Portabilidade com Eficiência da Advecção do Modelo BRAMS entre Arquiteturas Multi-Core e Many-Core

Manoel Baptista da Silva Jr.[1], Jairo Panetta[2], Stephan Stephany[3]

[1]Programa de Mestrado em Computação Aplicada (CAP)
Instituto Nacional de Pesquisas Espaciais (INPE) – São José dos Campos, SP – Brasil

[2]Divisão de Ciência da Computação (IEC)
Instituto Tecnológico de Aeronáutica (ITA) – São José dos Campos, SP – Brasil

[3]Laboratório Associado de Computação e Matemática Aplicada (LAC)
Instituto Nacional de Pesquisas Espaciais (INPE) – São José dos Campos, SP – Brasil

manoel.baptista@cptec.inpe.br,jairo.panetta@gmail.com,stephan@lac.inpe.br

Abstract. *This work investigates the possibility of obtaining portability with efficiency of an advection code to multi-core and many-core architectures using OpenMP and OpenACC directives. The advection code is part of the dynamics of the regional meteorological model BRAMS, executed daily in production mode at CPTEC/INPE, parallelized with the Message Passing Interface (MPI) library. We demonstrate that a single code with both directives obtains acceptable performance on both architectures.*

Resumo. *Este trabalho investiga a possibilidade de se obter portabilidade com eficiência do código da advecção de escalares para arquiteturas multi-core e many-core utilizando diretivas OpenMP e OpenACC. O código da advecção utilizado neste estudo é um trecho da dinâmica do modelo meteorológico regional BRAMS, correntemente executado em produção no CPTEC/INPE, paralelizado com a biblioteca de comunicação por troca de mensagens MPI. Demonstramos que é possível obter desempenho aceitável nas duas arquiteturas com uma única codificação contendo as duas classes de diretivas.*

1. Introdução

O aumento crescente do uso de aceleradores de processamento como GPGPUs (*General Purpose Graphics Processing Units*) na execução paralela de códigos extensos tais como modelos numéricos de previsão de tempo e clima impõe restrições de portabilidade desses programas entre arquiteturas *multi-core* e *many-core*, uma vez que exigem codificações diferentes para programas que têm centenas de milhares de linhas de código. Além disso, os custos de manutenção de versões diferentes são muitas vezes inviáveis. A linguagem CUDA se tornou a mais comum para uso de GPGPUs (Computação de Alta Performance, 2014), mas sendo derivada da linguagem C, obrigaria a recodificação dos códigos de modelos numéricos existentes, escritos preponderantemente em Fortran 90 ou suas versões mais recentes. Outra linguagem que suporta uma programação paralela heterogênea utilizando CPU e GPU é o OpenCL (OpenCL - The open standard for parallel programming of heterogeneous systems, 2015), mas é também derivada da linguagem C. Assim, padrões tais como o OpenMP (Chapman, Jost, & Van Der Pas, 2008) ou OpenACC (OpenACC Directives for

Accelerators, 2015), baseados na inserção de diretivas de paralelização, e consequentemente com mínima alteração do código sequencial, vem se difundindo, sendo o primeiro para arquiteturas de memória compartilhada (*multi-core*) e o segundo para arquiteturas com aceleradores (*many-core*), mas ambos baseados na execução concorrente de *threads*.

Este trabalho investiga a possibilidade de se obter um código único que seja portável para arquiteturas *multi-core* e *many-core*, com desempenho paralelo aceitável em ambas, por meio de diretivas de paralelização OpenMP e OpenACC. Tipicamente, a granularidade necessária para se obter melhor desempenho em arquiteturas *many-core* é maior que aquela em arquiteturas *multi-core*, dificultando a adoção de um código único. O código escolhido para estudo de caso é o módulo de advecção de escalares da dinâmica do modelo meteorológico regional BRAMS (*Brazilian Regional Atmospheric Modelling System*), que é utilizado operacionalmente no CPTEC (Centro de Previsão do Tempo e Estudos Climáticos) na previsão de tempo regional (Centro de Previsão de Tempo e Estudos Climáticos, 2015). Sua codificação em OpenACC é a primeira conhecida de um trecho da dinâmica de um modelo meteorológico regional.

É importante frisar que modelos existentes são paralelizados com a biblioteca de comunicação por troca de mensagens MPI e portá-los, mesmo parcialmente, para codificações híbridas com OpenMP ou OpenACC representa um esforço de programação substancial, sendo portanto desejável a adoção de um código único, tal como proposto neste trabalho.

2. Modelo meteorológico BRAMS

O BRAMS é um modelo regional baseado no modelo RAMS (*Regional Atmospheric Modeling System*), o qual começou a ser desenvolvido em 1970 por Willian R. Cotton e Roger A. Pielke. Sucessivas versões do BRAMS foram desenvolvidas desde 2003 até os dias de hoje pelo CPTEC para uso operacional (Panetta, 2012) ou para pesquisas em processamento paralelo (Fazenda, et al., 2011), (Rodrigues, et al., 2010) e em Meteorologia (Freitas S. R., et al., 2007), (Longo, et al., 2013), (Brazilian developments on the Regional Atmospheric Modelling System, 2015).

O código fonte do modelo BRAMS tem aproximadamente 350.000 linhas na linguagem Fortran 95 paralelizada com MPI no domínio horizontal. A escalabilidade dessa versão é boa até 9.600 núcleos computacionais, embora o supercomputador tupã que o executa disponha de mais de 30.000 núcleos de processamento (Panetta, 2012). Utiliza-se MPI para o paralelismo inter-nó e intra-nó, tendo este último seu desempenho limitado pelo *overhead* de comunicação entre processos de um mesmo nó. Assim, uma alternativa seria uma codificação híbrida para codificar o paralelismo intra-nó com OpenMP de forma a explorar as vantagens da arquitetura de memória compartilhada. Adicionalmente, seguindo a tendência mundial em supercomputadores, fazer uso de GPGPUs ou outros aceleradores.

2.1. Estrutura do código da advecção

Na dinâmica do modelo BRAMS, a advecção modela o transporte dos componentes do ar na atmosfera de um ponto de grade ao outro. Num dado instante de tempo, o *campo escalar* que representa cada componente do ar é definido por 3 *arrays* tridimensionais, relativos ao instante passado, ao presente e à correspondente tendência (derivada

temporal do instante presente para o futuro). A cada instante de tempo, a rotina *advectc* implementa a advecção de escalares executando uma inicialização comum a todos os campos escalares, seguida por um laço que atualiza todos os campos escalares, um campo por iteração do laço.

A inicialização computa variáveis intermediárias independentes do campo escalar, armazenadas em *arrays* tridimensionais que dependem da geometria da grade, da projeção estereográfica, do passo no tempo e da velocidade do vento, dentre outros. A inicialização elimina cálculos repetitivos, fatorando esses cálculos para fora do laço que percorre os campos escalares. Apesar da eliminação de cálculos repetitivos, a inicialização é custosa, pois calcula 11 campos tridimensionais por meio de 13 laços aninhados que percorrem todo o domínio tridimensional do processo MPI.

Cada iteração do laço que atualiza os campos escalares inicia pela cópia do campo escalar para um *array* temporário. Segue advectando o *array* temporário sucessivamente nas direções x, y e z. Termina calculando a tendência do *array* temporário advectado com relação ao campo original. Os três procedimentos que calculam a advecção unidimensional são compostos por dois aninhamentos que percorrem o domínio horizontal do processo MPI, sendo que o primeiro calcula os fluxos nas faces das células e o segundo utiliza este fluxos para calcular a advecção. O procedimento que calcula a tendência contém um único laço que percorre todo o domínio horizontal do processo MPI.

Neste trabalho, para a realização dos experimentos com OpenMP e OpenACC a subrotina *advectc* do modelo BRAMS, bem como os procedimentos associados foram incluídos num módulo que pode ser executado à parte do modelo e que fornece as mesmas saídas relativas à advecção a cada passo de tempo.

2.2. Formulação da advecção de campos escalares

A advecção tridimensional do BRAMS é obtida por 3 advecções unidimensionais sucessivas em x, y e z, as quais são acumulativas, ou seja, o campo resultante da advecção numa direção é utilizado para a advecção na direção seguinte, utilizando-se sempre a formulação de 2ª ordem unidimensional descrita em (Tremback, Powell, Cotton, & Pielke, 1987), sumarizada a seguir.

Seja $\emptyset$ um campo escalar e u a componente da velocidade na direção x. A equação da advecção unidimensional expressa em função do fluxo é:

$$\frac{\partial \emptyset}{\partial t} = \frac{-\partial u \emptyset}{\partial x}, \tag{1}$$

a qual é discretizada utilizando diferenças finitas por:

$$\emptyset_j^{n+1} = \emptyset_j^n + \frac{\Delta t}{\Delta x}\left[F_{j+\frac{1}{2}} - F_{j-\frac{1}{2}}\right], \tag{2}$$

onde $F_{j-\frac{1}{2}}$ e $F_{j+\frac{1}{2}}$ representam o fluxo $u\emptyset$ ao longo do tempo Δt nas bordas da célula discretizada. O fluxo é aproximado pela integral de um polinômio de Lagrange do 2º grau que interpola $\emptyset$:

$$F_{j+\frac{1}{2}} = \frac{1}{\Delta x}\int_{x_{j+\frac{1}{2}}-u\Delta x}^{x_{j+\frac{1}{2}}} P\left[x, \emptyset_j^n\right] dx, \tag{3}$$

A expressão resultante para o fluxo é:

$$F_{j+\frac{1}{2}}\frac{\Delta t}{\Delta x} = \frac{\propto}{2}\left(-\emptyset_j - \emptyset_{j+1}\right) + \frac{\propto^2}{2}\left(-\emptyset_j + \emptyset_{j+1}\right), \tag{4}$$

onde $\propto = u\frac{\Delta t}{\Delta x}$.

3. Desempenho paralelo com OpenMP

Os testes com OpenMP utilizam duas codificações distintas, uma relativa à paralelização por laços e outra à paralelização por telhas. Na primeira codificação, paralelização por laços, incluíram-se diretivas de paralelização OpenMP nos laços externos dos 13 aninhamentos da advecção, os quais não apresentavam dependência de dados entre iterações. Na execução de cada aninhamento, os pontos da grade horizontal bidimensional são divididos entre os *threads* (laços de pontos da grade horizontal). A cláusula *nowait*, que elimina a barreira implícita no fim de cada aninhamento, foi inserida nos aninhamentos independentes do próximo aninhamento.

Criou-se uma única região paralela, imediatamente antes do laço que invoca *advectc* a cada passo de tempo, de forma a evitar o *overhead* de abrir e fechar regiões paralelas a cada execução dessa rotina e dos 13 aninhamentos.

Na segunda codificação, paralelização por telhas, explora-se a técnica de blocagem para otimizar o acesso à memória. O domínio horizontal é então dividido em telhas. Cada telha abrange um conjunto de pontos da grade horizontal ampliado pela replicação dos pontos nas bordas de forma a permitir a execução independente da advecção na telha. Assim, na execução da advecção, as telhas são divididas entre os *threads* (laço de telhas).

Um parâmetro importante é o tamanho das telhas. A divisão do domínio bidimensional por telhas pequenas implica num maior número de telhas e permite mais paralelismo. Entretanto, a repetição de pontos de grade em telhas que são adjacentes nesse domínio espacial implica na repetição de cálculos, que penaliza o uso de telhas pequenas. Optou-se por testar telhas 2x2, que tem tamanho 3x3 com as bordas, totalizando 9 pontos da grade horizontal, e também telhas 4x4, que tem tamanho 5x5 com as bordas, totalizando 25 pontos.

Os testes de desempenho, apresentados na Figura 1, referem-se ao tempo de execução da advecção de campos escalares 10.800 vezes, correspondentes a uma simulação de 24 horas com uma grade com 5 km de resolução, mas para domínios diferentes, variando de grades horizontais a partir de 10x10 pontos até 100x100 pontos com incrementos de 10x10 pontos. Cada ponto da grade horizontal inclui a coluna da atmosfera com os correspondentes níveis verticais, em número fixo de 42. Os resultados de desempenho correspondem a valores médios de 5 execuções num nó XE6 de um sistema Cray, sendo observados coeficientes de variação inferiores a 2% na versão de laços e 5% para a versão por telhas.

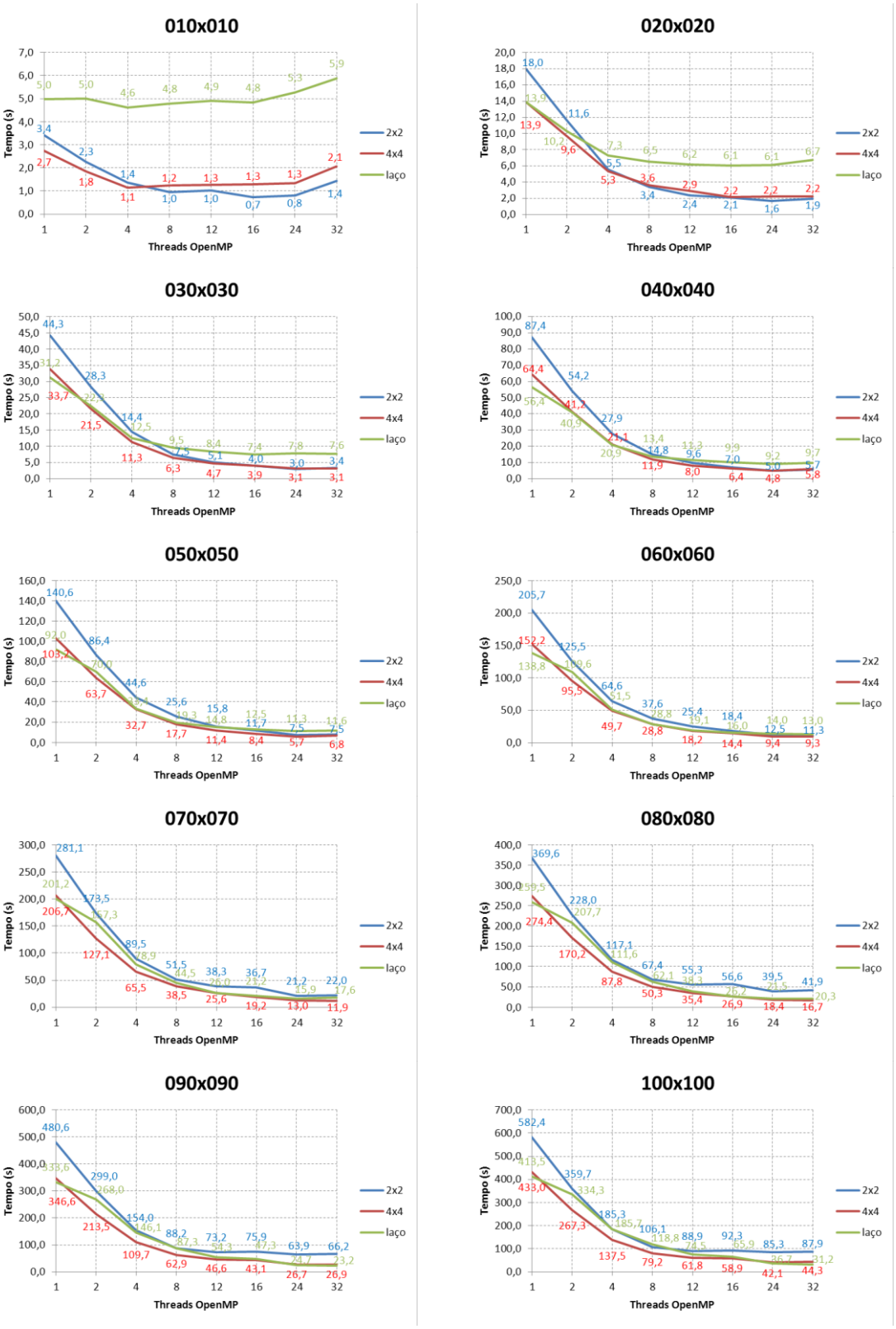

Figura 1. Tempo de execução da advecção em função do numero de *threads* OpenMP para as versões paralelizadas por telhas e por laços para diferentes tamanhos de grade.

Os tempos de execução da Figura 1 nas codificações por laços e por telhas são expressos em função do número de *threads* OpenMP para os diferentes tamanhos de grade, sendo consideradas telhas 2x2 e 4x4. Na versão codificada por laços, o tempo de execução com uma única *thread* tende a aumentar com o tamanho da grade, mas sempre menos que quadraticamente. Com relação ao desempenho paralelo, a versão por laços apresenta escalabilidade até 24 *threads* e, em alguns casos, até 32 *threads*, exceto para a grade 10x10, provavelmente devido à sua baixa granularidade.

Na versão codificada por telhas, observa-se um desempenho similar à versão por laços para as telhas 4x4, com ligeira vantagem para esta última. Entretanto, a versão com telhas 2x2 apresentou tempos maiores, principalmente para números de *threads* baixos, o que pode ser explicado pela repetição de cálculos relativos a pontos de grade nas bordas das telhas, que é relativamente maior do que nas telhas 4x4. Esse *overhead* tende a diluir-se para grades maiores e execuções com maior número de *threads*. Entretanto, nas grades 10x10 e 20x20, as telhas 2x2 tiveram o melhor desempenho a partir de 8 *threads*.

Em termos de escalabilidade, sempre tomando-se por referência a execução com um único *thread*, a versão codificada por laços obteve eficiências pouco acima de 50% nos melhores casos, enquanto que a versão com telhas 4x4, eficiências variando entre 40% e 50%. Ainda para as telhas 4x4, houve casos piores, como por exemplo, grade 100x100 e 32 *threads*, em que a eficiência foi de 30%, ou então, melhores, como para a grade 70x70 com 4 *threads*, em que foi de 80%.

4. Desempenho paralelo com OpenACC

A versão codificada em OpenACC foi derivada da versão OpenMP paralelizada por laços, exposta na sessão anterior, com a chave de compilação para OpenMP desabilitada. Optou-se por não utilizar telhas em função da menor granularidade resultante, a qual prejudicaria seu desempenho na execução em GPGPU. Utilizou-se a mesma metodologia e configuração dos experimentos em OpenMP, ou seja, 10.800 iterações da advecção, grades variando de 10x10 até 100x100 e resultados de desempenho que correspondem a valores médios de 5 execuções, porém num nó XK7 de um sistema Cray, sendo observados coeficientes de variação inferiores a 1%. O código OpenACC foi gerado com o compilador da *suite* CCE (*Cray Compiler Environment*).

A otimização de desempenho foi progressiva, podendo ser descrita em 5 etapas. A Figura 2 permite comparar os tempos de execução para a grade 40x40 relativos à versão OpenACC ao longo dessas etapas com o tempo de execução “sequencial”, que corresponde à versão OpenMP por laços executada por um único *thread*. Os tempos de execução nas demais grades são apresentados adiante, no final desta sessão (Figura 4).

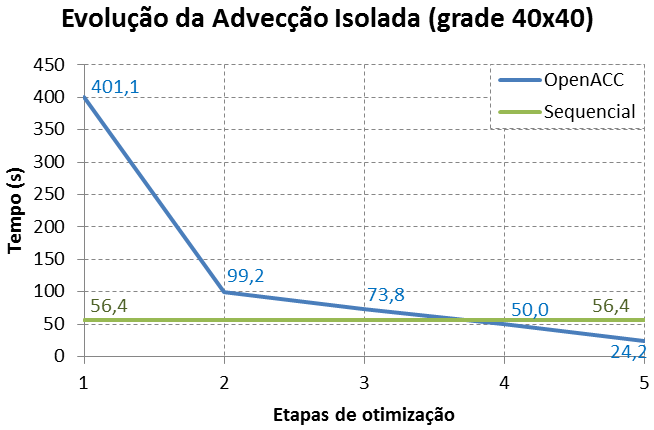

Figura 2. Tempos de execução da advecção isolada com grade 40x40 para as versões OpenACC de cada etapa de otimização, comparados ao tempo de execução da versão OpenMP com um thread.

A 1ª etapa de otimização resultou numa versão em que são definidas regiões paralelas com chamadas a *kernels* a serem executados na GPGPUs para cada um dos 13 aninhamentos de laços da rotina *advectc*, utilizando a diretiva *acc loop* no laço mais externo. Assim, o laço de iterações da advecção, bem como a chamada da *advectc*, são executados pela CPU, enquanto os aninhamentos internos a esta rotina são executados pela GPGPU. Todos os dados da *advectc* residem na memória acessada pela CPU, sendo utilizada a cláusula *data_copy* em cada região paralela, ou seja, o compilador seleciona quais variáveis serão copiadas da CPU para a GPGPU e vice-versa. Consequentemente, nenhuma variável persiste na memória da GPGPU entre execuções sucessivas de *kernels* e as cópias de dados entre a CPU e a GPGPU requerem sincronização na entrada e na saída de cada região paralela, Resulta o tempo de execução inaceitável dessa versão OpenACC, de 401,1 segundos face aos 56,4 segundos da execução sequencial (1 *thread*) da advecção isolada codificada por laços. O *profiler* CrayPat permite constatar que o baixo desempenho decorre do tempo gasto na cópia de dados entre as memórias da CPU e da GPGPU.

A 2ª etapa de otimização consistiu em fundir todas as regiões paralelas dentro da rotina *advectc* numa única região, de forma a diminuir o *overhead* das cópias de dados entre CPU e GPGPU, discriminando variáveis de entrada (atributo *copyin*) e saída (atributo *copyout*), sem no entanto discriminar as variáveis locais à GPGPU. Assim os dados requeridos são copiados da CPU para a GPGPU no início de cada execução de *advectc* e em sentido contrário ao seu término. Houve uma redução de tempo da ordem de 4 vezes, resultando no tempo de execução de 99,2 segundos, que ainda é substancialmente maior que o tempo sequencial de 56,4 segundos.

A 3ª etapa de otimização continuou reduzindo as cópias de dados entre CPU e GPGPU, explicitando ao compilador as variáveis referenciadas em uma execução da rotina *advectc* que estão residentes na memória da GPGPU desde a iteração anterior. Essa otimização foi implementada por meio do *inlining* dessa rotina e do uso do atributo *present* para as variáveis em questão. Além disso, optou-se por definir a região paralela externamente à rotina *advectc*, ou seja, englobando o laço das iterações da advecção. O tempo de execução caiu para 73,8 segundos, sendo ainda dominado pelas transferências de dados entre CPU e GPGPU.

A 4ª etapa de otimização utilizou o mesmo *profiler* CrayPat e informações obtidas por traçadores de eventos na GPGPU (*CRAY_ACC_DEBUG*) para constatar que ainda havia variáveis utilizadas unicamente na região paralela, mas que eram copiadas desnecessariamente para a CPU. Essas variáveis são utilizadas para cálculos intermediários em "*scratch areas*", e foram declaradas com o atributo *present_or_create*, de forma a eliminar essas cópias. Obteve-se um tempo de execução de 50 segundos, menor que a versão sequencial, mas ainda penalizado pela transferência de dados entre a CPU e GPGPU.

A 5ª etapa de otimização permitiu identificar cópias de dados desnecessárias entre a CPU e a GPGPU devidas a um tipo derivado da linguagem Fortran denominado *scalar_table* utilizado no código original, composto por um vetor de ponteiros para os valores presentes e os valores das tendências temporais de cada campo escalar. Esse vetor é dimensionado pelo número de campos escalares, que pode variar entre execuções do BRAMS em função das opções do usuário. Esse tipo derivado permite que se use um único código da rotina *advectc* mesmo que o número de campos escalares varie. Ocorre que tipos derivados (e ponteiros em geral) ainda não são suportados pelo padrão OpenACC atual, versão 2.0 de junho de 2013 (OpenACC Working Group and others, 2013). Alguns autores como (Beyer, Oehmke, & Sandoval, 2014) consideram que essa é a maior limitação para a popularização de OpenACC. Assim, o compilador utilizado neste trabalho copia todos os campos apontados nesse vetor da memória da CPU para a memória da GPGPU na entrada da região paralela e em sentido contrário, em sua saída. Essa restrição foi contornada pela substituição do tipo derivado por dois *arrays* tetradimensionais, um contendo os valores presentes dos campos escalares tridimensionais e o outro contendo suas tendências temporais. A quarta dimensão define a quantidade de campos escalares. Obteve-se uma redução significativa do tempo de execução, que caiu para 24,2 segundos, limitando as transferências de dados entre CPU e GPGPU ao início e ao final da região paralela.

Essa otimização permitiu explorar a opção de compilação *auto_async_kernel* do compilador OpenACC, em vez da opção *default auto_asyn_none* do compilador OpenACC (Cray CCE). A nova opção gera um código em que a CPU pode continuar a executar seu fluxo de instruções assincronamente em relação à GPGPU após efetuar uma chamada para execução de *kernels* nesta última. Além desta opção, há ainda uma terceira, *auto_asyn_all*, que permite também que as transferências de dados entre CPU e GPGPU sejam feitas assincronamente. O uso da opção *auto_async_kernel* não otimizava o desempenho das versões das etapas anteriores, mas permitiu reduzir significativamente o tempo de execução da versão desta etapa, de 24,2 para 13,7 segundos, conforme se pode observar na Figura 3, a qual também mostra que essa opção praticamente não fazia diferença na execução das versões das etapas anteriores.

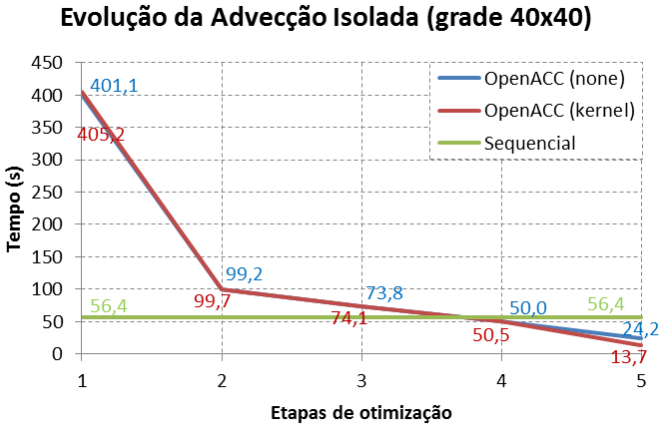

Figura 3. Tempos de execução das versões OpenACC com opções de compilação *async_kernel* e *async_none* para cada etapa de otimização, comparados ao tempo de execução sequencial.

Finalmente, sabe-se que um compilador OpenACC pode atribuir automaticamente o número de *gangs* (parâmetro *num_gangs*) e o número de *threads* de cada *gang* (parâmetro *vector_length*), mas é possível especificar manualmente estes parâmetros em função da arquitetura da placa. O modelo de GPGPU do nó XK7 tem 13 multiprocessadores, cada um com 192 núcleos, sendo que cada multiprocessador pode executar *threads* de diferentes *gangs*, mas cada *gang* somente pode ser executada por um mesmo multiprocessador. Existe ainda a limitação do *warp size*, que divide a execução de cada *gang* em rodadas de 32 *threads*. Constatou-se que a escolha dos parâmetros *num_gangs* igual a 3900 (múltiplo de 13) e *vector_length* igual a 64 (múltiplo do *warp size*) permitiu uma pequena melhora de desempenho (1,8%) em relação à escolha automática pelo compilador OpenACC, que definia um número de *gangs* de 128, 1444 ou 1600 conforme o laço considerado, mas sempre com 128 *threads* por *gang*.

A versão OpenACC da 5ª etapa compilada com a opção *auto_async_kernel* foi testada para as demais grades, conforme apresentado na Figura 4, que compara os tempos da execução dessa versão OpenACC com os tempos da execução da versão OpenMP paralelizada por laços em função do número de *threads* OpenMP para todas as grades consideradas. Constatou-se que o tempo de execução da versão OpenACC é cada vez mais competitivo em relação ao tempo de execução da versão OpenMP com o aumento do tamanho da grade, pois o número de *threads* necessário para que a versão OpenMP seja mais rápida que a versão OpenACC aumenta com o tamanho da grade. Assim, considerando-se a grade 100x100, a versão OpenMP tem melhor desempenho que a OpenACC apenas acima de 16 *threads*.

A título de exemplo, apresenta-se na Figura 5 um trecho do código desta ultima versão, onde aparecem diretivas OpenMP e OpenACC relativas a um aninhamento.

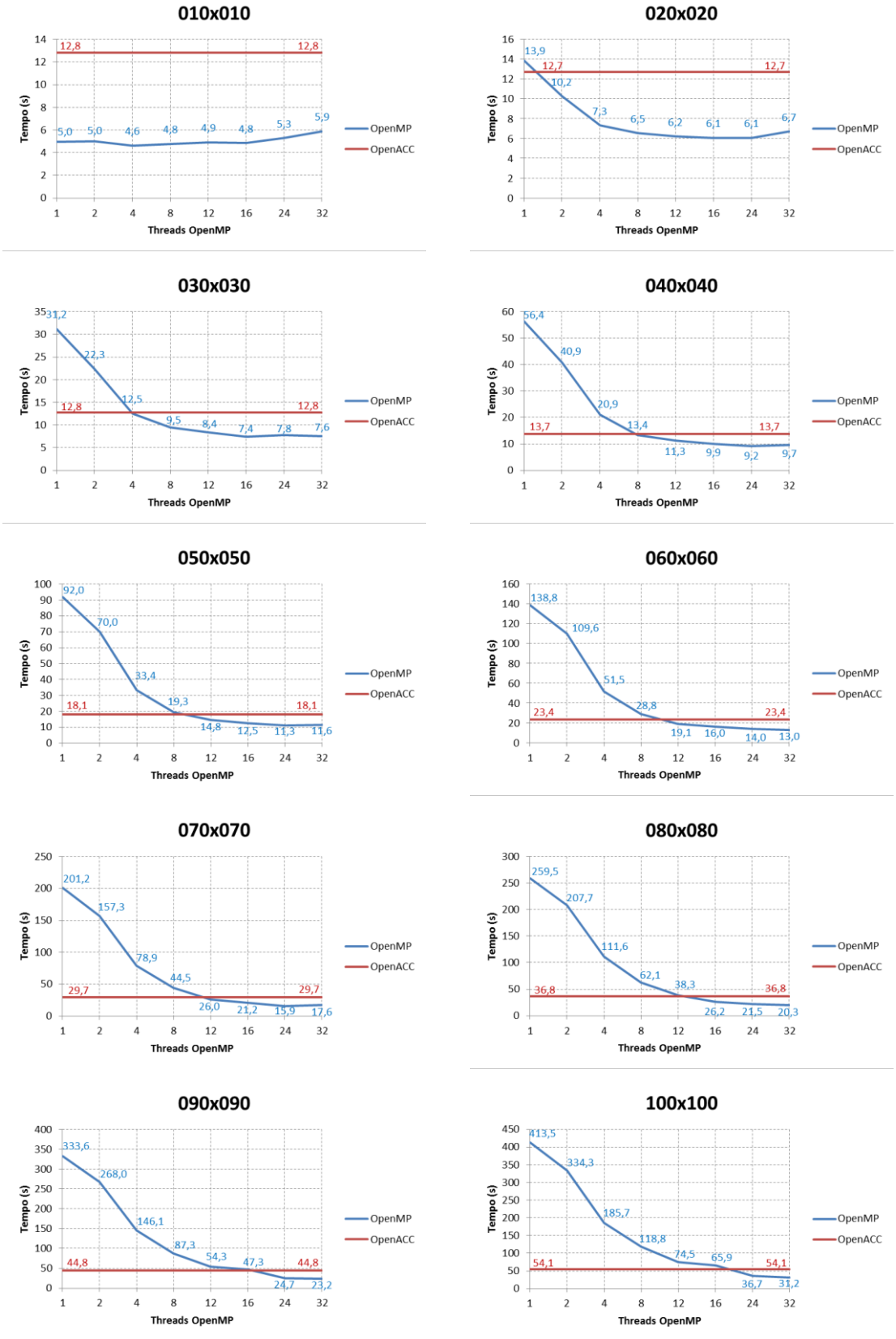

Figura 4. Comparação dos tempos de processamento da melhor versão OpenACC com a versão OpenMP codificada por laços para diferentes números de *threads*.

```
!$acc parallel present(vt3da,vt3db,vt3dc,uc,up,vc,vp,wc,wp), num_gangs(4096), vector_length(64)
!$acc loop private(j,i,k), collapse(2)
!$omp do private(j,i,k), collapse(2)
do j = ja-1, jz+1
   do i = ia-1, iz
      do k = 1,mzp
         vt3da(k,i,j) = (up(k,i,j) + uc(k,i,j)) * dtlto2
      end do
   end do
end do
!$omp end do nowait
!$acc end loop
    .
    .
    .
!$acc end parallel
```

Figura 5. Trecho de código da 5ª etapa da versão OpenACC ilustrando o uso de diretivas OpenMP e OpenACC relativas a um aninhamento.

5. Conclusões

Este trabalho demonstrou que é possível gerar um único código para execução paralela em arquiteturas *multi-core* e *many-core*, por meio da inserção de diretivas de paralelização OpenMP e OpenACC, respectivamente, obtendo-se um desempenho razoável em ambas arquiteturas. Entretanto, a obtenção de bom desempenho na versão OpenACC depende fortemente da minimização da transferência de dados entre CPU e GPGPU, sendo essa otimização condicionada a limitações dos compiladores OpenACC, tais como não suportar tipos derivados ou ponteiros. O código escolhido foi uma parte da dinâmica do modelo BRAMS, a advecção, e sua codificação em OpenACC é a primeira já realizada da dinâmica de um modelo regional.

Referências

Beyer, J., Oehmke, D., & Sandoval, J. (2014). Transferring user-defined types in OpenACC. *CUG - Cray User Group*, 15.

Brazilian developments on the Regional Atmospheric Modelling System. (2015). Acesso em 11 de Março de 2015, disponível em BRAMS: http://brams.cptec.inpe.br

Centro de Previsão de Tempo e Estudos Climáticos. (2015). Acesso em 01 de 02 de 2015, disponível em CPTEC: http://www.cptec.inpe.br/

Chapman, B., Jost, G., & Van Der Pas, R. (2008). *Using OpenMP: portable shared memory parallel programming* (Vol. 10). MIT press.

Computação de Alta Performance. (2014). Acesso em 05 de Janeiro de 2014, disponível em NVIDIA: http://www.nvidia.com.br/object/tesla-supercomputing-solutions-br.html

Fazenda, A. L., Panetta, J., Katsurayama, D. M., Rodrigues, L. F., Motta, L., & Navaux, P. (2011). Challenges and solutions to improve the scalability of an operational regional meteorological forecasting model. *International Journal of High Performance Systems Architecture, 3*(2), 87-97.

Freitas, S. R., Longo, K. M., Dias, M. S., Chatfield, R., Dias, P. S., Artaxo, P., . . . others. (2007). The Coupled Aerosol and Tracer Transport model to the Brazilian developments on the Regional Atmospheric Modeling System (CATT-BRAMS) Part 1: Model description and evaluation. *Atmospheric Chemistry and Physics Discussions, 7*(3), 8525-8569.

Lee, S., Min, S.-J., & Eigenmann, R. (2009). OpenMP to GPGPU: A compiler framework for automatic translation and optimization. *ACM Sigplan Notices, 44*(4), 101-110.

Longo, K. M., Freitas, S. R., Pirre, M., Marécal, V., Rodrigues, L. F., Panetta, J., . . . others. (2013). The chemistry CATT-BRAMS model (CCATT-BRAMS 4.5): a regional atmospheric model system for integrated air quality and weather forecasting and research. *Model Dev. Discuss, 6*, 1173-1222.

OpenACC Directives for Accelerators. (2015). Acesso em 17 de Março de 2015, disponível em OpenACC: http://www.openacc-standard.org/

OpenACC Working Group and others. (2013). *The OpenACC Application Programming Interface*. Acesso em 05 de Janeiro de 2014, disponível em OpenACC Home: http://www.openacc.org/sites/default/files/OpenACC%202%200.pdf

OpenCL - The open standard for parallel programming of heterogeneous systems. (2015). Acesso em 18 de 03 de 2015, disponível em Khronos Group: https://www.khronos.org/opencl/

Panetta, J. (2012). *Historico de Desenvolvimento do CCATT-BRAMS*. Acesso em 01 de Fevereiro de 2015, disponível em https://projetos.cptec.inpe.br/attachments/237/JairoPanetta-I.pdf

Rodrigues, E. R., Navaux, P. O., Panetta, J., Fazenda, A. L., Mendes, C., & Kale, L. V. (2010). A comparative analysis of load balancing algorithms applied to a weather forecast model. In: *Computer Architecture and High Performance Computing (SBAC-PAD), 2010 22nd International Symposium on* (pp. 71-78). IEEE.

Tremback, C. J., Powell, J., Cotton, W. R., & Pielke, R. A. (1987). The forward-in-time upstream advection scheme: Extension to higher orders. *Monthly Weather Review, 115*(2), 540-555.