Otimização de Aplicações Paralelas em Aceleradores Vetoriais NEC SX-Aurora*

Félix D. P. Michels[1], **Matheus S. Serpa**[1], **Danilo Carastan-Santos**[1]
Lucas M. Schnorr[1], **Philippe O. A. Navaux**[1]

[1] Instituto de Informática – Universidade Federal do Rio Grande do Sul (UFRGS)
Caixa Postal 15.064 – 91.501-970, Porto Alegre – RS – Brasil
Programa de Pós-Graduação em Computação

{felix.junior, msserpa, danilo.csantos, schnorr, navaux}@inf.ufrgs.br

Abstract. *By design, Vector processors favor an instruction being executed on multiple data, increasing performance in real numerical applications, such as simulations of fluid mechanics and wave propagation. The present work addresses a performance analysis of the SX-Aurora TSUBASA architecture. For this task, we used the NAS benchmark and a real wave propagation application, which is essential for the oil and prospecting industry. By applying simple optimization techniques such as loop unrolling and inlining, we achieved performance improvements with the SX-Aurora TSUBASA up to* $7,8\times$ *with the NAS benchmark, and up to* $1,9\times$ *with the real application, when compared to the performance of the original versions of the applications.*

Resumo. *Aceleradores vetoriais, por conta do modo que foram projetados, favorecem a execução de um mesmo conjunto de instruções sobre múltiplos dados, aumentando o desempenho de aplicações reais, como previsão do tempo e prospecção de petróleo. Neste trabalho, avaliamos o desempenho de aplicações paralelas executadas na arquitetura NEC SX-Aurora. Para tanto, foram utilizados como estudo de caso, o benchmark NAS e uma aplicação real de migração sísmica, utilizada pela indústria de petróleo e gás. Resultados experimentais mostraram que as técnicas de otimização loop unrolling e inlining, melhoraram o desempenho do benchmark NAS em até* $7,8\times$ *e da aplicação real de migração sísmica em até* $1,9\times$*, em comparação com o desempenho das versões originais.*

1. Introdução

A computação de alto desempenho é indispensável para diversas indústrias, setores comerciais e pesquisas atuais. Ela provê uma abundância de benefícios os quais vão desde facilitar a análise de dados, até possibilitar novas simulações e modelagens [Ezell and Atkinson 2016]. Assim, avanços na área de CAD (Computação de Alto Desempenho) são de extrema importância. Aumentando as dimensões e diminuindo o tempo de execução podemos, então, beneficiar diversas áreas da sociedade.

*Este trabalho foi parcialmente financiado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Código de Financiamento 001, pelo projeto Petrobras (2016/00133-9, 2018/00263-5) e pelo projeto "GREEN-CLOUD: Computação em Cloud com Computação Sustentável" (#16/2551-0000 488-9), da FAPERGS e do CNPq, programa PRONEX 12/2014.

O desempenho de aplicações depende em grande parte da arquitetura utilizada e da maneira que o programa foi codificado para executar nela. Atualmente, para fins de aceleração, são utilizadas diferentes arquiteturas, como *multicore*, *manycore*, aceleradores específicos e GPUs (*Graphics Processing Units*).

O número grande de arquiteturas, ao mesmo tempo que é atrativo e flexível, impõe novos desafios para o programador [Mittal and Vetter 2015]. O aumento da complexidade da arquitetura também aumentará a dificuldade da implementação da aplicação. Vale ressaltar a ideia de que existem diversos gargalos comuns, como os encontrados no subsistema de memória, os quais incluem poluição de *cache*, *thrashing*, entre outros.

Além do uso de novas arquiteturas, diversas técnicas são empregadas para impulsionar o desempenho, sendo algumas exclusivas de certas arquiteturas. Por exemplo, a vetorização permite que uma instrução atue sobre múltiplos dados, porém nem todas as arquiteturas implementam suporte para isso. Outras otimizações incluem aumentar a taxa de acerto da memória *cache*, sendo essa possível em diferentes arquiteturas.

Nesse sentido, a empresa NEC Corporation, lançou uma nova arquitetura, um processador vetorial intitulado SX-Aurora. Esse processador possui 8 núcleos de processamento a 1,6 GHz e 3 níveis de memória cache [Komatsu et al. 2018]. Uma das vantagens dessa arquitetura em relação as outras existentes, é o tamanho das unidades vetoriais da mesma. Além disso, o compilador da NEC toma decisões automaticamente, ou seja, identifica áreas vetorizáveis e gera código para isso. Entretanto, o compilador ainda necessita de ajuda do programador para facilitar a interpretação do código, além de aprimorar a vetorização automática, seguindo diretrizes específicas e, neste caso, utilizar técnicas de otimização como *loop unrolling* e *inlining*.

Neste trabalho, otimizamos o desempenho de aplicações reais e de um conjunto de *benchmarks* utilizando técnicas de *loop unrolling* e *inlining* na nova arquitetura da NEC, a SX-Aurora TSUBASA, procurando melhorar a vetorização automática desempenhada pelo compilador. Mais precisamente, o presente trabalho apresenta as seguintes contribuições:

- Efetuamos uma análise experimental de desempenho do acelerador vetorial NEC SX-Aurora TSUBASA. Utilizamos um *benchmark* de aplicações sintéticas paralelas e uma aplicação real.
- Mostramos que as técnicas de *loop unrolling* e *inlining*, são capazes de melhorar significativamente o desempenho das aplicações, quando executadas na SX-Aurora TSUBASA, em situações em que o ganho de desempenho não se dá de forma automática pelo compilador da mesma. Resultados experimentais evidenciaram ganhos de desempenho de até $7,8\times$, quando comparado a versão original.

O artigo foi organizado da seguinte forma. A seção 2 apresenta e discute trabalhos relacionados. Na seção 3 exibe-se uma multiplicação de matrizes, a qual é aplicada simples técnicas de otimização, como motivação para esse trabalho. A metodologia, fluxo de trabalho e o ambiente de execução são apresentados na seção 4. Já a seção 5 retrata os resultados experimentais do *benchmark* NAS e da modelagem RTM. Por fim, é exposto na seção 6 a conclusão e possíveis trabalhos futuros.

Com o intuito de reprodutibilidade, o material associado com esse trabalho está disponível publicamente em https://gitlab.com/koty25/wscad-2020-sx-aurora1, contendo

os dados, scripts e imagens utilizadas.

2. Trabalhos Relacionados

Existem diversos estudos na área de computação paralela e de alto desempenho. A seguir são ressaltados alguns estudos na área, importantes para o presente trabalho, principalmente no que condiz a técnicas de otimização.

É proposto por Serpa et al. diversas estratégias de otimização para um modelo de propagação de onda em diversas arquiteturas. O estudo mostra que utilizando técnicas como *loop interchange*, vetorização e mapeamento de dados e *threads*, foi possível alcançar um aumento de desempenho de até 8,5 vezes [Serpa et al. 2017].

Técnicas de alinhamento de memória e *cache blocking* são apresentadas por Castro et al., aprimorando o desempenho de uma aplicação de propagação de ondas acústicas [Castro et al. 2016]. Semelhante a Castro et al., com o intuito de alcançar melhor desempenho, neste trabalho será implementado duas técnicas, *loop unrolling* e *inlining*.

Uma visão geral para técnicas de otimização para a memória cache é feita por Kowarschik e Weiß, demonstrando técnicas como: *loop interchange*, *loop fusion*, *loop blocking*, entre outros [Kowarschik and Weiß 2003]. É desenvolvido, no presente trabalho, algumas dessas técnicas, observando seus efeitos em uma arquitetura vetorial.

Jacquelin et al. propõem algoritmos que utilizam a memória cache para arquitetura *multicore* e que minimizam o tempo de acesso de memória, assim, diminuindo a taxa de falha da memória *cache*, bem como o tempo de acesso a memória compartilhada e cache [Jacquelin et al. 2009]. O impacto da taxa de falha da memória *cache* também é explorado no presente artigo, focando agora em uma arquitetura vetorial.

No trabalho de Sherlon et al. são comparados diferentes técnicas de otimização, como *loop unrolling* e *loop tiling*. É demonstrando seu impacto em aplicações paralelas, alcançando desempenhos de até 20 vezes ao original [da Silva et al. 2016]. Neste trabalho pretende-se analisar o impacto de *loop unrolling* e *inlining* em uma arquitetura vetorial, diferente do foco no processador Xeon de [da Silva et al. 2016].

O trabalho de Komatsu et al. evidencia o potencial da arquitetura SX-Aurora TSUBASA, comparando-a a outras arquiteturas, entre elas Intel Xeon Phi 7290, NVIDIA Tesla V100 e SX-Ace. Essa comparação é feita através da execução de aplicações de *benchmark* e duas aplicações reais. Os resultados mostram que a arquitetura da SX-Aurora é capaz de executar eficientemente, até $3,5\times$, além de obter um speedup maior de até $2,8\times$ [Komatsu et al. 2018].

Já Yokokawa et al., demonstra a capacidade da arquitetura SX-Aurora TSUBASA para aplicações de I/O, comparando o sistema de I/O da nova arquitetura com sistemas de I/O normais. A distinta função acelerada de I/O presente na arquitetura, para certos casos, triplica o desempenho em MB por segundo [Yokokawa et al. 2020].

Ao comparar os trabalhos apresentados anteriormente e o presente trabalho, o foco na nova arquitetura SX-Aurora TSUBASA é um diferencial, devido a sua essência de processador vetorial e o seu compilador única, que, por exemplo, promove vetorização automática. Além disso, mesmo nos trabalhos apresentados sobre a nova arquitetura, nenhum deles implementa técnicas simples e conhecidas de otimização, que é o caso das

aplicadas nesse trabalho, as técnicas de *loop unrolling* e *inlining*.

3. Motivação

Com uma nova arquitetura, novos desafios surgem, como sua paralelização e eficiência energética, e novas respostas podem ser desenvolvidas, bem como aplicar soluções tradicionais. Assim com o intuito de testar os limites da nova arquitetura SX-Aurora TSUBASA, um teste de multiplicação de matrizes foi desenvolvido. Esse experimento foi escolhido, pois a multiplicação de matrizes, devido a sua natureza vetorial, é ideal para se utilizar um acelerador vetorial.

Inicialmente, desenvolveu-se uma aplicação de uma multiplicação de matrizes. Em seguida, aplicou-se a técnica de *loop interchange*, a qual visa mudar a ordem dos laços para melhorar o acesso aos dados em memória. Então, adiciona-se a técnica de *loop unrolling*, a qual expande o laço para que ele execute mais de uma operação por iteração. Na próxima etapa adiciona-se a paralelização *OpenMP*, utilizando 8 cores, junto com o *loop interchange*, e em seguida adiciona-se novamente o *loop unrolling*.

O Resultado para uma entrada de uma matriz de *floats* de 2048 X 2048 é apresentado na Figura 1. No eixo y temos o tempo de execução, no eixo x o tamanho da entrada, sendo cada cor das barras relacionada a uma das implementações: vermelha é a original, azul é com apenas *loop interchange*, verde adiciona-se *loop interchange* e *loop unrolling* juntos, roxo é a paralelização com OpenMP e *loop interchange*, e por fim, a laranja engloba paralelização com OpenMP, *loop interchange* e *loop unrolling*.

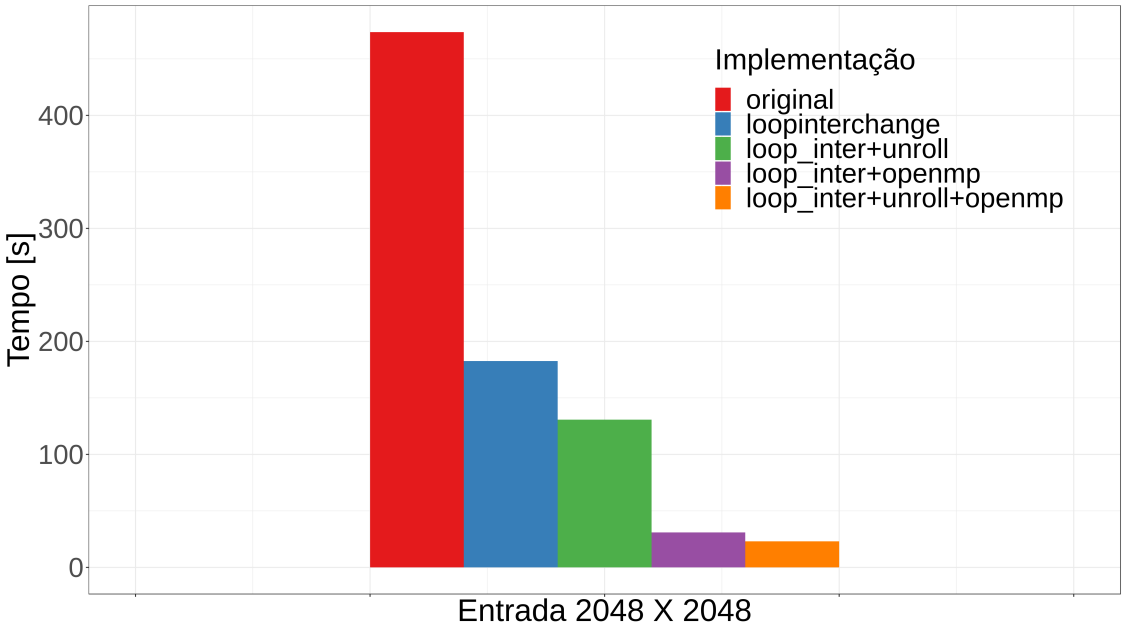

Figura 1. Multiplicação de matrizes

Aqui percebe-se o impacto de cada técnica. Resultando no fim em uma diminuição de aproximadamente 39 vezes. Com esse pequeno exemplo é demonstrado uma grande melhora no uso dos recursos computacionais, podendo assim aplicar tais técnicas em aplicações reais e consequentemente economizar em recursos físicos, como energia e tempo de execução.

4. Metodologia e Ambiente de Execução

O fluxo de trabalho inicial é dividido em: preparações do ambiente e da coleta de dados, execução dos experimentos, coleta de dados, análise, otimização.

Primeiro, a preparação do ambiente e da coleta de dados condiz em criar as rotinas, tanto para a compilação/execução quanto para a visualização dos dados. Então, executa-se no ambiente de trabalho os experimentos selecionados na etapa anterior. A coleta de dados e sua manipulação são feitas utilizando R. Em seguida, é feita uma análise da execução, utilizando os dados de compilação, bem como as métricas da execução. Em seguida, utiliza-se técnicas de otimização já conhecidas para aprimorar o desempenho das aplicações.

Por fim, espera-se obter resultados, das coletas de dados, como taxa de erro da memória *cache*, operações de ponto flutuante por segundo (FLOPS) e taxa de operações vetoriais, e partindo desses dados, otimizar os códigos utilizando as técnicas de *loop unrolling* e *inlining*.

Os experimentos utilizaram o ambiente SX-Aurora TSUBASA utilizando os recursos da infraestrutura PCAD, http://gppd-hpc.inf.ufrgs.br, no INF/UFRGS. Visualizando a Figura 2, percebe-se um ambiente com 8 *cores*, memória global e cache L3, cada core com, memória cache L1 e L2, uma unidade de processamento escalar (SPU) e uma unidade de processamento vetorial (VPU), sendo que cada VPU contém *load buffer*, *store buffer* e 32 *pipeline* paralelo vetorial (VPP) [NEC 2020d]. A tabela 1 mostra as especificações detalhadas da arquitetura, contendo as especificações dos processadores, memórias *cache* e global.

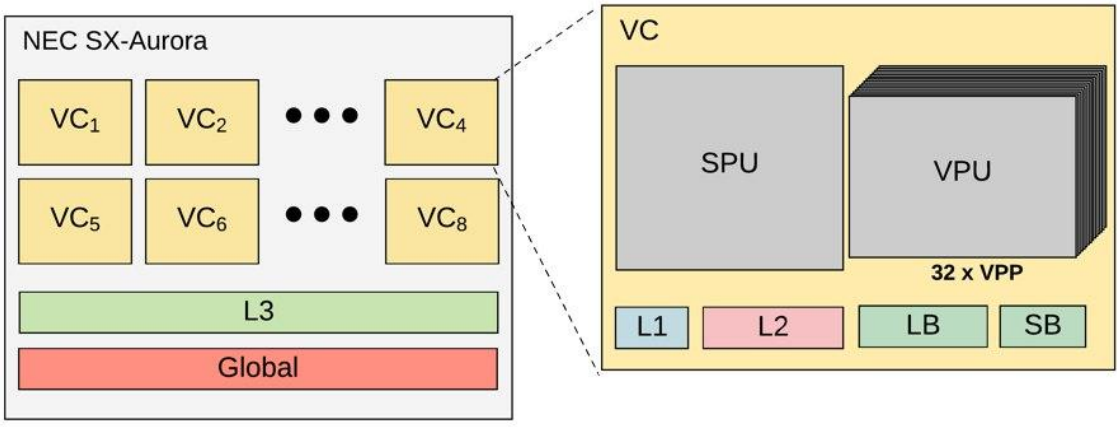

Figura 2. Visão detalhada de cada *core* do acelerador SX-Aurora TSUBASA.

Para a avaliação da SX-Aurora, utilizou-se o *benchmark* NAS [Bailey et al. 1991] e uma aplicação real utilizada pela indústria de petróleo para migração sísmica, denominada Reverse Time Migration (RTM) [Zhou et al. 2018, Fowler et al. 2010, Fletcher et al. 2009]. O benchmark NAS é uma série de programas desenhados especificamente para avaliar o desempenho de computadores paralelos. A modelagem RTM constitui a simulação da propagação de ondas através do tempo, utilizando a equações de acústica e o fato de diferentes camadas geológicas possuem velocidade distintas.

Assim, as aplicações utilizadas para o *benchmark* NAS foram: BT, CG, EP, FT,

Tabela 1. Arquitetura SX-Aurora.

Vector Engine Type 10BE	Processador	8 *cores* @ 1408 MHz
	Microarquitetura	SX-Aurora
	Cache	8 X 32 KB L1I 8 X 32 KB L1D; 8 X 256 KB L2; 8 X 2 MB L3
	Memória	HBM2 48 GB, 900 MHz

IS, MG, SP, UA. A classe de dados de entrada usada é a B. As especificações desses programas se encontram na documentação da NAS [Bailey et al. 1991]. Todos os experimentos, *benchmark* NAS e modelagem RTM, foram executados 10 vezes, tomando, então, a média e o erro padrão.

Todas as aplicações receberão as mesmas otimizações. A primeira técnica é a de *inlining* que consiste em substituir a chamada de função pela função propriamente dita. Neste trabalho a técnica de *inlining* foi feita através da combinação de dois modos, utilizando a *flag* `-finline-functions` e de forma manual, e assim, pode-se aplicar a técnica para funções específicas, explorando apenas os benefícios da técnica. Além disso, a técnica de *loop unrolling* também foi aplicada, a qual é a ação de desenrolar o laço. Nas seções a seguir, temos exemplos da técnica de *loop unrolling*, especificamente o Algoritmo 3 e Algoritmo 6, que mostram a técnica aplicada no *benchmark* NAS e na modelagem RTM.

Como mencionado anteriormente, *inlining* é a técnica de colocar uma cópia de uma função quando ela é chamada. Isso aprimora a execução eliminando *overheads* desnecessário, como instruções de chamada de função relacionados a utilização de registradores e *stacks*. Porém, se a função a ser aplicada a técnica for muito grande, ou seja, ocupara muito espaço, *inlining* pode acabar consumindo muito da memória *cache* de instrução, deteriorando o desempenho da aplicação. Assim, é recomendado usa-la para funções que ocupam pouco espaço.

Novamente, a técnica de *loop unrolling* visa transformar um laço a modo que ele aplique mais de uma execução por iteração. Isso minimiza o número de saltos e de instruções. Além disso, pode facilitar a paralelização e vetorização, desde que as variáveis do laço sejam independentes.

5. Resultados das otimizações *loop unrolling* e *inlining*

A seguir são apresentados os experimentos, bem como seus resultados. Inicialmente, é exposto os resultados dos experimentos e análise, junto com as otimizações implementadas. Finalmente, uma discussão dos resultados obtidos, comparando o otimizado e original. Repetindo-se para ambos, o *benchmark* NAS e a modelagem RTM.

5.1. Otimização *inlining* e *loop unrolling* na NAS *Benchmark*

O experimento utilizando o *benchmark* NAS é apresentado a seguir. A versão utilizada é a NPB 3.4.1 *OpenMP* [Bailey et al. 1991]. Foi executado as aplicações, BT, CG, EP, FT, IS, MG, SP e UA em suas versões originais. O programa BT é um solucionador tri-diagonal com blocos. CG consiste no método do gradiente conjugado, com acesso a memória e comunicação irregulares. Já EP é uma aplicação *Embarrassingly parallel*. FT

é a transformada de Fourier rápida em 3D. *Integer sorting* é implementado no programa IS. MG é um solucionador de um campo potencial 3D. SP é semelhante ao BT, sendo um solucionador escalar penta diagonal. Por último temos UA, solucionador para uma malha adaptativa não estruturada.

A compilação utiliza o compilador exclusivo da NEC e algumas *flags* desse compilador, sendo elas `-O2` e `-fopenmp` [NEC 2020b]. É denominado aqui como versão "original", o código NAS não modificado, apenas aplicado a vetorização automática da máquina SX-Aurora TSUBASA. A "versão otimizada", consiste na aplicação das otimizações apresentadas anteriormente, *inlining* e *loop unrolling*.

A coleta de dados é feita através do programa de perfilamento da NEC, PROGINF [NEC 2020c]. Esse programa nos entrega diversas informações referentes a arquitetura, e para esse estudo, principalmente as FLOPS e a taxa de falha da memória cache.

O compilador, ao inserir *flags* específicas, neste caso `-report-all`, `-fdiag-inline=2`, `-fdiag-parallel=2` e `-fdiag-vector=2` [NEC 2020b], faz um relatório completo, contendo, por exemplo, laços vetorizados e não vetorizados, funções que bloqueiam a vetorização, laços aninhados, laços estendidos, entre outros. Isso ajuda na decisão de onde aplicar as otimizações. Por exemplo, para o programa BT, o compilador não estava vetorizando algumas das áreas de computação, especificamente as partes de subtração e multiplicação das matrizes. Ao analisar o código e utilizando o guia desenvolvido pela NEC [NEC 2020b] é possível, então, aplicar as técnicas de otimização, já discutidas, e diretrizes para facilitar a vetorização feita pelo compilador. O Algoritmo ilustrado na Figura 3 mostra a aplicação da técnica de *loop unrolling* no programa BT.

```
do k=ksize-1,0,-1
   do m=1,BLOCK_SIZE
      do n=1,BLOCK_SIZE, 2
         rhs(m,i,j,k) = rhs(m,i,j,k)
            lhs(m,n,cc,k)*rhs(n,i,j,k+1)
         rhs(m,i,j,k) = rhs(m,i,j,k)
            lhs(m,n+1,cc,k)*rhs(n+1,i,j,k+1)
      enddo
   enddo
enddo
```

Figura 3. Trecho de código-fonte Fortran, ilustrando um *Loop unroling* aplicado no benchmark NAS.

Assim, aplicando as técnicas de otimização previamente mencionadas, *inlining* e *loop unrolling*, no *benchmark* NAS, e a comparando com a implementação original, obtém-se a Figura 4, sendo o eixo x cada aplicação do *benchmark* NAS utilizado nesse teste, o eixo y mostra as FLOPS, a barra vermelha equivale a implementação original e a barra azul representa a versão otimizada.

Percebe-se que as FLOPS aumentam consideravelmente, chegando até $7,8\times$ maior para a aplicação otimizada do *benchmark* BT. Comparando o aumento entre a versão original e a otimizada, temos uma média de, respectivamente, 204,42 MFLOPS e 1599,18 MFLOPS.

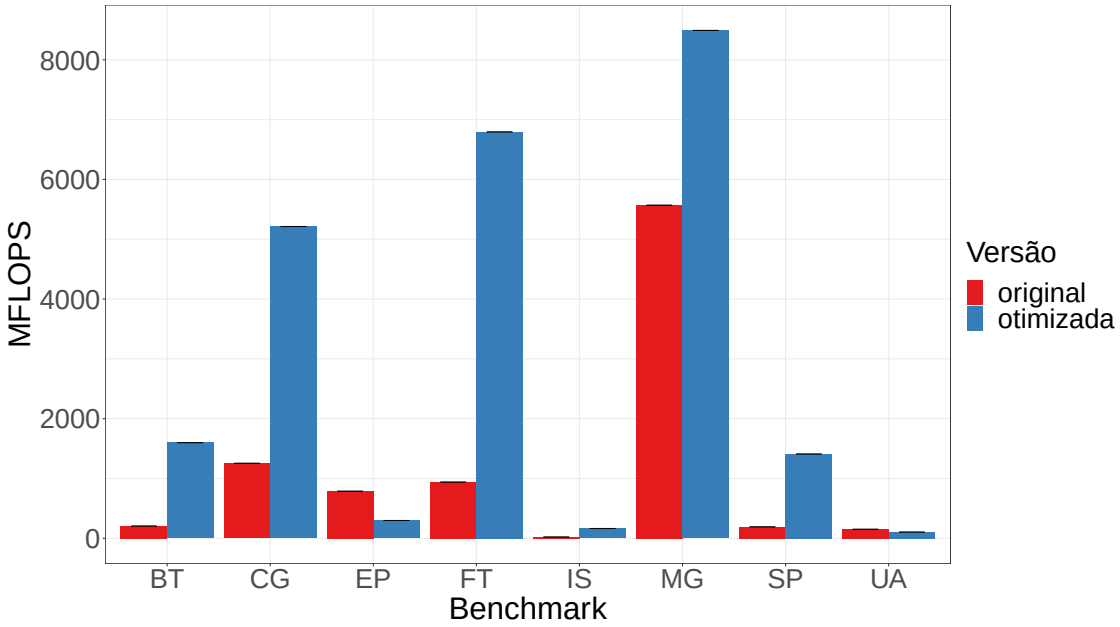

Figura 4. MFLOPS X Aplicação NAS

Na Figura 5 temos a taxa de operações vetoriais no eixo y para cada aplicação do *benchmark* NAS no eixo x, sendo em vermelho as aplicações originais e em azul a versão otimizada. O programa BT no código otimizado possui uma taxa de operações vetoriais superior ao original, aproximadamente de 20% para 50%, mais que o dobro de operações vetoriais. Esse grande aumento das operações é devido ao *loop unrolling* e *inlining* ajudarem o compilador na sua vetorização automática. Além disso, para o programa BT, por exemplo, o tamanho médio de cada vetorização também sofreu um aumento de aproximadamente 31%, indo de 92 elementos para 120 elementos.

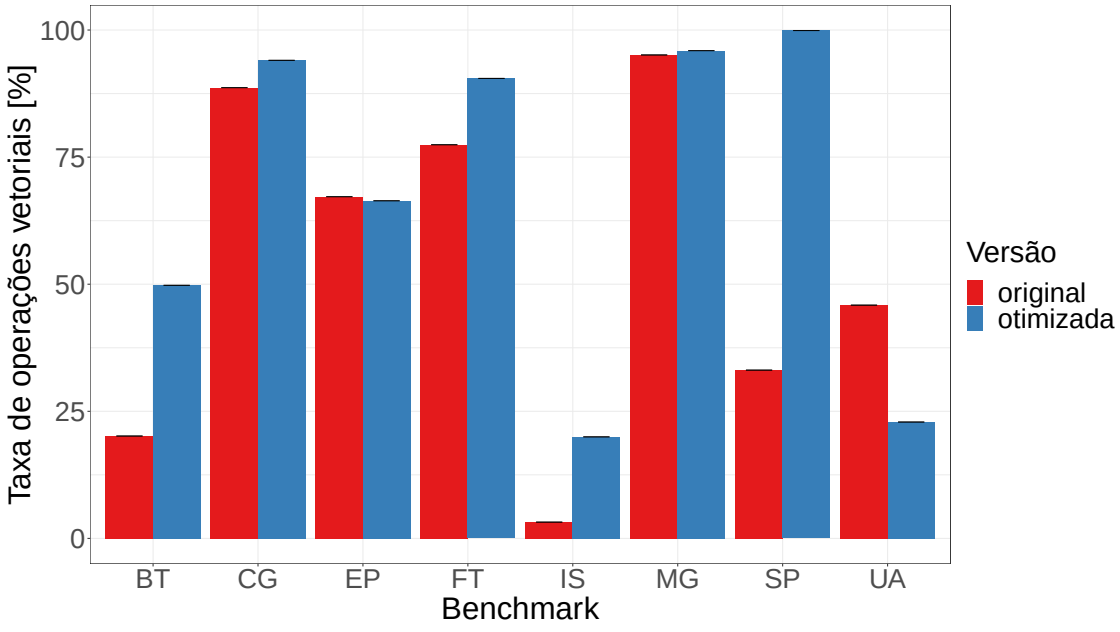

Figura 5. NAS *benchmark*: Taxa de operações vetoriais [%] X Aplicação NAS

Além disso, vale ressaltar que a memória *cache* também se beneficiou das otimizações, novamente, tomando como exemplo o programa BT, obteve-se um aumento de aproximadamente 10% na taxa de acerto da *cache* L3.

5.2. Otimização *inlining* e *loop unrolling* na modelagem RTM

A seguir, o experimento utilizando a modelagem RTM é apresentado, executado com a versão original *OpenMP*.

A configuração utilizada para a aplicação RTM é a mesma da otimização com o NAS *benchmark*. Novamente, a compilação faz uso do compilador exclusivo da NEC e as mesmas *flags* utilizadas no experimento anterior. Igual a NAS *benchmark*, a versão "original", condiz com o código RTM não modificado, aplicado a vetorização automática da máquina SX-Aurora TSUBASA. A versão otimizada, equivale a implementação das técnicas apresentadas anteriormente, *inlining* e *loop unrolling*.

A coleta de dados, do mesmo modo na otimização NAS, utiliza-se do programa de perfilamento da NEC, PROGINF. Através desse programa diversas informações relacionadas com a arquitetura e execução, principalmente as FLOPS e a taxa de falha da memória cache.

Outra vez, o compilador, faz um relatório completo, contendo diversas informações referente a compilação. Exemplificando, para melhorar a vetorização, *loop unrolling* foi utilizado em diversas funções. Novamente utilizou-se a guia providenciada pela NEC [NEC 2020a] e as técnicas previamente mencionadas para otimizar a aplicação. O Algoritmo ilustrado na Figura 6 demonstra a aplicação de *loop unrolling*.

```
for (i=1; i<sx*sy*sz; i+=4) {
  maxvel=fmaxf(maxvel,vpz[i]*sqrt(1.0+2*epsilon[i]));
  maxvel=fmaxf(maxvel,vpz[i+1]*sqrt(1.0+2*epsilon[i+1]));
  maxvel=fmaxf(maxvel,vpz[i+2]*sqrt(1.0+2*epsilon[i+2]));
  maxvel=fmaxf(maxvel,vpz[i+3]*sqrt(1.0+2*epsilon[i+3]));
}
```

Figura 6. Trecho de código-fonte C, ilustrando um *loop unrolling* aplicado a RTM

Portanto, a Figura 7 mostra o experimento aplicado para a aplicação RTM, exibindo as FLOPS no eixo y em relação ao tamanho da entrada no eixo x, sendo em azul a versão otimizada e em vermelho a versão original. Os FLOPS variam consideravelmente da menor entrada para a maior. Analisando a versão original e a nova versão otimizada, para a maior entrada de dados 504×504, temos 2429,83 MFLOPS e 4574,75 MFLOPS, respectivamente, um aumento de aproximadamente $1,9\times$. Isso ocorre devido a vetorização automática feita pelo compilador da NEC, aprimorada pelas técnicas aplicadas de *loop unrolling* e *inlining*. Também deve-se ressaltar que *loop unrolling* diminuiu o número de instruções de execução não vetorizadas.

A taxa de operações vetoriais é demonstrada na Figura 8. No eixo y temos a taxa de operações vetoriais em relação ao tamanho de entrada no eixo x, sendo em azul a versão otimizada e em vermelho a versão original. Comparando a maior entrada de dados 504×504 para a versão original e a nova versão otimizada, temos um aumento

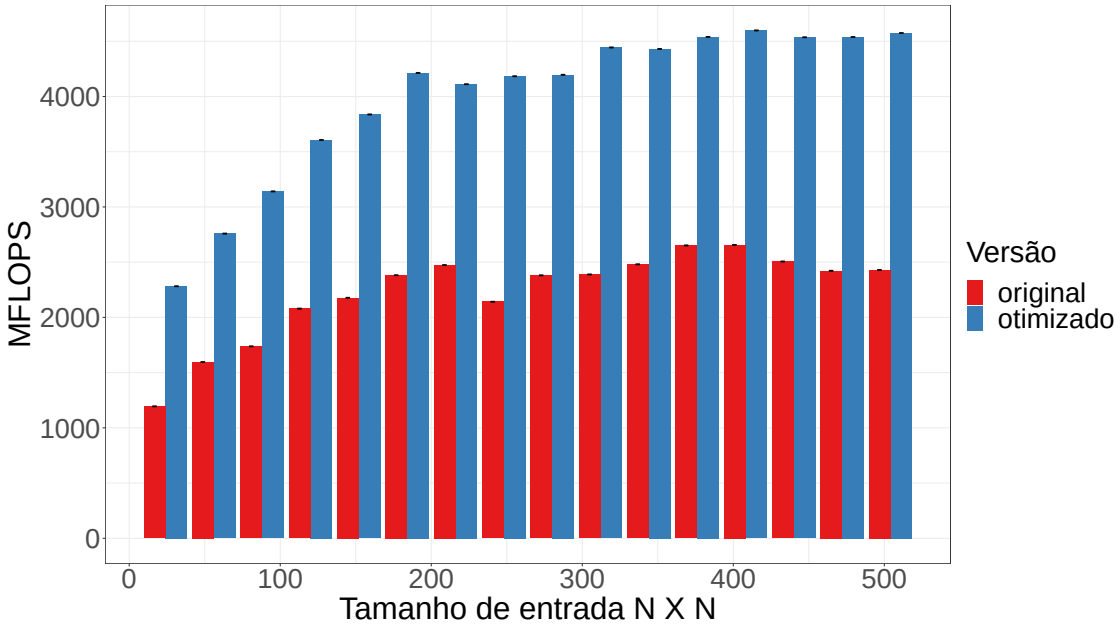

Figura 7. RTM: MFLOPS X Tamanho da entrada

de 5%, aproximadamente. Além disso, deve-se apresentar um pequeno aumento para o acerto da memória cache L3 de aproximadamente 2%, e analisando a cache L1, temos uma diminuição de 50%, aproximadamente, no tempo de falha da memória cache L1 e 20% para a cache L2. Devido a técnica de *loop unrolling*, diminui-se os *overheads* de instrução do laço, bem como *inlining* minimiza as instruções de chamada de função.

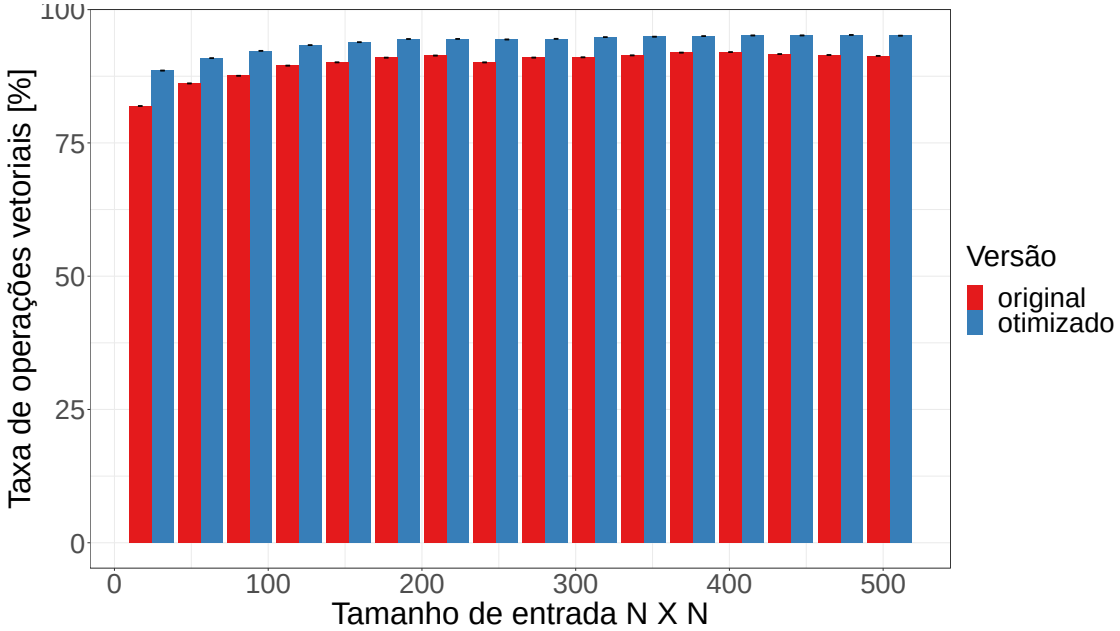

Figura 8. RTM: Taxa de operações vetoriais [%] X Tamanho da entrada

6. Conclusão e Trabalhos Futuros

Dispositivos aceleradores de desempenho tais como *Graphics Processing Units* (GPUs), são componentes essenciais para o alto desempenho dos supercomputadores modernos. Sob essa motivação, existem grandes esforços no desenvolvimento de processadores vetoriais para atuarem também como dispositivos aceleradores de desempenho. Além de possuírem uma alta capacidade de processamento *Single-Instruction-Multiple-Data* (SIMD), aceleradores vetoriais se destacam por permitirem que aplicações se beneficiem de ganhos de desempenho, com pouca ou nenhuma intervenção no código-fonte das aplicações, fato que é menos frequente na aceleração com GPUs.

Neste trabalho, fizemos uma das primeiras análises de desempenho do recém-lançado acelerador vetorial SX-Aurora TSUBASA [Komatsu et al. 2018, Yokokawa et al. 2020]. Para essa tarefa utilizamos o NAS [Bailey et al. 1991], que é um notório *benchmark* de aplicações paralelas, como também utilizamos uma aplicação real de migração sísmica, denominada RTM [Zhou et al. 2018, Fowler et al. 2010, Fletcher et al. 2009]. Nosso objetivo principal foi verificar, em comparação com a CPU, o quanto pode-se ganhar de desempenho com a SX-Aurora TSUBASA em duas situações, a saber: (i) sem nenhuma intervenção no código-fonte e unicamente com as otimizações automáticas do compilador da SX-Aurora TSUBASA e (ii) com técnicas de otimização simples e conhecidas da literatura, em situações onde o ganho de desempenho não foi automático.

Realizamos uma extensa campanha experimental e conseguimos mostrar evidências de ganhos desempenho utilizando a SX-Aurora TSUBASA, sem nenhuma intervenção no código-fonte de algumas aplicações. Além disso, mostramos também que, com pequenas intervenções no código-fonte, usando técnicas de otimização bem conhecidas na literatura por facilitar a computação SIMD, tais como *loop unrolling* e *inlining*, conseguimos melhorar o desempenho (aqui denominado como a taxa de operações de ponto flutuante por segundo (FLOPS)) da aplicação real RTM em até $1,9\times$ e em até $7,8\times$ para o *benchmark* NAS.

Para trabalhos futuros, estenderemos nossa análise para mais aplicações reais, notadamente aplicações de Aprendizado de Máquina e Inteligência Artificial, como também verificaremos se mais técnicas de otimização de desempenho, tais como *loop tiling* e *loop interchange* podem também se apresentar vantajosas para se obter ganhos de desempenho no acelerador vetorial SX-Aurora TSUBASA.

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., et al. (1991). The nas parallel benchmarks. *The International Journal of Supercomputing Applications*, 5(3):63–73.

Castro, M., Francesquini, E., Dupros, F., Aochi, H., Navaux, P. O., and Mehaut, J.-F. (2016). Seismic wave propagation simulations on low-power and performance-centric manycores. *Parallel Computing*, 54:108–120.

da Silva, S. A., da Silva Serpa, M., and Schepke, C. (2016). Técnicas de otimização loop unrolling e loop tiling em multiplicações de matrizes utilizando openmp. In *Workshop de Iniciação Científica do WSCAD*, pages 13–18.

Ezell, S. J. and Atkinson, R. D. (2016). The vital importance of high-performance computing to us competitiveness. *Information Technology and Innovation Foundation, April*, 28.

Fletcher, R. P., Du, X., and Fowler, P. J. (2009). Reverse time migration in tilted transversely isotropic (tti) media. *Geophysics*, 74(6):WCA179–WCA187.

Fowler, P. J., Du, X., and Fletcher, R. P. (2010). Coupled equations for reverse time migration in transversely isotropic media. *Geophysics*, 75(1):S11–S22.

Jacquelin, M., Marchal, L., and Robert, Y. (2009). Complexity analysis and performance evaluation of matrix product on multicore architectures. In *2009 International Conference on Parallel Processing*, pages 196–203. IEEE.

Komatsu, K., Momose, S., Isobe, Y., Watanabe, O., Musa, A., Yokokawa, M., Aoyama, T., Sato, M., and Kobayashi, H. (2018). Performance evaluation of a vector supercomputer sx-aurora tsubasa. In *SC18: International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 685–696. IEEE.

Kowarschik, M. and Weiß, C. (2003). An overview of cache optimization techniques and cache-aware numerical algorithms. In *Algorithms for memory hierarchies*, pages 213–232. Springer.

Mittal, S. and Vetter, J. S. (2015). A survey of cpu-gpu heterogeneous computing techniques. *ACM Computing Surveys (CSUR)*, 47(4):1–35.

NEC (2020a). How to use c/c++ compiler for vector engine. `https://www.hpc.nec/api/v1/forum/file/download?id=pgNh9b`. Acessado em: 08/2020.

NEC (2020b). How to use fortran compiler for vector engine. `https://www.hpc.nec/api/v1/forum/file/download?id=pRdhmv`. Acessado em: 08/2020.

NEC (2020c). Proginf/ftrace user's guide. `https://www.hpc.nec/documents/sdk/pdfs/g2at03e-PROGINF_FTRACE_User_Guide_en.pdf`. Acessado em: 08/2020.

NEC (2020d). Sx-aurora tsubasa a100-1 series user's guide. `https://www.hpc.nec/documents/guide/pdfs/A100-1_series_users_guide.pdf`. Acessado em: 08/2020.

Serpa, M. S., Cruz, E. H., Diener, M., Krause, A. M., Farrés, A., Rosas, C., Panetta, J., Hanzich, M., and Navaux, P. O. (2017). Strategies to improve the performance of a geophysics model for different manycore systems. In *2017 International Symposium on Computer Architecture and High Performance Computing Workshops (SBAC-PADW)*, pages 49–54. IEEE.

Yokokawa, M., Nakai, A., Komatsu, K., Watanabe, Y., Masaoka, Y., Isobe, Y., and Kobayashi, H. (2020). I/o performance of the sx-aurora tsubasa. In *2020 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pages 27–35. IEEE.

Zhou, H.-W., Hu, H., Zou, Z., Wo, Y., and Youn, O. (2018). Reverse time migration: A prospect of seismic imaging methodology. *Earth-Science Reviews*, 179:207–227.