Universidade Federal do Pampa
Matheus Beniz Bieger

Estudo das Rotinas MPI do Modelo Atmosférico OLAM através de Ferramentas de *Tracing* e Visualização

Alegrete
2015

Matheus Beniz Bieger

Estudo das Rotinas MPI do Modelo Atmosférico OLAM através de Ferramentas de *Tracing* e Visualização

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Ciência da Computação da Universidade Federal do Pampa como requisito parcial para a obtenção do título de Bacharel em Ciência da Computação.

Orientador: Claudio Schepke

Alegrete

2015

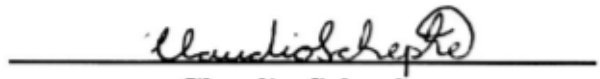
Matheus Beniz Bieger

Estudo das Rotinas MPI do Modelo Atmosférico OLAM através de Ferramentas de *Tracing* e Visualização

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Ciência da Computação da Universidade Federal do Pampa como requisito parcial para a obtenção do título de Bacharel em Ciência da Computação.

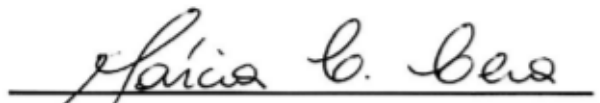
Trabalho de Conclusão de Curso defendido e aprovado em 30 de Novembro de 2015.

Banca examinadora:



Claudio Schepke
Orientador

Arthur Francisco Lorenzon
Universidade Federal do Rio Grande do Sul



Márcia Cristina Cera
Universidade Federal do Pampa

Dedico este trabalho à Micheli Weyh.
(in memoriam)

Agradecimentos

À minha família, por acreditar e investir em mim.

À minha mãe, Izaura, por ser esta pessoa determinada, que não mede esforços para ajudar os filhos.

Ao meu pai, Renato, pelo apoio e conselhos ao longo da minha vida.

Ao meu irmão Thiago, que sempre foi fonte de inspiração e referência intelectual.

Aos meus professores, por exercerem um papel fundamental na minha formação.

Ao meu orientador, Claudio Schepke, pelas orientações e aconselhamentos.

Aos meus amigos. Dos mais antigos aos que fiz durante a graduação. Todos são especiais para mim e de alguma forma contribuiram com a minha formação.

À Deus, por ser um alicerce na minha vida.

À todos que participaram direta ou indiretamente desta conquista.

"Eu Acredito, que às vezes são as pessoas que ninguém espera nada que fazem as coisas que ninguém consegue imaginar."
(Alan Turing)

Resumo

A previsão do tempo e clima é um fator relevante para sociedade pois impacta diretamente em importantes fatores econômicos e sociais. As previsões são realizadas através de modelos atmosféricos que simulam as condições atmosféricas correntes. Por utilizar uma quantidade considerável de dados, a execução deste modelo tende a ser lenta, podendo levar dias. Neste contexto, este trabalho realiza uma análise das rotinas MPI implementadas no modelo atmosférico OLAM (*Ocean-Land Atmosphere Model*). Para tanto, utilizou-se ferramentas de perfilamento, rastreamento e visualização de rotinas MPI. Como resultados, analisou-se através de balanceamento de carga, que a rotina `MPI_Wait` teve o maior impacto no tempo de processamento das rotinas MPI. Representou em torno de 100% no tempo de processamento dentre todas as rotinas MPI analisadas. A compreensão do funcionamento da implementação paralela e a identificação de rotinas que representam gargalo de desempenho contribuem para a melhoria do modelo.

Palavras-chave: *Ocean-Land Atmosphere Model (OLAM). Message Passing Interface (MPI). Profiling.*

Abstract

The weather and climate is an important factor in society as it impacts directly on important economic and social factors. Predictions are made using atmospheric models that simulate the current weather conditions. To use a considerable amount of data, the execution of this model tends to be slow and could take days. In this context, this paper makes an analysis of the MPI routines implemented in the atmospheric model OLAM (*Ocean-Land Atmosphere Model*). To this end, we used profiling tools, tracking and viewing MPI routines. As a result, it was examined through load balancing, the routine `MPI_Wait` had the greatest impact on the processing time of the MPI routines. Represented around 100% in processing time among all MPI routines analyzed. Understanding the operation of the parallel implementation, and the identification of routines that represent performance bottleneck contribute to the improvement of the model.

Key-words:Ocean-Land Atmosphere Model (OLAM). Message Passing Interface (MPI). Profiling.

Lista de ilustrações

Figura 1 – Estrutura de um Icosaedro 26
Figura 2 – Icosaedro com subdivisões da malha global 26
Figura 3 – Representação dos níveis verticais do modelo OLAM 27
Figura 4 – Representação do refinamento de malha dinâmico 28
Figura 5 – Representação do processo de execução do modelo OLAM 29
Figura 6 – Representação da Comunicação entre dois Processos 30
Figura 7 – Estrutura de um programa em MPI 33
Figura 8 – Representação dos elementos do modelo OLAM 37
Figura 9 – Tela do programa Intel Trace Analyzer 45
Figura 10 – Tempo de computação e Rotinas MPI do modelo OLAM 48
Figura 11 – Balanceamento de carga - 2 Processos 51
Figura 12 – Balanceamento de Carga - 4 processos 51
Figura 13 – Balanceamento de carga - 8 processos 52
Figura 14 – Balanceamento de carga - 16 processos 53
Figura 15 – Visualização dos Arquivos de Traços MPI - 2 processos 54
Figura 16 – Visualização dos Arquivos de Traços MPI - 4 processos 54

Lista de tabelas

Tabela 1 – Estrutura de dados do modelo OLAM 38
Tabela 2 – Estrutura de Dados do Buffer 38
Tabela 3 – Configuração das Execuções do modelo OLAM 46
Tabela 4 – Percentual de Computação dos módulos e bibliotecas no Modelo OLAM 47
Tabela 5 – Desbalanceamento no tempo de execuçao das Rotinas MPI 48
Tabela 6 – Troca de mensagens entre 4 processos 49
Tabela 7 – Troca de mensagens entre 8 processos 49
Tabela 8 – Troca de mensagens entre 16 processos 50
Tabela 9 – Referência entre cores e rotinas 53

Lista de siglas

CPU *Central Processing Unit*

MPI *Message-Passing Interface*

OLAM *Ocean-Land Atmosphere Model*

PAD Processamento de Alto Desempenho

RAMS *Regional Atmospheric Modeling System*

Sumário

1 INTRODUÇÃO 23
1.1 Problema 23
1.2 Objetivos 24
1.3 Estrutura do Trabalho 24

2 OCEAN-LAND ATMOSPHERE MODEL 25
2.1 Principais Características 25
2.2 Estrutura da Malha 25
2.3 Refinamento Dinâmico 27
2.4 Implementação 28
2.5 Execução do modelo OLAM 28
2.6 Comunicações MPI no modelo OLAM 29
2.7 Considerações sobre o Capítulo 30

3 MESSAGE PASSING INTERFACE 31
3.1 Visão Geral 31
3.2 Conceitos básicos 32
3.3 Comunicações MPI 32
3.4 Principais Rotinas MPI do Modelo OLAM 33
3.5 Considerações sobre o Capítulo 36

4 IMPLEMENTAÇÃO MPI NO MODELO OLAM 37
4.1 Principais módulos 37
4.2 Rotinas de inicialização e Finalização 39
4.3 Rotina de alocação de memória 40
4.4 Rotinas de Comunicação 40
4.4.1 Rotinas de Envio 40
4.4.2 Rotinas de Recebimento 41
4.5 Considerações sobre o Capítulo 42

5 MATERIAIS E MÉTODOS 43
5.1 Intel Parallel Studio XE 43
5.1.1 Intel Vtune Amplifier XE 43
5.1.2 Intel Trace Analyzer and Collector 44
5.1.3 Collector 44
5.1.4 Trace Analyzer 44

5.2 Configurações do modelo OLAM para execução 45
5.3 Ambiente de Testes 46
5.4 Considerações sobre o Capítulo 46

6 RESULTADOS OBTIDOS 47
6.1 Análise do Perfil do modelo OLAM 47
6.2 Análise das Rotinas MPI do Modelo OLAM 47
6.2.1 Matriz de Troca de Mensagens entre Processos 49
6.2.2 Balanceamento de Carga entre Processos 50
6.2.3 Perfil das Comunicações MPI 52

7 CONSIDERAÇÕES FINAIS 55

Referências 57

Índice 59

1 Introdução

As transformações socioeconômicas que ocorreram nos últimos séculos, bem como o avanço das ciências e a descoberta de novas tecnologias causaram profundas mudanças no modo de ver e interagir do ser humano com a natureza. A necessidade de entender os fenômenos naturais e de prevê-los com maior precisão sempre permeou entre as preocupações humanas sendo decisiva, muitas vezes até mesmo para a sobrevivência de povos e comunidades.

As recorrentes discussões sobre mudanças climáticas e a previsão de cenários futuros podem ser qualificadas com modelos numéricos de previsão de tempo e clima que são amplamente utilizadas atualmente para esse propósito. Modelos de previsão numérica auxiliam na detecção e prevenção de catástrofes, aquecimento global, temperatura, entre outros fatores relevantes para a comunidade global (SILVA R.;DIAS, 2009).

A utilização de processamento de alto desempenho para suprir as necessidades computacionais de modelos numéricos de previsão climática têm sido largamente utilizada no contexto atual. Deste modo, busca-se obter o máximo de retorno de recursos físicos (*hardware*) através do uso de interfaces de programação paralela.

Neste trabalho, utilizou-se interface de programação paralela - *Message-Passing Interface* (MPI). Esta interface tira proveito das *Central Processing Units* (CPUs) e de memórias compartilhada e distribuída. Pode prover bons ganhos de desempenho em aplicações científicas por fornecer rotinas que permitem a distribuição da computação entre diferentes processos, que podem ser executados em diferentes computadores.

1.1 Problema

Os modelos atmosféricos, responsáveis pela previsão do tempo, são *softwares* de modelagem numérica que efetuam simulações do mundo real através de equações físicas. Essas equações retratam fenômenos físicos, como: conservação de massa e *momentum*, energia, densidade, temperatura potencial, entre outros (SCHEPKE, 2012).

A simulação de fenômenos físicos trabalha com um grande volume de dados, o que torna a execução do modelo lenta. Esse processo pode levar dias para executar. Neste contexto, buscam-se alternativas para para melhorar o desempenho deste tipo de aplicação através do emprego de técnicas de programação paralela e alto desempenho.

1.2 Objetivos

O objetivo principal deste trabalho é realizar uma análise das rotinas MPI no modelo atmosfério *Ocean-Land Atmosphere Model* (OLAM). Para tanto, utilizou-se ferramentas de análise como: *profiler*, gerador e visualizador de traços. A análise busca identificar as rotinas MPI que têm maior impacto no desempenho do modelo.

Para prover esta análise são empregados seguintes os objetivos específicos:

- Estudo do perfil da aplicação com diferentes processos;
- Identificação dos pontos críticos (*hotspots*) na execução;
- Identificação do padrão da troca de mensagens;
- Análise o balanceamento de carga da aplicação;

1.3 Estrutura do Trabalho

Este trabalho está estruturado em sete capítulos.

Capítulo 2: Apresenta o modelo atmosférico OLAM e suas características.

Capítulo 3: Descreve em detalhes a interface de programação paralela MPI.

Capítulo 4: Descreve a implementação MPI do modelo OLAM.

Capítulo 5: Descreve os métodos e ferramentas utilizadas no trabalho.

Capítulo 6: Apresenta e analisa os resultados obtidos.

Capítulo 7: Conclui o trabalho e apresenta as considerações finais.

2 Ocean-Land Atmosphere Model

Este capítulo descreve o modelo atmosférico OLAM e suas principais características estruturais. Está dividido em sete seções. A seção 2.1, descreve as principais características do modelo. A seção 2.2, descreve a estrutura da malha. A seção 2.3, demonstra como é realizado o refinamento dinâmico e a seção 2.4 descreve detalhes de implementação.A seção 2.5 aborda os aspectos relacionados a execução do modelo e a seção 2.6 descreve como é realizada a comunicação entre processos. Por fim, a seção 2.7 descreve as considerações finais sobre o capítulo.

2.1 Principais Características

O modelo OLAM é um modelo de simulação numérica baseado no *Regional Atmospheric Modeling System* (RAMS) (PIELKE R., 1992). Ele estende as capacidades do RAMS para modelagem de um domínio global e tira proveito de várias de suas características como: parametrizações, métodos de inicialização, assimilação dos dados, estrutura lógica e do código e formatos de entrada e saída (WALKO R.;AVISSAR, 2008). O modelo pode representar fenômenos de escala global e regional simultaneamente, além de possuir processos dinâmicos de configuração da grade, estrutura de memória e técnicas de solução numérica (SILVA R.;DIAS, 2009). Além disso, o OLAM utiliza técnicas de discretização de volumes finitos (MARSHALL J.; HEISEY C., 1997) e possui uma compreensão total das equações de Navier-Stokes.

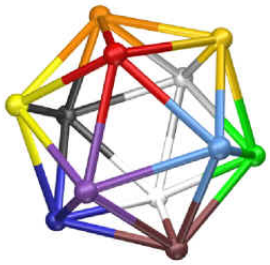
A principal característica do modelo OLAM é a capacidade de simulação de fenômenos em escala global e local. Para tanto, o modelo utiliza um refinamento dinâmico das malhas (WALKO R.;AVISSAR, 2008). Esse refinamento baseia-se na subdivisão dos triângulos que compõe a região desejada em mais triângulos, aumentando a resolução dessa região. Quanto menor for a distância horizontal entre dois triângulos, maior é a resolução do modelo e mais dados são computados. O refinamento dinâmico pode ocorrer em vários níveis, permitindo que sejam refinados dois ou mais pontos distintos no globo simultaneamente.

2.2 Estrutura da Malha

A estrutura global do modelo OLAM consiste de uma malha não estruturada de células triangulares. Estas células são dispostas horizontalmente sobre o globo terrestre e são constituidas de camadas verticais. A construção da malha possui um formato inicial de icosaedro. Um icosaedro é uma estrutura regular que consiste em 20 triângulos

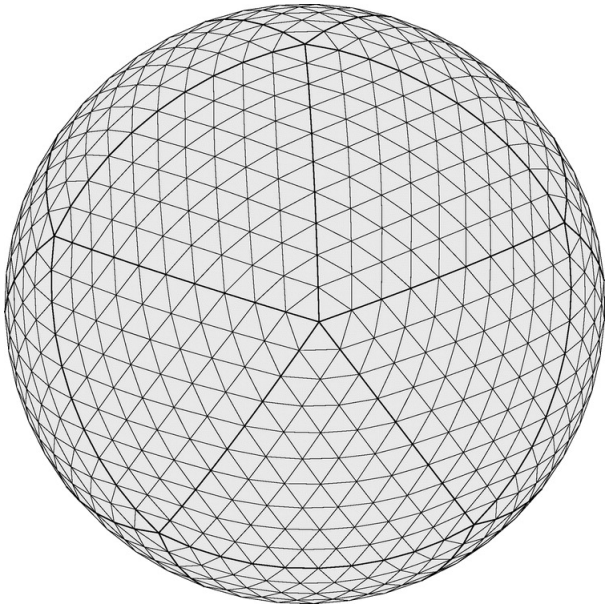
equiláteros, possui 30 arestas e 12 vértices. Cada vértice reúne 5 arestas. A Figura 1 e Figura 2 representam a estrutura do icosaedro e as subdivisões de seus triângulos, respectivamente.

Figura 1 – Estrutura de um Icosaedro



Fonte: Schepke (2012)

Figura 2 – Icosaedro com subdivisões da malha global

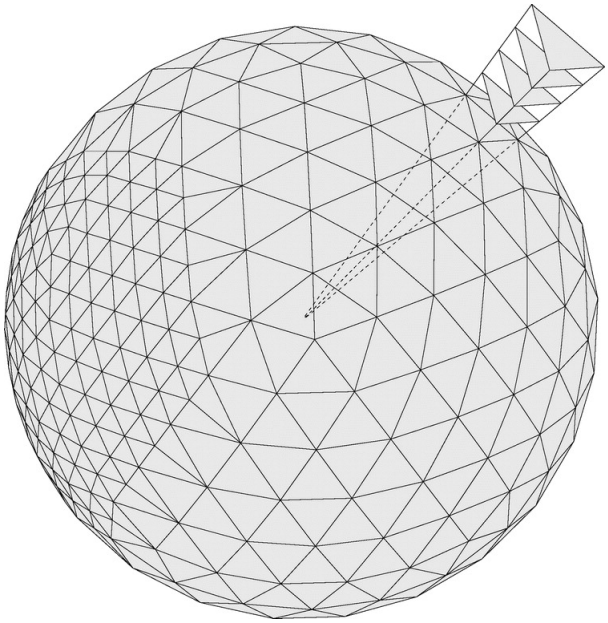


Fonte: Walko R.;Avissar (2008)

Com a divisão das células triangulares em camadas verticais, essas células formam um prisma cuja base encontra-se no centro do globo terrestre. Na Figura 3 é demonstrado um exemplo destes prismas.

Cada célula triangular representada no globo pode ser dividida, melhorando a resolução horizontal da malha. Como a malha não é estruturada, os triângulos não tem sua distribuição horizontal bem definida e suas coordenadas são armazenadas computaci-

Figura 3 – Representação dos níveis verticais do modelo OLAM



Fonte: Walko R.;Avissar (2008)

onalmente para serem posteriormente utilizadas na integração numérica (SILVA R.;DIAS, 2009).

2.3 Refinamento Dinâmico

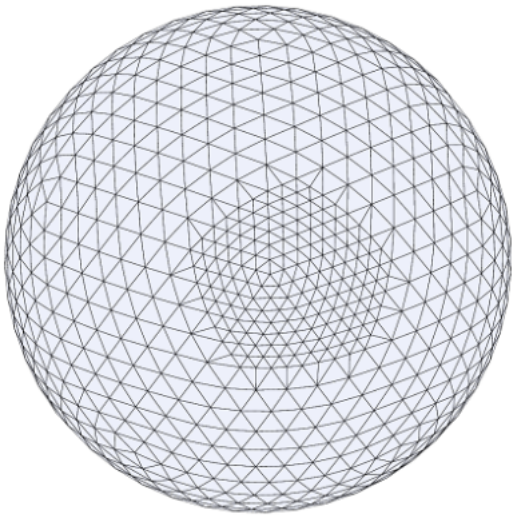
O modelo OLAM possui uma malha não estruturada, em que cada índice é representado por um único índice horizontal (WALKO R.;AVISSAR, 2008). Isso deve-se ao fato do refinamento de malhas local não poder ocorrer em uma malha estruturada.

O refinamento da malha local sempre ocorre após a construção da malha global. Quando realizado, este refinamento seque um padrão. Segundo (SCHEPKE, 2012), cada triângulo deve compartilhar uma aresta com outros três triângulos.

Na Figura 4, é representado um refinamento local de uma malha. No exemplo, a resolução é duas vezes maior que a original. Isso ocorre com a divisão de cada triângulo em dois menores. Para atender esse critério, é necessária a inserção de novas arestas, afim de manter o padrão de refinamento, ou seja, três triângulos vizinhos a cada triângulo.

Para realização da transição entre uma resolução mais grossa, triângulos maiores, para um resolução mais fina, triângulos menores, são utilizados vértices com mais de seis arestas no lado grosso e menos de seis no lado fino da transição. Além disso, quando for necessário um refinamento mais intensivo, este pode ser obtido utilizando vértices com mais de 7 e menos de 5 arestas, porém com redução da acurácia das simulações numéricas.

Figura 4 – Representação do refinamento de malha dinâmico



Fonte: Walko R.;Avissar (2008)

2.4 Implementação

No projeto de implementação do modelo OLAM, é realizada a divisão em três partes: a parametrização da inicialização, a simulação em si e a escrita dos resultados de saída.

Segundo Schepke (2012), na inicialização ocorre o pré-processamento, onde são lidas as configurações iniciais e é realizado o processamento de informações sobre a Terra, vegetação, solo e mar e são feitas alocações de memória para a execução.

O principal ponto da implementação está na parte iterativa, que engloba a simulação das parametrizações físicas. Essas parametrizações são similares ao modelo atmosférico RAMS, no qual o OLAM é baseado. É na parte iterativa que as informações calculadas no pré-processamento são inseridas na execução. Ao final de cada iteração é calculado o tempo decorrido.

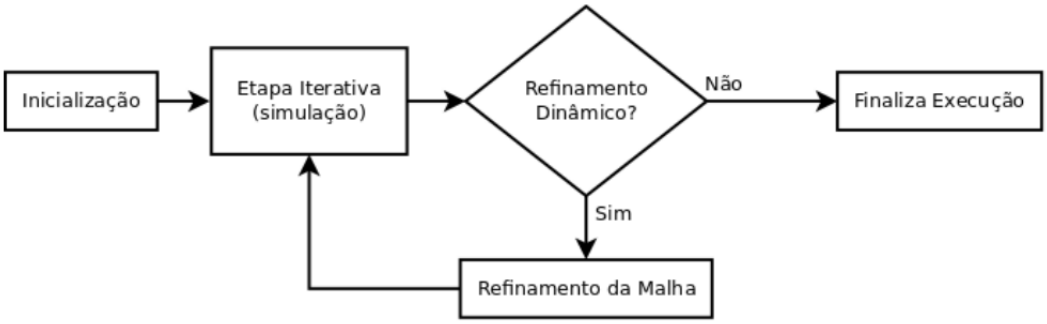
Após as iterações, os resultados são escritos em arquivos específicos. Esses arquivos armazenam valores das condições físicas da atmosfera para uma fatia de tempo determinada.

2.5 Execução do modelo OLAM

O processo de execução do modelo OLAM é realizado basicamente em 4 etapas: Inicialização, iteração, refinamento dinâmico e finalização. O refinamento dinâmico é

opcional, ocorrendo somente quando solicitado pelo operador. Na Figura 5, é ilustrado um fluxograma que representa os passos que ocorrem na execução do modelo OLAM.

Figura 5 – Representação do processo de execução do modelo OLAM



Fonte: Vargas e Schepke (2014)

Segundo Silva R.;Dias (2009), na inicialização são realizadas a leitura das opções do usuário, a configuração da grade e a definição das opções de refinamento. Além disso, é realizado o processo de alocação de memória e o pré-processamento da topologia, vegetação, solos, entre outros fatores.

Na parte iterativa, são realizados todos os cálculos relativos a simulação numérica, sendo a parte que demanda maior desempenho, por ser bastante demorada.

A finalização faz o registro da execução em arquivos de saída. É gerado um arquivo por processo em execução. Estes arquivos possuem as mesmas informações que são dispostas na saída padrão, porém no contexto de um processo em específico.

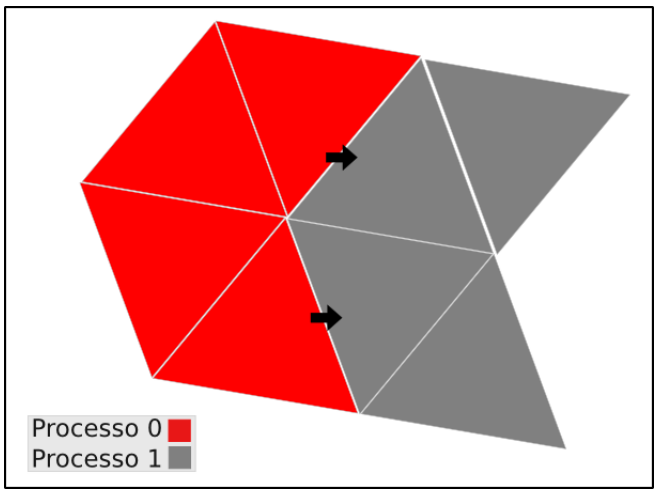
2.6 Comunicações MPI no modelo OLAM

As comunicações entre os processo MPI do modelo OLAM são realizadas basicamente por três rotinas de envio e três de recebimento. Segundo Schepke (2012) as rotinas de comunicação `mpi_send/recv_u()` e `mpi_send/recv_uf()` são responsáveis por associar dados das propriedades físicas às arestas da malha e as rotinas `mpi_send/recv_w` associam as propriedades aos triângulos.

A Figura 6, demonstra como é realizada a comunicação entre diferentes processos no modelo.

Na Figura, que representa uma pequena parte do globo terrestre, os triângulos em vermelho e cinza representam diferentes processos. As setas indicam onde acontece a comunicação. A troca de mensagem sempre será realizada em uma região de borda entre dois processos.

Figura 6 – Representação da Comunicação entre dois Processos



Embora representados de maneira horizontal, estes triângulos são prismas. Possuem um determinado número de níveis verticais que representam pontos específicos da atmosfera. Neste sentido, para a realização da comunicação, em um primeiro momento é alocado um espaço na memória para armazenar as propriedades de cada prisma. Em seguida, a comunicação é realizada.

2.7 Considerações sobre o Capítulo

Este capítulo apresentou as principais características relacionadas ao modelo OLAM. Descreveu as principais estruturas, as resoluções e detalhes técnicos. Também descreveu o fluxo de execução e o refinamento de malhas.

A compreensão do aspecto estrutural do modelo é importante para entender como ocorrem as comunicações MPI. No Capítulo 3, é feita uma abordagem sobre a interface de programação MPI e as principais rotinas relacionadas ao modelo. No Capítulo 4 são fornecidos detalhes da implementação paralela do modelo, e a sua compreensão depende do entendimento da estrutura do modelo OLAM.

3 Message Passing Interface

Em Processamento de Alto Desempenho (PAD) existem inúmeras interfaces de programação que auxiliam na parelização de programas. Neste trabalho, foca-se na interface MPI. A seção 3.1 apresenta uma visão geral sobre a interface. A seção 3.2 descreve os conceitos básicos. A seção 3.3 são apresentados os principais tipos de comunicação. As principais rotinas relacionadas ao modelo OLAM são descritas no seção 3.4. Por fim, o seção 3.5 descreve algumas considerações finais sobre o capítulo e a relação com os próximos capítulos.

3.1 Visão Geral

Segundo Barney (2014), MPI é uma padronização de uma interface de troca de mensagens, *message passing*, em máquinas paralelas com memória distribuída ou compartilhada. Essa especificação não deve ser confundida com um compilador ou biblioteca, pois apenas estabelece padrões de como uma biblioteca deve ser.

Existem diversas implementações de bibliotecas MPI disponíveis no mercado, e atualmente, possui suporte para uma grande variedade de linguagens de programação. Entre as principais bibliotecas atuais estão *Open MPI*, atualmente na versão 1.10.1 e *MPICH*, na versão 3.2.

Em MPI, todo paralelismo é explícito. O programador é responsável para identificação dos pontos críticos para a paralelização e pela implementação das rotinas MPI na aplicação (SNIR et al., 1998).

Conforme descrito por Barney (2014), é possível citar seis principais razões para a utilização dessa interface:

- **Padronização:** MPI é a única biblioteca de troca de mensagens que pode ser considerada padrão. É suportada pela maioria das plataformas de alto desempenho disponíveis e substitui praticamente todas bibliotecas de troca de mensagens anteriores;
- **Portabilidade:** Para portar uma aplicação em MPI são necessárias poucas modificações no código fonte;
- **Desempenho:** Tem boa oportunidade de obter desempenho paralelo, tendo em vista que os fabricantes tem produzido *hardware* compatíveis;
- **Funcionalidade:** Na versão atual, MPI-3, a interface possui mais de 440 rotinas;

- **Implementação:** Há uma variedade enorme de implementações dessa especificação disponíveis no mercado;

3.2 Conceitos básicos

Em MPI, as tarefas são divididas para executar em diferentes processos. Segundo Gropp, Lusk e Skjellum (1999), um par (*group*, *rank*) especifica um processo, mas um processo não determina um único par, uma vez que pode pertencer a diversos *groups*.

O *rank* é um número iniciado em zero que identifica unicamente um determinado processo. Quando os processos são iniciados, é função do sistema atribuir um valor para cada um deles.

Um *group*, é um conjunto ordenado de N processos. Todo grupo está associado a um *communicator*, e por padrão, todos os processos pertencem inicialmente a um comunicador chamado `MPI_COMM_WORLD`.

A função do *communicator*, ou comunicador, é a criação de um contexto entre os processos. Ele define uma coleção de processos que poderão se comunicar entre si (PACHECO, 1996). Dessa maneira, há uma garantia de comunicação segura e os problemas de envios de mensagens entre processos são evitados.

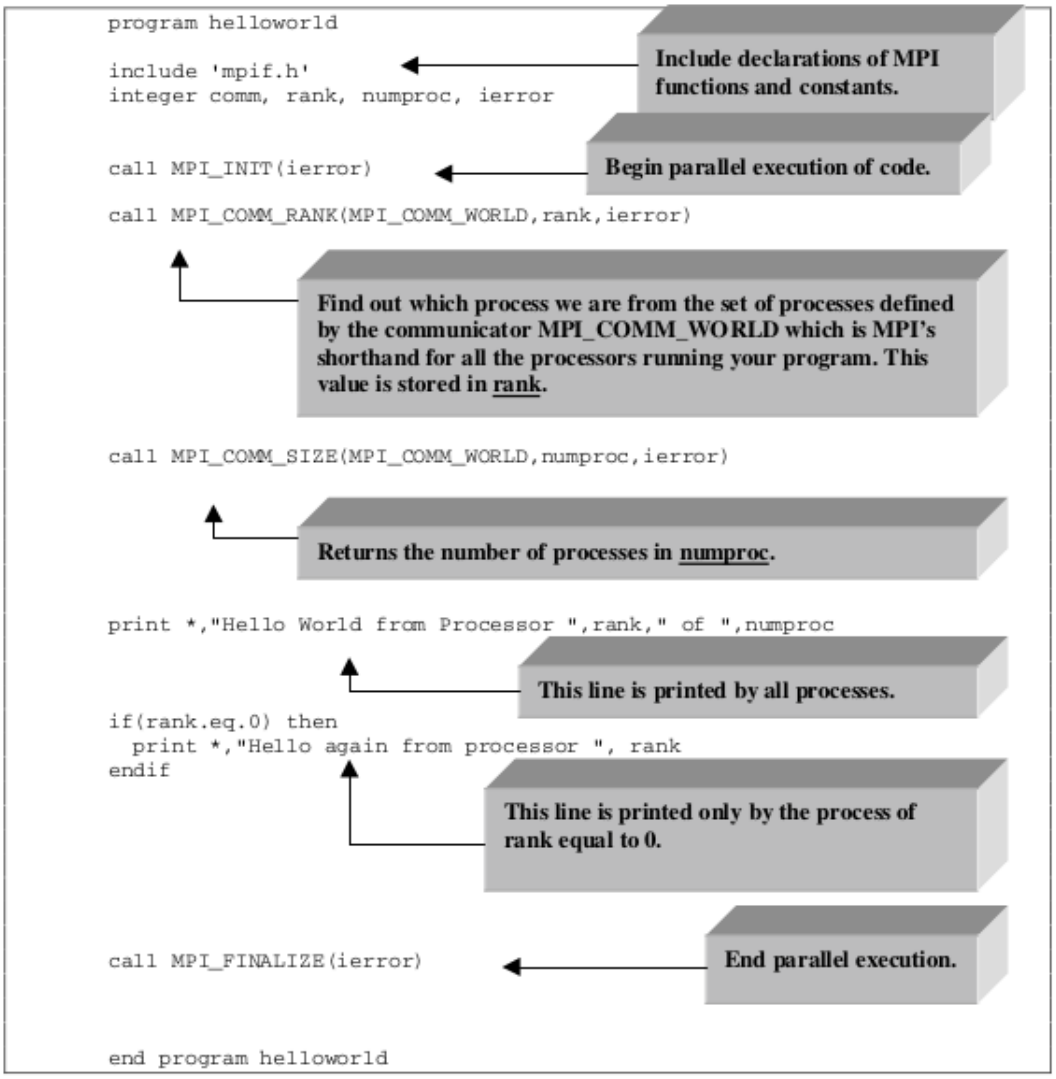
Na Figura 7 é ilustrada a estrutura básica de um programa MPI. O algoritmo define uma região paralela e identifica o processo atual através do identificador (*rank*). Também verifica quantos processos estão executando e finaliza a execução.

3.3 Comunicações MPI

- **Comunicação bloqueante e não bloqueante:** Segundo CEPANAD (2012), uma comunicação MPI bloqueante ocorre quando a finalização da rotina depende de eventos externos. Nesse caso, espera-se por uma determinada ação antes de liberar a continuação do processamento. Em contrapartida, as rotinas de comunicação não-bloqueantes podem finalizar a execução sem a dependência de outros eventos. O processo continua sendo executado normalmente, sem esperar por nenhuma ação externa.

- **Comunicação síncrona e assíncrona:** As comunicações assíncronas ocorrem no processo de envio de mensagens. Nesse caso, o processo que realiza o envio não aguarda um sinal de recebimento do destinatário. No caso da comunicação síncrona, o processo que envia a mensagem não retorna a sua execução normal até que haja um sinal de recebimento por parte do destinatário (GROPP; LUSK; SKJELLUM, 1999).

Figura 7 – Estrutura de um programa em MPI



Fonte: Pineda e Smith (1998)

- **Comunicação ponto a ponto e coletiva:** A comunicação ponto a ponto ocorre quando é realizada a troca de mensagens entre somente dois processos. Um processo realiza o envio e o outro o recebimento. Nesse caso, a comunicação pode ser bloqueante ou não bloqueante.

  A comunicação coletiva ocorre quando um processo emissor realiza o envio de uma mensagens para vários destinatários. Todos os processos em um mesmo comunicador irão receber a mensagem. Neste sentido, este tipo de comunicação é muito utilizada quando faz-se necessário a coordenação de um grupo de processos (SNIR, 1998).

3.4 Principais Rotinas MPI do Modelo OLAM

Nesta seção são abordadas todas as principais rotinas MPI FORTRAN no contexto do modelo OLAM. Além disso, são demonstrados os parâmetros utilizados em cada rotina.

Rotina MPI_Init: A rotina `MPI_Init` inicializa um ambiente de execução MPI. É a primeira rotina a ser chamada no programa paralelo. O parâmetro `ierror`, é um valor inteiro. O retorno de `MPI_SUCCESS` é a garantia de que a função executou corretamente. A chamada dessa função deve ocorrer um única vez no programa (PINEDA; SMITH, 1998).

Protótipo: `MPI_INIT(ierr)`
`ierr`: *Flag* de erros.

Rotina MPI_Comm_rank: Esta função é responsável pela rotulação dos processos MPI. Essa rotulação ocorre através da atribuição de um número, *rank*, para cada processo pertencente ao mesmo comunicador. O parâmetro de entrada da função é o comunicador, seguido de dois parâmetros de retorno, o *rank* e o erro, caso houver (PINEDA; SMITH, 1998).

Protótipo: `MPI_COMM_RANK(comm,rank,ierr)`
`comm`: Representa o comunicador;
`rank`: Representa o rank do processo;
`ierr`: *Flag* de erro;

Rotina MPI_Comm_size: Esta rotina é utilizada para conhecer o número total de processos que estão executando no programa. Este número é retornado através do parâmetro *size*. Os parâmetros *comm* e *ierror*, tem a mesma função da rotina descrita anteriormente (PINEDA; SMITH, 1998).

Protótipo: `MPI_COMM_SIZE (comm,size,ierr)`
`comm`: Comunicador compartilhado pelos processos;
`size`: Número total de processos internos ao comunicador;
`ierr`: *Flag* de erros;

Rotina MPI_Send: Esta rotina é utilizada para o envio de uma mensagem entre dois processos MPI. O envio utilizado é bloqueante, ou seja, não há nenhum retorno até que o *buffer* de mensagens (`buf`) esteja disponível para uso, novamente. O parâmetro `buf` é um vetor que contém os dados a serem enviados e `count` é o número de elementos contido no `buf`. `Datatype` é um parâmetro do tipo inteiro que indica qual o tipo de dado armazenado no `buf`. Além desses parâmetros, `dest` armazena o número do *rank* do processo destinatário e a `tag` atua como um rótulo para combinar *sends* e *receives*

correspondentes (PINEDA; SMITH, 1998).

Protótipo: `MPI_SEND(buf, count, datatype, dest, tag, comm)`
`buf`: Representa o endereço inicial do *buffer* de envio;
`count`: Representa o número de elementos do *buffer*;
`datatype`: Representa o tipo de dados MPI de cada elemento do *buffer* de envio;
`dest`: Contém o *rank* do processo de destino;
`tag`: Um número intero que rotula a mensagem;
`comm`: O comunicar que será utilizado na troca de mensagens;

Rotina MPI_Recv: Esta rotina é utilizada para receber mensagens enviadas por rotinas de envio. Ela é bloqueante; não retorna até que um envio de correspondência tenha sido recebido. Seus parâmetros são semelhantes aos da rotina `MPI_Send`, divergindo somente no parâmetro `source`, que guarda o `rank` do emissor da mensagem e o parâmetro `status` que armazena informações detalhadas do `status` da comunicação (SNIR, 1998).

Protótipo: `MPI_RECV (buf,count,datatype,source,tag,comm,status,ierr)`
`buf`: Representa o endereço inicial do *buffer* de recebimento;
`count`: Representa o número de elementos do *buffer*;
`datatype`: Representa o tipo de dados MPI de cada elemento do *buffer* de recebimento;
`source`: Possui o endereço do *rank* do processo de origem;
`tag`: Um número inteiro que rotula a mensagem;
`comm`: O comunicador que será utilizado para a troca de mensagens;
`status`: Status da comunicação;
`ierr`: *Flag* de erros;

Rotina MPI_Isend: A rotina `MPI_Isend` inicia o envio de uma mensagem não bloqueante. Os seus parâmetro de entrada são semelhantes ao `MPI_Send`, divergindo apenas no parâmetro `request` (SNIR, 1998). Este parâmetro é utilizado

Protótipo: `MPI_ISEND(buf, count, datatype, dest, tag, comm, request)`
`buf`: Endereço da localização do *buffer* de envio;
`count`: Número de elementos no *buffer* de envio;
`datatype`: O tipo de dados MPI de cada elemento no *buffer* de envio;
`dest`: O identificador do destinatário;
`tag`: Rótulo da mensagem;
`comm`: Comunicador a ser utilizado na comunicação;
`request`: Solicitação de envio;

Rotina MPI_Irecv: Esta rotina é uma versão não bloqueante da `MPI_Recv`. Os

seus argumentos são idênticos exceto por um argumento. O parâmetro `status` é substituido por `request`. O parâmetro `request` é utilizado para consultar o *status* da comunicação ou esperar até sua conclusão. Esta rotina está diretamente ligada a `MPI_Wait` (PINEDA; SMITH, 1998).

Protótipo:`MPI_IRECV (buf, count, datatype, source, tag, comm, request)`
`buf`: Representa o endereço inicial do *buffer* de recebimento;
`count`: Descreve o número de elementos no *buffer*;
`datatype`: Define o tipo de dados de cada elemento do *buffer* de recebimento;
`source`: Representao o número do *rank* do processo de origem.
`tag`: Rótulo da mensagem;
`comm`: Comunicador a ser utilizado na comunicação;
`request`: Solicitação de recebimento;

Rotina MPI_Wait: Esta rotina tem a função de aguardar o fim da transmissão de rotinas de comunicação não bloqueantes, como: `MPI_Isend` e `MPI_Irecv`. O parâmetro *request* é enviado através dessas rotinas (SNIR, 1998).

Protótipo: `MPI_WAIT(request, status)`
`request`: Solicitação do processo;
`status`: status;

Rotina MPI_Finalize: Esta rotina tem por função terminar um ambiente de execução MPI. A sua chamada está restrita ao final do programa. Uma vez chamada, encerram-se os processos que executam em paralelo.

Protótipo: `MPI_FINALIZE (ierr)`
`ierr`: Parâmetro que retorna uma notificação caso ocorra erro na chamada.

3.5 Considerações sobre o Capítulo

Neste capítulo foram apresentados os principais conceitos referentes à interface de programação MPI. Descreveu-se os principais conceitos, os tipos de comunicação bloqueante, não bloqueante, síncrona, assíncrona, ponto a ponto e coletiva. Também detalhou-se as principais rotinas MPI no contexto do modelo OLAM.

Este capítulo é fundamental para a compreensão do próximo, em que é abordada a implementação do modelo OLAM. No Capítulo 4 as rotinas MPI são discutidas de maneira trivial, tornando-se necessário um conhecimento prévio sobre o assunto.

4 Implementação MPI no modelo OLAM

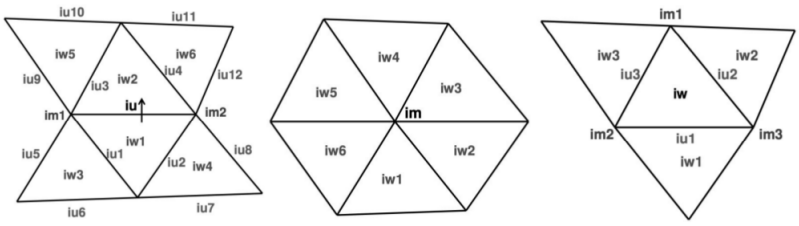
O modelo atmosférico OLAM implementa as rotinas MPI em outras subrotinas maiores, que além de comunicação, possuem funções de alocação de memória e empacotamento de dados. Além disso, esta subrotinas utilizam módulos externos que contém estruturas de dados que são utilizadas nas rotinas MPI. Neste contexto, este capítulo descreve as principais estruturas de dados, módulos e rotinas de comunicação implementadas pelo modelo.

4.1 Principais módulos

Esta seção descreve dos principais módulos implementado no modelo OLAM. Nos itens em sequência são descritas as funções que cada módulo exerce:

Módulo mem_ijtabs: O módulo `mem_ijtabs` é uma das principais seções de código que compõe o modelo OLAM. Contém as principais estruturas da discretização e decomposição do domínio do modelo (triângulos, vértices e pontos). A Figura 8 ilustra estas estruturas.

Figura 8 – Representação dos elementos do modelo OLAM



Fonte: Lopes (2010)

Os pontos `W` são representados pela estrutura de dados definida por `itab_w_vars`. Esta estrutura descreve os pontos `W`, define seus vizinhos, o número do *rank* do processo e o identificador global.

Os pontos `U` são representados pela estrutura `itab_u_vars`. Esta estrutura descreve os pontos `U`, que representam as aresta dos prismas triangulares pelos quais o OLAM é formado. Define identificadores para os triângulos adjacentes, para os vértices (`m`) e valores que serão utilizados na paralelização.

A estrutura de dados `itab_m_vars` representa os pontos `M`, que repreentam os vértices. Esta estrutura armazena vetores que indicam quais são seus vizinhos `w` e `u`.

A Tabela 1 descreve os principais os tipos de elementos que representam a estrutura do modelo OLAM:

Tabela 1 – Estrutura de dados do modelo OLAM

Nome	Tipo de elemento	# Elementos	# sub-elementos	Apontadores
$itab\_m$	Vértices M	nma	7	$iw(maxtpn)$, $iu(maxtpn)$
$itab\_u$	Faces U	nua	64	$im1$, $im2$, $iu1$, $iu2$, $iu3$, $iu4$, $iu5$, $iu6$, $iu7$, $iu8$, $iu9$, $iu10$, $iu11$, $iu12$, $iw1$, $iw2$, $iw3$, $iw4$, $iw5$, $iw6$
$itab\_w$	Triângulos W	nwa	55	$im1$, $im2$, $im3$, $iu1$, $iu2$, $iu3$, $iu4$, $iu5$, $iu6$, $iu7$, $iu8$, $iu9$, $iw1$, $iw2$, $iu3$

Fonte: Lopes (2010)

A tabela descreve o nome da estrutura de dados que representa cada elemento. Adicionalmente, descreve o número de sub-elementos e os apontadores, que representam informações sobre os pontos vizinhos. O número de sub-elementos é formado pela quantidade de elementos que compõem os vizinhos.

Módulo mem_para: Este módulo possui a definição das variáveis que são utilizadas para a alocação posterior de memória para as rotinas de comunicação `send` e `receive`. Além disso, possui uma estrutura de dados que específica para representar o *buffer*. Esta estrutura é representada pela Tabela 2.

Tabela 2 – Estrutura de Dados do Buffer

Type nodebuffs
character, allocatable :: buff(:)
integer :: nbytes
integer :: irequest
integer :: iremote
End type nodebuffs

Neste módulo também são definidos vetores do tipo `nodebuffs`, descrito na Tabela 2, que representam os buffers de envio e recebimento das estruturas descritas no módulo `mem_ijtabs`. Estes espaços de memória são denominados: `send_u`, `send_w`, `send_uf`, `recv_u`, `recv_w` e `recv_uf`.

Módulo mem_grid: Neste módulo encontram-se as informações e declarações de variáveis referentes à grade formada de triângulos. Possui variáveis que alocam o

número de pontos verticais (*nza*) e o número de pontos das estruturas `M`, `U` e `W`.

Módulo misc_coms: É uma interface para as variáveis inseridas no arquivo de configuração `OLAMIN`. Dentre as mais importantes estão a variável `nxp`, que delimita a resolução horizontal e a variável `nzp`, que delimita os níveis verticais. Este módulo também possui a variável `iparallel`, que habilita a paralelização por meio de condicionais inseridos no código.

Módulo micro_coms: É o módulo onde são definidas as constantes que são utilizadas para o cálculo das propriedades físicas envolvidas no modelo numérico. Como são constantes, estes valores podem somente ser lidos e não escritos durante a execução.

Módulo mem_basic: É onde está definido o conjunto de variáveis, estruturas e alocações de memória que são utilizadas pelas propriedades físicas.

4.2 Rotinas de inicialização e Finalização

A rotina de inicialização da implementação paralela do modelo OLAM é realizada pela rotina `olam_mpi_init()`. Esta rotina tem por função principal a alocação de memória para os vetores correspondentes aos declarados no módulo `mem_para`. Neste sentido, são alocados espaços na memória para os vetores de envio: `send_u`, `send_w`, `send_uf`; e vetores de recebimento: `recv_u`, `recv_w` e `recv_uf`. O tamanho desses vetores é definido pela variável `mgroupsize` que armazena o número de processos definidos no momento da execução.

No interior da função `olam_mpi_init()` são chamadas as rotinas MPI:

`MPI_Init(ierr)`: Define o início de uma execução paralela. Possui uma variável que retorna um erro caso ocorra.

`MPI_Comm_size(MPI_COMM_WORLD, mgroupsize, ierr)`: Esta rotina define o tamanho do comunicador; o número de processos que podem comunicar-se entre si. Este valor é armazenado na variável `mgroupsize`, que é chamada anteriormente às alocações.

`MPI_Comm_rank(MPI_COMM_WORLD, myrank, ierr)`: Retorna o número do processo que está executando.

Caso a execução seja sequencial, para a variável `mgroupsize` é atribuído o valor 1 e a variável `myrank` passa a valer 0.

Para finalizar a execução é utilizada a rotina `olam_mpi_finalize()`. Esta função chama a rotina `MPI_Finalize(ierr)` para terminar a execução dos processos. Além disso, libera o espaço de memória dos vetores alocados pela rotina de inicialização.

4.3 Rotina de alocação de memória

A alocação da memória para as estruturas de dados de cada ponto da malha é realizada pela rotina `olam_alloc_mpi()`. Esta rotina possui dois parâmetros de entrada: `mza`, que recebe o número de níveis verticais e `mrls`, que define o número de níveis de refinamento da malha. No contexto deste trabalho, não utilizou-se mais de um nível de refinamento.

Inicialmente, a função `olam_alloc_mpi()` invoca a rotina `MPI_Pack_size()` que define um limite superior para o espaço necessário para empacotar uma mensagem. Neste caso, as variáveis `nbytes_int`, `nbytes_real` e `nbytes_real8` definem este limite. Além disso, é necessário que seja determinado o número de bytes que serão enviados por coluna (triângulos/arestas x níveis verticais). Neste sentido, são definidas as variáveis `n_bytes_per_iu`, `n_bytes_per_uf` e `n_bytes_per_w`.

Com finalidade de evitar sobrecarga na comunicação, ainda são definidos três laços de repetição referentes a `u` (arestas), `uf` (arestas) e `w` (triângulos). Estes realizam chamadas de comunicação `send/receive` nas quais comunicam aos processos vizinhos o tamanho do `buffer` e a quantidade de dados que o processo que recebe deve estar habilitado a suportar.

4.4 Rotinas de Comunicação

O modelo OLAM possui seis rotinas de comunicação: três de envio e três de recebimento. Essas rotinas são específicas para a comunicação entre processos adjacentes das estruturas `u` (arestas), `uf` (arestas) e `w` (triângulos). Todas as rotinas MPI de comunicação dentro dessas funções são não bloqueantes. Independente de fazerem envio ou recebimento, conciliam a execução da aplicação enquanto estão se comunicando. Na subseções 4.4.1 e 4.4.2, são descritas as rotinas de envio e recebimento, respetivamente.

4.4.1 Rotinas de Envio

Com uma análise das rotinas de envio do modelo OLAM, percebeu-se que ambas possuem partes da implementação em comum. Por exemplo, todas são iniciadas com um laço de repetição em que é executada uma rotina de recebimento. A única diferença está no nome do *buffer*. Isso ocorre porque antes de cada envio é necessário verificar se não há um recebimento.

Nos tópicos seguintes aborda-se as características de cada uma das rotinas de envio:

mpi_send_u(sendgroup): Esta rotina recebe como parâmetro a variável `sendgroup`. Quando a variável `I` é passada como parâmetro, há uma cláusula condicional que delimita

o nível de refinamento em 1.

Basicamente a estrutura interna desta rotina é formada por dois laços condicionais aninhados. No laço externo ocorre a chamada da rotina `MPI_Pack()` que empacota dados referentes à estrutura de dados `jtab_u`, a qual armazena dados do índice global da aresta `U`. No laço interno são empacotados dados referentes às propriedades físicas `umc` e `uc`, além do índice global. Por fim, realiza a comunicação enviando uma mensagem `Isend`, não bloqueante, ao processo destinatário.

mpi_send_uf(hcnum_u, hcnum_w e hflux_t): Os parâmetros dessa rotina são propriedades físicas que são atribuídas aos pacotes da rotina `MPI_Pack()`.

A estrutura desta rotina também é formada por dois laços aninhados e segue o mesmo padrão da rotina `mpi_send_u`. No laço externo os dados empacotados são referentes à estrutura de dados `jtab_u`. No laço interno, são realizados empacotamentos que carregam no *buffer* as informações das propriedades físicas e o índice global.

mpi_send_w(sendgroup, wmarw): Os parâmetros dessa rotina são o `sendgroup`, que condiciona as rotinas de empacotamente e `wmarw`, que configura-se como uma propriedade física associada.

O funcionamento desta rotina depende diretamente do parâmetro de entrada que indica o grupo de envio `sendgroup`. Neste caso, esse parâmetro pode tratar os valores: `I`,`S`, `T`, `K` e `P`. Para cada um desses valores há uma condicional que determina quais propriedades físicas serão empacotadas para o envio.

Como as rotinas anteriores, esta também é formada por dois laços aninhados. No laço externo chama uma rotina de que empacota dados referentes à estrutura `jtab_w`. No laço interno, são declaradas condicionais que permitem a seleção do empacotamento de diferentes propriedades físicas.

Por fim, os dados empacotados são enviados através da rotina de envio não bloqueante `MPI_Isend`.

4.4.2 Rotinas de Recebimento

Nas rotinas de recebimento, o modelo OLAM também possui trechos da implementação comum à todas as funções. No início de cada uma, é chamada a rotina `MPI_Wait()` que verifica se todos os envios foram finalizados e desalocados. Dessa maneira, faz-se o teste para verificar se os *buffers* das rotinas não bloqueantes já podem ser reutilizados.

Nos tópicos seguintes aborda-se as características de cada uma das rotinas de recebimento:

mpi_recv_u(recvgroup): Assim como a rotina de envio `mpi_send_u`, há um parâmetro de entrada que condiciona a execução da rotina. Neste caso, o parâmetro

recvgroup atribui um valor a variável `mrl`, a qual representa o nível de refinamento.

A rotina possui dois laços aninhados. No laço externo realiza uma chamada de espera (`MPI_Wait()`), que aguarda a conclusão do recebimento de dados. Uma vez realizada esta execução, o laço interno desempacota as informações referentes a variável `iuglobe`, e as propriedades físicas `umc` e `uc`.

mpi_recv_uf(hcnum_u,hcnum_w,hflux_t): Esta rotina possui parâmetros que correspondem a propriedades físicas. Estas são desempacotadas e utilizadas na sequência da execução. Além disso, possui dois laços de repetição aninhados que realizam as chamadas de espera (`wait`) e o desempacotamento dos dados referentes ao índice global (`uglobe`) e as propriedades definidas nos parâmetros.

mpi_recv_w(recvgroup, wmarw): Esta subrotina difere-se das demais por condicionar quais serão as propriedades físicas que serão desempacotadas pela rotina `MPI_Unpack()`. Trata de maneira diferente os parâmetros de entrada `S`, `I`, `K`, `P` e `T`.

Quando recebe o parâmetro `S`, desempacota os dados da variável `vtab_var`. Para os demais parâmetros, são definidas as propriedades físicas que serão desempacotadas, como: `rho`, `wc`, `thil`, `hkm`, `vkm`, `vkm_sfc`, `wmarw` e `press`.

4.5 Considerações sobre o Capítulo

Este capítulo descreveu detalhes da implementação do modelo OLAM. Para tanto, abordou os principais módulos, estruturas de dados e subrotinas. Também descreveu em detalhes as subrotinas de comunicação e as suas rotinas MPI internas.

A compreensão deste capítulo facilita a assimilação dos termos empregados nos próximos capítulos. As rotinas apresentadas servem como base para o entendimento dos resultados.

5 Materiais e Métodos

Este capítulo descreve o ambiente de execução, as configurações da simulação do modelo OLAM e as ferramentas para analisar o perfil e traços da aplicação. A seção 5.1 descreve a suíte de aplicativos *Intel Parallel Studio XE 2016* e as ferramentas de perfilamento e traços acopladas. A seção 5.2 descreve as configurações utilizadas na execução do modelo. O ambiente de testes é descrito na seção 5.3. Por fim, a seção 5.4 descreve algumas considerações importantes sobre o capítulo.

5.1 Intel Parallel Studio XE

Para a realização da análise do perfil e traços do modelo OLAM utilizou-se a suíte de aplicativos Intel Parallel Studio XE (INTEL, 2015b). Esta suíte simplifica o desenvolvimento de aplicações paralela através da depuração e ajuste de código. Possui um conjunto de bibliotecas, compiladores e ferramentas de análise de aplicações paralelas. Atualmente, existem três edições disponíveis: Composer edition, Professional Edition e Cluster Edition. Neste trabalho utilizou-se a edição voltada para clusters (Cluster Edition) que contempla as principais ferramentas utilizadas para a análise de rotinas do modelo OLAM, como: Intel Vtune Amplifier XE, que cria o perfil da aplicação e Intel Trace Analyzer and Collector, que faz uma análise das rotinas MPI. A subseção 5.1.1 e 5.1.2 descrevem estas ferramentas.

5.1.1 Intel Vtune Amplifier XE

Intel Vtune é uma ferramenta de perfilamento de aplicações seriais e paralelas. Realiza a captura de estatísticas sobre o desempenho de uma aplicação. Além disso, foi projetada para uso em ambientes de memória compartilhada e tem suporte para as interfaces OpenMP e MPI.

Através de captura de dados sobre o desempenho (runtime) e de uma interface gráfica com vários recursos de visualização dos dados gerados, esta ferramenta ajuda a identificar e categorizar gargalos de desempenho. Além disso, identifica pontos críticos (hotspots), que são seções do código onde ocorre muita computação (INTEL, 2015e).

A ferramenta também possui um gráfico das chamadas de cada função. Localiza os pontos de entrada e saída das funções em tempo de execução. A instrumentação é binária e de baixa intrusão. O gráfico gerado é utilizado para conhecimento do fluxo de execução, funções críticas e saber a sequencia em que as funções são chamadas.

5.1.2 Intel Trace Analyzer and Collector

A ferramenta Intel Trace Analyzer and Collector é utilizada para entender o comportamento de aplicações MPI. É dividida em duas principais funções: a de geração de arquivos de rastros MPI e a visualização e análise dos arquivos gerados.

Segundo Intel (2015a), as principais características da ferramenta são:

- Visualizar e entender o comportamento de aplicações paralelas.
- Avaliação das estatísticas e balanceamento de carga das aplicações.
- Analisar o desempenho de subrotinas ou blocos de código.
- Entender os padrões de comunicação, parâmetros e dados sobre o desemepenho.
- Identificar pontos críticos de comunicação.
- Aumentar a eficiência da aplicação e diminuir o tempo de execução.

As subseções 5.1.3 e 5.1.4 abordam a criação de rastros MPI e visualização, respectivamente.

5.1.3 Collector

A ferramenta *Intel Trace Collector* tem função de capturar informações sobre a aplicação. Essas informações são compiladas em arquivos de traços, de extensão `.stf`, que armazenam informações como fluxo de execução e invocações de rotinas.

A instrumentação, adequação da aplicação à geração dos traços, pode ser realizada de forma automática ou manual. Neste trabalho, utilizou-se a instrumentação automática (INTEL, 2015d).

Para a geração dos traços utilizou-se a flag `-trace` durante a execução do programa MPI. Neste contexto, os arquivos registraram eventos referentes a toda a execução. Também utilizou-se a flag `-trace-pt2pt`, em que somente as rotinas de comunicação ponto a ponto foram rastreadas. Com este filtro, gerou-se arquivos de tamanhos menores e mais específicos.

Para a geração de traços das aplicações utilizou-se a biblioteca `libVT.so`, que refere-se ao *VampirTrace*(DRESDEN, 2014).

5.1.4 Trace Analyzer

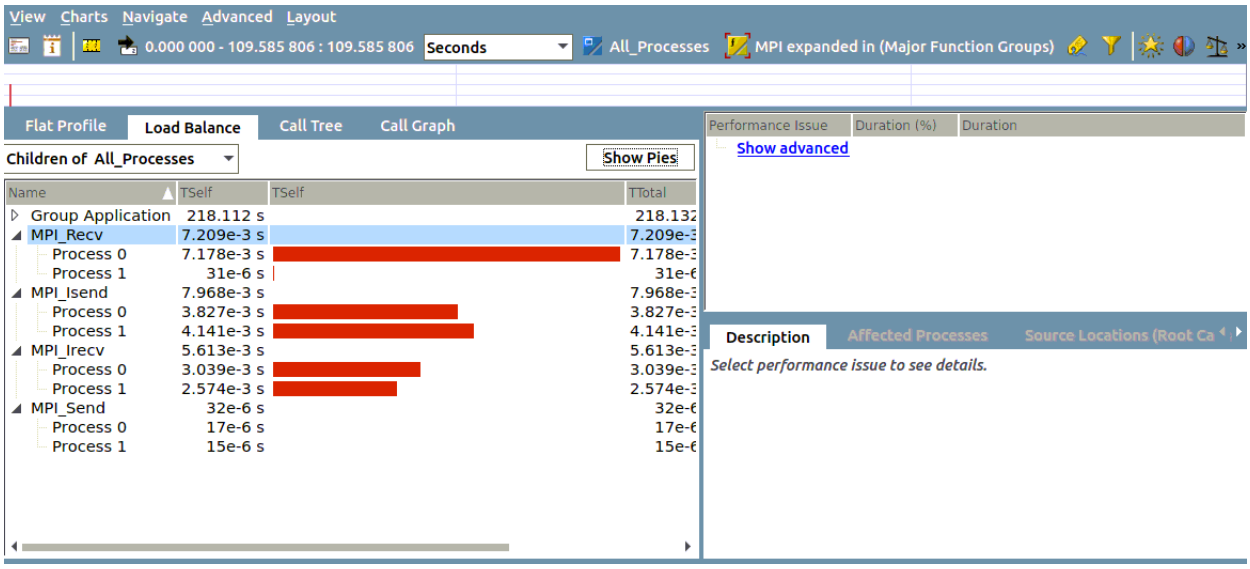
A ferramenta *Intel Trace Analyzer* permite realizar a visualização dos arquivos de traços (*.stf*) gerados. Isto é possível devido a uma interface gráfica com vários recursos

de análise. Auxilia no entendimento do comportamento da aplicação alvo. Pode detectar problemas de desempenho e erros de programação (INTEL, 2015c) .

Como alternativa ao modo gráfico, a ferramenta possui bibliotecas auxiliares em linha de comando. A principal é a *stftool* que permite converter os arquivos rastreados para a codificações ASCII, o que torna possível a visualização de alguns recursos em arquivos de texto comuns.

A Figura 9 demonstra a tela principal da ferramenta.

Figura 9 – Tela do programa Intel Trace Analyzer



Fonte: Intel (2015c)

Entre as principais funcionalidade está a possibilidade de visualização do balanceamento de carga entre diferentes processo e o gráfico de chamadas à cada rotina. Além disso, é possível verificar o tempo em que cada função executou e identificar possível pontos críticos das aplicações.

5.2 Configurações do modelo OLAM para execução

Na análise das rotinas MPI do modelo OLAM realizou-se uma única execução com finalidade de criar um perfil da aplicação e os traços para serem analisados pela ferramenta *Trace Analyzer and Collector*. Além disso, a execução ocorreu com diferentes números de processos. A Tabela 3 demonstra a configuração padrão utilizada nas execuções.

De modo geral, as configurações do modelo OLAM são feitas no arquivo `OLAMIN`. Neste arquivo, configurou-se a resolução da malha horizontal, através da variável `NXP` e

Tabela 3 – Configuração das Execuções do modelo OLAM

Descrição	Configuração
Número de execuções	única
Quantidade de processos	2, 4, 8 e 16
Resolução Horizontal da malha	100 km
Resolução Vertical da malha	28 níveis
Tempo de simulação	1 hora (3600s)

os níveis verticais, através da variável `NZP`. Além disso, na variável `TIMMAX` definiu-se o tempo de simulação. A variável `NXP` foi ajustada para 50 (5050/50 = 100 km). A variável `NZP` foi ajustada para 28 e a variável `TIMMAX` para 3600 (3600s = 1 hora).

5.3 Ambiente de Testes

Para realização dos testes foi utilizada uma *Workstation Dell Precision* T7600 da Universidade Federal do Pampa - Campus Alegrete. A máquina possui dois processadores Intel Xeon E5-2650 de 2.0 GHz e oito núcleos de processamento, cada. Com a utilização da tecnologia *HyperThreading* este número é estendido para 16 núcleos por processador. No total são 32 *cores*, 16 físicos e 16 lógicos. Além disso, a arquitetura possui 128 GB de memória.

Como o modelo OLAM é implementado em Fortran 90, utilizou-se o compilador Intel `ifort` para compilar os arquivos. Para a execução das rotinas MPI implementadas, explorou-se a implementação *Intel MPI Library*, na versão 5.1.2.

5.4 Considerações sobre o Capítulo

Este capítulo descreveu as principais ferramentas e métodos utilizados na realização deste trabalho e a importância de cada uma na análise dos resultados. Apresentou o ambiente em que os testes foram executados e como realizou-se a geração dos arquivos de traço. Também apresentou a ferramenta *Intel Trace Analyzer*, que permite a visualização de eventos em implementações MPI.

A apresentação das ferramentas descritas neste capítulo é importante para o entendimento dos resultados, considerando-se que os resultados gerados partiram do uso dessas ferramentas e métodos.

6 Resultados Obtidos

Este capítulo apresenta os principais resultados e contribuições deste trabalho. Está dividido em duas seções. A seção 6.1 avalia as execuções do modelo OLAM com um diferente número de processos. A seção 6.2 realiza a análise das rotinas MPI implementadas através da visualização do balanceamento de carga, matriz de troca de mensagens e tempo de computação.

6.1 Análise do Perfil do modelo OLAM

Como uma análise preliminar utilizou-se a ferramenta *Intel Vtune* para criação do perfil da aplicação. Neste sentido, buscou-se a identificação dos pontos críticos (*hotspots*). Estes pontos são regiões do código onde se concentram a maior parte do processamento. A utilização de paralelismo nestas regiões pode garantir um melhor ganho de desempenho global da aplicação.

No tempo de execução total, com um ou mais processos, identificou-se que embora o modelo OLAM tenha o tempo de computação em torno de 65%, a biblioteca padrão consome cerca de 35% do tempo de processamento total. Esse fator deve-se ao fluxo de Entrada e Saída e escrita dos dados resultantes em arquivos. A Tabela 4 demonstra o percetual de computação para cada módulo.

Tabela 4 – Percentual de Computação dos módulos e bibliotecas no Modelo OLAM

Módulo	**Sequencial**	**2 processos**	**4 processos**	**8 processos**
Executável do OLAM	63.9%	65%	64.7%	64.3%
Biblioteca (libc)	35.8%	34.7%	35%	35.5%
Biblioteca (libpthread)	0.2%	0.3%	0.2%	0.3%

Nas funções internas à execução do olam, identificou-se a rotina `prog_wrtu()` como ponto crítico no desempenho da aplicação. Do tempo de processamento total, cerca de 15% correspondem à execução desta rotina, mesmo na versão paralela.

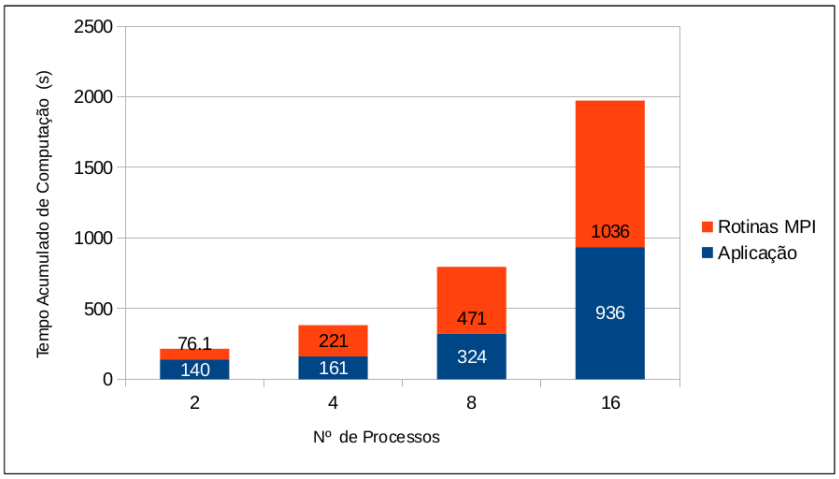
6.2 Análise das Rotinas MPI do Modelo OLAM

Para analisar as rotinas MPI do modelo OLAM utilizou-se os arquivos de traços gerados em diferentes execuções: com 2, 4, 8 e 16 processos. Estes arquivos armazenaram dados referentes à cada espaço de tempo da execução. Dessa maneira, possibilitou-se realizar uma avaliação da execução de maneira cronológica.

A visualização dos arquivos gerados tornou-se possível com a utilização da ferramenta *Intel Trace Analyzer and Collector*. Nesta ferramenta, utiliza-se uma abordagem diferente para mensurar os resultados, em que é realizado o somatório do tempo de execução dos diferentes processos. Para exemplificar este cálculo, é suposto que há dois processos em execução. Neste contexto, é realizada a soma do tempo de processamento do processo de *rank* 0 com o processo de *rank* 1. O mesmo é feito com o somatório das rotinas MPI dos diferentes processos.

Na Figura 10 é ilustrado o tempo de computação na execução da aplicação. A barra azul representa o tempo de computação efetivo, em que o modelo é executado. A barra vermelha representa o tempo que demora para execução das rotinas MPI.

Figura 10 – Tempo de computação e Rotinas MPI do modelo OLAM



Realizando uma análise da Figura 10, é possível verificar que salvo a execução com dois processos, nas demais execuções o tempo acumulado de computação das rotinas MPI é predominante. Isto ocorre devido ao fato da rotina `MPI_Wait` aguardar as rotinas de comunicação `MPI_Isend` e `MPI_Irecv` que são não bloqueantes e são chamadas com frequência. A Tabela 5 demonstra que a rotina `MPI_Wait` corresponde a aproximadamente 100% do processamento dentre as rotinas MPI.

Tabela 5 – Desbalanceamento no tempo de execuçao das Rotinas MPI

Nº processos	**MPI_Wait()**	**Demais Rotinas**
2	99.50%	0.50%
4	99.65%	0.35%
8	99.70%	0.30%
16	99.70%	0.30%

6.2.1 Matriz de Troca de Mensagens entre Processos

A análise da matriz de troca de mensagens é importante para verificar se há um padrão de comunicação entre os processos. Dessa maneira é possível visualizar quais processos comunicam-se entre si. Nessa perspectiva, analisou-se a comunicação entre 2, 4, 8 e 16 processos.

A troca de mensagens entre dois processos é simples. Ambos enviam e recebem mensagens entre si e não há um desbalanceamento quanto ao número de processos que se comunicam.

Nas comunicações realizadas com quatro processos nota-se que os processos de *rank* 1 e 2 realizam um número maior de envios que os demais processos. Por consequência, também recebem mensagens de mais processos. Este fator pode ser visualizado na Tabela 6.

Tabela 6 – Troca de mensagens entre 4 processos

Send/Receive	**0**	**1**	**2**	**3**
0		x	x	
1	x		x	x
2	x	x		x
3		x	x	

Na Tabela 7, a matriz representa a troca de mensagens em uma execução do oito processos. Nota-se que metade dos processos realiza um envio e recebimento a mais que os demais. Este fator não está ligado a um processos específico, é aleatório entre processos que compõem o comunicador MPI. Nesta execução, os processos 1, 2, 5 e 6 realizam uma comunicação a mais que os demais processos.

Tabela 7 – Troca de mensagens entre 8 processos

Send/Receive	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**
0		x	x		x	x	x	
1	x		x	x	x		x	x
2	x	x		x	x		x	x
3		x	x			x	x	x
4	x	x	x			x	x	
5	x	x		x	x		x	x
6	x		x	x	x	x		x
7		x	x	x		x	x	

Na análise da Tabela 8 constatou-se que não há um padrão definido nas trocas de mensagens com dezesseis processos. Há processos que comunicam-se com 5, 6, 7 ou 8 demais processos.

Tabela 8 – Troca de mensagens entre 16 processos

S/R	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
0			x		x		x		x				x			
1				x		x			x	x	x	x	x	x		
2	x				x		x		x		x				x	
3		x						x	x	x	x	x			x	x
4	x		x				x		x				x			
5		x						x	x	x			x	x	x	x
6	x		x		x						x		x		x	
7				x		x					x	x	x	x	x	x
8	x	x	x	x	x	x					x		x			
9		x		x		x						x		x		x
10		x	x	x			x	x	x						x	
11		x		x				x		x				x		x
12	x	x			x	x	x	x	x						x	
13		x				x		x		x		x				x
14			x	x		x	x	x			x		x			
15				x		x		x		x	x	x		x		

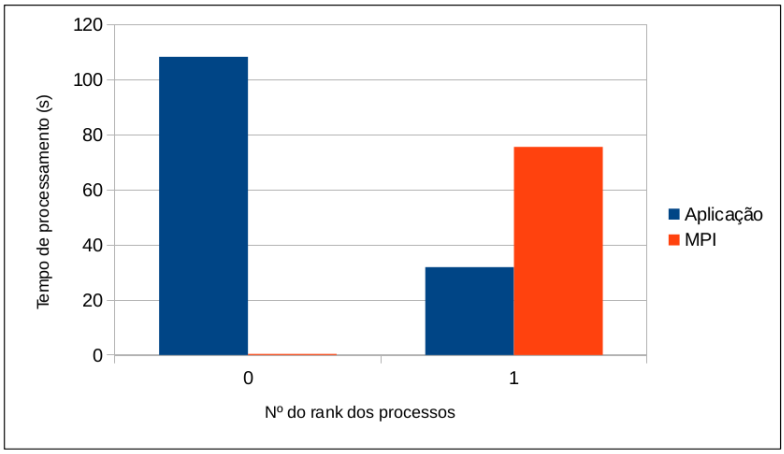
Com a análise das matrizes de trocas de mensagens verificou-se que não é possível determinar antes da execução com quais outros processos um processo específico pode se comunicar. Apesar de presumir-se que seja um evento aleatório, pode ter uma explicação aceitável. A decomposição do domínio entre os processos é realizada de modo a minimizar o tamanho das bordas entres os processos. Essa minimização não se preocupa com a diminuição do número de processos distintos com os quais haverá comunicação. Neste sentido, as comunicações são realizadas com os processos que fazem fronteira entre si.

6.2.2 Balanceamento de Carga entre Processos

Para analisar o balanceamento de carga utilizou-se os dados coletados na execução da ferramenta *Intel Trace Analyzer and Collector*. Nesta ferramenta são demonstrados os valores do custo da execução da aplicação e das chamadas de rotinas MPI para cada processo. Neste contexto, tornou-se possível a verificação do balanceamento de carga computacional de cada processo.

A Figura 11 descreve o balanceamento de carga na execução com dois processos. Na ilustração a barra azul representa o tempo de processamento (em segundos) que cada processo utiliza na execução. A barra vermelha representa os tempo referente às chamadas de rotinas MPI. Analisando o gráfico nota-se que o tempo de processamento das rotinas MPI do processo de *rank* 0 é praticamente nulo. A soma do processamento de todas rotinas chamadas por este processo resultaram em 0.39 segundos, enquanto no processo de *rank* 1 esta soma resultou em aproximadamente 75.7 segundos. Neste caso, nota-se que há um desbalanceamento entre os processos; o de *rank* 0 possui predominância na

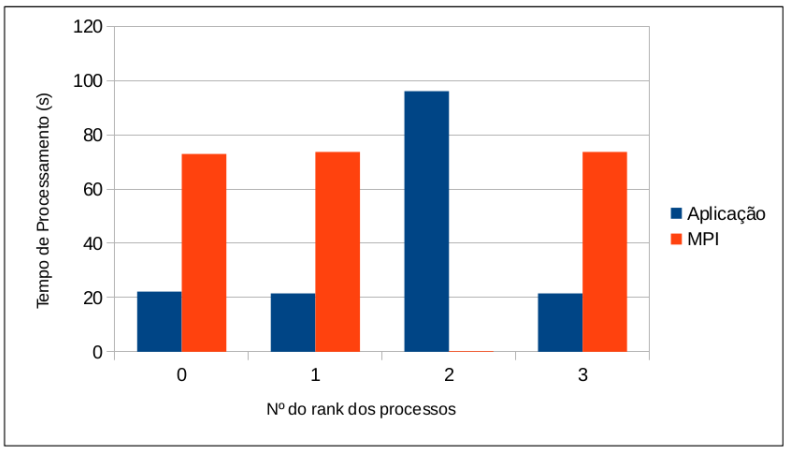
Figura 11 – Balanceamento de carga - 2 Processos



execução da aplicação e o processo de *rank* 1 predomina na execução das rotinas MPI.

A Figura 12 ilustra o balanceamento de carga em uma execução com quatro processos. A barra azul representa o tempo de execução da aplicação e a barra vermelha o tempo da invocação das rotinas MPI.

Figura 12 – Balanceamento de Carga - 4 processos

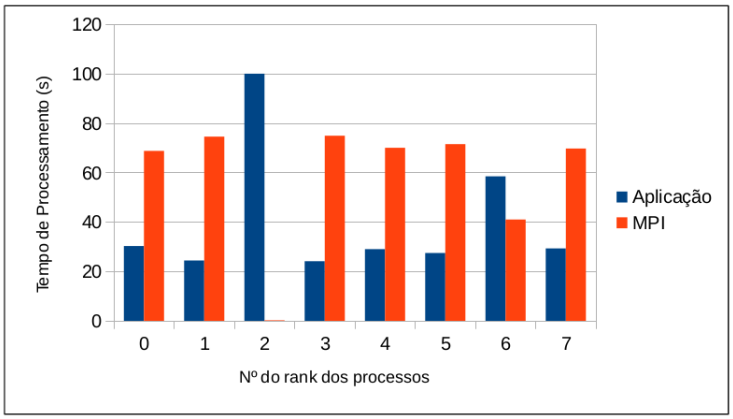


Através do gráfico verifica-se que o balanceamento de carga da execução e das rotinas MPI dos processos com *ranks* 0, 1 e 3 é similar. Entretanto, no processo de *rank* 2 houve predominância no tempo de processamento.

A Figura 13 ilustra o balanceamento da carga de processamento em uma execução com oito processos. As barras azuis e vermelhas, representam o tempo de processamento

da aplicação e das rotinas MPI, respectivamente.

Figura 13 – Balanceamento de carga - 8 processos



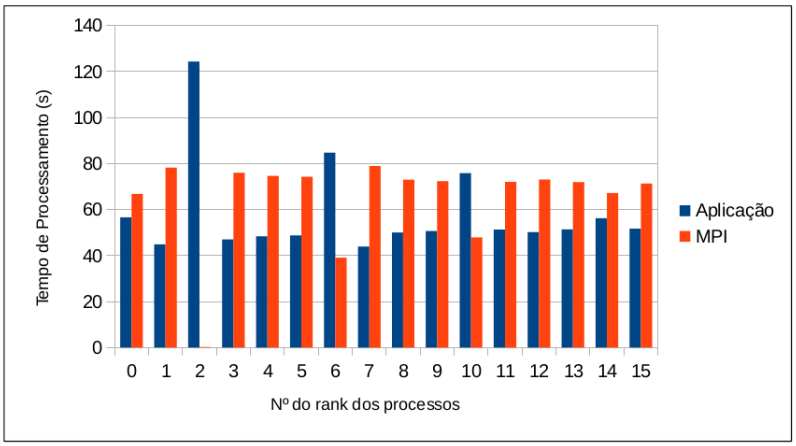
O gráfico resultante demonstra que o tempo acumulado das rotinas MPI é maior na maioria dos processos. Os processos de *ranks* 2 e 6, em contrapartida, diferem-se dos demais apresentando um tempo superior no processamento da aplicação (barra azul). No processo de *rank* 2 o tempo de processamento representou praticamente 100% do processamento total realizado por este processo. No processo de *rank* 6, o tempo de processamento permaneceu em 60%.

Na execução com dezesseis processos, ilustrada pela Figura 14, há um comportamento similar à execução com oito processos. Os processos de *rank* 2, 6 e 10 apresentam um tempo de processamento da aplicação maior que o tempo das rotinas MPI. Em contrapartida, nos demais processos há um predomínio no tempo das rotinas MPI. Através da visualização do balanceamento de carga em diferentes processos em ambas execuções (com 2, 4 ,8 e 16 processos), notou-se que há uma relação entre as diferentes execuções. Em todas, há um processo em que a carga computacional referente à execução de rotinas MPI é irrelevante. Através da ferramenta de visualização e arquivos de traços pode-se notar que este processo chama as rotinas MPI o mesmo número de vezes que alguns dos processos do mesmo grupo, porém ainda assim, na rotina `MPI_Wait` este processo não aguarda por muito tempo. A hipótese a se considerar é que este processo é o primeiro a realizar o envio da mensagem e portanto recebe o retorno mais rápido.

6.2.3 Perfil das Comunicações MPI

O modelo OLAM encapsula as rotinas de comunicação MPI em funções mais complexas. Além do envio e recebimento de mensagens, estas funções realizam o empacotamento e desempacotamento de dados. Também realizam a alocação de memória para

Figura 14 – Balanceamento de carga - 16 processos



as estruturas que montam a grade e empacotam cálculos referentes à propriedades físicas. Neste contexto, para analisar como as comunicações MPI são realizadas no modelo, utilizaram-se arquivos de traços e a ferramenta de visualização *Intel Trace Analyzer and Collector*.

As Figuras 15 e 16, representam trechos da execução do modelo OLAM com 2 e 4 processos, respectivamente. Para melhor compreensão, a Tabela 9 faz a referência entre as cores e as rotinas MPI apresentadas.

Tabela 9 – Referência entre cores e rotinas

Cor	**Rotina**
Azul Claro	Aplicação
Azul Escuro	MPI_Isend
Verde	MPI_Pack
Verde Claro	MPI_Unpack
Bordô	MPI_Wait
Rosa	MPI_Irecv

Em ambas ilustrações nota-se que há um padrão nas comunicações. Sempre que é realizada uma comunicação, chama-se a seguinte sequência de rotinas: `MPI_Irecv`, `MPI_Pack`, `MPI_Isend`, `MPI_Wait` e `MPI_Unpack`.

A rotina `MPI_Irecv` recebe um aviso sobre as rotinas que deve receber e continua executando, por ser não bloqueante. Em seguida, é realizado o empacotamento dos dados pela rotina `MPI_Pack`. Estes dados são enviados pela rotina `MPI_Isend`, que é chamada na sequência. Após o envio dos dados, a rotina `MPI_Wait` é invocada para que o recebimento de dados seja realizado antes do avanço na execução da aplicação. Após este

Figura 15 – Visualização dos Arquivos de Traços MPI - 2 processos

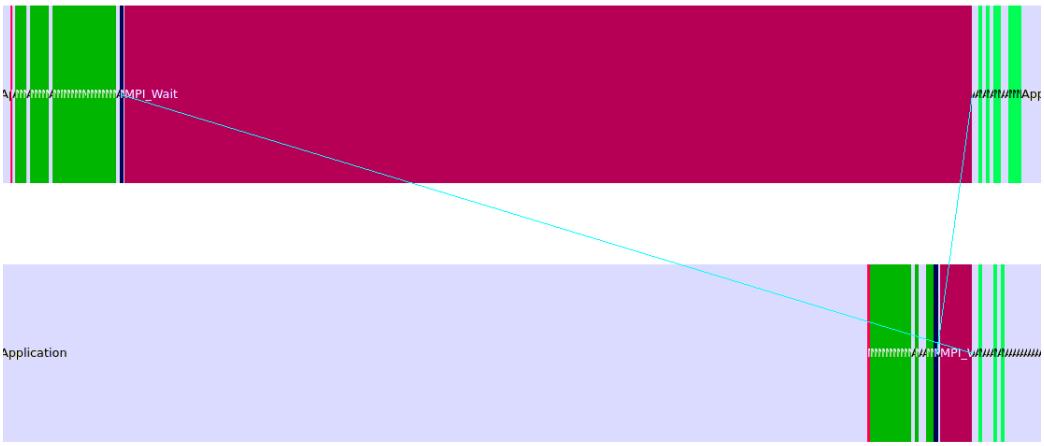
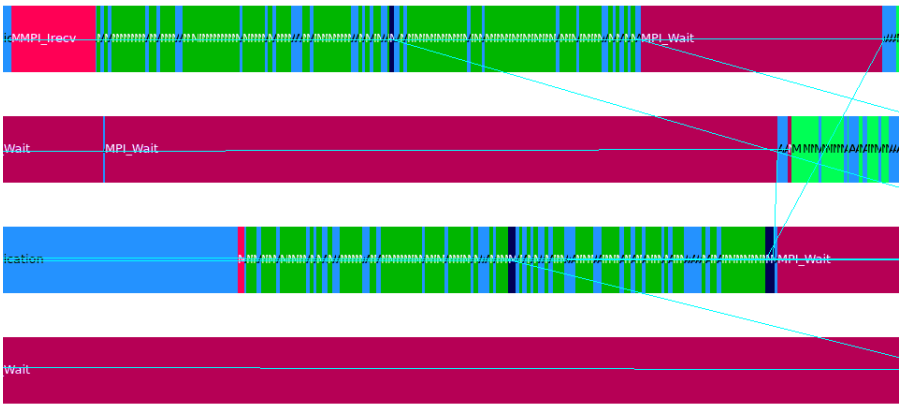


Figura 16 – Visualização dos Arquivos de Traços MPI - 4 processos



período aguardando, a mensagem é recebina e invoca-se a rotina de desempacotamento (`MPI_Unpack`.

A visualização do arquivos de traços com 8 e 16 processos também apresentaram o mesmo padrão nas rotinas de comunicação. Como demonstrado na Tabela 5, a rotina `MPI_Wait` é a que apresenta maior impacto dentre as rotinas MPI chamadas durante a execução do modelo OLAM. No entanto, no grupo de processos, em um desses processos as chamadas da rotina `MPI_Wait` é irrelevante no contexto geral da aplicação por ser o primeiro que realiza a comunicação.

7 Considerações Finais

A utilização de modelagem numérica para a previsão do tempo vem sendo muito utilizada nas últimas décadas. Pela quantidade massiva de dados que são computados nesses modelos, essa área está intimamente ligada ao processamento de alto desempenho. Há anos que pesquisadores tem estudado sobre a utilização de técnicas de paralelismo aplicadas a esse tipo de aplicação.

Neste contexto, este trabalho realizou um estudo das rotinas MPI implementadas no modelo atmosférico OLAM. Para tanto, utilizou-se as ferramentas *Intel Vtune Amplifier* e *Intel Trace Analyzer and Collector* para avaliar o perfil da execução do modelo com diferentes processos. A geração e visualização de arquivos de traços da aplicação possibilitou uma análise do balanceamento de carga entre processos, a identificação da relação de comunicação dos processos na execução e a identificação da rotina MPI que elevou o tempo do processamento dentre as rotinas MPI.

Os resultados identificaram uma anomalia no balanceamento de carga dos diferentes processos. Um dos processos em específico, dentre o grupo de processos, apresentou um tempo de computação das rotinas MPI próximo de zero. Além disso, identificou-se o motivo pelo qual na maioria dos processos o tempo acumulado das rotinas MPI foi superior ao tempo de computação da aplicação. Esse fator ocorreu devido às esperas realizadas pela rotina de aguardo `MPI_Wait`, a qual é chamada para gerenciar as rotinas de envio e recebimento não bloqueantes. Identificou-se também, que no processos que apresentaram anormalidades, o tempo em que as rotinas de aguardo executam é aproximadamente 100% inferior. Este cálculo é realizado com base na razão do tempo de processamento pelo número de chamadas da rotina.

O estudo das rotinas MPI através das ferramentas de visualização de traços contribuiu para uma melhor compreensão do funcionamento da implementação paralela do modelo OLAM. A identificação de rotinas que representam gargalos de desempenho podem auxiliar na melhoria do desenvolvimento do modelo.

Referências

BARNEY, B. *Message Passing Interface (MPI)*. 2014. Https://computing.llnl.gov/tutorials/mpi/. Lawrence Livermore National Laboratory. Citado na página 31.

CEPANAD. *Apostila de Treinamento: Introdução ao MPI*. http://www.cenapad.unicamp.br/servicos/treinamentos/apostilas/apostila_MPI.pdf, 2012. Citado na página 32.

DRESDEN, T. U. *VampirTrace*. 2014. Disponível em: <http://tu-dresden.de/die_tu_dresden/zentrale_einrichtungen/zih/forschung/projekte/vampirtrace>. Citado na página 44.

GROPP, W.; LUSK, E.; SKJELLUM, A. *Using MPI: Portable Programming with the Message Passing Interface*. 2º edição. ed. [S.l.]: The MIT Press, 1999. Citado na página 32.

INTEL. *Getting Started with Intel® Trace Analyzer and Collector*. 2015. Disponível em: <https://software.intel.com/en-us/get-started-with-itac>. Citado na página 44.

INTEL. *https://software.intel.com/en-us/intel-parallel-studio-xe*. 2015. Disponível em: <https://software.intel.com/en-us/intel-parallel-studio-xe>. Citado na página 43.

INTEL. *Intel® Trace Analyzer 9.1: User and Reference Guide*. [S.l.], 2015. Disponível em: <https://software.intel.com/sites/default/files/ITA_User_and_Reference_Guide.pdf>. Citado na página 45.

INTEL. *Intel® Trace Collector 9.1: User and Reference Guide*. [S.l.], 2015. Disponível em: <https://software.intel.com/sites/default/files/ITC_User_and_Reference_Guide.pdf>. Citado na página 44.

INTEL. *Tutorial: Finding Hotspots*. [S.l.], 2015. Disponível em: <https://software.intel.com/sites/default/files/fort_hotspots_amplxe_lin.pdf>. Citado na página 43.

LOPES, P. P. *Análise de benefícios do paralelismo por comunicação unilateral em aplicações com grades não estruturadas*. Dissertação (Mestrado) — Universidade de São Paulo - Instituto de Matemática e Estatística, Junho 2010. Citado 2 vezes nas páginas 37 e 38.

MARSHALL J., A. A. H. C. P. L.; HEISEY C., e. a. A finite volume, incompressible navier stokes model for studies of the ocean on parallel computers. *Journal of Geophysical Research Oceans*, v. 102, p. 5753–5766, 1997. Citado na página 25.

PACHECO, P. S. *Parallel Programming with MPI*. 1º edição. ed. [S.l.]: Morgan Kaufmann, 1996. Citado na página 32.

PIELKE R., e. A. A comprehensive meteorological model system - rams. *Meteorology and Atmospheric Physics*, v. 49, p. 69–91, 1992. Citado na página 25.

PINEDA, A. C.; SMITH, B. *MPI Tutorial*. 1998. Disponível em: <https://www.cs.kent.ac.uk/people/staff/trh/MPI/mpitutorial.pdf>. Citado 3 vezes nas páginas 33, 34 e 35.

SCHEPKE, C. *Exploiting Multiple Levels of Parallelism and Online Refinement of Unstructured Meshes in Atmospheric Model Aplication*. Tese (Doutorado) — Universidade Federal do Rio Grande do Sul, 2012. Citado 5 vezes nas páginas 23, 26, 27, 28 e 29.

SILVA R.;DIAS, P. D. S. E. Modelo OLAM (Ocean-Land-Atmosphere-Model): Descricao, Aplicacoes, e perspectivas. *Revista Brasileira de Meteorologia*, p. 144–157, 2009. Citado 4 vezes nas páginas 23, 25, 26 e 29.

SNIR, M. *MPI-the Complete Reference: The MPI core*. [S.l.]: Mass, 1998. (MPI–the Complete Reference). ISBN 9780262692151. Citado 3 vezes nas páginas 33, 35 e 36.

SNIR, M. et al. *MPI: The Complete Reference*. 2º edição. ed. [S.l.]: The MIT Press, 1998. Citado na página 31.

VARGAS, F. C. d.; SCHEPKE, C. *Estudo de Técnicas de Otimização de Desempenho para GPUs Utilizando CUDA Aplicado a um Modelo Meteorológico*. 2014. Universidade Federal do Pampa. Citado na página 29.

WALKO R.;AVISSAR, R. The ocean land atmosphere model (olam). part i: Shallow-water tests. *American Meteorological Society*, v. 136, p. 4033–4044, 2008. Citado 4 vezes nas páginas 25, 26, 27 e 28.

Índice

CPU, 23

MPI, 23, 24, 29–37, 39, 40, 42, 44–55

OLAM, 24, 25, 27–31, 33, 36–43, 45–47, 53–55

PAD, 31

RAMS, 25, 28