Avaliação de Desempenho, Consumo de Energia e Gerenciadores de Frequência em Sistemas Transacionais

João Paulo Labegalini de Carvalho[1], **Alexandro Baldassin**[1]

[1]Departamento de Estatística, Matemática Aplicada e Computação
Universidade Estadual Paulista – UNESP
CEP 13506-900 – Rio Claro – SP – Brasil

{joaopaulo,alex}@rc.unesp.br

Abstract. *Companies such as IBM and Intel have added hardware support for transactional memory in their latest processors. Similarly to the initial research on software transactional memory, preliminar analyses conducted with the hardware support have focused on evaluating execution time. Given the importance of energy consumption in modern computer systems, it is vital the evaluation of its behavior and the respective performance/energy tradeoffs. With that in mind, this work evaluates not only application execution time, but also the energy consumption of both software transacional memory and Intel's Haswell new hardware support. Moreover, this paper also evaluates the impact of all five Linux kernel's frequency governors, observing EDP (Energy-Delay Product) gains of up to 20% in 2 out of the 8 studied applications.*

Resumo. *Empresas como a IBM e Intel adicionaram suporte em hardware para memória transacional em seus últimos processadores. Semelhante às pesquisas iniciais com memória transacional em software, análises preliminares conduzidas com o suporte em hardware focaram na avaliação do tempo de execução. Dada a importância do consumo de energia em sistemas computacionais modernos, é vital que também sejam avaliados seu comportamento e respectiva correlação desempenho/energia. Com isso em mente, este trabalho avalia não apenas o tempo de execução, mas também o consumo de energia de uma implementação em software de memória transacional e o novo suporte em hardware presente no processador Haswell da Intel. Além disso, este artigo também avalia o impacto de cinco gerenciadores de frequência presentes no kernel do Linux, observando ganhos em EDP (Produto Energia-Latência) de até 20% em 2 de 8 aplicações estudadas.*

1. Introdução

O desenvolvimento tecnológico permite à indústria de microprocessadores a construção de máquinas cada vez mais rápidas, seguindo a tendência da Lei de Moore [Moore 1965]. Nos últimos 40 anos, a pilha de software alcançou maior desempenho a cada nova geração de microprocessadores, sem modificação do código-fonte, devido principalmente ao aumento da frequência de operação e otimizações microarquiteturais, como ILP (*Instruction Level Parallelism*) por exemplo. No entanto, a observação do efeito *power wall* [Barroso and Holzle 2007] desacelerou o aumento da frequência de operação, enquanto que técnicas baseadas em ILP também produziram menos ganhos. Como alternativa às abordagens baseadas em um único fluxo de execução, foram construídas as microarquiteturas com múltiplas unidades de execução (*multicore*).

As arquiteturas *multicore* mostraram-se desafiadoras do ponto de vista de programação, fazendo-se necessária uma revolução na forma como pensamos sobre software. Os modelos e ferramentas atualmente disponíveis não são suficientes para permitir uma exploração significativa de todos os núcleos de execução, o que somente será possível com novas abordagens [Sutter and Larus 2005]. Um novo modelo de programação concorrente, conhecido como Memória Transacional (TM – *Transactional Memory*) [Harris et al. 2010], abstrai o controle de concorrência em sistemas de múltiplas linhas de execução (*threads*) usando o conceito de *transação*, largamento empregado em sistemas de banco de dados.

O investimento na área de TM não é restrito apenas à academia. Recentemente a indústria de microprocessadores mostrou interesse e investimentos concretos na área. Como exemplo pode-se citar o *Blue Gene/Q*™ [Wang et al. 2012], supercomputador destinado à computação de alto desempenho, lançado em 2012 pela IBM® que possui suporte em *hardware* para Memória Transacional (HTM). Outro exemplo de máquina com suporte à HTM é o processador *Haswell*® lançado pela Intel® em Junho de 2013. Diferentemente do supercomputador da IBM®, o processador da Intel® pode ser empregado em uma gama maior de dispositivos, desde máquinas para computação de alto desempenho até dispositivos móveis (*smartphones* e *tablets*).

Este trabalho avalia o impacto no consumo de energia e desempenho do suporte em *hardware* do processador *Haswell*® e de uma biblioteca transacional em *software*. A análise desse impacto é importante se TM pretende ser empregado em escala comercial, como apontado recentemente por empresas como a IBM e a Intel. A preocupação com questões energéticas não é algo novo [Barroso and Holzle 2007, Firasta et al. 2008], mas apenas recentemente tem-se observado uma atenção maior por parte da indústria e da academia em sua investigação. Uma manifestação concreta dessa preocupação é a interface RAPL (*Running Average Power Limit*) [David et al. 2010] adicionada à microarquitetura Intel® a partir do modelo *Sandy Bridge*™.

As contribuições deste trabalho, em resumo, são:

- Avaliação do consumo de energia e desempenho de todas as 8 aplicações do pacote transacional STAMP (*Stanford Transactional Applications for Multi-Processing*) [Minh et al. 2008] (Seção 4.2.1). Essa análise considera a utilização da biblioteca em software de TM, a TinySTM [Felber et al. 2008], e o suporte em *hardware* do *Haswell*®. Os resultados apontam que, quando comparado com o suporte em *hardware*, STM (*Software Transactional Memory*) tem melhor desempenho e custo energético em 6 das 8 aplicações;
- Análise do impacto de diversas políticas de gerenciamento de frequência no consumo de energia e desempenho (Seção 4.2.2). Esse estudo é pioneiro e até então inexistente na literatura no contexto de memória transacional em *hardware*. Constatou-se que o uso de uma política diferente da padrão em geral piora o EDP (*Energy Delay Product*), mas também pode alcançar melhoras de até 20%.

Os resultados apresentados aqui são uma atualização e extensão natural dos obtidos pelos autores em [Baldassin et al. 2012, Baldassin et al. 2013], onde apenas sistemas de TM em *software* foram considerados. Este trabalho está organizado da seguinte forma. A Seção 2 apresenta uma breve introdução sobre memória transacional e apresenta a extensão transacional presente no processador *Haswell*®. Os gerenciadores de frequência e

a interface RAPL são discutidos na Seção 3. Em seguida, a Seção 4 analisa os principais resultados obtidos para o pacote de aplicações STAMP, enquanto a Seção 5 apresenta as conclusões do trabalho.

2. Memória Transacional

Memória transacional é o modelo de computação paralela no qual operações na região compartilhada de memória são realizadas por meio de transações. A região compartilhada é abstraída como um banco de dados e as transações (leituras e escritas) possuem os seguintes atributos: atomicidade, consistência e isolamento. Um operação/transação em memória é atômica se nenhum de seus estados intermediários é observável pelo sistema. Ou seja, ou os efeitos da transação são observados como um todo, nesse caso dizemos que a transação efetivou suas modificações (*commit*), ou então a transação falha e suas modificações são descartadas e para o sistema é como se nada tivesse acontecido (*abort*). Consistência é a propriedade que garante que os estados observáveis das variáveis no sistema são sempre válidos, enquanto o isolamento dita que estados intermediários de processamento de uma transação não são observáveis por quaisquer outras.

Sistemas transacionais se diferenciam em quatro aspectos básicos: mecanismo de controle de concorrência, gerenciamento de versões, estratégia de detecção e resolução de conflitos [Harris et al. 2010]. O controle de concorrência pode acontecer de duas formas, ditas *pessimista* e *otimista*. Em uma implementação pessimista a detecção e resolução de um conflito ocorrem no momento em que a transação tenta acessar uma região de memória modificada por outra transação. Na abordagem otimista a detecção e resolução do conflito ocorre de forma tardia.

O mecanismo de versionamento gerencia as versões, ditas *corrente* e *especulativa*, durante o ciclo de vida de uma transação. A versão corrente é a observável globalmente por todas as transações e a especulativa é o resultado da modificação efetuada por uma transação ainda não finalizada. A gestão das versões pode ocorrer de maneira *preguiçosa* ou *ansiosa*. O versionamento preguiçoso armazena a versão especulativa em um *buffer* privado à transação e a versão corrente permanece na região compartilhada. No versionamento ansioso a versão especulativa é armazenada na região global e a versão corrente é armazenada localmente à transação.

O termo Memória Transacional foi cunhado em 1993 por Herlihy e Moss [Herlihy and Moss 1993]. As pesquisas iniciais se concentraram em implementações em *software*, sendo que apenas recentemente o suporte restrito para Memória Transacional foi adicionado a um processador convencional, o processador da nova microarquitetura Intel® de codinome *Haswell*®.

2.1. Extensão de Sincronização Transacional (TSX)

A extensão de sincronização transacional (TSX – *Transactional Synchronization Extensions*) é uma extensão do conjunto de instruções implementada no novo processador *Haswell*® da Intel. Nessa nova arquitetura a utilização do suporte de execução especulativa pode ocorrer de duas maneiras: utilizando a Omissão de Locks em Hardware (HLE – *Hardware Lock Elision*), detalhado em 2.1.1, ou utilizando as instruções do modelo Restrito de Memória Transacional (RTM – *Restricted Transactional Memory*), em 2.1.2. Em ambas as abordagens os dados especulativos são gerenciados através da cache L1,

privadas ao *core* e tipicamente de 32KB. A detecção de conflito ocorre na granularidade da linha de cache (64 bytes) pelo protocolo de coerência [Int 2013]. As diferenças, como detalhadas a seguir, estão na semântica e controle do pós-cancelamento e reinício da transação.

2.1.1. Omissão de Locks em Hardware (HLE)

Omissão de locks em hardware é um mecanismo de prefixação de instruções de escrita associadas às operações em variáveis de trava (*lock*). Ao encontrar o prefixo, o *hardware* omite essa escrita e inicia a execução da região crítica de maneira especulativa. Na visão global do sistema, a variável de trava associada tem estado "trava liberada", o que permite que outras *threads* entrem na região crítica também de maneira especulativa. Ao atingir a escrita de liberação da trava, caso nenhum conflito tenha sido detectado, as modificações privadas da transação tornam-se disponíveis globalmente. Caso contrário, as modificações são descartadas e a região crítica é reexecutada. No entanto, ao reiniciar, a escrita na variável de trava é efetuada, causando assim o cancelamento das execuções especulativas e forçando cada *thread* a adquirir a trava antes de entrar na região crítica.

A vantagem dessa estratégia é a possibilidade de exploração do paralelismo ao evitar a serialização das regiões críticas em tempo de execução. Além disso, o progresso da execução é garantido, uma vez que em caso de falha a região crítica é executada de maneira exclusiva. A utilização do mecanismo HLE traz também como vantagem a portabilidade do código, visto que as modificações necessárias serão realizadas nas bibliotecas de sincronização. A portabilidade se estende ao não exigir a recompilação de aplicações, visto que o mecanismo HLE não é composto de novas instruções e sim de prefixos de instruções que são ignorados em processadores sem esse suporte.

2.1.2. Memória Transacional Restrita (RTM)

A implementação de memória transacional restrita é a resposta da Intel® frente ao recente interesse pela indústria em processadores com suporte à execução transacional. A RTM é uma implementação de memória transacional de "melhor esforço" (*best effort*), ou seja, o *hardware* não garante o progresso do sistema. Um caminho de execução alternativo, chamado *fallback*, deve ser fornecido pelo programador e possuir a lógica necessária para garantir o progresso e a consistência do sistema. Uma limitação da implementação é a ausência de instruções de escrita não-transacionais, o que impossibilita a depuração direta do código.

As instruções que fazem parte da interface RTM são: `xbegin`, `xend` e `xabort`. A primeira inicializa a execução especulativa, salvando o estado atual da arquitetura. Essa instrução tem como parâmetro o endereço de *fallback* que será executado após um *abort*. A segunda realiza o *commit* do estado especulativo caso nenhum conflito seja detectado. Do contrário, o fluxo de execução é desviado para o endereço de *fallback*. Por fim, a instrução `xabort` aborta a transação e permite informar pelo registrador EAX a razão do *abort* com o valor do imediato passado como parâmetro. Os mesmos bits são definidos pelo *hardware* quando uma das situações de *abort* é alcançada: (i) conflito de dados, (ii) *overflow* da cache L1, (iii) nível máximo de aninhamento atingido, (iv) instrução de depuração ou (v) instrução `xabort` executada. Outras situações levam ao *abort* de uma

Tabela 1. Políticas para gerenciamento de frequência

Política	Descrição
powersave	Ajusta a frequência de operação para a mínima disponível. A frequência permanece a mínima independente da carga de trabalho da CPU.
userspace	Permite que qualquer aplicação executando com o UID 0 (privilégio de superusuário) escolha em qual frequência a CPU deve operar.
conservative	Aumenta e diminui, gradativamente, a frequência conforme a carga de trabalho da CPU.
ondemand	Similar à *conservative*, porém o aumento ou redução da frequência é discreto, ou seja, ocorre de forma direta.
performance	Similar à *powersave*, porém a máxima frequência disponível é fixada.

transação, como por exemplo chamadas de sistema, mas o hardware não as rotula com um valor específico, apenas aborta a transação e cabe ao programador identificá-las.

3. Controle e Perfilamento do Consumo de Energia

Restrições de consumo de energia em servidores e dispositivos móveis fez com que novos processadores fossem projetados com controle dinâmico de frequência e voltagem (DVFS [Macken et al. 1990]). Aos sistemas operacionais foram então adicionados mecanismos e políticas de gerenciamento de frequência. Na subseção 3.1 serão discutidos os gerenciadores de frequência utilizados nesse trabalho. Seguindo também a tendência de preocupação energética, foram adicionados elementos nas microarquiteturas modernas para auxiliar o perfilamento energético de aplicações. O suporte arquitetural utilizado para obtenção dos resultados apresentados na Seção 4 será discutido na subseção 3.2.

3.1. Gerenciadores de Frequência

O *kernel* do Linux atualmente implementa cinco políticas de gestão de frequência: *powersave*, *userspace*, *conservative*, *ondemand* e *performance*. Cada uma dessas políticas estende as funcionalidades do módulo *acpi-cpufreq* e, por estarem implementadas em módulos independentes, permitem o controle dinâmico da frequência. A forma de gestão da frequência é ditada pelas diretrizes de cada política, explicadas resumidamente na Tabela 1. O que motiva o controle de frequência dinâmico é que, durante períodos de ociosidade do sistema, a frequência pode ser diminuída para reduzir o consumo. Da mesma forma, em períodos de uso intenso a frequência pode ser aumentada visando completar as tarefas mais rapidamente, reduzindo o tempo de CPU e potencialmente o consumo de energia.

As políticas *conservative* e *ondemand* operam entre a frequência mínima e a máxima, se diferenciando apenas quanto ao salto de frequência que ocorre de forma gradativa e direta, respectivamente. A frequência é alterada de acordo com a carga de utilização da CPU, verificada em intervalos regulares (na ordem de alguns ms) pelo *kernel*. O passo da mudança de frequência e o intervalo de verificação da carga da CPU podem ser configurados. É importante salientar que as frequências disponíveis de operação da CPU são definidas de fábrica e variam de processador para processador.

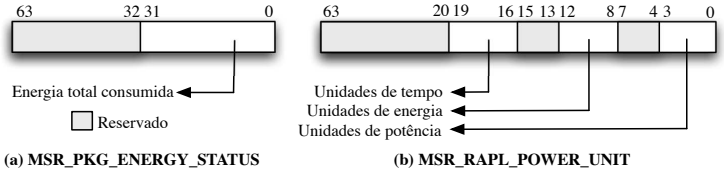

Figura 1. Registradores da interface RAPL: (a) registrador que armazena o contador; (b) registrador que armazena as unidades.

O padrão na maioria dos sistemas Linux é a política *ondemand*, mas a *conservative* é mais atraente em sistemas portáteis, nos quais o tempo de vida da bateria é importante. Há duas maneiras de alterar o gerenciador de frequência, daqui em diante referenciado por *governor*, vigente no sistema: usando as aplicações do pacote *cpufrequtils*; ou pela edição dos arquivos de configuração do módulos do *kernel*, comumente localizados em *"/sys/devides/cpu/"*.

3.2. Perfilamento Energético

Neste trabalho são utilizados os contadores de desempenho presentes na arquitetura dos processadores Intel para medição do consumo de energia. Os contadores de energia, presentes desde a microarquitetura *Sandy Bridge*™, são registradores específicos de modelo, ou MSRs (da sigla em inglês). MSRs são registradores de propósito específico adicionados à microarquitetura para facilitar a depuração de aplicações, permitir o monitoramento de desempenho e configurar as características da CPU. No manual do desenvolvedor, os MSRs de energia pertencem à interface RAPL (*Running Average Power Limit*) [David et al. 2010, Int 2013], que possui contadores de consumo de energia, dissipação de potência e tempo. A RAPL permite também a imposição de restrições mais severas de consumo de energia. Nesse trabalho é utilizado apenas o contador de consumo de energia representado na Figura 1.

O registrador da Figura 1a é um contador de 32 bits que é atualizado a cada $1ms$, aproximadamente. O registrador da Figura 1b armazena as unidades da interface RAPL. A energia consumida, em joules, por um trecho de programa é dada pela seguinte expressão:

$$E = (C_{depois} - C_{antes}) * 2^{(-EU)}$$

onde C_{depois} e C_{antes} representam os valores obtidos através da leitura do registrador de contagem (Figura 1a) depois e antes do trecho de código, e EU (*Energy Unit*) é dado pelos bits 12:8 do registrador `MSR_RAPL_POWER_UNIT` (Figura 1b). As unidades da interface RAPL são dependentes de arquitetura. Para o processador utilizado neste trabalho o valor de EU é 14, provendo uma granularidade de $61,0$ microjoules. É importante salientar que, devido à limitação de tamanho do contador (inteiro de 32 bits), se faz necessário um cuidado com o tempo de *wraparound* – aproximadamente $60s$, a partir do qual o contador é reiniciado devido ao limite de 32 bits e as informações podem ser perdidas.

4. Avaliação de Desempenho e Consumo de Energia

Nesta seção serão apresentados os resultados da avaliação do desempenho e consumo de energia de sistemas, em software e hardware, de memória transacional. Como parte da análise, serão comparados tempo de execução e consumo de energia das aplicações com cada um dos *governors* apresentados na subseção 3.1. Os dados de consumo de energia foram coletados utilizando os registradores da RAPL (vide subseção 3.2) conforme será discutido a seguir.

4.1. Metodologia Experimental

O pacote de aplicações transacionais conhecido como STAMP [Minh et al. 2008] foi o escolhido para condução da análise. Este pacote foi escolhido pois é composto de 8 aplicações com uma boa diversidade de parâmetros transacionais, tais como tempo de execução, tempo em transação e nível de contenção. A biblioteca transacional usada é a TinySTM [Felber et al. 2008], versão 1.0.4, considerada estado da arte da pesquisa em STM. A configuração usada dessa biblioteca é a padrão, usando tipo de versionamento *preguiçosa* e detecção de conflitos pessimista. O caminho alternativo (*fallback*) das transações em *hardware*, executado quando a transação não conseguiu finalizar após 5 tentativas, simplesmente serializa a execução utilizando uma trava global.

Os dados apresentados nessa seção foram obtidos em uma máquina com processador Intel® i7-4770 com *hyperthreading* habilitado, perfazendo um total de 8 núcleos. As frequências mínima e máxima de operação são iguais a 0.8 e 3.4Ghz, respectivamente. A máquina conta com um total de 16Gb de memória RAM e usa um ambiente Linux típico, com *kernel* versão 3.8.0. As aplicações e biblioteca STM foram compiladas usando o `GCC` versão 4.8.2, com nível três de otimização (`-O3`). Cada experimento foi executado 20 vezes para obtenção dos resultados apresentados nesta seção.

4.2. Resultados Experimentais

A análise será conduzida em duas partes. Na seção 4.2.1 será apresentado o resultado obtido quando se executa as aplicações STAMP, mantendo o gerenciador de frequência padrão (*ondemand*) habilitado, variando os seguintes mecanismos de paralelização: memória transacional em software (biblioteca TinySTM na configuração padrão), uso do suporte do *Haswell*® para memória transacional em hardware (RTM) e omissão de travas em hardware (HLE). Essa análise mostrará o tempo de execução e consumo de energia observados pela maioria dos usuários. Na segunda parte, apresentada na seção 4.2.2, os experimentos da primeira parte são repetidos variando-se o gerenciador de frequência do sistema. Dessa forma, é possível analisar o impacto no desempenho e custo energético das diretrizes de cada política.

4.2.1. Análise do Desempenho com Configuração Padrão

O gráfico da Figura 2 mostra, para cada aplicação estudada, o produto energia-latência (EDP) normalizado em relação à execução sequencial. São mostrados os resultados das execuções com omissão de travas em *hardware* (HLE), memória transacional em *hardware* (RTM) e biblioteca em *software* de TM (tinySTM), variando o número de *threads* entre 1 e 8. Como os valores de EDP estão normalizados com relação ao sequencial, valores maiores do que 1 representam um resultado melhor que o sequencial (e vice-versa). Nos gráficos de tempo e consumo de energia da Figura 3 é mostrada uma linha base referente à execução sequencial (SEQ). O tempo de execução é mostrado em segundos e o consumo de energia é mostrado em Joules.

O gráfico da Figura 2 mostra que o uso do STM apresentou melhor EDP em 6 das 8 aplicações STAMP, quando comparado com HLE e RTM. As exceções são as aplicações `K-Means` e `SSCA-2`, que apresentaram melhor EDP quando executadas com RTM e HLE, respectivamente. A justificativa para esse resultado é que em ambas

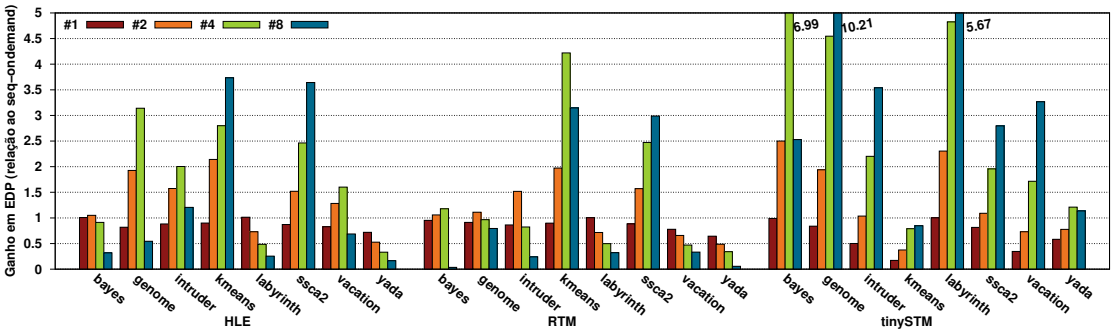

Figura 2. Ganho em EDP das aplicações STAMP com relação ao sequencial.

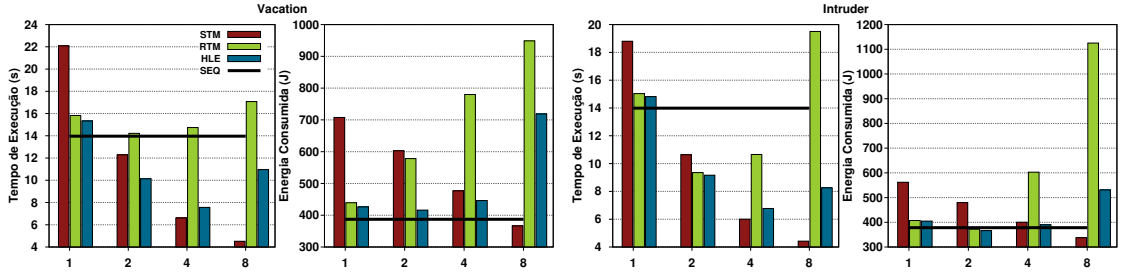

Figura 3. Detalhamento do tempo de execução e energia consumida.

aplicações a região crítica é breve, sendo assim o *overhead* introduzido pelo STM é considerável quando comparado ao baixo *overhead* das transações em *hardware*. As aplicações `Labyrinth`, `Vacation` e `Yada` foram as que mais claramente não obtiveram ganho em EDP. A principal razão é que o desempenho dessas aplicações foi limitado pelo tamanho do *buffer* de escrita especulativa (cache L1 de 32KB e 8 vias), visto que são essas as aplicações com maiores regiões críticas do ponto de vista de acesso à memória.

Com o objetivo de obter um maior entendimento da relação energia-desempenho, a Figura 3 mostra o tempo de execução e consumo de energia das aplicações `Vacation` e `Intruder`. Essas aplicações foram escolhidas por mostrarem claramente que não existe uma relação simples entre o tempo de execução e consumo de energia em aplicações transacionais, diferente do que se observa em aplicações convencionais. Apenas essas duas aplicações foram detalhadas devido à restrição de espaço para apresentação do trabalho. Um exemplo é a aplicação `Intruder` que, mesmo reduzindo o tempo de execução, apresenta aumento no consumo de energia. Foi observado que o uso do suporte em *hardware* do *Haswell*® gerou um aumento no consumo de energia com relação às execuções sequenciais e STM, mesmo quando o tempo de execução pouco se altera (vide aplicação `Vacation`). Acreditamos que esse acréscimo no consumo de energia deve-se ao custo das ações pós-cancelamento da transação (invalidação das linhas escritas especulativamente e limpeza dos bits nas linhas do conjunto de leitura) e da manutenção da consistência do estado especulativo.

4.2.2. Análise Comparativa com Múltiplos Gerenciadores de Frequência

A Figura 4 mostra o ganho em EDP, relativo ao gerenciador de frequência padrão (*ondemand*), das aplicações STAMP quando executadas com 4 *threads*. Desta forma, valores maiores do que 1 representam resultados melhores do que os obtidos com o *governor* padrão. Como nessa parte da análise o objetivo é analisar o impacto dos *governors*, escolhemos mostrar o resultado das execuções com 4 *threads*, o que tornando a comparação

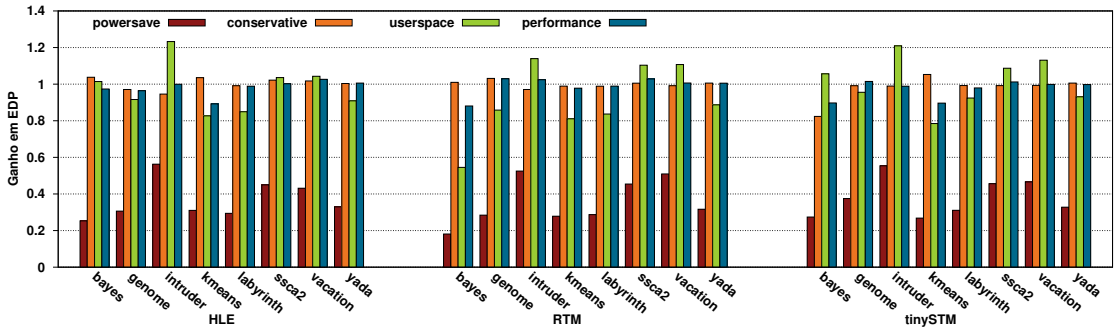

Figura 4. Ganho em EDP com 4 threads em relação ao gerenciador ondemand.

com STM mais justa, pois é o cenário no qual o suporte em *hardware* não é penalizado pelo compartilhamento da cache L1 no caso de duas *thread* estarem escalonadas no mesmo core. O módulo *userspace*, como já dito, permite a escolha da frequência de operação pelo usuário. Os resultados relativos ao *governor userspace* foram obtidos com a frequência fixada em 2.3Ghz. Essa configuração permite a análise de um cenário onde a frequência está em um valor intermediário ao observado quando as políticas *powersave* e *performance* estão ativas.

O gráfico mostra que, no geral, o EDP da aplicação piora com a mudança do *governor*. A redução da frequência de operação aumentou o tempo de execução e não reduziu o consumo energético significativamente, aumentando assim o EDP e reduzindo o ganho. O inverso ocorreu com o aumento da frequência de operação: maior consumo de energia sem redução do tempo de execução. As aplicações que obtiveram ganho foram as que apresentam a característica de estabilidade de consumo energético com o aumento do número de *threads*, como pode-se observar, por exemplo, na aplicação `Intruder` no gráfico da Figura 3, que obteve um ganho de cerca de 20%.

Outra razão que justifica a diminuição do ganho em EDP observado em quase todas as aplicações é o fato de sistemas transacionais funcionarem de maneira otimista. Ou seja, mesmo as *threads* das transações que não estão fazendo progresso permanecem ativas executando tarefas de gestão como, por exemplo, abortando e desfazendo modificações. Quando a taxa de contenção é alta, a repetição das tarefas de gestão tem impacto significativo no consumo de energia. Esse impacto pode ser agravado caso a janela de tempo entre o cancelamento e reinício da transação seja pequena.

5. Conclusão

A avaliação apresentada neste trabalho mostra que sistemas de memória transacional em *software* podem reduzir não apenas o tempo de execução assim como também o consumo de energia. Em 6 das 8 aplicações observou-se molhara no EDP comparando-se com os mecanismos de memória transacional em *hardware* (HLE e RTM). Esse resultado revela a maturidade dos sistemas em *software* de TM e que as restrições da implementação em *hardware* limitam severamente o desempenho. No entanto, o bom desempenho observado em 2 de 8 aplicações revela a viabilidade do suporte para transações pequenas e indica que algumas otimizações podem melhorar esse cenário.

Este artigo também analisou o impacto de diferentes políticas de controle de frequência em aplicações transacionais em sistemas de TM em *hardware* e software, estudo esse sem paralelo até então na literatura. Em especial, mostrou-se que a troca da

política padrão para uma outra piorou o EDP das aplicações consideradas. No entanto, foi observado em 2 das 8 aplicações uma melhora de até 20%. Isto parece indicar que tornar os sistemas de memória transacional cientes dessas políticas, e usá-las de maneira mais direta, pode melhorar o desempenho e consumo de energia de aplicações transacionais.

Referências

(2013). *Intel® 64 and IA-32 Architectures Software Developer's Manual*. Vol. 3B.

Baldassin, A., de Carvalho, J. P. L., and Azevedo, R. (2013). Reavaliando a eficiência energética de memória transacional em processadores convencionais. In *Anais do XIV Simpósio em Sistemas Computacionais (WSCAD-SSC)*, pages 69–76.

Baldassin, A., de Carvalho, J. P. L., Garcia, L. A. G., and Azevedo, R. (2012). Energy-performance tradeoffs in software transactional memory. In *Proceedings of the 2012 IEEE 24th International Symposium on Computer Architecture and High Performance Computing*, pages 147–154.

Barroso, L. A. and Holzle, U. (2007). The case for energy-proportional computing. *IEEE Computer*, 40(12):33–37.

David, H., Gorbatov, E., Hanebutte, U. R., Khanna, R., and Le, C. (2010). RAPL: memory power estimation and capping. In *Proceedings of the 16th ACM/IEEE international symposium on Low power electronics and design*, pages 189–194.

Felber, P., Fetzer, C., and Riegel, T. (2008). Dynamic performance tuning of word-based software transactional memory. In *Proceedings of the 13th Symposium on Principles and Practice of Parallel Programming*, pages 237–246.

Firasta, N., Buxton, M., Nasri, K., and Kuo, S. (2008). *Intel® AVX: New Frontiers in Performance Improvements and Energy Efficiency*.

Harris, T., Larus, J., and Rajwar, R. (2010). *Transactional Memory*. Morgan & Claypool Publishers, 2 edition.

Herlihy, M. and Moss, J. E. B. (1993). Transactional memory: Architectural support for lock-free data structures. In *Proceedings of the 20th Annual International Symposium on Computer Architecture*, pages 289–300.

Macken, P., Degrauwe, M., Van Paemel, M., and Oguey, H. (1990). A voltage reduction technique for digital systems. In *Solid-State Circuits Conference, 1990. Digest of Technical Papers. 37th ISSCC., 1990 IEEE International*, pages 238–239. IEEE.

Minh, C. C., Chung, J., Kozyrakis, C., and Olukotun, K. (2008). STAMP: Stanford Transactional Applications for Multi-Processing. In *Proceedings of the IEEE International Symposium on Workload Characterization*, pages 35–46.

Moore, G. E. (1965). Cramming more components onto integrated circuits. *Electronics*, pages 114–117.

Sutter, H. and Larus, J. (2005). Software and the concurrency revolution. *Queue*, 3(7):54–62.

Wang, A., Gaudet, M., Wu, P., Amaral, J. N., Ohmacht, M., Barton, C., Silvera, R., and Michael, M. (2012). Evaluation of Blue Gene/Q hardware support for transactional memories. In *Proceedings of the 21st International Conference on Parallel Architectures and Compilation Techniques*, pages 127–136.