Melhorando as Operações de E/S do Algoritmo RTM *

Pablo J. Pavan[1], **Matheus S. Serpa**[1], **Edson L. Padoin**[2],
Alexandre Carissimi[1], **Philippe O. A. Navaux**[1]

[1]Universidade Federal do Rio Grande do Sul (UFRGS) - Porto Alegre - RS - Brasil

[2]Universidade Reg. do Noroeste do Estado do Rio G. do Sul (UNIJUI) - Ijuí - RS - Brasil

{pablo.pavan,msserpa,asc,navaux}@inf.ufrgs.br, padoin@unijui.edu.br

Resumo. *Operações de leitura e escrita limitam o desempenho do algoritmo Reverse Time Migration (RTM). Desta forma, otimizar estas operações de E/S permite que simulações de maior escala sejam executadas em ambientes com recursos limitados. Os resultados preliminares mostram que a utilização de checkpoints e o aumento do tamanho da requisição reduzem o tempo de execução e o espaço de armazenamento em até* $17,3\%$ *e* $89,5\%$ *respectivamente.*

1. Introdução

As aplicações de geofísica que tratam da pesquisa de petróleo demandam grande capacidade computacional e geralmente manipulam grandes volume de dados. As imagens geradas nas simulações de levantamento sísmico do solo do fundo dos oceanos são de alta qualidade e demandam de grandes capacidade de armazenamento. Entretanto, as operações de entrada e saída (E/S) que precisam ser executadas muitas vezes geram um grande gargalo de desempenho para as aplicações.

Algumas aplicações de geofísica adotam como algoritmo o RTM (Reverse Time Migration) para calcular a condição final das imagens. Sendo um algoritmo bem consolidado, introduzido no final dos anos 1970, possui elevados requisitos tanto em termos de computação quanto de memória. Por outro lado, dado o aumento da capacidade computacional das máquinas atuais, esse algoritmo vem tornando-se mais viável. O núcleo do algoritmo é a correlação cruzada de dois campos de onda no mesmo nível de tempo, um computado adiantando-se no tempo, o outro computado retrocedendo no tempo. A propagação da onda *forward* geralmente é realizada primeiro, sendo que todo o histórico de tempo desta propagação deve ser disponibilizado durante a propagação da onda *backward* para calcular a condição final da imagem.

Essas disponibilidade do histórico de tempo da propagação, mostra-se como um problema de E/S - uma vez que estes dados do histórico podem demandar TeraBytes de espaço [Dussaud et al. 2008], obrigando a aplicação realizar a gravação e posteriormente, leitura desse histórico. Assim, este trabalho aplica otimizações que melhoram as operações de E/S, melhorando o desempenho de uma aplicação RTM disponibilizada pela empresa Petrobras.

2. Trabalhos Relacionados

Considerando trabalhos que buscam a paralelização de aplicações RTM, o trabalho de [Bartana et al. 2015] busca demostrar uma implementação paralela das derivadas uti-

*Trabalho desenvolvido com recursos do edital MCTIC/CNPq - Universal 28/2018 sob número 436339/2018-8 e parcialmente financiado pela Petrobras sobre o projeto 2016/00133-9.

lizadas no RTM para a utilização em GPU. Usando operadores recursivos de dispersão mínima, os resultados mostram que o uso dos operadores recursivos aumentam o custo computacional. Porém com o uso de GPUs isso pode ser reduzido. No trabalho de [Obregon et al. 2017] busca dividir os dados a serem calculados no RTM em diferentes GPUs utilizando MPI, os resultados mostram que o fator de *speedup* foi de 5:61 quando comparado três GPUs para somente uma. Já no trabalho de [Qawasmeh et al. 2017], utilizou-se OpenACC para paralelizar o modelo sísmico para o uso em GPUs, seus ganhos chegam a 10x vezes de *speedup* utilizando somente uma GPU.

Tendo em vista trabalhos relacionados que buscam melhorar o uso de E/S, os trabalhos de [Symes 2007, Dussaud et al. 2008, Clapp et al. 2009] propõem o uso de *checkpoints* de E/S durante a execução do RTM, fazendo que ocorra somente gravações e leituras a cada N iterações. Isso reduz a qualidade da imagem gerada pelo RTM, assim necessita-se encontrar um ponto de equilíbrio, onde se reduz a E/S e mantém-se uma imagem fiel. Tendo em vista que não há novos trabalhos com foco em E/S no RTM, buscamos pesquisas que investigam E/S para HPC em outros tipos de aplicações.

Alguns trabalhos relacionados propõem técnicas similares para agregação de pequenas requisições, uma vez que este é um problema comum entre aplicações científicas. No entanto, em relação a outras otimizações, essas abordagens diferem muito das nossas. No trabalho de [Rettenberger and Bader 2015] busca otimizar simulações sísmicas trabalhando em grandes malhas não estruturadas, e no trabalho de [Yu et al. 2012] trabalham com aplicações de cosmologia que usam árvores de refinamento adaptativo. Ambos consideram uma representação de dados diferente do que é usado pelo algoritmo RTM utilizado, portanto suas técnicas não podem ser aplicadas. Além disso, eles também mudam os formatos entradas e saída (incluído o número de arquivos), enquanto para nós o foco é o melhoramento do desempenho de E/S mantendo o formato original.

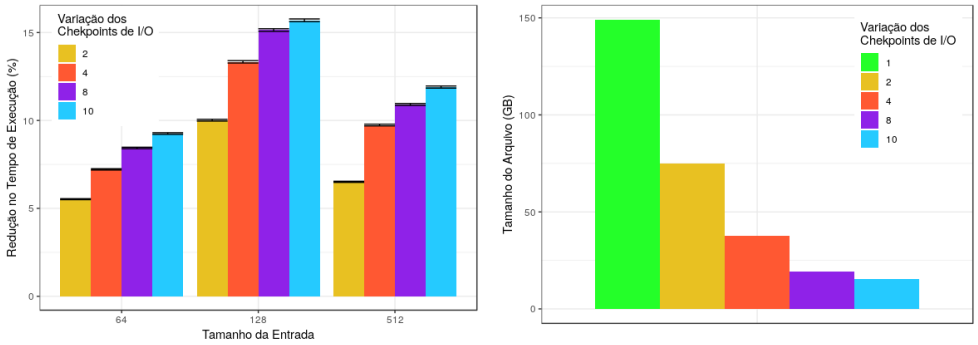
3. Metodologia Experimental

Os experimentos foram realizados em uma plataforma composta de dois processadores Intel Xeon modelo E5-2699 v4 de microarquitetura *Broadwell*. Cada processador possui 22 cores com 2 SMT/core, que totaliza 88 *threads* de 2.20 GHz de frequência. Este equipamento também possui 256 GB memória RAM DDR4 de frequência 1600 MHz. Para os testes, utilizou-se o sistema operacional Ubuntu 18.04 com versão de kernel $4.15.0$-24. A aplicação foi executada com g++ na versão 7.3 e paralelizada com OpenMP. Para os teste de E/S forma foi selecionado um dispositivo SSD fabricado pela empresa Samsung modelo 850 EVO com capacidade de 960 GB conectado na interface SATA3. O sistema de arquivo utilizado foi o *ext4*.

Nos nossos testes foram utilizados três tamanhos, 64^3, 128^3 e 512^3, variando-se as ocorrências dos *checkpoints* em 1, 2, 4, 8 e 10. Posteriormente buscamos utilizar o tamanho alvo da empresa Petrobras, para isso utilizou-se como entrada as dimensões de $1024 \times 1024 \times 512$, onde o tamanho em z é a metade do tamanho de x e y para simular a profundidade do oceano. Em todos os experimentos foi usado um maximo de iterações igual à 200 e somente uma imagem. Todas as execuções utilizaram os 88 cores disponíveis pelo ambiente de execução. Cada um dos testes realizados neste trabalho foram repetidos 10 vezes, os resultados apresentados são 99% de confiança estatística para uma *Student's t-distribution*.

4. Resultados Premilinares

Nesta seção são apresentados as avaliações de desempenho de E/S da aplicação na plataforma experimental apresentada, os primeiros resultados mostram os ganhos de desempenho quando utilizamos a otimização de *checkpoints* . Na Figura 1(a) observa-se uma redução do tempo de execução, a medida que é aumentado a distância entre os *checkpoints*. Quando aumentado de uma iteração para 10 iterações, consegue-se reduzir o tempo de execução de $41,8$ segundos para 38 segundos, o que representa um redução de $9,2\%$ para o tamanho 64. Nos testes com tamanho de 128, o tempo foi reduzido de $106,2$ para $89,5$ segundos, o que representa $15,6\%$. Por fim, utilizando tamanho de 512 a redução foi de $11,9\%$, uma redução de $1835,7$ para $1617,2$ segundos.



(a) Tempo de execução com os diferentes tamanhos e variações de *checkpoints*

(b) Tamanho dos dados intermediarios gerados durante a execução para o tamanho 512

Figura 1. Resultados dos experimentos utilizando *checkpoints*

Na Figura 1(b) são apresentados os tamanhos dos arquivos gerados para o tamanho de entrada 512. Para o tamanho do arquivo gerado a porcentagem de redução se mantém a mesma, sendo que para todos os casos onde variamos o *checkpoint* de 1 para 10 a redução foi de $89,5\%$. Já em relação o crescimento do arquivo, para o tamanho de 64 e *checkpoints* a cada 1 iteração, o arquivo tem o tamanho de $1,8$ GigaBytes, se comparado com o tamanho 512 com o mesmo *checkpoint* o tamanho foi de $149,1$ GigaBytes, isso representa um aumento de $7932,8\%$.

Nessa aplicação, o mesmo número de dados é escrito e lido, o que ocasiona o mesmo número de operações de E/S. A partir do número de operações, pode-se encontrar o tamanho das requisições que a aplicação fez. Utiliza-se o tamanho final de Bytes escritos e lidos dividido pelo número de operações. Nessa aplicação o tamanho das requisições das operações são de $273,3$ Bytes, assim realizou-se uma agregação destas requisições com o objetivo de diminuir o número de operações, onde agora a cada chamada de escrita e leitura manipula $2,7$ Gigabytes, reduzindo para 40 operações ao todo.

Com objetivo de avaliar essa otimização, utilizou-se as configurações para o tamanho alvo da Petrobras ($1024 \times 1024 \times 512$), com *checkpoints* a cada 10 iterações. Analisando o resultado encontrado com a utilização da agregação das requisições, obtivemos uma redução de $17,3\%$ no tempo final de execução. Esta redução no tempo de execução com a agregação das requisições acontece porque existem custos fixos associados a cada requisição, então é mais eficiente realizar um número menor de grandes requisições que um número maior de pequenas requisições [Boito et al. 2011].

5. Considerações Finais

Esse trabalho propôs otimizações das operações de E/S para uma aplicação real de geofísica disponibilizada pela empresa Petrobras. Quando aplicado *checkpoints*, conseguimos reduzir o tempo de execução em até $15,6\%$ e o tamanho do arquivo em até $89,5\%$. Originalmente o arquivo gerado era de $526,3$ GB e passou para 55 GB. Desta forma, o uso de *checkpoints* permite que simulações de maior escala possam ser executadas com uma menor limitação de espaço de armazenamento. Com a utilização de agregação das requisições, juntamente com o uso de *checkpoints*, o tempo total de execução foi reduzido em até $17,3\%$.

As pesquisas atuais, em conjunto com a empresa, concentram-se na busca de alternativas para otimização das operações de E/S dessa aplicação. Em trabalhos futuros pretende-se estender as melhorias já implementadas e avaliar a utilização de *buffers* para armazenar algumas partes do histórico da propagação em memória, reduzindo a necessidade da busca de dados no sistema de arquivo.

Referências

Bartana, A., Kosloff, D., Warnell, B., Connor, C., Codd, J., Kessler, D., Micikevicius, P., Mckercher, T., Wang, P., and Holzhauer, P. (2015). Gpu implementation of minimal dispersion recursive operators for reverse time migration. In *SEG Technical Program Expanded Abstracts 2015*, pages 4116–4120.

Boito, F. Z., Kassick, R. V., and Navaux, P. O. (2011). The impact of applications' i/o strategies on the performance of the lustre parallel file system. *International Journal of High Performance Systems Architecture*, 3(2-3):122–136.

Clapp, R. G. et al. (2009). Reverse time migration with random boundaries. In *2009 SEG Annual Meeting*.

Dussaud, E., Symes, W. W., Williamson, P., Lemaistre, L., Singer, P., Denel, B., and Cherrett, A. (2008). Computational strategies for reverse-time migration. In *SEG Technical Program Expanded Abstracts 2008*, pages 2267–2271.

Obregon, I., Salamanca, W., and Ramirez, A. B. (2017). Reverse time migration on a gpu cluster using the seismic data parallelism strategy. In *15th International Congress of the Brazilian Geophysical Society & EXPOGEF, RJ, Brazil*, pages 353–357.

Qawasmeh, A., Hugues, M. R., Calandra, H., and Chapman, B. M. (2017). Performance portability in reverse time migration and seismic modelling via openacc. *The International Journal of High Performance Computing Applications*, 31(5):422–440.

Rettenberger, S. and Bader, M. (2015). Optimizing i/o for petascale seismic simulations on unstructured meshes. In *2015 IEEE International Conference on Cluster Computing (CLUSTER)*, pages 314–317.

Symes, W. W. (2007). Reverse time migration with optimal checkpointing. *Geophysics*, 72(5):SM213–SM221.

Yu, Y., Rudd, D. H., Lan, Z., Gnedin, N. Y., Kravtsov, A., and Wu, J. (2012). Improving parallel io performance of cell-based amr cosmology applications. In *Parallel & Distributed Processing Symposium (IPDPS), 2012 IEEE 26th International*, pages 933–944.