UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO

ANDRÉ RAMOS CARNEIRO
JEAN LUCA BEZ

Relatório de Pesquisa 373

Otimização do Desempenho de E/S de uma Aplicação de Modelagem Numérica Atmosférica

Orientador
Prof. Dr. Philippe Olivier Alexandre Navaux

Porto Alegre, 4 de agosto de 2017.

RESUMO

Neste trabalho avaliamos o desempenho das operações de E/S de uma aplicação de modelagem numérica atmosférica, o OLAM (Ocean-Land-Atmosphere Model). Com o auxílio da ferramenta de profiling Darshan, buscamos caracterizar as operações de E/S da aplicação considerando diferentes cenários de execução no supercomputador brasileiro Santos Dumont. Investigamos problemas e perdas de desempenho relacionadas às operações de E/S da aplicação utilizando diferentes implementações MPI. Constatamos que o OLAM faz um grande número de requisições pequenas (aproximadamente 1300 bytes), que impacta diretamente no seu desempenho e na sua escalabilidade, principalmente ao se utilizar a implementação BullxMPI.

1 INTRODUÇÃO

Aplicações científicas como simulações atmosféricas, de fluxos ou de terremotos, demandam cada vez mais requisitos computacionais para simular esses fenômenos complexos. Durante suas execuções tais aplicações necessitam acessar dados, sejam arquivos de entrada da simulação, *checkpoints* ou arquivos de saída. Por executarem em ambientes de High-Performance Computing (HPC), como clusters de larga escala ou supercomputadores, estas aplicações normalmente realizam suas operações de entrada e saída (E/S) em um sistema de arquivos paralelo que é compartilhado por todas elas.

Essas operações de E/S são um gargalo para um número crescente de aplicações devido à diferença existente entre as velocidades de processamento e de acesso aos dados. Um exemplo de tais aplicações é o OLAM (Ocean-Land-Atmosphere Model) [1].

O OLAM é um sistema de simulação de tempo que busca prover as vantagens de modelos regionais através de um modelo de execução global. Isso é obtido usando um *grid* global que pode ser localmente refinado em regiões de interesse. Usando essa técnica, fenômenos locais podem ser adequadamente simulados e não há a necessidade de execuções prévias de modelos globais de maior granularidade, já que as condições nas bordas são geradas durante a execução do modelo.

Implementações de alto desempenho desses modelos são fundamentais para as atividades de previsão de tempo e clima. Primeiramente, a complexidade computacional de tais modelos é $O(n^4)$, onde n representa o número de pontos em um eixo do domínio. Esse número influencia a precisão dos resultados, já que define a resolução espacial. Adicionalmente, esses modelos devem concluir a execução o mais rápido possível, para que seus resultados possam ser aproveitados em tempo hábil. É desejável obter o resultado de uma simulação de previsão do tempo antes que o evento que se está tentando prever ocorra. Portanto, uma melhora no desempenho resulta em uma maior resolução alcançável, maior precisão dos resultados.

O OLAM é uma das aplicações que estão sendo otimizadas no contexto do projeto High Performance Computing for Energy[2] (HPC4E), que é um projeto envolvendo a União Europeia (UE) e o Brasil, estabelecido através do Programa Horizon 2020 da UE e do Ministério da Ciência, Tecnologia, Inovações e Comunicações do Brasil. O projeto HPC4E visa melhorar a eficiência do setor energético, por meio da utilização da computação de alto

desempenho para desenvolver novas aplicações e aprimorar ferramentas de simulação computacional para diferentes fontes de energia: projeto e produção de energia eólica, sistemas eficientes de combustão de combustíveis derivados de biomassa (biogás) e geofísica de exploração para reservatórios de hidrocarbonetos.

Para atingir esses objetivos, diversas instituições brasileiras e europeias (universidades, centros de pesquisa, instalações de supercomputação e indústria) estão cooperando no aprimoramento de ferramentas de computação de alto desempenho para a melhoria da cadeia do uso de diferentes fontes de energia. Dentre essas instituições podemos citar o Barcelona Supercomputing Center, o INRIA, o CIEMAT, a COPPE, o ITA, o LNCC e a UFRGS. Diversas pesquisas do HPC4E são realizadas no Supercomputador SDumont [3], instalado na sede do LNCC.

2 OBJETIVOS

O objetivo deste trabalho foi avaliar o desempenho das operações de E/S de uma aplicação de modelagem numérica atmosférica, o OLAM. Neste contexto, a ferramenta Darshan [4] foi empregada para caracterizar as operações de E/S da aplicação considerando diferentes cenários de execução, com um sobrecusto desprezível [4]. Desta forma buscou-se investigar problemas e perdas de desempenho relacionados às operações de E/S da aplicação.

Esperamos assim poder propor otimizações que poderão possivelmente reduzir o tempo de E/S e permitir maior frequência do uso destas operações, por exemplo, através de *checkpoints* mais frequentes.

O restante deste relatório está organizado da seguinte forma. A Seção 3 apresenta a aplicação OLAM utilizada nesta pesquisa. A Seção 4 descreve o ambiente utilizado para a realização dos experimentos, assim como as ferramentas auxiliares. Os resultados obtidos e as avaliações de desempenho são apresentadas na Seção 5. A última seção apresenta as conclusões e os trabalhos futuros.

3 OLAM

O OLAM foi desenvolvido para estender as funcionalidades do Regional Atmospheric Modeling System (RAMS)[5] para um domínio global. O OLAM utiliza muitas das funções do RAMS, incluindo as parametrizações de física, assimilação de dados, métodos de inicialização, lógica e estrutura de código, e os formatos das operações de E/S [6]. Ele introduz um novo núcleo dinâmico, baseado em uma grade geodésica global com células de malhas triangulares. Além disso, também utiliza uma discretização de volume finito das equações compressíveis de Navier Stokes. Refinamentos locais podem ser definidos para abranger áreas geográficas específicas com uma maior resolução. Recursividade pode ser aplicada ao refinamento local. A grade global e seus refinamentos definem uma única grade, ao contrário das grades aninhadas dos modelos regionais. As células refinadas da grade não sobrepõe as células da grade global – elas as substituem.

O modelo consiste essencialmente de uma malha de grade global, com células triangulares, com a capacidade de refinamento global, todas as equações compressíveis de Navier-Stokes não-hidrostáticas, uma formulação em volume finito de leis de conservação para massa, impulso e temperatura potencial, e operadores numéricos que incluem divisão do tempo para termos acústicos. O domínio global amplia o alcance dos sistemas atmosféricos e as interações em escala que podem ser representadas no modelo, que foi o principal motivo do desenvolvimento do OLAM.

O OLAM foi desenvolvido em FORTRAN 90 e paralelizado com o Message Passing Interface (MPI), sob o modelo Single Program Multiple Data (SPMD). Ele é um modelo interativo, onde a cada passo de tempo pode ocorrer a escrita dos dados de saída, conforme definido nos seus parâmetros de entrada. O seu fluxo de trabalho é ilustrado na Figura 1.

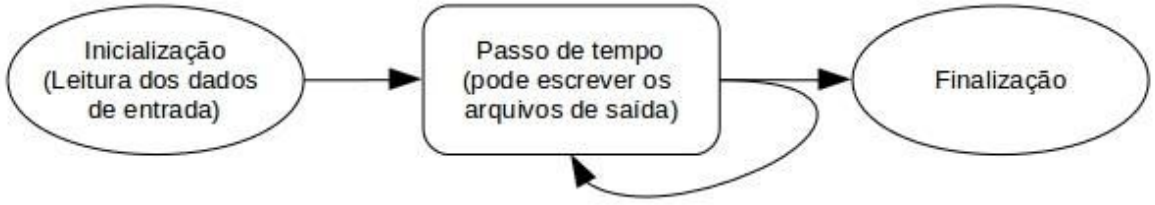


Figura 1: Organização Interativa do OLAM

Os arquivos de entrada do OLAM não são particionados para o processamento paralelo, visto que cada processo lê os arquivos de entrada por completo. Típicos arquivos de entrada são: condições de inicialização global em uma determinada data e hora, e mapas globais descrevendo a topografia, tipo de solo, áreas com cobertura de gelo, conjunto de dados de vegetação, profundidade do solo, temperatura da superfície do mar e o índice de vegetação também conhecido por Normalized Difference Vegetation Index (NDVI). A leitura dos dados do globo inteiro ocorre somente para arquivos que são lidos durante o momento da inicialização. Para os arquivos que também precisam ser lidos durante a integração do modelo (SST e NDVI), pelo fato de seus valores serem dependentes do tempo, eles são organizados em áreas geográficas separadas de 30 x 30 graus. Um processo MPI individual não precisa ler os dados para todo o globo.

Após essa fase, ocorre o processamento e a escrita dos dados de saída de forma alternada: durante cada fase de processamento o OLAM simula um determinado número de passos de tempo, evoluindo as condições atmosféricas em unidades de tempo. Após cada passo de tempo, os processos trocam mensagens com seus vizinhos para manter o estado da atmosfera consistente.

Após executar um determinado número de passos de tempo, as variáveis representando a atmosfera (pressão atmosférica, temperatura, velocidade do vento, precipitação, etc) são escritas no arquivo de saída. Durante essa fase, o OLAM utiliza o HDF5 para realizar operações de E/S coletivas, escrevendo um único arquivo compartilhado. Esses arquivos de saída possuem tamanho variado, indo de poucos MB a alguns GB, dependendo da definição da grade e do número de grades.

A partir da versão 4 do OLAM ele passou a utilizar as operações de escrita paralelas do HDF5 [7] para gerar os arquivos de saída. O HDF5 é uma biblioteca de E/S portável, muito empregada por diversas aplicações científicas para armazenar dados de uma forma organizada. Ele possui suporte ao paralelismo através da interface MPI-IO, possibilitando obter um maior desempenho ao acesso aos dados utilizando sistemas de arquivos paralelos.

Hoje, no Brasil, é feito um esforço integrado entre CPTEC/INPE, IAG/USP, LNCC, IME/USP, UFCG, UFSC e EMBRAPA para o desenvolvimento do modelo do OLAM. Como exemplo podemos citar o projeto “Desenvolvimento de funcionalidades, aumento da escalabilidade e estudos numéricos com modelos atmosféricos de alta resolução: BRAMS e OLAM (BRAMSOLAM)”[8], coordenado pelo pesquisador Pedro Leite da Silva Dias [9] e

que é parte do HPC4E. O projeto BRAMSOLAM é desenvolvido no SDumont e está pesquisando e implementando melhorias em diversas áreas do OLAM, tais como aumento de escalabilidade e utilização de aceleradores (GPU e Xeon PHI).

Os experimentos iniciais do OLAM realizados pelo projeto BRAMSOLAM no SDumont apresentaram um desempenho que foi seriamente comprometido pelas suas operações de E/S, como pode ser observado no gráfico da Figura 2, que detalha o tempo de execução da aplicação em segundos (eixo Y) variando o número de processos de 250 a 440 (eixo X), utilizando 20 nós do cluster. A barra vermelha representa o tempo gasto em I/O e a barra verde o tempo gasto em computação.

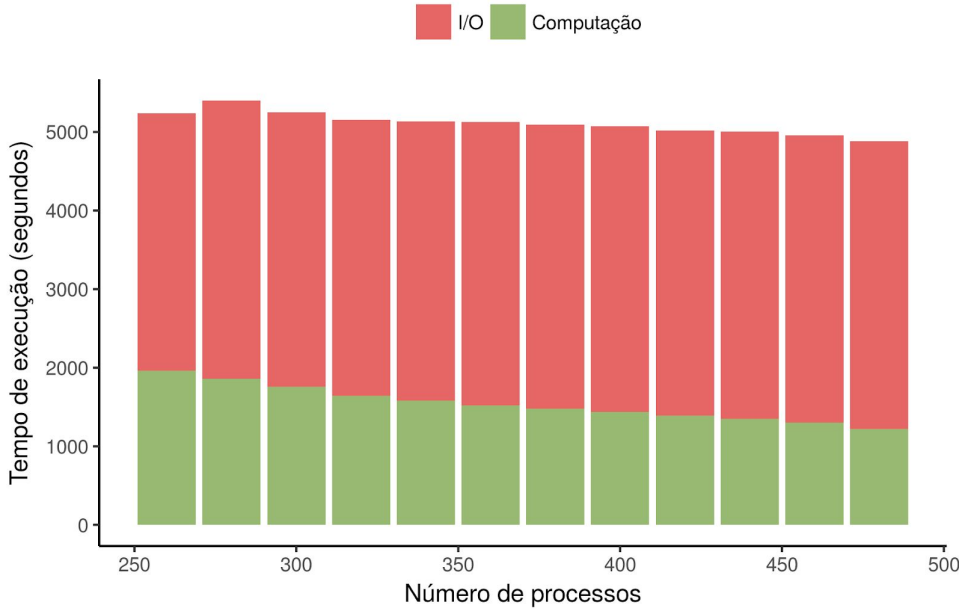


Figura 2. Relação entre tempo de execução e I/O

Ao utilizar o Darshan para obter o perfil de desempenho das operações de E/S, pode-se observar que as operações de E/S correspondem a até 73% do tempo de execução do OLAM. Outro ponto a se observar nesse gráfico é que, ao aumentar o nível de paralelismo, o tempo de computação cai, mas essa tendência não é necessariamente acompanhada pelo tempo gasto em E/S, comprometendo a escalabilidade da aplicação. Portanto, melhorar o desempenho destas operações é crucial para melhorar o desempenho da aplicação.

4 EXPERIMENTOS

Nesta seção são apresentados maiores detalhes do ambiente foco do estudo deste trabalho, juntamente com as ferramentas e bibliotecas empregadas na análise. Por fim, no item 4.3 apresentamos os cenários de testes utilizando o OLAM.

4.1 O Supercomputador SDumont

Todos os experimentos apresentados neste relatório foram executados no Supercomputador SDumont, projetado pela empresa francesa ATOS/BULL, e que está localizado na sede do LNCC. O SDumont possui um total de 18.144 núcleos de CPU, distribuídos da seguinte forma:

- 504 nós de computação B710 (thin node);
- 198 nós de computação B715 (thin node) com 2 x GPUs K40;
- 54 nós de computação B715 (thin node) com 2 x co-processadores Xeon Phi.

Todos os 756 nós possuem 2 processadores Intel Xeon E5-2695v2 Ivy Bridge, 2,4GHZ, cada com 12 núcleos (totalizando 24 núcleos por nó), 64GB de memória RAM DDR3 e um disco SSD local de 128GB. O SDumont ainda conta com 1 nó de computação MESCA2 com memória compartilhada de grande capacidade, com 16 processadores Intel Xeon Ivy E7-2870, 2,4GHZ, cada 15 núcleos (totalizando 240), e 6 TB de memória RAM.

Todos os equipamentos são interligados por uma rede de interconexão Infiniband FDR com uma topologia *fat-tree full-nonblocking*, com 1.944 portas. Possui um desempenho de 58Gb/s e 0,7us por porta, e banda passante total de 112.752 Gb/s. A vazão por porta é de 137 milhões de mensagens por segundo.

Por fim, o SDumont dispõe de um sistema de arquivos paralelo Lustre [10], v2.1 instalado no ClusterStor v1.5.0 da Xyratex/Seagate. A arquitetura de um sistema de arquivos Lustre é fundamentada sobre o armazenamento distribuído baseado em objetos, onde os metadados são separados dos dados, e ambos são armazenados em servidores separados. Os metadados são armazenados em dispositivos chamados MDTs (*Metadata Targets*), fornecidos e gerenciados pelos Servidores de Metadados (MDS). Os dados dos arquivos são

armazenados nos dispositivos OSTs (*Object Storage Targets*), que são administrados pelos Servidores de Armazenamento de Objetos (OSS). Há ainda um servidor de gerenciamento (MGS), que armazena e realiza a gerência das configurações do sistema de arquivo.

O Lustre do SDumont é formado por 1 servidor MGS e 1 servidor MDS, trabalhando em uma redundância Ativo/Ativo alternado, e 10 servidores de armazenamento (OSSs), organizados em 5 pares Ativo/Ativo, com cada um fornecendo 1 OST. Ele é integrado à rede Infiniband e possui capacidade bruta de armazenamento da ordem de 1,7 PBytes. Os nós computacionais utilizam o cliente Lustre na versão 2.4.3. O sistema de arquivos Lustre é montado como o volume `/scratch` e com a flag *flock* ativada no SDumont.

Por padrão, o `/scratch` está configurado para dividir os arquivos armazenados nele em pedaços, chamados de *stripes*, de até 1MB (1048576 bytes). Esse tamanho é configurado através do parâmetro *stripe size* do Lustre. Quando um arquivo atinge tamanho maior que o *stripe size*, ele será dividido e continuará sendo armazenado em um OST diferente, caso o parâmetro *stripe count* (utilizado para definir em quantas partes o arquivo será dividido) estiver configurado como maior que 1.

Todos os experimentos foram configurados para gerar os arquivos de saída em um diretório dentro do volume `/scratch`, o qual foi configurado para dividi-los por 10 OSTs, através do parâmetro *stripe count*. O *stripe size* não foi alterado.

4.2 Ferramentas de Análise de Desempenho das Operações de E/S

Para obter o perfil das operações de E/S do OLAM foi utilizado o Darshan v 3.1.4. O Darshan é um conjunto de bibliotecas que podem caracterizar operações de E/S MPI-IO e POSIX dentro de aplicações. Ele é uma ferramenta leve e que opera de uma maneira não intrusiva, sendo utilizado para investigar o comportamento de programas que utilizam o MPI para troca de mensagens e paralelização. Através do Darshan é possível obter os padrões de acesso das operações de E/S, como as operações de leitura, escrita e de metadados que são realizadas pela aplicação, além de informar se o acesso é baseado em MPI-IO ou POSIX.

Durante a execução da aplicação, o Darshan coleta as estatísticas, que são armazenadas em um arquivo por processo. Quando a aplicação finaliza, através da chamada `MPI_Finalize()`, uma rotina de encerramento é executada para combinar todos os arquivos.

4.3 Experimentação com o OLAM

Os experimentos com o OLAM foram configurados para a execução do domínio global, com 6 níveis verticais de atmosfera, e 7 grades (grade 1 sendo a global e mais 6 grades aninhadas), configuradas como:

- Grade 1: Resolução espacial horizontal de 200KM
- Grade 2: Resolução espacial horizontal de 100KM
- Grade 3: Resolução espacial horizontal de 50KM
- Grade 4: Resolução espacial horizontal de 25KM
- Grade 5: Resolução espacial horizontal de 12,5KM
- Grade 6: Resolução espacial horizontal de 6,25KM
- Grade 7: Resolução espacial horizontal de 3,125KM

Todas as simulações utilizaram o passo de tempo de 10 segundos, realizando uma integração de 2 dias e gerando 49 arquivos de saída, cada um sendo escrito a cada uma hora de integração, com aproximadamente 1.1GB. Essa configuração do OLAM está sendo utilizada para estudos e análises pelo projeto de pesquisa “BRAMSOLAM”.

A versão do OLAM[11] utilizada nos experimentos foi a 4.10 (r544), compilada utilizando o Intel Parallel Studio XE 2017 update 1 e utilizando as bibliotecas HDF5 v1.8.18 e NCARG v5.2.0.

5 RESULTADOS

Os resultados abaixo são a média aritmética de 5 execuções de cada um dos experimentos e foram obtidos utilizando 10 nós (sdumont[5004-5013]) e 24 cores por nó, totalizando 240 cores. Estes nós foram selecionados por fazerem parte de uma fila especial restrita, utilizada para testes de novas aplicações.

Como o objetivo é avaliar o comportamento padrão, tanto da aplicação como da implementação MPI, não foram realizadas alterações nos parâmetros (chamados de `hints`) que configuram o funcionamento do interno do MPI-IO de cada implementação MPI. Para assegurar que não haveria diferenças nas configurações dos `hints` entre as implementações MPI realizamos experimentos para coletar os parâmetros utilizados por padrão em cada uma delas. Assim foi possível observar que todas utilizam os mesmos parâmetros.

5.1 OLAM com 7 grades

Os resultados apresentados na Figura 2 foram obtidos utilizando o BullxMPI v1.2.8.4 como biblioteca de comunicação interprocessos. Ele que é uma implementação do padrão MPI baseado no OpenMPI, personalizado pela BULL/ATOS para a sua plataforma de Software.

Inicialmente optamos por realizar novos experimentos utilizando o BullxMPI e, adicionalmente, experimentos também utilizando a implementação MPI OpenMPI 1.10, por ela não possuir as alterações de código implementadas pela BULL/ATOS.

Com base nos dados coletados pelo Darshan dos experimentos realizados, podemos observar através da Figura 3 que ao utilizar o OpenMPI-1.10 o tempo gasto com as operações de E/S reduziu drasticamente, sendo responsável por aproximadamente 12% do tempo total de execução, enquanto que o BullxMPI novamente apresentou um baixo desempenho nas operações de E/S, que consumiram uma grande parcela do tempo de execução.

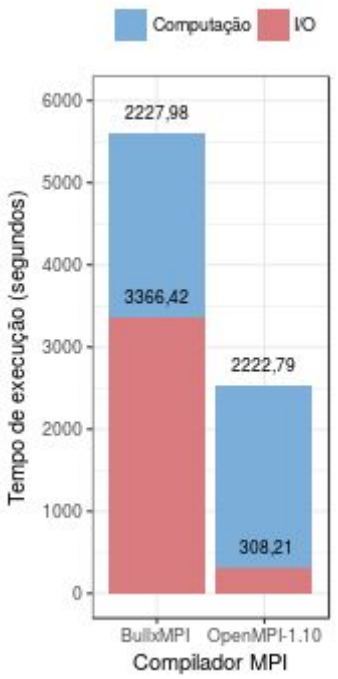
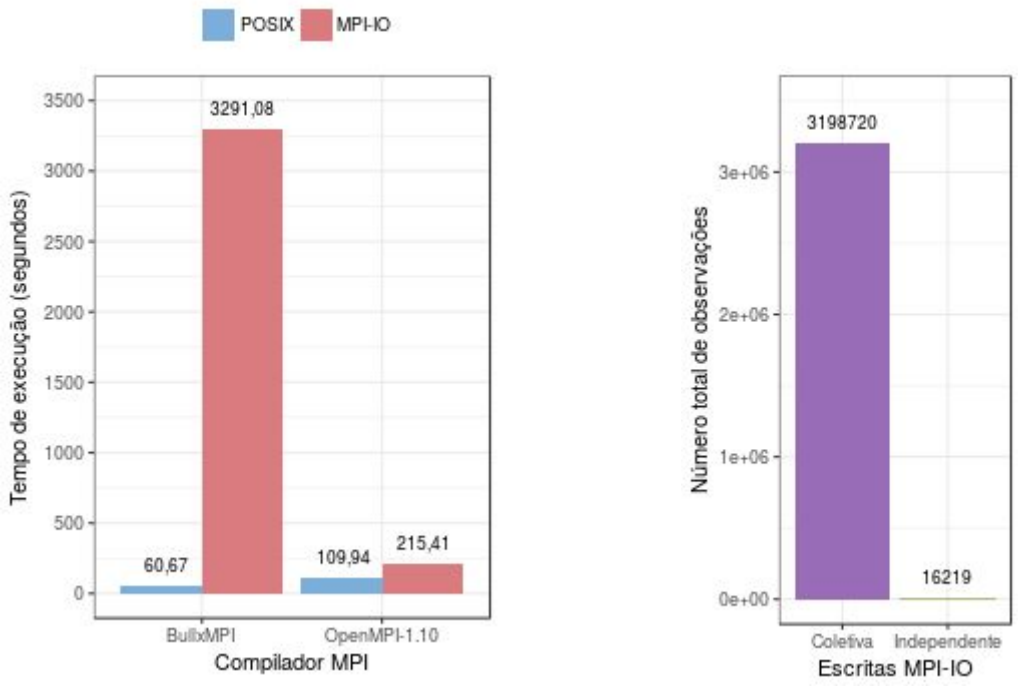


Figura 3. Comparação entre tempo de execução e tempo em operações de entrada e saída.

Importante observar que o tempo gasto com a computação foi praticamente o mesmo para ambas as versões, sendo de 2227 segundos com o BullxMPI e 2222 com o OpenMPI-1.10.

Ao analisarmos o tempo gasto com as operações de E/S, conforme ilustra a Figura 4.a é possível observar que a maior parte do tempo é gasto realizando em operações utilizando o HDF5 através da API MPI-IO. Detalhando o tipo da operação, se coletiva ou não, a Figura 4.b demonstra que foram observadas mais chamadas de escritas utilizando operações MPI-IO coletivas. Esse é o momento onde os arquivos de saída são escritos no disco.



a. Tempo de E/S por API (POSIX e MPI-IO) b. Total de operações MPI-IO

Figura 4. Comparação entre tempo de E/S por API (POSIX e MPI-IO).

Outra informação interessante que foi possível obter através do Darshan é em relação ao tamanho das requisições realizadas pelo OLAM. Pela Tabela 1 podemos observar que a maioria das requisições são de tamanhos pequenos, em torno de 1300 bytes, nas operações realizadas através do MPI-IO.

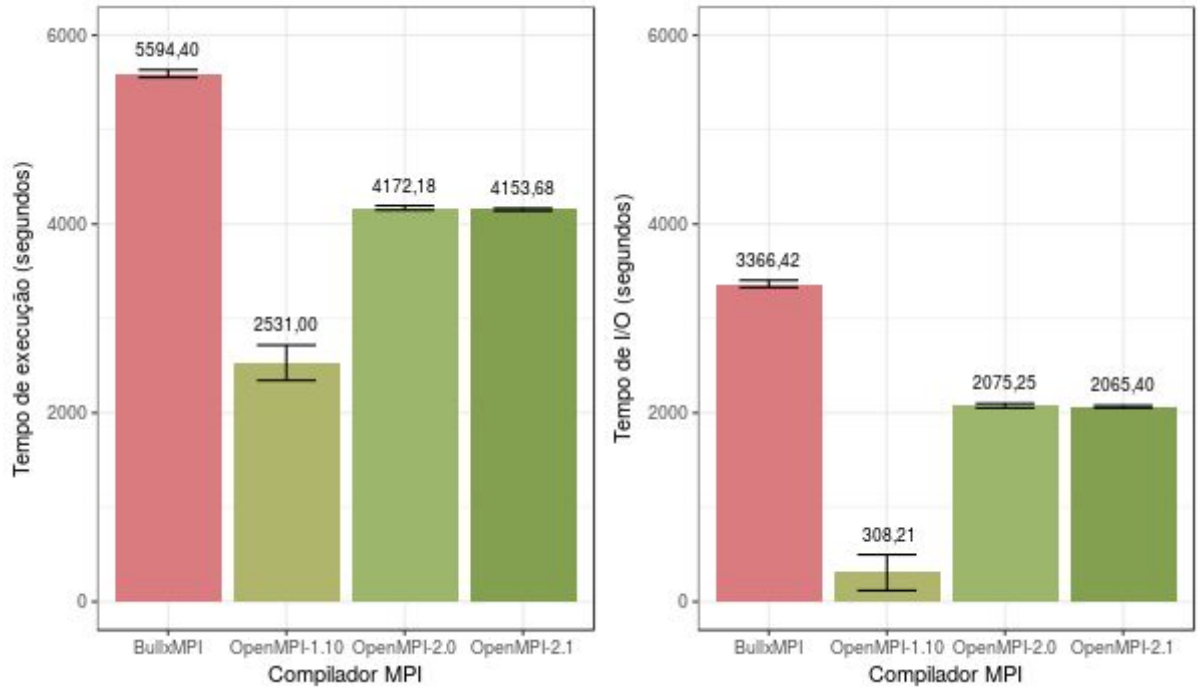
Tabela 1: Tamanhos de requisições mais frequentes reportados pelo Darshan

API	BullxMPI		OpenMPI-1.10	
	Tamanho (b)	**Quantidade**	**Tamanho (b)**	**Quantidade**
POSIX	8192	8585280	8192	8585280
	8190	6785760	8190	6785760
	512	126186	512	126186
	1048576	46011	1048576	46011
MPI-IO	1308	94521	1308	110397
	1312	55713	1312	94521
	1316	44394	1316	63798
	59040	43904	58860	59584

A grande diferença de desempenho entre os experimentos do OLAM utilizando o BullxMPI e o OpenMPI 1.10 foi inesperada pelo fato de que apenas alteramos a implementação MPI, sem nenhuma outra modificação. Dessa forma optamos por realizar novos experimentos utilizando todas as implementações do padrão MPI disponíveis no SDumont, com finalidade de verificar se haveria diferença no desempenho. Os novos experimentos utilizaram as versões 2.0 e 2.1 do OpenMPI e o IntelMPI, que é uma Implementação MPI baseada no MPICH e disponível no Intel PSXE 2017.1.

Ao realizar os experimentos com as outras implementações MPI não foi possível obter o arquivo de análise do Darshan. Durante o processo de escrita dos arquivos de saída utilizando o IntelMPI ocorre um erro e a simulação aborta, não sendo possível obter os arquivos do Darshan. Ao utilizar as implementações OpenMPI-2.X, a simulação do OLAM transcorre normalmente sem erros, mas os arquivos do Darshan não são gerados.

Para contornar esse problema e obter o tempo gasto com as operações de E/S, foi realizada uma modificação no código do OLAM, adicionando temporizadores para contabilizar o tempo das operações. Através dessa modificação foi possível obter o tempo para as versões 2.0 e 2.1 do OpenMPI para a configuração com 7 grades do OLAM.



a. Tempo total de execução do OLAM

b. Tempo total gasto em operações de E/S

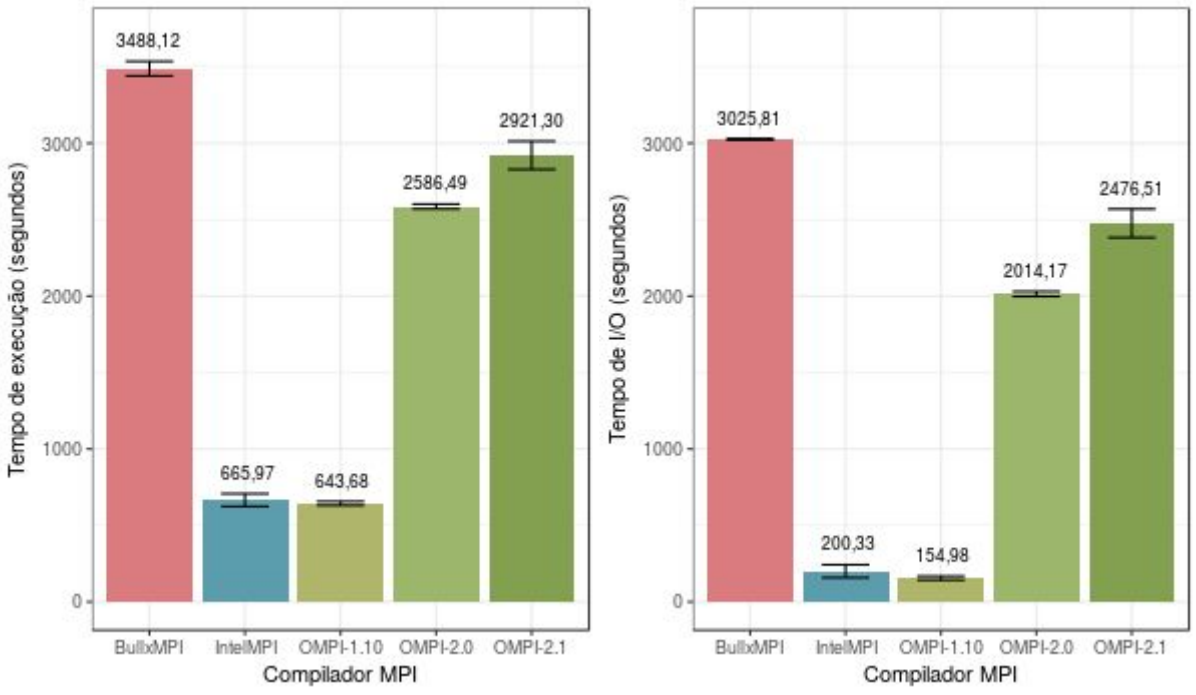
Figura 5. Métricas com a aplicação OLAM utilizando 7 grades

Através da Figura 5 podemos observar que o desempenho do OLAM utilizando as versões 2.0 e 2.1 do OpenMPI foi melhor em relação ao BullxMPI, mas ainda ficou abaixo do OpenMPI-1.10.

5.2 OLAM com 4 grades

Após alguns testes alterando a configuração do OLAM descobrimos que ao reduzir o número de grades do OLAM de 7 para 4, a simulação com a versão IntelMPI transcorre normalmente, sem apresentar os erros da configuração com 7 grades. Desta forma, utilizam-se as grades 1 a 4 com as mesmas configurações descritas na seção 4.3 deste relatório.

Neste sentido, optamos por repetir a comparação nessa escala/configuração para poder ter uma visão geral do comportamento da aplicação nos diferentes cenários utilizados neste estudo. É importante ressaltar que os resultados apresentados nesta seção utilizam a versão modificada do OLAM para coletar os tempos das fases de I/O da aplicação.



a. Tempo total de execução do OLAM

b. Tempo total gasto em operações de E/S

Figura 6. Métricas com a aplicação OLAM utilizando 4 grades

A Figura 6 demonstra que o tempo gasto nas operações de E/S pelo BullxMPI continua maior que as demais implementações MPI testadas. Além disso, neste cenário a aplicação passa aproximadamente 87% do seu tempo total de execução em operações de entrada e saída. Novamente, o OpenMPI-1.10 apresentou melhores resultados, em tempos absolutos. Apenas 24% do seu tempo total de execução é referente às operações de E/S. Utilizando os *timers* foi possível observar também o desempenho do IntelMPI, que demonstrou ser similar ao observado com o OpenMPI-1.10.

5.3 Experimentação com o NPB BT-IO

Para verificar se a grande diferença entre as implementações MPI ocorre com outra aplicação, realizamos experimentos utilizando o benchmark BT-IO[12] do NAS Parallel Benchmarks (NPB)[13]. O NPB é um conjunto de pequenos programas desenvolvidos pela divisão de supercomputação da NASA projetados para ajudar a avaliar o desempenho de supercomputadores paralelos.

O BT-IO implementa um solver de Bloco Tridiagonal que utiliza uma complexa decomposição de domínio chamado de muiti-particionamento diagonal. Cada processador é responsável por vários subconjuntos cartesianos de todo o conjunto de dados, cujo número aumenta como a raiz quadrada do número de processadores que participam da computação. A cada 5 passos de tempo, todo o campo da solução, que consiste de cinco palavras de dupla precisão por ponto de malha, deve ser gravado em arquivo. Após todos os passos de tempo serem finalizados, os dados devem ser ordenados pela componente do vetor, sendo respectivamente a coordenada *x*, coordenada *y* e coordenada *z*.

Os experimentos utilizaram a classe de problema D, que configura a execução do BT-IO com uma grade de 408 x 408 x 408 pontos, realizando 250 interações, com passo de tempo 20 μs e gerando um arquivo de aproximadamente 135 GB. Pelo fato do BT-IO necessitar que o número total de processadores possua raiz quadrada inteira, os experimentos foram executados em 8 nós (sdumont[5004-5011]) e 18 cores por nó, totalizando 144 cores, que proporcionou a utilização de um maior número de nós em cada experimento.

Nos resultados reportados pela saída do BT-IO não foi observado a grande diferença no tempo utilizado para realizar as operações de E/S entre as implementações MPI, conforme

ocorreu no OLAM. A análise do Darshan dos experimentos realizados com o BullxMPI e com o OpenMPI apontaram que todas as requisições realizadas pelo BT-IO utilizando o MPI-IO para ler e escrever os arquivos utilizaram o tamanho de transferência de 18865920 bytes, um valor muito maior do que o utilizado pelo OLAM.

5.4 Experimentação com o IOR

Para tentar investigar se o problema do desempenho está relacionado ao tamanho das requisições utilizadas pelo OLAM e melhor compreender os resultados até o momento apresentados foi utilizado o *benchmark* IOR [14], versão 2.10.3, buscando simular o comportamento das operações paralelas de E/S do OLAM e realizar um comparativo do desempenho das implementações MPI utilizando o sistema de arquivo Lustre.

O Interleaved-Or-Random (IOR) foi desenvolvido pelo Lawrence Livermore National Laboratory (LLNL), projetado para realizar testes de desempenho de sistemas de arquivo paralelos para *clusters* de alto desempenho. Ele pode ser utilizado em qualquer plataforma que possua uma implementação MPI.

Este *benchmark* fornece a capacidade de testar as taxas agregadas das operações de E/S através de várias bibliotecas, incluindo o MPI-IO, o HDF5, o NetCDF e o POSIX. Através dos parâmetros de entrada é possível realizar diversas configurações, variando o tamanho geral de E/S, o tamanho das transferências individuais, o modo de acessar o arquivo (um único arquivo compartilhado ou um arquivo por cliente/processo), e se os dados serão acessados de forma sequencial ou aleatória. Esses e outros parâmetros podem ser utilizados para simular os padrões das operações de E/S de aplicações reais.

Os experimentos com IOR foram configurados para realizar testes de leitura e escrita, configurados com o tamanho de transferência de dados de 1312 bytes, que é o tamanho mais utilizado pelo OLAM nas suas requisições. Importante notar que esse tamanho não é divisor ou múltiplo do *stripe size* configurado por padrão no `/scratch`, (1048576 bytes), gerando acessos desalinhados no sistema de arquivos do Lustre. Dessa forma, também foi testado um tamanho próximo, de 1024 bytes, que não leva a acessos desalinhados, visando avaliar se o fato do OLAM realizar acessos desalinhados resultaria em perda de desempenho. Também optamos por realizar testes com os tamanhos de 58864 bytes, que é o quarto tamanho mais

utilizado pelo OLAM e também gera acessos desalinhados, e com o tamanho de 65536 bytes, pelo mesmo motivo de utilizar 1024 bytes.

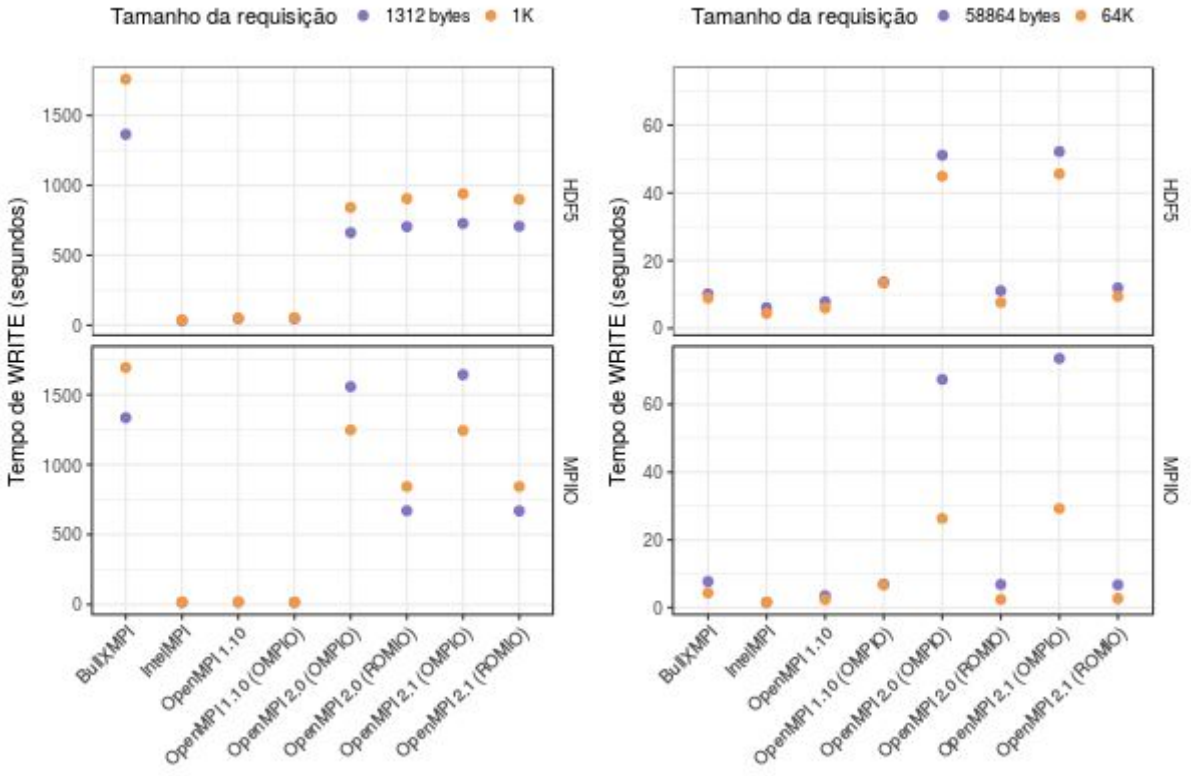
Para cada uma das implementações MPI foram realizados 5 experimentos utilizando a API do MPI-IO e 5 experimentos utilizando a do API do HDF5. Todos os experimentos realizaram operações coletivas, gerando um arquivo com o tamanho final de aproximadamente 1,5 GB.

O IOR organiza o arquivo a ser utilizado no teste em segmentos, que representam os *datasets* de uma aplicação. Cada segmento é dividido igualmente entre os processadores em unidades chamadas de "blocos", que representam o tamanho do subdomínio do *dataset* utilizando em cada processador. O processo com *rank* 0 recebe o primeiro bloco do seguimento, o processo com o *rank* 1 recebe o segundo bloco do segmento, e assim por diante. Cada bloco é subdividido em unidades de transferência, que é a quantidade de dados transferidos em cada chamada de função de E/S. A tabela 2 detalha os valores, em bytes, dos segmentos e blocos para cada um dos tamanhos de transferência escolhidos para os experimentos.

Tabela 2: Configuração dos Segmentos, Blocos e Unidades de Transferência

Tamanho da Unidade de Transferência	Tamanho do Bloco	Tamanho do Segmento[1]	Número de segmentos
1024	1024	245760	6554
1312	1312	314880	5116
58864	58864	14127360	116
65536	65536	15728640	104

Os resultados abaixo são a média aritmética de 5 execuções de cada um dos experimentos e também utilizaram os 10 nós (sdumont[5004-5013]) e 24 cores por nó, totalizando 240 cores.



a. Requisições de 1312 bytes e 1K

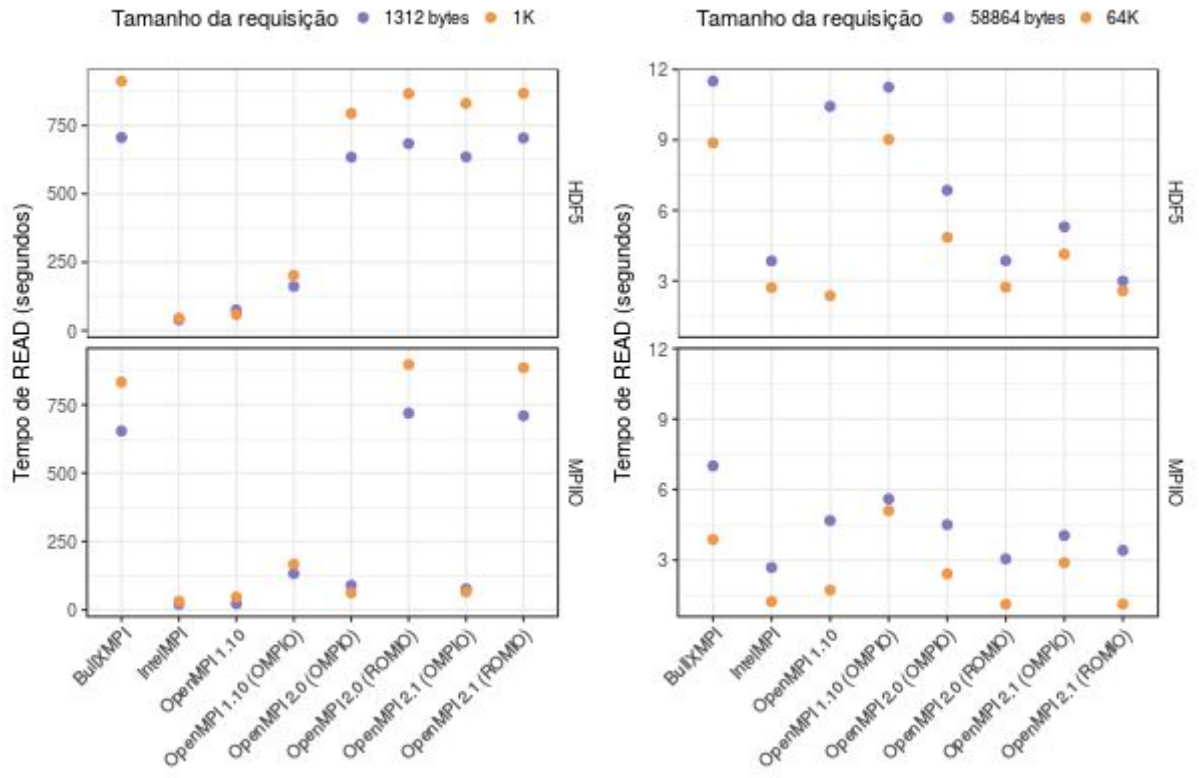
b. Requisições de 58864 bytes e 64K

Figura 7. Tempo de requisições de escrita com o IOR, diferentes versões de MPI (a escala de tempo utilizada nos dois gráficos não é a mesma)

É possível observar pelos resultados obtidos com o IOR que novamente o BullxMPI apresentou o pior desempenho, gastando mais tempo nas operações de escrita com tamanhos pequenos de transferência (1024 bytes – 1312 byes), com as versões 2.0 e 2.1 do OpenMPI bem próximas, sendo que o ROMIO foi mais rápido do que o OMPIO.

Ao aumentar os tamanhos de requisições para 58864 e 65536 bytes, todas as implementações MPI obtiveram uma redução nos tempos de leitura e escrita. Os resultados das implementações que utilizam o ROMIO ficaram mais próximos, com o OMPIO apresentando o pior resultado. O OpenMPI-1.10 e o IntelMPI apresentaram os melhores resultados no geral, com tempos bem próximos.

A implementação que apresentou a menor redução no tempo de escrita, ao aumentar o tamanho da requisição de 1024 bytes para 65536 bytes, foi o IntelMPI. A diferença foi de 15,3 segundos. A implementação que mais se beneficiou com o aumento no tamanho da requisição foi o BullxMPI, que teve uma redução 1689 segundos, sendo 99.7% mais rápido.



a. Requisições de 1312 bytes e 1K

b. Requisições de 58864 bytes e 64K

Figura 8. Tempo de requisições de leitura com o IOR, diferentes versões de MPI (a escala de tempo utilizada nos dois gráficos não é a mesma)

As operações de leitura utilizando tamanhos pequenos de requisições seguem o mesmo padrão do resultado das operações de escrita. Porém ao utilizar tamanhos maiores, as versões 2.0 e 2.1 apresentaram os melhores tempos, com o OpenMPI-1.10 e o IntelMPI ficando bem próximos. Também observamos uma grande redução no tempo utilizado para realizar a operação de leitura., com o IntelMPI sendo o que apresentou a menor redução (30,5 segundos), e o OpenMPI-2.0 (ROMIO) é o que obteve a maior redução do seu tempo (896 segundos).

Através da Figura 9 podemos observar o ganho de desempenho obtido ao aumentar o tamanho da requisição para as operações de escrita. No melhor resultado, obtido com o IntelMPI, podemos ver o desempenho saltar de 90 MB/s para 1014 MB/s, representando um ganho de 11 vezes.

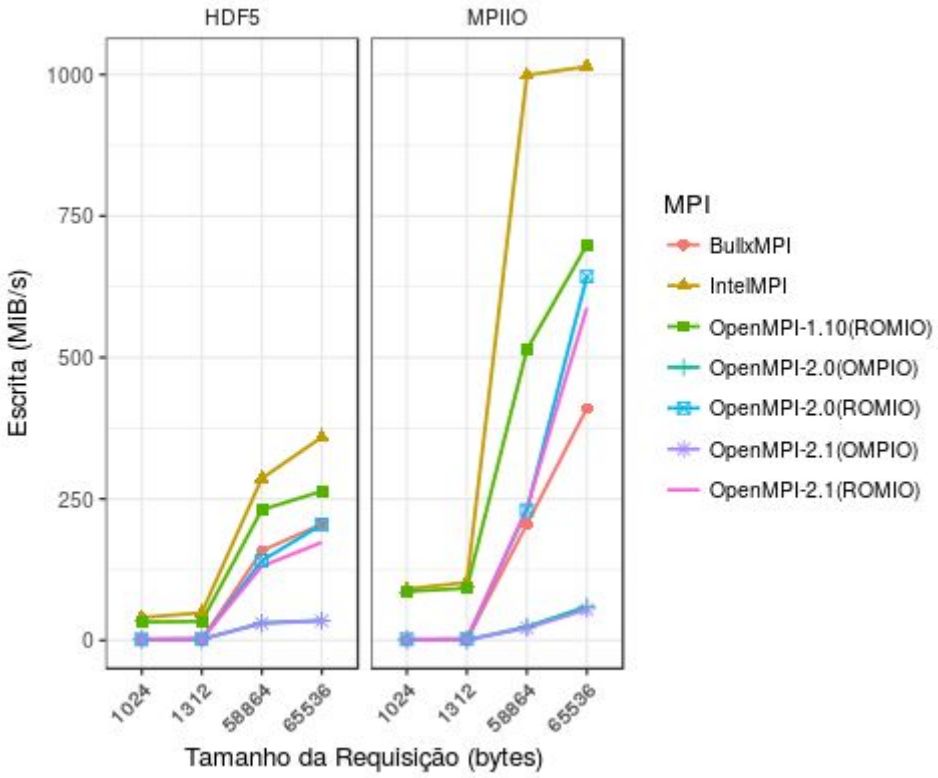


Figura 9. Desempenho das operações de escrita em relação ao tamanho das requisições

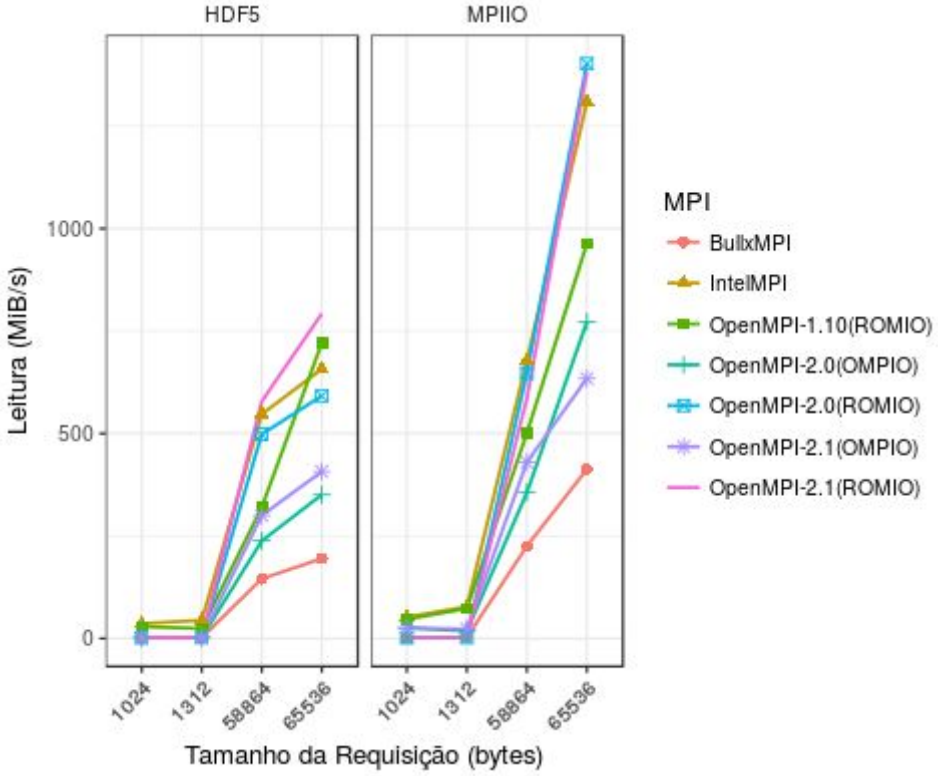


Figura 10. Desempenho das operações de leitura em relação ao tamanho das requisições

O desempenho das operações de leitura também apresentaram um grande aumento no desempenho, como pode ser observado através da Figura 10, sendo que a versão 2.0 do OpenMPI (ROMIO) apresentou os melhores resultados finais. O salto de desempenho foi de 1,7 MB/s para 1402 MB/s, apresentando um ganho de desempenho de 818 vezes.

Os resultados do IOR demonstraram que o baixo desempenho das operações de E/S do OLAM está relacionado ao grande número de requisições de tamanhos pequenos, especialmente quando utiliza o BullxMPI, que apresentou o pior desempenho nas operações de escrita. Também podemos observar que os acessos desalinhados de tamanhos pequenos impactaram negativamente o desempenho apenas das versões 2.0 e 2.1 do OpenMPI ao utilizar o OMPIO. As implementações MPI que utilizaram o ROMIO tiveram um desempenho melhor ao aumentar o tamanho da transferência de 1024 bytes para 1312 bytes.

Em relação ao desempenho das APIs utilizadas, o HDF5 foi geralmente inferior ao MPI-IO. Isso possivelmente está relacionado ao *overhead* das operações de E/S ao utilizar a biblioteca.

5.5 Investigação do Tempo de E/S

Buscando melhor compreender o tempo gasto nas operações de E/S coletivas já que, conforme apontado pelos resultados apresentados com o IOR na Seção 5.3, elas são as responsáveis pelas diferenças observadas nos tempos entre as diferentes implementações MPI, desenvolvemos um *microbenchmark* que mede cada etapa de uma operação de escrita coletiva.

Esse *microbenchmark* foi construído tendo como base a implementação de operações de escritas coletivas utilizada pelo ROMIO, onde as diferentes fases são implementadas utilizando outras rotinas MPI (e.g. MPI_Allgather e MPI_Allreduce). Assim, é possível determinar o custo destas operações no contexto da escrita coletiva, determinando quais apresentam diferenças no tempo devido a implementação utilizada.

O ROMIO utiliza o conceito de "*Two-Phase I/O"* [15]. No caso de escritas, a primeira fase consiste na troca de mensagens relativas aos *offsets* de início e fim acessados por processo, a quantidade e tamanhos de acessos e por fim a troca de dados entre os agregadores e os processos com base nas informações obtidas até então. A segunda fase, por

sua vez, é onde a operação de escrita é feita ao sistema de arquivo paralelo pelos agregadores. A Figura 11 detalha estas etapas.

Etapa 1	Troca de mensagens entre todos os processos para que cada um tenha conhecimento do *offset* de início e fim das requisições (`MPI_Allgather`)
Etapa 2	Troca de mensagens entre todos os processos para que cada um tenha conhecimento do número e tamanhos dos acessos (`MPI_Allreduce`)
Etapa 3	Troca de mensagens entre agregadores e processos para comunicar os *offsets* e tamanhos do acesso (`MPI_Isend` e `MPI_Irecv`)
Etapa 4	Troca de mensagens entre agregadores e processos para que cada agregador obtenha os dados para realizar a operação de escrita (`MPI_Isend` e `MPI_Irecv`)
Etapa 5	Processos agregadores executam a operação de I/O no sistema de arquivos paralelos (`MPI_File_write_at`)

Figura 11. Etapas do *microbenchmark* que simula operações coletivas

Este experimento foi configurado para realizar requisições de 1312 byes e utilizou 10 nós (sdumont[5004-5013]) e 24 cores por nó, totalizando 240 cores. A Figura 12 ilustra a mediana de 5 repetições. É possível ver que a maior parte do tempo é gasto pela Etapa 3, onde há a comunicação entre os processos e agregadores para obter os *offsets* e tamanhos de acesso. Além disso, nesta mesma etapa, o BullxMPI apresenta um tempo significativamente maior que as demais implementações MPI. O mesmo pode ser visto, em escala menor, na Etapa 4. Este resultado é inesperado pois ambas as fases são essencialmente iguais, a única diferença é que na Etapa 4 as mensagens são maiores do que as utilizadas na Etapa 3.

Desta forma, foi possível comparar os resultados desta análise com os obtidos com o OLAM, demonstrando que as diferenças observadas entre as implementações MPI consideradas neste estudo estão relacionadas com a troca de pequenas mensagens (para comunicar os offsets e tamanhos de acessos) entre os processos agregadores e os demais processos envolvidos na comunicação coletiva.

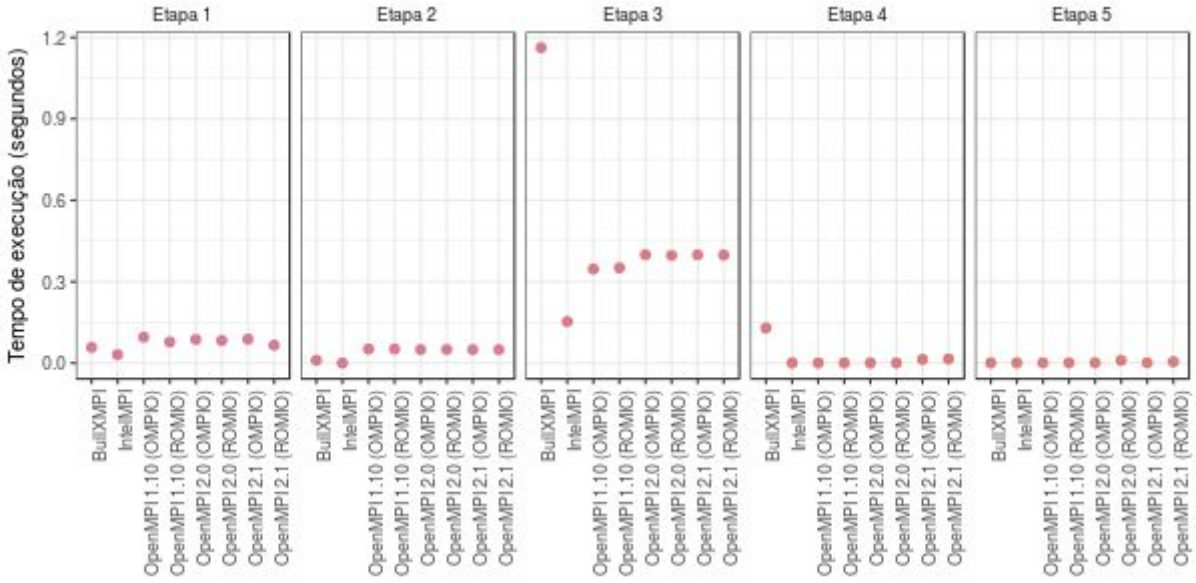


Figura 12. Tempo gasto em cada uma das etapas da operação coletiva

6. CONCLUSÃO E TRABALHOS FUTUROS

Neste trabalho buscou-se investigar as operações de E/S da aplicação OLAM, identificando possíveis gargalos. As análises iniciais apontaram para uma grande variação no desempenho do OLAM ao utilizar diferentes implementações MPI, com o BullxMPI apresentando os piores resultados. Através do Darshan foi possível obter o perfil das operações de E/S e constatamos que o OLAM realiza um grande número de requisições de tamanhos pequenos, em torno de 1300 bytes. Esse tamanho impacta diretamente o desempenho e a escalabilidade do OLAM, principalmente quando é utilizado com o BullxMPI, conforme foi comprovado através dos resultados obtidos com o IOR. Buscando identificar a causa deste comportamento criou-se um microbenchmark para cronometrar as diferentes etapas da comunicação coletiva, conforme implementada pelo ROMIO. Foi possível observar que o problema do BullxMPI aparentemente está relacionado com a troca de mensagens entre os processos agregadores e os demais processos da aplicação, conforme descrito na seção 5.5. O mesmo baixo desempenho e a alta variação entre as implementações MPI não foi observada no benchmark BT-IO, que utiliza tamanhos muito maiores de requisições.

Analisando também o impacto de diferentes tamanhos das requisições com o IOR foi possível constatar que ao aumentar o tamanho da requisição o desempenho das operações de E/S também aumenta. Foram observados ganhos de até 818 vezes para leitura (OpenMPI-2.0 com ROMIO) e desempenho de 1014 MB/s para a escrita (IntelMPI). Também foi possível observar o overhead em se utilizar a biblioteca HDF5.

Analisando também o impacto dos tamanhos das requisições feitas pelo OLAM, simulando tal comportamento com o IOR foi possível constatar que o tamanho da requisição tem grande influência no desempenho da aplicação. Foram observados ganhos de até 818 vezes para leitura (OpenMPI-2.0 com ROMIO) e desempenho de 1014 MB/s para a escrita (IntelMPI). Também foi possível observar o *overhead* em se utilizar a biblioteca HDF5. Buscamos também verificar se outros usuários que utilizam a implementação BullxMPI reportaram semelhante comportamento, porém não foi possível encontrar nenhum relato neste sentido. Desta forma, reportamos a investigação conduzida neste trabalho, juntamente com suas conclusões à Bull. Neste sentido também será sugerido aos usuários do OLAM e de outras aplicações que também fazem escritas pequenas utilizando operações coletivas no

SDumont que, momentaneamente, não utilizem tal implementação MPI para que não sejam impactadas no seu tempo de execução.

Desta forma, como trabalhos futuros, pretende-se modificar o OLAM para realizar requisições maiores, buscando determinar o melhor tamanho do ponto de vista da aplicação e sistema de arquivos. Além disso, pretende-se considerar os benefícios e impactos de outros *hints* no tempo das operações de E/S da aplicação. Para finalizar, o estudo da escalabilidade das operações de E/S ao utilizar um número maior de nós e uma maior quantidade maior de dados a serem escritos pode demonstrar outros problemas no desempenho das operações de escrita do OLAM. Neste sentido, poderia-se considerar utilizar os SSDs locais, presentes nos nós de computação do SDumond, como *buffers* temporários para que as operações de I/O sejam realizadas localmente, de forma mais rápida, e somente posteriormente escrever tais arquivos no sistema de arquivos paralelo.

BIBLIOGRAFIA

[1] OLAM Model: https://sourceforge.net/projects/olam-model/. Acessado em agosto de 2017.

[2] High Performance Computing for Energy: https://hpc4e.eu Acessado em agosto de 2017.

[3] Santos Dumont: http://sdumont.lncc.br. Acessado em agosto de 2017.

[4] SNYDER, Shane; CARNS, Philip; HARMS, Kevin; ROSS, Robert. "Modular HPC I/O characterization with Darshan". In Proceedings of the 5th Workshop on Extreme-Scale Programming Tools (ESPT '16). IEEE Press, Piscataway, NJ, USA, pp. 9-17.

[5] Pielke, R.A. et al. (1992). "A Comprehensive Meteorological Modeling System – RAMS", in: Meteorology and Atmospheric Physics. 49(1), pp. 69-91

[6] Walko, R.L.& Avissar, R. The Ocean-Land-Atmosphere Model (OLAM). Part I: Shallow-Water Tests. Monthly Weather Review 136:4033-4044, 2008

[7] The HDF Group. 1997-2016. Hierarchical Data Format, version 5. (1997-2016).

[8] SDumont: Projetos em Andamento: http://sdumont.lncc.br/projects_ongoing.php?pg=projects#

[9] Pedro Leite da Silva Dias. Instituição: IAG/USP Endereço do CV Lattes: http://lattes.cnpq.br/9273702863744424

[10] LUSTRE home page. http://www.lustre.org/. Acessado em julho de 2017.

[11] Ocean Land Atmosphere Model: https://sourceforge.net/projects/olam-model/files/

[12] Wong, Parkson; VanderWijngaart, Rob F. & Biegel, Bryan A. "NAS Parallel Benchmarks I/O Version 2.4. 2.4", in NAS Technical Report NAS-03-002. NASA Ames Research Center, Moffett Field, CA 94035-1000, January 2003.

[13] Nas Parallel Benchmarks (NPB) - NAS Home: https://www.nas.nasa.gov/publications/npb.html Acessado em agosto de 2017.

[14] Shan, Hongzhang; & Shalf, John. (2007). "Using IOR to analyze the I/O Performance for HPC Platforms". Lawrence Berkeley National Laboratory. Disponível em: http://escholarship.org/uc/item/9111c60j

[15] Thakur, R., Lusk, E., & Gropp, W. (1997). "Users guide for ROMIO: A high-performance, portable MPI-IO implementation". United States. doi:10.2172/564273

[16] Chaarawi M., Gabriel E., Keller R., Graham R.L., Bosilca G., Dongarra J.J. (2011) "OMPIO: A Modular Software Architecture for MPI I/O." In: Cotronis Y., Danalis A., Nikolopoulos D.S., Dongarra J. (eds) Recent Advances in the Message Passing Interface. EuroMPI 2011. Lecture Notes in Computer Science, vol 6960. Springer, Berlin, Heidelberg