	5641	5485	0.58	0.97
dc.B.x	0.45	28571	25531	24000	0.91	0.94	0.17	17954	11675	8484	0.65	0.73
LavaMD	1.18	1721	241	87	0.14	0.36	0.86	2060	576	190	0.28	0.33
hotspot	0.95	216666	77499	775	0.24	0.01	0.98	2602735	494413	6742	0.19	0.01
cfd	1.05	11948	4950	99	0.39	0.02	0.89	8751	2101	813	0.24	0.39
ep.C.x	0.61	210	36	11	0.19	0.3	0.59	2020	850	148	0.42	0.17
cg.Cx	0.35	81395	71789	13640	0.86	0.19	0.22	60530	36002	8015	0.59	0.22
bt.C.x	1.08	43311	19656	7273	0.45	0.37	0.73	419140	123924	47931	0.30	0.39
is.C.x	0.49	504	545	202	0.2	0.89	0.40	250	65	54	0.26	0.83
lu.C.x	0.90	24133	11581	4285	0.33	0.39	0.22	533986	199656	104426	0.37	0.52
ua.C.x	0.53	22231	11864	4390	0.68	0.44	0.24	156982	74331	70919	0.47	0.95
ft.C.x	0.76	16015	4891	1810	0.5	0.25	0.57	72970	28790	4346	0.39	0.15

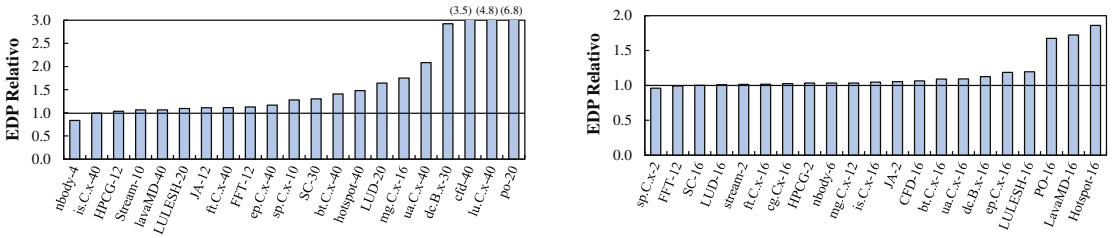
Desta maneira, para aplicações que não escalam com o número de *threads*, recursos computacionais (i.e., núcleos e memória cache) poderão ficar ociosos quando esta é executada de maneira ótima. Por exemplo, se a aplicação *sp.C.x* for executada no processador Intel Xeon, apenas 10 núcleos de processamento estarão sendo utilizados. Este comportamento oferece a oportunidade de explorar a execução de mais de uma aplicação paralela ao mesmo tempo, cada uma com recursos computacionais distintos, com o objetivo de otimizar a utilização dos recursos computacionais e reduzir o EDP total da execução de tais aplicações. Neste sentido, apresentamos na próxima Seção uma análise da execução de múltiplas aplicações paralelas de maneira concorrente.

4.2. Análise da Execução Concorrente de Aplicações Paralelas

Para analisar o comportamento da execução concorrente de aplicações paralelas, primeiramente nós avaliamos as características de cada aplicação sendo executada com o número de *threads* ideal. A Tabela 2 apresenta as seguintes informações de cada aplicação: IPC médio; número de acessos às memórias cache L2 e L3 e memória principal (RAM); e taxa de *misses* nas memórias cache L2 e L3. Por exemplo, para o processador Intel Xeon, a aplicação *PO* apresentou o maior IPC (3.31) enquanto que a aplicação *cg.C.x* apresentou o menor valor de IPC (0.35). Neste sentido, com base nas informações apresentadas na Figura 2 e Tabela 2, nós definimos seis cenários de testes de acordo com o grau de exploração do paralelismo, IPC e número de acessos à memória compartilhada. Para os experimentos de cada cenário, cada aplicação é executada com o número de *threads* que obteve o melhor EDP quando executada de maneira individual (Figura 2).

4.2.1. Cenário I: Alto TLP e número de acessos à memória compartilhada

Com este cenário, objetivamos encontrar quais aplicações podem se beneficiar da execução concorrente com aplicações que possuem alto grau de exploração de parale-



(a) Xeon: *cg.C.x-40*

(b) AMD: *lu.C.x-16*

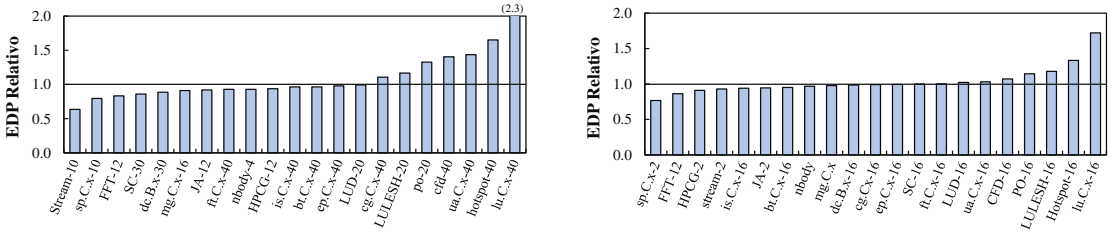
Figura 5. Resultados de EDP do Cenário I.

lismo e número de acessos à memória compartilhada (e.g., cache L3 e memória RAM). Para tanto, selecionamos as aplicações *cg.C.x* e *lu.C.x* para a execução concorrente com as demais aplicações nos processadores Intel e AMD, respectivamente. A Figura 5 mostra os resultados de EDP da execução concorrente da aplicação alvo (*cg.C.x* no Intel e *lu.C.x* no AMD) com cada aplicação. O EDP é normalizado pela execução das duas aplicações de maneira separada (representado pela linha preta). Portanto, valores menor que 1.0 significa que a execução concorrente apresentou melhores resultados. Esta mesma organização de gráfico é utilizada nos demais experimentos desta Seção.

Para este cenário, podemos observar que, na grande maioria dos casos, a execução concorrente de aplicações paralelas não possibilitou melhores resultados, levando a um aumento significativo no EDP total. Este comportamento ocorre pois, além das aplicações alvo *cg.C.x* e *lu.C.x* (no Intel e AMD, respectivamente) apresentarem alto grau de exploração de paralelismo (ambas atingem o melhor resultado de EDP com o número máximo de *threads* disponíveis em cada arquitetura), elas também possuem bastante acessos à memória compartilhada. Com isto, a CPU e memória (utilizada para troca de dados) estão sendo completamente utilizados pelas aplicações alvo. O resultado deste comportamento é a falta de recursos (e.g., CPU e memória) que podem ser utilizados por outras aplicações, ao mesmo tempo, o que afeta negativamente o EDP da execução concorrente das aplicações na grande maioria dos casos. Em casos restritos da execução concorrente com aplicações que possuem escalabilidade baixa (*nbody* no Intel e *sp.C.x* no AMD), a execução concorrente possibilitou pequenos ganhos de EDP.

4.2.2. Cenário II: Alto TLP com baixo número de acessos à memória compartilhada

Este cenário objetiva identificar quais aplicações paralelas se beneficiam da execução concorrente com aplicações que apresentam o melhor resultado de EDP com o maior número possível de *threads* e que possui um número muito baixo de acessos à memória compartilhada. Dentre as aplicações com tal características, selecionamos a aplicação *LavaMD* para analisar o comportamento em ambos os processadores. Conforme pode ser observado na Figura 6, aplicações com alto número de acessos à memória compartilhada são beneficiadas da execução concorrente (e.g., *stream*, *FFT*, *HPCG*, *sp.C.x*). No caso mais significativo, executando a aplicação *LavaMD* (40 *threads*) de maneira concorrente com a aplicação *Stream* (10 *threads*) no processador Intel Xeon, foi capaz de reduzir o EDP em 37% quando comparado a execução de ambas as aplicações de maneira individual. Este comportamento mostra que a grande maioria das aplicações com alta taxa de acessos à memória compartilhada, independentemente do nível de TLP, podem ser executadas de maneira concorrente com aplicações CPU-Intensiva com o objetivo de melhor aproveitar os recursos computacionais. Adicionalmente, quanto menor o nível de TLP associado à essas aplicações, maiores são as melhorias no EDP.



(a) Xeon: *LavaMD-40* (b) AMD: *LavaMD-16*

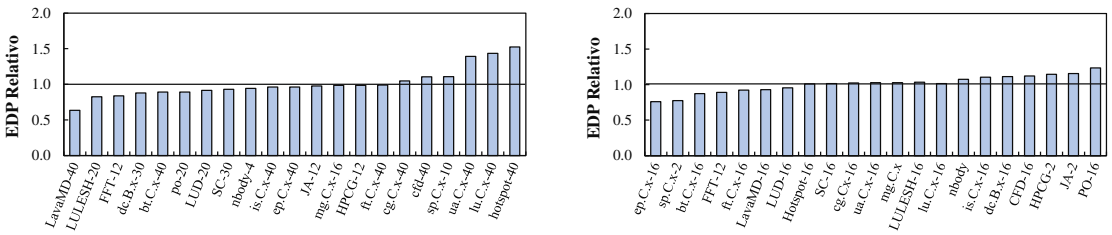
Figura 6. Resultados de EDP do Cenário II.

4.2.3. Cenário III: Baixo TLP com alto número de acessos à memória

Neste cenário, objetivamos encontrar as aplicações paralelas que se beneficiam da execução concorrente com uma aplicação que possuí baixo grau de exploração do paralelismo, mas que estressa bastante a memória compartilhada. Para isto, usamos a aplicação *Stream* para execução concorrente nos processadores Intel e AMD. Conforme podemos observar na Figura 7, diversas aplicações se beneficiam da execução concorrente. No entanto, as aplicações que mais se beneficiam são aquelas com (*i*) alto TLP; (*ii*) baixo número de acessos à memória compartilhada; e/ou (*iii*) alto IPC. No caso mais significativo do processador AMD Ryzen (*ep.C.x*), o EDP é reduzido em 25%. Por outro lado, ao executar este tipo de aplicação concorrentemente com uma aplicação que também possuí alta taxa de acessos à memória compartilhada (e.g., *JA* e *HPCG*, no AMD Ryzen), pode afetar negativamente o EDP.

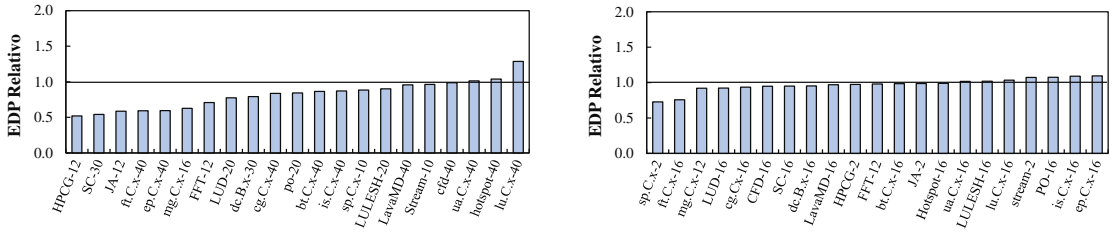
4.2.4. Cenário IV: Baixo TLP e número de acessos à memória

Com este cenário, buscamos identificar as aplicações que podem se beneficiar da execução concorrente com aplicações paralelas onde o melhor número de *threads* é baixo e que estressa pouco a memória compartilhada. Para tanto, selecionamos a aplicação *nbody* em ambos os processadores. Conforme mostrado na Figura 8, a maioria das aplicações se beneficiam da execução concorrente com este tipo de aplicação. No caso mais significativo do processador Intel Xeon (e.g., execução concorrente de *HPCG* e *nbody*), o EDP é otimizado em 49%. No entanto, as aplicações com alto grau de TLP e que são CPU-intensiva com considerável quantidade de acessos à memória compartilhada, não são capazes de se beneficiar da execução concorrente com aplicações de baixo TLP e número de acessos à memória compartilhada.



(a) Xeon: *Stream-10* (b) AMD: *Stream-2*

Figura 7. Resultados de EDP do Cenário III.



(a) Xeon: *nbody-4* (b) AMD: *nbody-6*

Figura 8. Resultados de EDP do Cenário IV.

4.2.5. Cenário V: Médio TLP com alto IPC

Para identificar as aplicações que podem se beneficiar da execução concorrente com aplicações de médio TLP mas com alto IPC, selecionamos as aplicações *PO* e *ft.C.x* no processador Intel e AMD, respectivamente. Conforme observado na Figura 9, os melhores resultados da execução concorrente são obtidos com aplicações que apresentam (*i*) baixo/médio TLP e/ou (*ii*) alta taxa de misses no último nível da memória cache (e.g., *mg.C.x* no AMD Ryzen). Neste caso, o EDP pode ser otimizado em 39% no AMD.

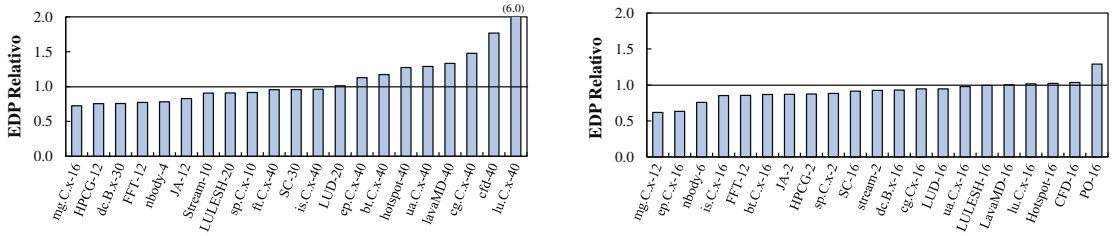
4.2.6. Cenário VI: Médio TLP com baixo IPC

Por fim, selecionamos as aplicações *dc.B.x* e *mg.C.x* no Intel e AMD, respectivamente, para identificar qual tipo de aplicação pode se beneficiar da execução concorrente com aplicações que possuem este tipo de característica. Neste cenário, aplicações com baixo IPC e acessos à memória compartilhada são beneficiadas, independente do nível de TLP, conforme observado na Figura 10, para o processador Intel Xeon. No caso mais significativo, ao executar a aplicação *dc.B.x* de maneira concorrente com a aplicação *FFT* no processador Intel Xeon, o EDP é otimizado em 43%.

4.3. Diretivas para os Desenvolvedores de Aplicações Paralelas

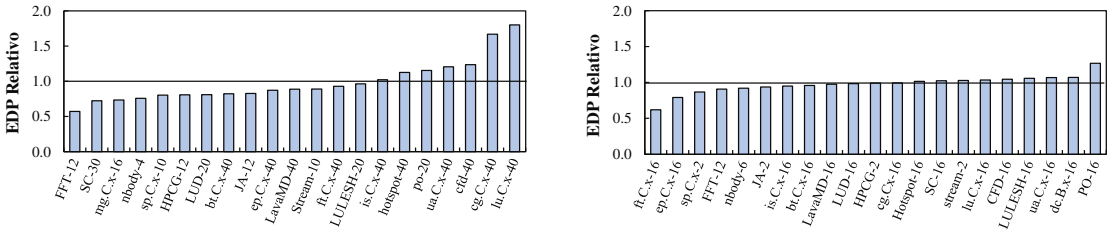
Através da exploração de espaço e projeto da execução de vinte e duas aplicações com diferentes características de acesso a memória cache, IPC e grau de exploração de TLP em dois processadores *multicore*, as seguintes diretivas podem ser utilizadas por desenvolvedores de aplicações paralela com o objetivo de otimizar o uso dos recursos computacionais:

- A execução concorrente de aplicações com baixa taxa de acessos à memória compartilhada e baixo TLP com aplicações que possuem diferentes características de



(a) Xeon: *PO-20* (b) AMD: *ft.C.x-12*

Figura 9. Resultados de EDP do Cenário V.



(a) Xeon: *dc.B.x-30* (b) AMD:*mg.C.x-12*

Figura 10. Resultados de EDP do Cenário VI.

acesso à memória e graus de TLP possibilita melhor uso dos recursos computacionais, apresentando otimização no EDP.

- Quando aplicações com médio TLP e alto IPC são executadas concorrentemente com aplicações com baixo/médio TLP e alta taxa de misses no último nível da memória cache, os recursos computacionais são melhores utilizados e o EDP é otimizado.
- Ao executar aplicações com médio TLP e baixo IPC concorrentemente com aplicações que apresentam baixo IPC e acessos à memória compartilhada, o trade-off entre desempenho e consumo de energia é otimizado.
- Quando as aplicações possuem alto grau de exploração de TLP e alto número de acessos à memória compartilhada, o indicado é que tais aplicações sejam executadas de maneira individual, pois apresentam melhor resultado de EDP do que se executadas concorrentemente com qualquer outra aplicação.

5. Conclusão

Este artigo apresentou um estudo da execução concorrente de aplicações paralelas em arquiteturas *multicores*. Realizamos uma extensa exploração de espaço e projeto para mostrar quais aplicações paralelas podem ser executadas de maneira concorrente e as razões para tal comportamento considerando diferentes métricas de *hardware* e *software*: acessos à memória compartilhada (e.g., RAM), IPC e grau de exploração do paralelismo. Embora diferentes trabalhos tenham focado em otimizar o uso dos recursos computacionais, nenhum havia realizado uma análise das potenciais aplicações que podem ser executadas de maneira concorrente. Como trabalhos futuros, pretende-se (*i*) definir um modelo algorítmico para encontrar a melhor combinação de aplicações paralelas que podem ser executadas de maneira concorrente e (*ii*) implementar o algoritmo de otimização de recursos internamente a um sistema de *runtime* para realizar a otimização de maneira autônoma.

Referências

Bailey, D. H., Barszcz, E., Barton, J. T., Browning, D. S., Carter, R. L., Dagum, L., Fatoohi, R. A., Frederickson, P. O., Lasinski, T. A., Schreiber, R. S., Simon, H. D., Venkatakrishnan, V., and Weeratunga, S. K. (1991). The nas parallel benchmarks and summary and preliminary results. In *ACM/IEEE SC*, pages 158–165, USA. ACM.

Che, S., Boyer, M., Meng, J., Tarjan, D., Sheaffer, J. W., Lee, S.-H., and Skadron, K. (2009). Rodinia: A benchmark suite for heterogeneous computing. In *IEEE Int. Symp. on Workload Characterization*, pages 44–54, DC, USA. IEEE Computer Society.

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.

Creech, T., Kotha, A., and Barua, R. (2013). Efficient multiprogramming for multicores with scaf. In *46th Annual IEEE/ACM Int. Symp. on Microarchitecture*, MICRO-46, page 334–345, New York, NY, USA. ACM.

dos Santos Marques, W., de Souza, P. S. S., Lorenzon, A. F., Schneider Beck, A. C., Beck Rutzig, M., and Diniz Rossi, F. (2017). Improving edp in multi-core embedded systems through multidimensional frequency scaling. In *2017 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1–4.

Gonzalez, R. and Horowitz, M. (1996). Energy dissipation in general purpose microprocessors. *IEEE Journal of solid-state circuits*, 31(9):1277–1284.

Hackenberg, D., Ilsche, T., Schone, R., Molka, D., Schmidt, M., and Nagel, W. E. (2013). Power measurement techniques on standard compute nodes: A quantitative comparison. In *IEEE ISPASS*, pages 194–204.

Hähnel, M., Döbel, B., Völp, M., and Härtig, H. (2012). Measuring energy consumption for short code paths using rapl. *SIGMETRICS Performance Evaluation Rev.*, 40(3):13–17.

Harris, T., Maas, M., and Marathe, V. J. (2014). Callisto: Co-scheduling parallel runtime systems. In *Proceedings of the Ninth European Conference on Computer Systems*, EuroSys '14, New York, NY, USA. Association for Computing Machinery.

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*, 38:258–269.

Lorenzon, A. F. and Beck, A. C. S. (2019). *Parallel Computing Hits the Power Wall - Principles, Challenges, and a Survey of Solutions*. Springer Briefs in Computer Science. Springer.

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

Lorenzon, A. F., Dellagostin Souza, J., and Schneider Beck, A. C. (2017). Laant: A library to automatically optimize edp for openmp applications. In *Design, Automation Test in Europe Conference Exhibition (DATE), 2017*, pages 1229–1232.

Lorenzon, A. F., Sartor, A. L., Cera, M. C., and Beck, A. C. S. (2015). The influence of parallel programming interfaces on multicore embedded systems. In *IEEE COMPSAC*, volume 2, pages 617–625. IEEE.

O'brien, K., Pietri, I., Reddy, R., Lastovetsky, A., and Sakellariou, R. (2017). A survey of power and energy predictive models in hpc systems and applications. *ACM Computing Surveys (CSUR)*, 50(3):1–38.

Raasch, S. E. and Reinhardt, S. K. (2003). The impact of resource partitioning on smt processors. In *PACT*, pages 15–25.

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

Suleman, M. A., Qureshi, M. K., and Patt, Y. N. (2008). Feedback-driven threading: Power-efficient and high-performance execution of multi-threaded workloads on cmps. *SIGARCH Computer Architecture News*, 36(1):277–286.

Tousimojarad, A. and Vanderbauwhede, W. (2014). An efficient thread mapping strategy for multiprogramming on manycore processors. *CoRR*, abs/1403.8020.

Varisteas, G. (2015). *Effective cooperative scheduling of task-parallel applications on multiprogrammed parallel architectures*. PhD thesis. QC 20151016.