6

Aplicações de Grades Computacionais: Estudos de Caso em Meteorologia e Hidrologia

Professores:

Juliana Kaizer Vizzotto[1]
(juvizzotto@inf.ufsm.br)
Andrea Schwertner Charão[2]
(andrea@inf.ufsm.br)
Roberto Pinto Souto[3]
(roberto.souto@crs.inpe.br)
Haroldo Fraga de Campos Velho[4]
(haroldo@lac.inpe.br)

Resumo:

Grades computacionais podem ser definidas como um conjunto de computadores, possivelmente separados geograficamente, ligados em rede, permitindo assim o compartilhamento de recursos computacionais em larga escala. A possibilidade de aproveitar uma grande quantidade de recursos faz

[1] Graduada em Ciência da Computação pela Universidade Católica de Pelotas, com mestrado e doutorado em Ciência da Computação pela Universidade Federal do Rio Grande do Sul. Atualmente, é professora na Universidade Federal de Santa Maria. Atua principalmente nos seguintes temas: teoria da computação com ênfase em semântica de linguagens de programação funcional, programação quântica e lógica para computação.

[2] Graduada em Informática pela Universidade Federal de Santa Maria, com mestrado em Ciência da Computação pela Universidade Federal do Rio Grande do Sul e Doutorado em Informatique, Systèmes et Communications pelo Institut National Polytechnique de Grenoble. Atualmente, é professora na Universidade Federal de Santa Maria, atuando principalmente nos seguintes temas: processamento de alto desempenho, programação de aplicações paralelas e distribuídas, grades computacionais e virtualização.

[3] Graduado em Matemática Aplicada Computacional pela Universidade Federal do Rio Grande do Sul, com mestrado em Sensoriamento Remoto e doutorado em Computação Aplicada pelo Instituto Nacional de Pesquisas Espaciais. Atualmente, é pesquisador associado do Instituto Nacional de Pesquisas Espaciais, atuando principalmente nos seguintes temas: processamento paralelo, computação em grade, problemas inversos e modelos meteorológicos computacionais.

[4] Graduado em Engenharia Química pela Pontifícia Universidade Católica do Rio Grande do Sul, com mestrado e doutorado em Engenharia Mecânica pela Universidade Federal do Rio Grande do Sul. Atualmente, é pesquisador titular do Instituto Nacional de Pesquisas Espaciais, atuando principalmente nos seguintes temas: problemas inversos, assimilação de dados, modelos de turbulência para atmosfera, redes neurais artificiais e métodos numéricos.

das grades computacionais uma plataforma vantajosa para execução de aplicações paralelas. O objetivo deste capítulo é descrever dois casos de utilização de grades computacionais. Ambos os casos são aplicações computacionalmente intensivas que foram paralelizadas para execução em clusters e posteriormente configuradas como tarefas a serem distribuídas em grade. O texto inicia apresentando as metodologias e ferramentas utilizadas nestes casos, para depois descrever as aplicações e os resultados obtidos graças ao uso da computação em grade.

6.1. Introdução

Grades de computadores têm se difundido como plataformas de execução capazes de agregar e aproveitar recursos distribuídos. O termo grade foi cunhado por Foster et al. [Foster and Kesselman 1999, Foster 2002] no final da década de 90, baseado numa analogia inovadora entre a rede elétrica (*power grid*) que oferece energia e as redes de computadores que provêem poder computacional. Embora o trabalho de Foster tenha se tornado um marco na área, pode-se dizer que a ideia geral de agregar recursos distribuídos já era um anseio antigo. Na época, Foster levantou vários requisitos desejáveis para tornar a analogia uma realidade, a fim de lidar com problemas de autorização, autenticação, segurança, escalonamento de tarefas, contabilização, gerenciamento, entre outras. Desde então, a intensa pesquisa na área possibilitou atender muitos desses requisitos e tornar correntes certas aplicações de grades.

Existem vários tipos de grades: grades de dados, grades computacionais, etc. As grades computacionais, que são o objeto deste minicurso, têm o foco no compartilhamento de poder de processamento. Elas se destinam geralmente à execução de aplicações computacionalmente intensivas. Em muitos casos, as grades computacionais são fisicamente formadas por clusters de computadores localizados em diferentes instituições. Para tornar uma grade computacional operacional, necessita-se de ferramentas que possibilitem o gerenciamento dos recursos e facilitem o acesso a esses por parte dos usuários, oferecendo a imagem de um recurso único. Esse conjunto de ferramentas, geralmente denominadas *middleware* de grade, fornecem uma infra-estrutura lógica para o funcionamento da grade. Exemplos de *middleware* de grade são OurGrid e Globus, entre vários outros.

A concepção de aplicações que se beneficiem efetivamente das grades requer conhecimento das características da grade e do comportamento da aplicação. Por exemplo, dependendo da velocidade da rede que interliga os nós da grade, pode-se executar aplicações com mais ou menos dependências entre tarefas. Em grades cujos nós são clusters, uma abordagem muito comum é executar ao mesmo tempo várias instâncias de uma mesma aplicação paralela sobre a grade, sendo cada instância mapeada num cluster. Há portanto vários níveis de paralelismo em jogo e é importante seguir uma metodologia para obter o aproveitamento esperado da grade.

Muitas organizações, principalmente científicas, têm unido esforços para compor grades computacionais ou outros tipos de grades. Assim surgiram projetos como EGEE (Enabling Grids for E-sciencE), E-Infrastructure shared between Europe and Latin America (EELA), NorduGrid, Grid5000, entre outros. As aplicações visadas por esses projetos são bastante variadas, compreendendo áreas como física nuclear, cosmologia, biologia, química, hidrologia e economia.

Embora as grades computacionais venham se proliferando, sua exploração eficiente requer conhecimentos sobre sua infra-estrutura, em particular sobre as ferramentas e metodologias empregadas. Em termos de ferramentas, é imprescindível dominar os recursos do *middleware* utilizado pela grade. Apesar da variedade de ferramentas, muitas delas têm pontos em comum e têm buscado seguir padronizações. Quanto à metodologia,

convém conhecer estratégias bem estabelecidas de paralelização de aplicações computacionalmente intensivas.

Dado o cenário descrito acima, este minicurso se propõe a apresentar dois casos de sucesso no uso de grades computacionais. Ambos os casos são aplicações computacionalmente intensivas que foram paralelizadas para execução em clusters e posteriormente configuradas como tarefas a serem distribuídas em grade. A primeira aplicação é um sistema de previsão meteorológica e climática regional, utilizado em produção por várias instituições, em especial pelo Centro de Previsão de Tempo e Clima (CPTEC) do Instituto Nacional de Pesquisas Espaciais (INPE), que é um órgão brasileiro de referência na área em questão. A segunda aplicação é um sistema formado para investigar uma nova técnica de pesquisa em ótica hidrológica, visando a reconstrução tomográfica 3D de imagens de satélite para determinar a concentração de clorofila no oceano.

Para apoiar a apresentação dos casos, este texto inicia fornecendo uma visão geral das arquiteturas, metodologias e ferramentas utilizadas em clusters e grades (seção 6.2.), para depois descrever as aplicações e os resultados obtidos graças ao uso da computação em grade (seções 6.3. e 6.4.). Para melhor compreensão do conteúdo do curso, sugere-se um conhecimento básico sobre arquiteturas paralelas e distribuídas, bem como sobre programação paralela, conforme o conteúdo dos cursos permanentes da Escola Regional de Alto Desempenho (ERAD). Não é objetivo deste curso entrar em detalhes técnicos das aplicações e ferramentas ou abordar exaustivamente as questões metodológicas envolvidas. Para aprofundar essas questões, sugere-se a consulta à bibliografia de referência citada ao longo do texto.

6.2. Arquiteturas, Metodologias e Ferramentas

Esta seção visa prover um embasamento sobre arquiteturas de clusters e grades, além de metodologias e ferramentas para desenvolvimento de aplicações para estas plataformas paralelas.

6.2.1. Clusters e Grades computacionais

As arquiteturas paralelas e distribuídas evoluíram bastante ao longo do tempo e foram alvo de diversas iniciativas de classificação. Uma das classificações precursoras foi a de Flynn [Flynn 1972], baseada no número de fluxos de dados e de instruções. Hoje em dia, esta classificação costuma ser complementada usando outros critérios de distinção entre os vários tipos de arquiteturas [Skillicorn 1988]. Dentre esses critérios, um dos mais marcantes diz respeito à organização de **memória**, que pode ser **compartilhada** ou **distribuída** entre as unidades de processamento. Como exemplo de arquitetura paralela com memória compartilhada, tem-se as arquiteturas multicore, que são bastante populares atualmente. Já com relação às arquiteturas com memória distribuída, destacam-se os clusters e as grades, que são objeto desse minicurso e serão discutidas nas seções a seguir.

6.2.1.1. Clusters

Um cluster de computadores se caracteriza por múltiplos nós de processamento interligados por uma rede local. Os nós são computadores independentes e as configurações variam conforme o tipo de aplicação: cluster de alto desempenho, de alta disponibilidade, de armazenamento, etc. Em clusters de alto desempenho, os nós são geralmente homogêneos, constituídos por servidores especialmente projetados para serem agrupados em *racks*. Esses *racks* são padronizados para acomodar servidores com medidas em *rack units*. Uma *rack unit*, ou 1U, corresponde a 1,75 polegadas (44.45 mm) de altura, sendo que é comum encontrar servidores de 1U, 2U ou 4U.

Ainda num cluster de alto desempenho, é comum existir um nó *front-end*, com hardware ligeiramente diferente dos demais, usado para prover acesso externo aos nós do cluster. Esse nó costuma possuir pelo menos 2 interfaces de rede: uma rede rápida (alta vazão e baixa latência) interligando os nós do cluster, usada para comunicação durante o processamento paralelo, e uma rede que pode ser mais lenta e que serve para ligar o nó *front-end* a uma outra rede, pela qual se dará o acesso ao cluster. O nó *front-end* geralmente executa um software de gerenciamento do cluster e costuma prover serviços aos demais nós (por exemplo, um servidor de arquivos). Os demais nós muitas vezes são desprovidos de disco.

As configurações de hardware podem variar bastante mesmo em clusters voltados para aplicações de alto desempenho. Isso pode ser facilmente constatado examinando-se a lista dos TOP 500 supercomputadores da atualidade [Dongarra et al. 2009]. Na lista lançada em novembro de 2009, cerca de 80% dos sistemas instalados possuem arquitetura do tipo cluster, com desempenho variando de 20 a 1042 TFlops.

Na figura 6.1 tem-se uma imagem de um dos clusters usados nos casos deste minicurso, instalado no Centro Regional Sul (CRS) do INPE. O cluster em questão possui 21 nós HP ProLiant DL145 de 1U, cada nó com um processador AMD-Opteron a 2.6GHz. Embora esta configuração possua apenas um núcleo (*core*) de processamento em cada nó, é cada vez mais comum encontrar clusters com nós *multicore*.

6.2.1.2. Grades Computacionais

Grades são plataformas computacionais geograficamente distribuídas, acessíveis a seus usuários por meio de uma interface única [Foster and Kesselman 1999, Skillicorn 2001]. A motivação inicial para esse tipo de plataforma foi agregar recursos heterogêneos e fisicamente distribuídos entre diferentes organizações, como se fossem um único recurso, oferecendo um poder computacional que dificilmente seria obtido com um sistema centralizado.

O conceito de grade assume algumas variações em diferentes contextos, podendo por exemplo designar um conjunto de clusters interligados ou mesmo um grupo de computadores pessoais ociosos num certo período. Mesmo com essa diversidade, existe consenso sobre certos aspectos que caracterizam as grades [Foster and Kesselman 1999]:

Dinamicidade: os recursos de uma grade podem variar durante a execução da aplicação;

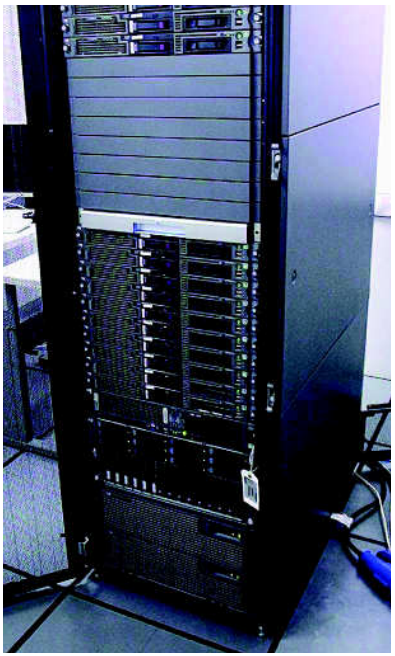

Figura 6.1. Cluster do CRS-INPE.

Heterogeneidade: os recursos podem ser bastante diversos em termos de hardware e software;

Distância: os recursos estão distribuídos e potencialmente distantes uns dos outros, de modo que a comunicação pode ter alto custo;

Propriedade: os recursos pertencem a diferentes organizações ou proprietários, que podem ter diferentes políticas administrativas e de acesso.

Estas características estão presentes em grades orientadas a diferentes aplicações. Sob esse ponto de vista, há alguns tipos comuns de grades: as **grades computacionais**, voltadas para aplicações de alto desempenho; as **grades de acesso**, cujo foco é disponibilizar recursos para um grande número de usuários e as **grades de dados** (*data grids* ou grades de armazenamento), cujo recurso principal são grandes bases de dados.

Mesmo que as grades possam ter diferentes objetivos, é possível descrever uma arquitetura geral de hardware e software para este tipo de infra-estrutura. A figura 6.2 ilustra uma das primeiras propostas de arquitetura para computação em grade [Foster et al. 2001]. Esta arquitetura é organizada em camadas que têm correspondência com a arquitetura de protocolos Internet.

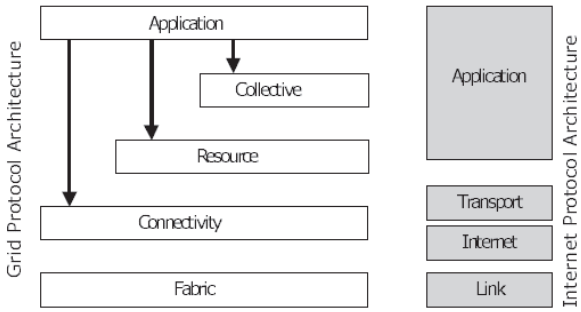

Figura 6.2. Arquitetura de uma grade. Fonte: [Foster et al. 2001]

A camada de mais baixo nível, **Fabric**, provê recursos que compõem a grade: processadores, sistemas de armazenamento, recursos de rede, equipamentos, etc. A camada seguinte, **Connectivity**, define protocolos de comunicação e autenticação necessários para transações em rede na grade, enquanto a camada **Resource** trata do gerenciamento individual de recursos da grade incluindo iniciação, monitoramento, controle, segurança e contabilidade de cada recurso. A coordenação dos múltiplos recursos da grade é responsabilidade da camada **Collective**, que agrupa serviços e protocolos tais como: serviços de diretório, alocação de recursos e escalonamento, monitoramento e diagnóstico, replicação de dados, ambientes de programação para a grade, gerenciamento da carga de trabalho, contabilidade geral de recursos, entre outros. Por fim, a última camada, **Application**, compreende as aplicações dos usuários da grade, que interagem com ferramentas e serviços das outras camadas.

6.2.2. Metodologias: Paralelização

A evolução da engenharia de software tem mostrado a importância das abordagens metódicas de programação. No caso do desenvolvimento de programas paralelos, a metodologia frequentemente parte da análise de um programa ou algoritmo originalmente sequencial, para então proceder à sua **paralelização**, ou seja, à elaboração um programa paralelo que produza o mesmo resultado do programa sequencial, mas com melhor desempenho em uma arquitetura paralela. Esta abordagem justifica-se pela grande quantidade de aplicações e sistemas legados construídos a partir de algoritmos sequenciais e que podem se beneficiar da paralelização. Outra abordagem consiste em explorar o paralelismo desde a concepção do algoritmo que soluciona determinado problema. Com essa abordagem, é possível desde o início escolher métodos de resolução que se comportam eficientemente em paralelo. Nos casos apresentados neste minicurso, foi utilizada a primeira abordagem.

Embora o estudo de algoritmos paralelos tenha começado há vários anos, as referências que enfatizam metodologias e ferramentas para paralelização são mais recentes. Nesse sentido, uma importante obra é o livro produzido por Foster [Foster 1995], que está disponível *on-line* na íntegra. Neste livro, Foster propõe que programas paralelos sejam

projetados em quatro etapas: **particionamento**, **comunicação**, **aglomeração** e **mapeamento**. As duas primeiras etapas têm foco na descoberta do paralelismo no problema, enquanto as duas últimas se preocupam com seu desempenho na arquitetura alvo.

A metodologia proposta por Foster é abrangente e mantém-se válida mesmo com o passar dos anos. Conforme a área foi se consolidando, algumas estratégias de paralelização se mostraram eficientes para determinadas arquiteturas e classes de aplicações. Alguns autores passaram então a classificar as aplicações e a documentar **padrões** (*patterns*) de programação paralela.

Nas seções a seguir, apresenta-se brevemente as etapas da metodologia proposta por Foster, seguidas de alguns padrões documentados em livros-texto de programação paralela.

6.2.2.1. Particionamento

Nesta etapa do projeto de um programa paralelo, busca-se identificar as oportunidades de paralelismo no problema ou algoritmo, decompondo-o em partes que possam ser processadas ao mesmo tempo. Cada partição ou tarefa é composta por um (sub-)conjunto de dados e um (sub-)conjunto de operações sobre tais dados. Para realizar o particionamento, geralmente segue-se uma das abordagens abaixo:

Particionamento de dados: nesta abordagem, também conhecida como decomposição de dados ou decomposição de domínio[5], analisa-se primeiramente as estruturas de dados que compõem o problema. Por exemplo, em uma operação de multiplicação de matrizes, os dados de entrada são duas matrizes A de ordem $N_1 \times M_1$ e B de ordem $N_2 \times M_2$ ($M1 = N2$) e o dado de saída é uma matriz C de ordem $N_1 \times M_2$. Dados intermediários também podem ter que ser considerados. Ao analisar as estruturas de dados, busca-se identificar diferentes itens de dados que possam ser processados em paralelo. No exemplo da multiplicação de matrizes, todos elementos c_{ij} da matriz resultante C podem ser calculados simultaneamente, desde que estejam disponíveis os elementos da linha i de A e da coluna j de B. As tarefas paralelas são então compostas por esses elementos de dados e suas operações associadas.

Particionamento de tarefas: nesta abordagem, também conhecida como decomposição ou particionamento funcional, analisa-se primeiramente as operações que precisam ser realizadas, buscando-se identificar diferentes conjuntos de operações que possam ser processados simultaneamente. Por exemplo, em uma tarefa que exija diversas operações com matrizes A e B (multiplicações, inversões, etc.), é possível realizar o particionamento funcional quando não houver uma sequência obrigatória para estas operações. As tarefas paralelas são então compostas pelas operações e pelos dados sobre os quais elas atuam, podendo ser os mesmos dados ou dados diferentes.

[5] A expressão "decomposição de domínio" também pode designar uma classe de métodos matemáticos para resolução numérica de equação diferenciais parciais. Tais métodos são bem adaptados ao processamento paralelo. Neste minicurso, a expressão é utilizada no seu sentido mais geral, de forma independente do problema.

Estas abordagens de particionamento são complementares. Independentemente da abordagem, busca-se produzir sub-tarefas de tamanho semelhante, ou seja, que trabalhem com quantidades semelhantes de dados e operações. Isso facilita a distribuição de tarefas sobre os processadores e, consequentemente, aumenta as chances de uma paralelização eficiente.

Geralmente, o particionamento de dados é a solução mais simples quando as estruturas de dados são bem conhecidas e regulares. O particionamento funcional pode ser muito trabalhoso em aplicações de médio e grande porte, pois requer uma análise aprofundada das dependências entre as operações. Mesmo quando as estruturas de dados são irregulares (por exemplo, um grafo qualquer ao invés de uma matriz), pode-se facilmente adotar o particionamento de dados com a ajuda de ferramentas que auxiliam na decomposição equilibrada (por exemplo, Metis [Karypis and Kumar 1995]).

6.2.2.2. Comunicação

Nesta etapa do projeto de um programa paralelo, identifica-se a necessidade de interação entre as tarefas paralelas criadas na fase de particionamento. Na maioria dos casos, as tarefas paralelas precisam, em algum momento, de dados associados a outras tarefas. Essa comunicação é necessária, mas deve ser minimizada tanto quanto possível, pois é um fator limitante da eficiência da paralelização.

As necessidades de interação podem ter diferentes formas, dando origem a padrões conceituais de comunicação entre tarefas. Nesta etapa, ainda não é importante definir como essa comunicação será implementada, embora se saiba que isso dependerá em grande parte da arquitetura-alvo.

Os padrões de comunicação considerados por Foster são classificados em quatro eixos:

Comunicação local X global: na comunicação dita local, cada tarefa comunica-se com outras poucas tarefas (tipicamente apenas uma); já na comunicação global, uma tarefa precisa comunicar-se com muitas outras tarefas (possivelmente todas as demais);

Comunicação estruturada X não-estruturada: a comunicação estruturada caracteriza-se por um padrão regular de interligação entre as tarefas, como por exemplo uma árvore em que uma tarefa-mãe só interage com as tarefas-filhas; a comunicação não-estruturada, por outro lado, ocorre quando as tarefas interagem formando grafos arbitrários;

Comunicação estática X dinâmica: na comunicação estática, a identidade das tarefas comunicantes é conhecida de antemão e não se altera durante a execução paralela; já na comunicação dinâmica, essa identidade é computada durante a execução e pode variar bastante;

Comunicação síncrona X assíncrona: na comunicação síncrona, as tarefas cooperam de uma forma coordenada, por exemplo com uma tarefa agindo como emissora e

outra como receptora num determinado momento; já na comunicação assíncrona, a troca de dados pode ocorrer em momentos diferentes, sem que uma tarefa coopere diretamente com a outra.

6.2.2.3. Aglomeração e Mapeamento

Nestas etapas do projeto de um programa paralelo, considera-se a arquitetura na qual ocorrerá a execução, buscando-se usar estratégias que garantam a eficiência paralela em tal arquitetura.

A **aglomeração** consiste em revisar o resultado das etapas de particionamento e comunicação, considerando o tipo de arquitetura. Nesta revisão, pode-se adotar diferentes estratégias para ajustar a quantidade e o tamanho das tarefas paralelas, bem como sua relação com a quantidade de comunicação. Por exemplo, para arquiteturas com memória distribuída, a comunicação pode ter alto custo, portanto pode ser vantajoso replicar dados ou operações em mais de um processador para evitar comunicações. Outra questão a considerar nesta etapa é a relação entre o número de tarefas paralelas e o número de processadores disponíveis. Por exemplo, se houver um grande número de pequenas tarefas e poucos processadores, a execução paralela tende a ser ineficiente.

Na etapa de **mapeamento**, o objetivo é decidir onde cada tarefa será executada, de forma a minimizar o tempo total de execução. Para isso, a estratégia básica consiste em mapear sobre diferentes processadores as tarefas que podem executar simultaneamente, de forma a maximizar a **concorrência**. No entanto, ao considerar também as necessidades de comunicação, pode-se mapear sobre um mesmo processador as tarefas que precisam comunicar-se frequentemente. Com isso aumenta-se a **localidade** e reduz-se o custo de comunicação, o que é favorável à obtenção de eficência paralela.

6.2.2.4. Padrões de Programação Paralela

Padrões de projeto constituem uma estratégia para lidar com a complexidade crescente dos sistemas de software. Um padrão descreve metodicamente a solução para um problema recorrente num determinado domínio, de forma que essa solução possa ser aproveitada em problemas semelhantes [Gamma et al. 1994].

Na programação paralela, os padrões são úteis para facilitar a obtenção de uma solução eficiente para um problema, mesmo que o programador não tenha grande experiência com metodologias de paralelização [Mattson et al. 2004]. A publicação de padrões de projeto começou a ocorrer sistematicamente na segunda metade da década de 90, sendo que os padrões paralelos estão entre os mais recentes.

Atualmente, há uma grande quantidade de padrões paralelos já documentados, abrangendo diversos espaços de projeto (descoberta da concorrência, estruturação do algoritmo, entre outros). A seguir descreve-se brevemente alguns padrões comuns aplicáveis em diferentes espaços de projeto citepatterns-mattson, wilkinson:

***Embarassingly Parallel*:** este padrão designa problemas trivialmente paralelizáveis, em

que o trabalho pode ser dividido em várias tarefas independentes que realizam as mesmas operações. Este padrão é encontrado por exemplo em algumas operações de processamento de imagens e em determinadas simulações usando o método de Monte Carlo. Problemas que se enquadram neste padrão frequentemente atingem alta eficiência paralela.

***Divide and Conquer*:** este padrão refere-se a problemas que podem ser divididos (*split*) em sub-problemas independentes, cujos resultados são combinados (*merge*) mais adiante para produzir a solução final. A divisão pode ser recursiva e precisa ser controlada adequadamente. Este padrão é frequentemente usado em problemas de ordenação de dados.

***Pipeline*:** este padrão refere-se a problemas em que as operações são feitas em estágios ordenados mas independentes sobre um conjunto de dados. A solução segue a mesma estratégia usada em arquiteturas de computadores, em que a saída de um estágio é a entrada de um estágio subsequente. Este padrão é frequentemente usado em processamento de sinais.

***Synchronous*:** este padrão refere-se a problemas em que as operações são feitas em iterações ordenadas. Uma iteração não pode iniciar antes da outra, mas as operações a cada iteração podem ser realizadas independentemente. Este padrão é comumente encontrado na paralelização de métodos numéricos de resolução de equações.

***Master-Worker*:** este padrão refere-se a uma forma de estruturar o programa paralelo, em que um processo mestre possui uma lista de tarefas que são distribuídas a processos trabalhadores. Cada trabalhador obtém uma tarefa, executa o trabalho associado e devolve seu resultado ao mestre. Isso continua até que o mestre detecta uma condição de término. Este padrão está num nível mais baixo de abstração que os padrões anteriores, podendo ser combinado com estes outros padrões para produzir um programa paralelo.

SPMD (*Single Program Multiple Data*): este padrão refere-se a outra forma de estruturar um programa paralelo, em que são lançadas múltiplas cópias de um único programa, cada uma com sua própria visão dos dados. Cada cópia é um processo e todos processos têm as mesmas atribuições, isto é, não há um processo mestre.

6.2.3. Metodologias: Modelos de Programação para Grades

O desenvolvimento de programas para grades se parece, em alguns aspectos, com o desenvolvimento de programas paralelos. Uma etapa importante nesta contexto continua sendo o particionamento de um problema em sub-problemas, ou seja, a identificação de tarefas paralelas.

No caso de grades, este particionamento deve gerar tarefas de tamanho considerável, que possam ser distribuídas entre nós distantes sem que o custo de movimentação de dados seja proibitivo. A comunicação costuma penalizar fortemente o desempenho de

aplicações em grades, por isso é comum que algumas grades se restrinjam a aplicações trivialmente paralelizáveis ou chamadas *bag-of-tasks*, em que as tarefas são independentes e por isso não têm necessidade de comunicação.

Em grades computacionais, os padrões de coordenação das tarefas (por exemplo: *manager-worker*) geralmente ficam a cargo de um serviço de gerenciamento de recursos da grade, que realiza o escalonamento das tarefas sobre os nós disponíveis. Também existem grades ditas *peer-to-peer*, que exploram o compartilhamento de recursos de forma descentralizada, levando a aplicações que precisam ser construídas sem ponto único de coordenação.

Por fim, vale também mencionar os modelos de programação orientados a serviços, orientados a objetos e orientados a componentes, que contrastam com os modelos de mais baixo nível baseados em troca de mensagens e memória compartilhada usados tipicamente em clusters. Esses outros modelos oferecem um nível mais alto de abstração para a construção de aplicações distribuídas, em que a interação entre tarefas se dá pela interação entre serviços, objetos ou componentes.

6.2.4. Ferramentas para Clusters e Grades

Quando se trabalha com aplicações paralelizadas para execução em clusters e grades, é necessário conhecer e utilizar diversas ferramentas que auxiliam desde o desenvolvimento até a execução. Em alguns casos, existem possibilidades de integração entre as ferramentas em clusters e grades. Esta seção apresenta algumas dessas ferramentas e fornece ponteiros para se conhecer mais sobre elas.

6.2.4.1. Ferramentas para Clusters

Um cluster deve dispor no mínimo de ferramentas de suporte à programação e execução paralelas. Além disso, pode ser necessária uma ferramenta de gerenciamento dos recursos e das execuções de tarefas no cluster. No caso de clusters interligados numa grade, esse gerenciamento geralmente é integrado ao *middleware* da grade.

MPI (*Message Passing Interface*): Esta interface de programação surgiu em 1994 e é a mais utilizada atualmente para implementar a comunicação entre processos paralelos [Forum 1994, Forum 1997]. Trata-se, portanto, de uma ferramenta para programação em clusters com memória distribuída. MPI oferece um suporte flexível para implementar os padrões de projeto descritos na seção 6.2.2.4. e os padrões de comunicação discutidos no item 6.2.2.2.. Existem basicamente 2 versões populares (1.2 e 2.1) e importantes implementações abertas desta interface (MPICH [Gropp et al. 1996, Gropp and Lusk 1996] e OpenMPI [Gabriel et al. 2004]), disponíveis para linguagens como C, C++, Fortran, Python e Java. Na sua versão mais moderna (2.1), sua interface foi estendida com funções para gerenciamento dinâmico de processos, entrada/saída paralela e acesso a memória remota, além de ser mais adaptada a arquiteturas com memória híbrida (compartilhada entre os núcleos

e distribuída entre os nós). Existem atualmente ótimos livros-texto sobre programação paralela usando MPI [Pacheco 1996, Quinn 2003, Wilkinson and Allen 2004], além de tutoriais *on-line* bastante completos [Barney 2009].

OpenMP: Esta ferramenta é voltada para a programação paralela em arquiteturas com memória compartilhada, usando *threads* [Chandra et al. 2001, Chapman et al. 2007]. Ela consiste em um conjunto de diretivas de compilação e algumas funções de biblioteca que permitem facilmente transformar um programa sequencial em um programa paralelo *multithread*. Trata-se de um padrão relativamente recente, cuja implementação necessita do suporte do compilador para reconhecimento das diretivas. É possível desenvolver aplicações que utilizam MPI e OpenMP concomitantemente, mas esse nicho da programação ainda é pouco explorado em comparação com programas que usam puramente MPI ou OpenMP.

TAU (*Tuning and Analysis Utilities*): Este pacote de ferramentas é uma das opções para a análise de desempenho de programas paralelos [Shende and Malony 2006]. Ferramentas tradicionais de *profiling* (prof, gprof, etc.) são geralmente usadas antes da paralelização, para determinar partes de código onde há processamento intensivo. Após a paralelização, torna-se necessário empregar ferramentas que levem em conta o caráter distribuído e concorrente da aplicação paralela. Este é o caso de TAU, que permite rastrear e visualizar a execução de programas MPI, a fim de determinar perfis de desempenho e possíveis gargalos. Além de TAU, existem várias outras ferramentas com propósito semelhante, tais como como Jumpshot [Zaki et al. 1999] e VAMPIR [Nagel et al. 1996]. A figura 6.3 fornece uma imagem de um dos utilitários incluídos em TAU (ParaProf).

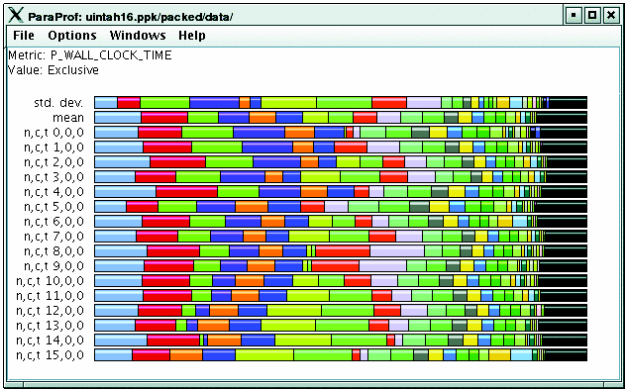

Figura 6.3. ParaProf, uma das ferramentas incluídas em TAU.

PTP (*Eclipse Parallel Tools Platform*): Esta ferramenta é um ambiente integrado (IDE) para desenvolvimento de programas paralelos [Watson et al. 2006, Watson 2010]. Trata-se de uma extensão ainda em desenvolvimento do popular ambiente Eclipse,

que pretende reunir vários recursos para programação paralela em um só IDE. Dentre esses recursos, tem-se: interface única de uso de diferentes ferramentas (MPI, OpenMP, etc.), facilidades para análise e refatoração de código, depurador paralelo e interface para interação do usuário com a plataforma paralela. Embora ainda não esteja concluído, possui uma comunidade ativa de desenvolvedores e já oferece um conjunto de recursos que facilitam o trabalho com programas paralelos.

6.2.4.2. Ferramentas para Grades

Comparativamente com clusters, a concretização de grades computacionais depende de várias ferramentas adicionais, uma vez que seus objetivos vão além da obtenção de desempenho e exigem um maior grau de gerenciamento de recursos.

Dada a diversidade de ferramentas, atualmente é difícil apontar exemplos dominantes. No entanto, pode-se analisar as ferramentas existentes sob os seguintes pontos de vista:

***Middleware*:** constitui uma infra-estrutura genérica que se estende sobre os recursos fisicamente distribuídos e provê a base para tais recursos sejam explorados por aplicações. Há muitos exemplos de *middleware* para grades computacionais, provendo infra-estruturas mais amplas ou mais restritas a um determinado tipo de aplicação. Dentre alguns populares, pode-se citar: BOINC, Globus, gLite, Sun Grid Engine e OurGrid.

Serviços: há ferramentas que implementam serviços específicos, que podem ou não ser independentes do *middleware*. Exemplos de serviços são: gestão de segurança, gestão de licenças, gerenciamento de tarefas, gestão do contrato de serviço (*Service Level Agreement*), portais para grades, entre outros.

Padrões: existem alguns esforços de padronização de componentes e serviços para grades, liderados por uma comunidade denominada Open Grid Forum (OGF). Um importante padrão produzido pelo OGF é o *Open Grid Services Architecture* (OGSA), que especifica uma arquitetura orientada a serviços Web para grades. Outro exemplo de especificação visando padronização é DRMAA (*Distributed Resource Management Application*), para submissão e controle de tarefas numa grade.

Considerando esses pontos de vista e os casos escolhidos para este minicurso, descreve-se a seguir algumas ferramentas representativas para grades computacionais.

Globus Toolkit: trata-se de um *middleware* de código aberto que implementa vários padrões do OGF e provê um vasto conjunto de serviços, como por exemplo: submissão e controle de aplicações, movimentação de dados, segurança e descoberta de recursos [I. Foster and C. Kesselman 1997]. Essa diversidade de serviços oferece a vantagem da flexibilidade à grade, permitindo executar aplicações com diferentes características. Por outro lado, essa flexibilidade confere complexidade ao *middleware*, dificultando seu uso em aplicações que requerem poucos serviços, como os casos apresentados neste minicurso.

OurGrid: é um *middleware* para grades computacionais *peer-to-peer*, em que usuários participantes disponibilizam seus recursos ociosos a outros usuários [OurGrid 2008]. Essa plataforma, também de código aberto, só pode ser usada para executar aplicações do tipo *bag-of-tasks*, ou seja, aplicações paralelas cujas tarefas são independentes e não se comunicam entre si.

Gridsphere: esta ferramenta destina-se à construção de portais Web para grades computacionais [Novotny et al. 2004, GridSphere Project 2007]. Um portal Web é uma forma conveniente de prover acesso a uma grade, pois é de fácil utilização por diferentes perfis de usuários e não requer a instalação de software específico por parte do usuário. Gridsphere é baseado em padrões Java para aplicações Web e oferece vários serviços prontos, como autenticação e gerenciamento de perfis de usuários.

6.3. Aplicação em Meteorologia: previsão de clima com o BRAMS

O objetivo dessa seção é descrever um caso de utilização de uma grade computacional para uma aplicação em meteorologia. Em síntese, apresenta-se a grade computacional utilizada no projeto da *Rede Estadual de CLImatologia no Rio grande do Sul* (RECLIRS), financiado pela Financiadora de Estudos e Projetos (FINEP) e executado em conjunto pelas seguintes instituições: Faculdade de Meteorologia da Universidade Federal de Pelotas (UFPel), Centro Regional Sul (CRS) e Laboratório Associado de Computação (LAC), ambos do INPE, Universidade Federal de Santa Maria (UFSM) e Instituto de Informática da Universidade Federal do Rio Grande do Sul (UFRGS). O projeto RECLIRS tinha como principal objetivo montar uma rede estadual de climatologia no Rio Grande do Sul, provendo assim um sistema para operacionalizar previsão climática sazonal de meso-escala, feita mensalmente para o estado do Rio Grande do Sul. Dentre os objetivos específicos do projeto, estava a realização de climatologia de meso-escala, para um período de 30 anos (1970 a 2000). Nesse contexto, nessa seção discute-se, especificamente, o arcabouço experimental para a execução da climatologia de 10 anos (1992 -2001) no RS em uma grade computacional.

Essa seção está estruturada como segue. Primeiramente, na seção 6.3.1. apresenta-se brevemente o modelo de previsão de tempo e clima utilizado na previsão, o BRAMS. A seguir, discutem-se os aspectos importantes da paralelização do modelo BRAMS, na seção 6.3.2..

6.3.1. BRAMS

Modelos meteorológicos podem ser vistos como sistemas computacionais que realizam uma simulação numérica do comportamento da atmosfera. Em especial, os modelos meteorológicos **prognósticos** resolvem as equações de conservação do estado da atmosfera com incremento de tempo, possibilitando assim a previsão do tempo e clima. Tais mo-

delos podem prever fenômenos atmosféricos da microescala até a escala sinótica com confiabilidade. Os modelos regionais de previsão de tempo e clima trabalham com fenômenos de mesoescala (i.e, intermediária entre micro e sinótica), tal que as dimensões horizontais geralmente oscilam de um a centenas de quilômetros. Exemplos de fenômenos de mesoescala são complexos convectivos de mesoescala, as brisas do mar, linhas de instabilidade, etc.

Diversos países têm desenvolvido modelos regionais de previsão meteorológica, sendo que um exemplo importante é o *Regional Atmospheric Modeling System* (RAMS) [Pielke 1984], desenvolvido na Colorado State University/USA. O modelo RAMS está implementado na linguagem de programação Fortran, com apenas algumas rotinas de entrada e saída de dados implementadas com a linguagem de programação C. Essencialmente, o RAMS contém três partes principais: i) a implementação das equações dos fenômenos de mesoescala calibradas para a região dos USA; ii) um módulo de assimilação de dados para inicialização do sistema; iii) e uma interface com ferramenta de visualização de mapas meteorológicos.

O Brasil, no início dos anos 90, junto ao Centro de Previsão de Tempo e Clima do Instituto Nacional de Pesquisas Espaciais (CPTEC/INPE) começou a utilização do RAMS para algumas previsões específicas em São Paulo e região Nordeste. Entretanto, um modelo leva em consideração condições bem específicas de vegetação, relevo e clima da região. Assim, desenvolveu-se a partir do RAMS o *Brazilian Regional Atmospheric Modeling System* (BRAMS) [Fazenda et al. 2007]. O BRAMS simula a circulação da atmosfera para uma área geográfica limitada (i.e., é um modelo de mesoescala) adaptado às condições de vegetação, relevo e clima do Brasil.

A atmosfera é um fluido. Assim, a ideia básica na predição numérica do tempo é identificar o estado do fluido em um dado tempo e utilizar as equações da dinâmica dos fluidos e termodinâmica para estimar o estado do fluido em tempos futuros.

Portanto, como em todo o modelo **computacional**, o BRAMS é implementado através da utilização de métodos numéricos para se construir algoritmos (discretos) que simulem o comportamento das equações contínuas de dinâmica dos fluidos e termodinâmica, que descrevem o comportamento da atmosfera. Assim, métodos numéricos obtêm soluções **aproximadas** para as equações, e diferentes modelos utilizam diferentes métodos de soluções.

Essencialmente, no BRAMS, a região da atmosfera a qual deseja-se simular o comportamento (por exemplo na figura 6.4) é divida em uma malha com pontos finitos (como na figura 6.5). Essa malha então pode ser refinada como nas figuras 6.6 e 6.7.

Para calcular as propriedades da atmosfera nos pontos das malhas mais finas (e.g., figuras 6.6 e 6.7) o BRAMS utiliza um método numérico chamado de interpolação com diferenças finitas [Kaw and Kalu 2008]. Basicamente, dados valores de uma função para um número de pontos, a interpolação define os valores da função para outros pontos entre os pontos dados inicialmente.

Em geral, devido ao considerável número de cálculos requeridos, métodos numéricos são computacionalmente dispendiosos. No caso da interpolação no BRAMS, isso acontece especialmente quando a malha vai ficando mais fina e o número de pontos de

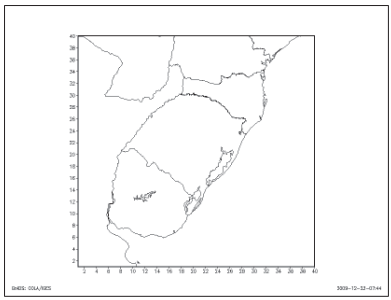

Figura 6.4. *Seleção de uma região do globo.*

Figura 6.5. *Definição dos pontos da malha.*

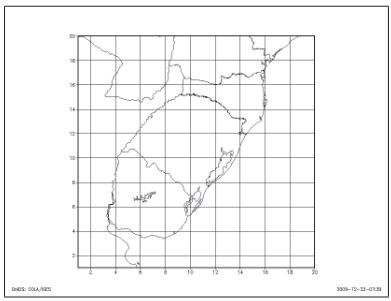

Figura 6.6. *Novos pontos na malha.*

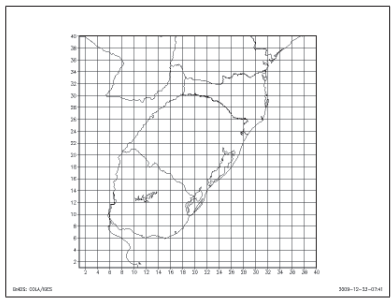

Figura 6.7. *Malha ainda mais fina.*

cálculo aumenta consideravelmente, aumentando assim a complexidade computacional do modelo. O modelo BRAMS, que emprega o método das diferenças finitas, tem complexidade $\mathbf{O}(n^3)$, tal que n é a resolução. Voltaremos a falar na interpolação na próxima seção, quando descreveremos aspectos da paralelização do BRAMS.

Para sua execução, o modelo BRAMS deve ser **inicializado**. Os dados para inicialização variam e podem ser obtidos de um modelo de escala global do CPTEC, capaz de simular o comportamento de toda atmosfera, de rádio-sondas, de satélites de tempo, de plataformas de observação, etc. Esses dados são então utilizados como ponto de partida para a previsão. Em geral, o BRAMS é inicializado com dados do Modelo Global, os quais têm valores definidos em uma malha com resolução bem mais grossa que o modelo de mesoescala. A interpolação então define os novos valores para a malha mais fina do modelo regional e o processamento de previsão do tempo é então executado calculando-se novos valores para os pontos de grande em tempos futuros.

Após a integração da previsão, a execução do BRAMS termina gerando dado bruto, o qual é pós-processado utilizando-se o RAMSPOST (do inglês *RAMS POSTpro-*

cessing), que é um pacote para gerar representação gráfica da saída do modelo. Um exemplo de saída de dados pós-processado do BRAMS pode ser visualizado na figura 6.8, que mostra linhas de temperatura plotadas no mapa. O mapa mostra uma região de mesoescala compreendendo uma região grande do Brasil e localidades de países vizinhos.

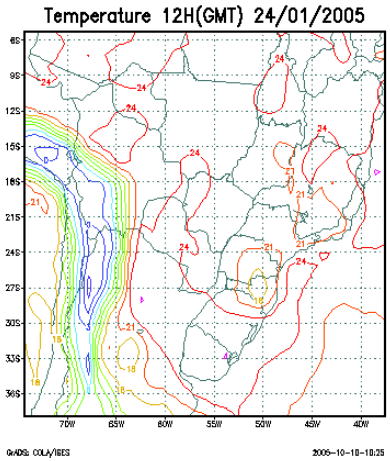

Figura 6.8. Saída pós-processada do modelo BRAMS.

Além de executar a previsão do **tempo** o BRAMS pode ser utilizado para previsão do **clima**. Ambas previsões de tempo e clima focam em previsões de temperatura, vento, nuvens, chuvas, etc. Entretanto, a maneira como elas são utilizadas é um pouco diferente. Segundo o NCAR (*The National Center for Atmospheric Research*) o tempo é uma mistura de eventos que acontecem diariamente na atmosfera e clima é o padrão médio do tempo em vários anos em um determinado lugar.

6.3.2. Paralelização do BRAMS

Como já mencionado acima, o modelo BRAMS é uma aplicação que demanda desempenho computacional. Além da complexidade computacional, outros fatores, como tempo de integração e tamanho do *ensemble* (no caso de previsão climatológica, conforme descrito na seção 6.3.3. mais adiante), também são norteadores para demanda de poder computacional [Panetta et al. 2008a].

Segundo [Panetta et al. 2008b] a complexidade computacional do BRAMS é $\mathbf{O}(n^3 v)$, tal que n^2 é o número de pontos na malha horizontal e v é o número de verticais. A terceira potência em n advém do passo de tempo que decresce com o aumento da resolução. Logo, fixos o domínio geográfico e o período de previsão, a quantidade de passos no tempo é proporcional ao número de pontos em uma dimensão da malha.

Assim, torna-se claro que uma estratégia importante para diminuir o tempo de processamento do modelo é dividir (particionar) a malha de cálculos entre os processadores. Segue-se a abordagem de particionamente de dados mencionada na seção 6.2.2., que discute metodologias para paralelização de programas.

Para a paralelização do modelo BRAMS, o conjunto de pontos de malha (ver figura 6.7), que pode ser pensado como um bloco tridimensional (i.e., colunas verticais sobre pontos de malha), é transformado em subconjuntos retangulares de pontos de malha (sub-domínios).

Nesta estrutura de sub-domínios, emprega-se o modelo mestre-escravo (*master-worker*), em que um processo mestre controla a execução dos demais processos (escravos), que são efetivamente os responsáveis pela execução do modelo [Panetta et al. 2008a].

O processo mestre inicia a computação, lendo os dados de entrada (campos meteorológicos), fazendo a divisão de domínio e enviando para os escravos (trabalhadores) o estado inicial da atmosfera apenas no trecho do domínio de cada escravo.

Os processos escravos avançam o estado da atmosfera na sua partição de domínio em passos de tempo discretos. Enquanto os escravos avançam o estado da atmosfera, o mestre nada faz. Durante cada passo de tempo, escravos comunicam com seus vizinhos para atualizar o estado da atmosfera na fronteira de seu domínio (chamadas *ghost zones*).

De tempos em tempos, escravos enviam o estado da atmosfera na sua partição de domínio para o mestre, que compõe o estado da atmosfera na totalidade do domínio e salva-o em disco (são previsões em intervalos de tempo selecionados pelo usuário, tipicamente a cada 3 horas). Também de tempos em tempos, o mestre lê novas condições de contorno (como o BRAMS é um modelo de área limitada, o estado da atmosfera na fronteira do domínio deve ser atualizado ao longo do tempo) que são comunicadas aos escravos (novamente, em intervalos de tempo selecionados pelo usuário, tipicamente um a cada 6 horas). Toda a comunicação entre escravos e entre mestre-escravo é efetuada utilizando a biblioteca de troca de mensagens MPI.

6.3.3. Execução do BRAMS para Climatologia em uma Grade Computacional

Nessa seção discute-se o arcabouço experimental do Projeto RECLIRS para a execução da climatologia de 10 anos (1992 -2001) no RS, em uma grade computacional.

O domínio ou malha computacional do projeto poder ser visualizado na figura 6.9. Vimos na figura 6.7 que, para trabalhar com uma malha computacional consideravelmente fina, deve-se refinar aos poucos o domínio de previsão. Assim, o projeto utilizou três malhas aninhadas: a malha maior englobando parte da América do Sul com 160km de resolução, a malha intermediária com parte do Brasil e países vizinhos com 80km de resolução e enfim a malha de interesse do projeto, representada na figura 6.10, considerando RS, SC e Uruguai com 20km de resolução.

Como já mencionado, além executar a previsão do **tempo** o BRAMS pode ser utilizado para previsão do **clima**, i.e., o padrão médio do tempo em vários anos em um

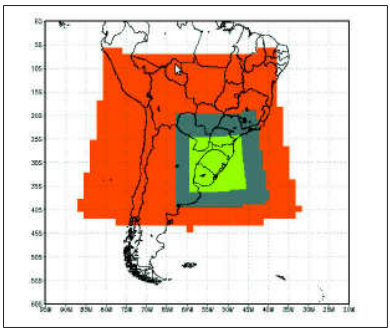

Figura 6.9. *Malhas do Projeto RECLIRS.*

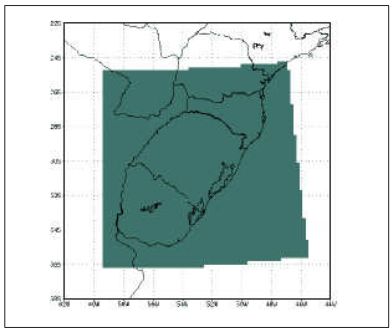

Figura 6.10. *Domínio do Projeto RECLIRS.*

determinado lugar.

Em geral, para se fazer previsão climatológica, a qual requer a execução do modelo por um período mais longo de tempo, também utiliza-se a técnica de *ensemble*. Um *ensemble* corresponde a múltiplas execuções do modelo com pequenas mudanças no estado inicial da atmosfera. Assim, utilizando-se essa estratégia constrói-se a média das previsões em um *ensemble*, minimizando o viés (ou desvio) inerente ao modelo. Portanto, simulações climatológicas requerem um alto poder computacional devido a grande quantidade de dados que necessita ser processada.

A climatologia regional no projeto RECLIRS foi obtida utilizando-se uma metodologia de *ensemble* do *European Centre for Medium-Range Weather Forecasts* (ECMWF). No ECMWF, cada ano de previsão é calculado em 12 períodos de três meses mais um mês de *spin-up*. *Spin-up* é o período necessário para o modelo se ajustar à condição inicial fornecida e estabilizar. Cada período (linha) na figura 6.11 representa uma rodada (*job*) do modelo, **independente** das demais. Os quadros com contorno vermelho representam o *spin-up* utilizado para estabilizar a próxima rodada ou período.

As figuras 6.12 e 6.13 mostram mais detalhadamente as rodadas independentes da metodologia ECMWF para cada ano. Especificamente, a figura 6.12 representa uma rodada de três meses começando em dezembro e a figura 6.13 representa uma rodada de três meses começando em janeiro. Para cada ano executa-se 12 rodadas como essas. Ao final, monta-se um *ensemble* constituído de três membros para cada mês.

Neste contexto, a estratégia do projeto foi então executar cada rodada independente em uma grade computacional composta por três nodos, sendo que cada nodo executava o BRAMS em paralelo como descrito na seção 6.3.2..

A arquitetura da grade computacional é ilustrada na figura 6.14. A grade foi constituída por quatro nodos: um cluster do convênio CRS-INPE/UFSM, formado por 21 nós HP ProLiant DL145 (AMD Opteron 2.6 GHz), um cluster da UFPel, formado por 3 nós SunFire X2200, cada um com 4 processadores *dual-core* (AMD Opteron 2.6GHz),

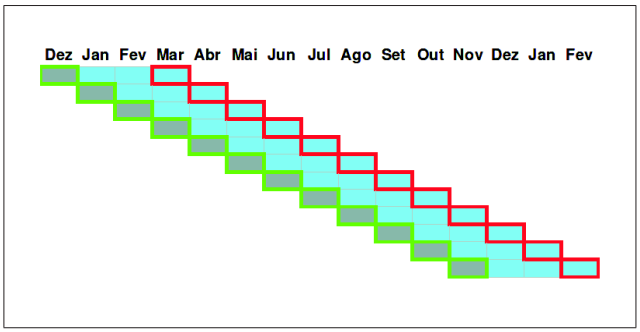

Figura 6.11. *Metodologia ECMWF.*

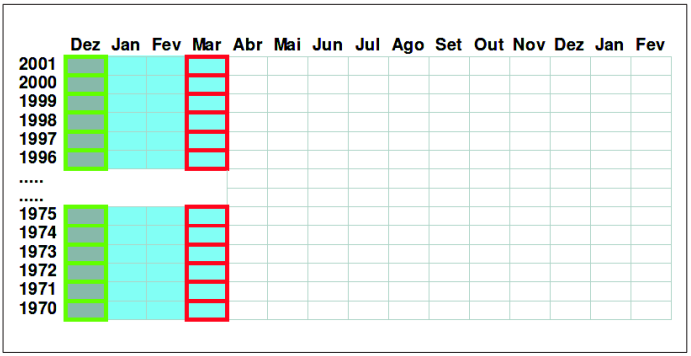

Figura 6.12. *Metodologia ECMWF.*

um cluster Cray/XD1 do LAC/INPE e outro cluster Cray/XD1 da UFRGS. Cada cluster Cray/XD1 possui 12 processadores *dual-core* (AMD Opteron 2.6GHz).

O *middleware* de grade computacional utilizado no projeto RECLIRS foi o OurGrid. O MyGrid (distribuído com o OurGrid) foi o escalonador responsável por enviar os *jobs* prontos para serem executados da base de dados para os nós da grade.

A interface da grade com os meteorologistas e pesquisadores foi desenvolvida utilizando-se a ferramenta GridSphere para construção de portais Web. A utilização do portal permite a criação de *jobs* para execução na grade e a visualização dos resultados obtidos após as execuções dos mesmos, como pode ser observado nas figuras 6.15 e 6.16. A figura 6.15 ilustra a página de entrada do portal Web do projeto e a 6.16 mostra a página com alguns *jobs* já executados, bem como a interface para adicionar novos *jobs*.

Os 10 anos de climatologia do projeto RECLIRS compreenderam um total de 360

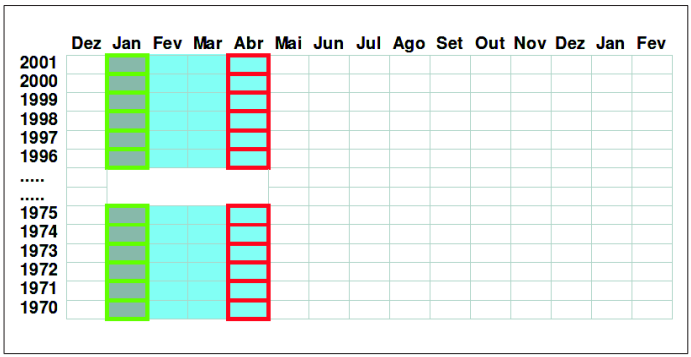

Figura 6.13. *Metodologia ECMWF.*

jobs. Esses *jobs* foram distribuídos entre 4 máquinas da grade, conforme detalhado na tabela 6.1.

Cluster	Jobs	Tempo (dias)
CRS/INPE	136 (J1)	0.64 (T1)
II-UFRGS	120 (J2)	0.52 (T2)
LAC/INPE	79 (J3)	0.63 (T3)
Met-UFPel	25 (J4)	0.76 (T4)

Tabela 6.1. Tempo de execução médio.

Portanto, o tempo total (TT) de execução sequencial dos *jobs* pode ser obtido como segue:

$$TT = J1 * T1 + J2 * T2 + J3 * T3 + J4 * T4 = 218.86 \texttt{ dias}$$

Para calcular o tempo de execução com a grade computacional, calcula-se primeiro a soma dos tempos médios:

$$STM = T1 + T2 + T3 + T4 = 2.55$$

Logo, a média ponderada de tempo de execução na grade computacional é:

$$MD = TT/STM = 85.81$$

Assim, sem a grade computacional, os *jobs* levariam 219 dias aproximadamente para executar. Com a utilização da grade, os *jobs* executaram em aproximadamente 86 dias[6]. Houve portanto um ganho de 2.5 sobre a estimativa de tempo total.

[6]Esta é uma estimativa caso as máquinas da grade estivessem sempre disponíveis para o projeto.

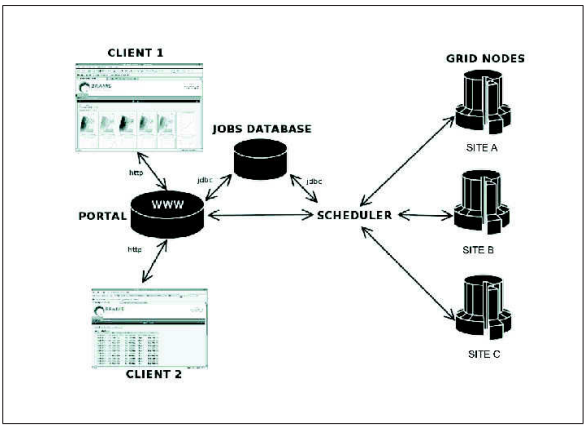

Figura 6.14. *Configuração da grade.*

6.4. Aplicação em Hidrologia: Ótica Hidrológica

Esta seção apresenta outro caso em que a computação em grade foi utilizada com sucesso para avançar uma pesquisa na área de hidrologia. No caso em questão, formou-se um sistema para investigar experimentalmente um novo método aplicado à ótica hidrológica computacional. Entende-se por ótica hidrológica uma área de estudo interdisciplinar que se interessa pelas interações entre a luz e a água, considerando sua composição e propriedades.

O método computacional em questão tem como objetivo geral determinar a **concentração de clorofila no oceano**, partindo da intensidade de radiação da luz emergente na superfície. Com este método, obtém-se uma reconstrução tomográfica 3D da concentração de clorofila em função da profundidade do oceano.

Nesta seção, discute-se primeiramente o método de reconstrução, incluindo sua justificativa e objetivo (seção 6.4.1.), para depois apresentar-se a paralelização deste método (seção 6.4.2.) e sua extensão para execução em grade (seção 6.4.3.).

6.4.1. Tomografia 3D da concentração de clorofila no oceano

A determinação da concentração de clorofila no oceano é uma importante aplicação da ótica hidrológica, com grande impacto em estudos do meio ambiente. A clorofila em águas marinhas é usada por fitoplânctons para realizar a fotossíntese e serve como indicador da "saúde"do oceano. Em condições ambientais favoráveis, estes micro-organismos se multiplicam nas águas oceânicas e ajudam a manter o equilíbrio do ecossistema. Desta

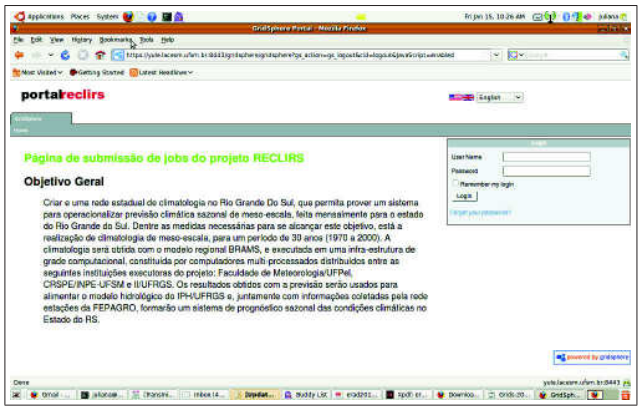

Figura 6.15. *Página de entrada no portal Web do Projeto RECLIRS.*

forma, o monitoramento da concentração de clorofila presta-se a vários setores de atividade humana, dentre eles a indústria pesqueira (onde há mais fitoplânctons há mais peixes), a meteorologia (os fitoplânctons demarcam trajetórias das correntes marítimas em imagens de satélite) e as pesquisas do setor energético (há estudos sobre a geração de biocombustíveis a partir dos fitoplânctons).

Nesta seção, discute-se o método teórico para reconstrução do perfil de clorofila apresentado em [Souto et al. 2004b]. Essencialmente, a determinação da concentração de clorofila no oceano é um problema de transferência radioativa. Em geral, são utilizadas duas abordagens para os cálculos de transferência radioativa:

- Método **direto**: o problema de transferência radioativa direta na ótica hidrológica envolve a determinação da distribuição de radiância no corpo d'água, dadas condições de contorno, propriedades ótica inerentes do meio e coefientes de absorção e espalhamento.

- Método **inverso**: o problema de transferência radioativa inversa surge quando propriedades físicas, origem de luz interna e condições de contorno precisam ser estimados a partir de medidas radiométricas do corpo de luz embaixo d'água e na sua superfície.

Em [Souto et al. 2004b] utilizou-se um método inverso para reconstrução do perfil de clorofila a partir da radiação multi-espectral que emerge da superfície do oceano, como ilustrado na figura 6.17. A radiância emergente é detectada por sensores e, a partir dessa informação, determina-se o perfil de clorofila representado pela curva abaixo da superfície do oceano.

A solução para esse problema a partir da radiância observada é dada por um algoritmo de otimização, i.e., busca-se a melhor solução entre candidatas. A técnica utilizada

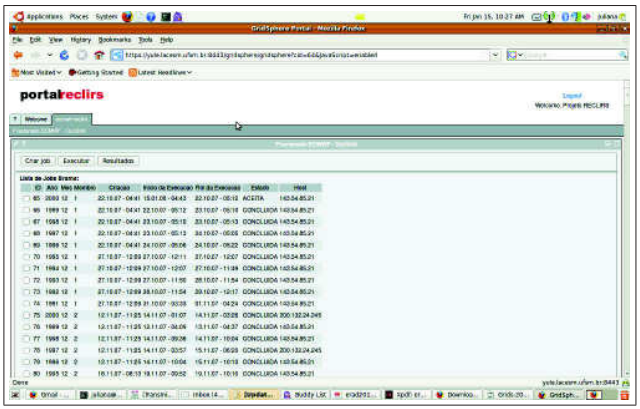

Figura 6.16. *Jobs no portal Web do Projeto RECLIRS.*

para esse problema de otimização chama-se algoritmo da **colônia de formigas** (do inglês, *ant-colony*) [Dorigo et al. 1996]. Basicamente, o algoritmo de otimização de colônia de formigas imita formigas chegando até a comida, ou seja uma geração aprendendo o caminho com a outra. No mundo real, as formigas inicialmente circulam randomicamente à procura de comida. Então quando elas encontram comida, elas retornam a sua colônia deixando trilhas de feromônio (i.e., substância química que permite o reconhecimento mútuo e sexual entre alguns animais). Assim, se outras formigas encontram esse caminho, elas o seguem ao invés de circularem randomicamente.

Entretanto, a trilha de feromônio evapora com o tempo. Quanto maior a trilha menor a intensidade do feromônio, quanto menor a trilha maior a intensidade do feromônio. Assim, trilhas mais curtas ficam marcadas por mais tempo. Logo, quando uma formiga encontra um bom caminho (i.e., curto) que leva a colônia até a origem de comida, outras formigas também seguirão aquele caminho. Formigas também seguem um *feedback* positivo, i.e., após encontrar um caminho curto todas as outras formigas seguem aquele caminho. Portanto, a ideia do algoritmo da **colônia de formigas** é imitar esse comportamento através da simulação de formigas no grafo representando as possíveis soluções do problema.

No método de otimização da colônia de formigas utiliza-se várias gerações de formigas. Para cada geração, uma quantidade fixa de formigas é avaliada. Cada formiga é associada com um caminho possível, o qual representa uma possível solução do problema. Cada caminho é visto como uma ligação entre os nodos no grafo que contém todas as possíveis soluções do problema. A melhor formiga de cada geração é então escolhida, e a melhor formiga escolhida então influencia a criação de formigas na nova geração. Ao final do algoritmo, encontra-se o melhor caminho no grafo de possíveis soluções.

Sumarizando a técnica, dado um valor de radiância observada, $L^{exp}(\Delta, \lambda)$ busca-se

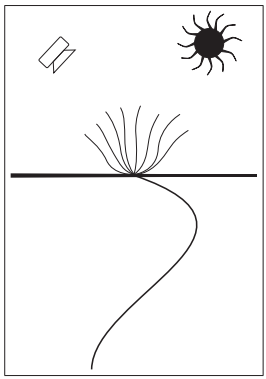

Figura 6.17. *Método inverso para reconstrução do perfil de clorofila.*

encontrar a função objetiva, $f_{obj}(C)$, que **melhor** descreva o perfil de clorofila utilizando o algoritmo da colônia de formigas. Supõe-se um valor inicial para C e, utilizando-se um valor de radiância do modelo, aplica-se a fórmula:

$$|L^{exp}(\Delta,\lambda) - L^{mod}(\Delta,\lambda)| = f_{obj}(C)$$

iterativamente até encontrar o valor de C que mais se aproxime a realidade.

6.4.2. Paralelização da aplicação

A partir da explicação do método de otimização da colônia de formigas na seção anterior, pode-se perceber que este é um tipo de método de busca de menor caminho em grafos, o qual tem um custo computacional intensivo. Uma implementação paralela deste método foi apresentada em [Souto et al. 2004a], onde foi paralelizado o trecho do código que avalia a função objetiva de todas as soluções/formigas geradas em cada iteração do método.

No estudo de caso apresentado neste minicurso, simulou-se um caso específico com dimensões de 60km $\times$ 60km e 40m de profundidade para o domínio espacial do oceano. Assim, o domínio horizontal é uniformemente distribuído em 36 regiões menores de 10km $\times$ 10km, conforme ilustrado na figura 6.18.

Neste caso, tem-se três perfis de clorofila no domínio total para serem reconstruídos. A distinção entre cada perfil pode ser observada de acordo com a coloração das regiões na figura 6.18. Como pode ser observado, existem 20 sub-regiões com perfil-1, 12 com perfil-3 e 4 sub-regiões que correspondem ao perfil-2. Para cada perfil existe um conjunto de valores de radiância multi-espectral que vem da superfície do oceano.

Com o objetivo de melhorar a solução inversa, utilizou-se uma estratégia em dois passos: no $passo-1$ calcula-se uma primeira solução para o problema e, no $passo-2$,

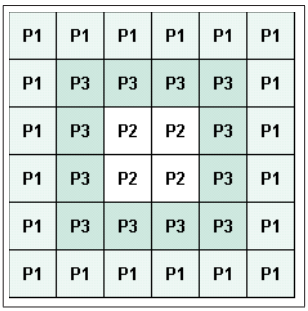

Figura 6.18. *Domínio espacial da distribuição dos perfis.*

processa-se a reconstrução para a camada de maior profundidade, fixando-se os valores obtidos na camada de menor profundidade durante o $passo - 1$. Em outras palavras, o $passo - 2$ é um outro problema inverso, porém mais simples que o problema original, pois o problema neste segundo passo é de menor dimensão e parte de uma melhor estimativa inicial.

O ponto chave neste esquema é que a reconstrução dos perfis de clorofila em cada região é independente das outras regiões, i.e., o conjunto de cálculos para solução do problema inverso de determinação do perfil de clorofila pode ser tratado com uma aplicação do tipo "mala de tarefas" (do inglês *bag-of-tasks*).

Portanto, o uso de um ambiente de grade computacional para executar o cálculo da inversão do domínio espacial completo é uma escolha natural.

6.4.3. Execução em grade computacional

Nesse experimento, configurou-se uma grade computacional com três clusters geograficamente separados no Brasil: Computação da UFSM, Instituto de Informática da UFRGS e LAC/INPE, localizados em Santa Maria, Porto Alegre e São José dos Campos, respectivamente. A descrição do hardware da grade é apresentada na tabela 6.2.

Tabela 6.2. Descrição do hardware utilizado na grade

Instituição	Equipamento
Computação/UFSM	SGI Altix XE server with 8 cores (2 Intel Xeon quad-core 2.0 GHz)
II/UFRGS	Cluster Cray XD1 with 4 processors AMD Opteron, 2.8 GHz
INPE/LAC	Cluster Cray XD1 with 8 processors AMD Opteron, 2.8 GHz

O *midleware* de grade instalado foi o OurGrid. Essencialmente, utilizou-se três componentes do OurGrid:

- **mygrid**: a interface que o usuário utiliza, na sua máquina *home*, para submissão e execução de *jobs*;
- **peer**: fornece os computadores conectados na máquina *home*;
- **useragent**: executa em cada máquina da grade. São as máquinas que executam as tarefas no OurGrid.

Para esse trabalho, o componente *peer* foi instalado em máquinas que não fizeram parte do OurGrid (nos *gateways* da Internet), e o *useragent* foi instalado no restante das máquinas da grade.

Cada *job* da grade correspondeu a um *pixel* ou sub-domínio da superfície do oceano, vetorizando-se as radiâncias associadas a cada *pixel* e preparando-se tais radiâncias para a execução do problema inverso.

Foram executados um total de 56 *jobs* na grade computacional, tal que 36 foram relativos ao $passo-1$ de reconstrução dos perfis e 20 foram relativos ao $passo-2$ somente para o perfil-1.

Define-se o tempo acumulado que o *midleware* executou nos clusters, T_p, como o tempo total de execução dos *jobs* submetidos aos nodos da grade através do OurGrid. Esse tempo acumulado corresponde a execução sequencial de um conjunto de *jobs*. A tabela 6.3 apresenta as medidas de T_p para cada cluster durante o experimento.

Tabela 6.3. Tempo acumulado de uso dos clusters (horas)

Cluster	Jobs	T_p (hh:mm)	T_p/Jobs (hh:mm:ss)
Computação-UFSM	30	07:20	00:14:40
II-UFRGS	9	06:48	00:45:20
INPE/LAC	17	07:26	00:26:11
	56	21:34	00:23:06

O experimento consistiu de 56 *jobs* e levou $21hs$ e $34min$ para completar, como ilustra a tabela 6.3. Todas as medidas apresentadas nesse texto foram extraídas dos arquivos de *log* do OurGrid, o qual registra os tempos de início e fim de cada *job*.

Também apresenta-se na tabela 6.3 o tempo médio de duração de um único *job* em cada cluster. Essa análise mostra que cada cluster tem seu tempo médio de execução diferente dos outros. O primeiro cluster (computação da UFSM) apresenta um tempo de execução menor. Isso pode ser explicado pela arquitetura da máquina multicore com memória compartilhada, a qual reduz o custo de comunicação na implementação paralela do algoritmo de otimização da colônia de formigas.

Como as computações foram eventualmente executadas simultaneamente (dependendo do número de nodos da grade disponíveis), o tempo de uso da grade necessário

para executar todo o conjunto de *jobs* é definido como tempo total da grade, T_g, o qual é naturalmente menor que T_p. Para esse experimento, considerou-se que o tempo total da grade foi 07 : 43 horas. O tempo ganho com a execução na grade pode ser obtido pela média T_p/T_g, o qual é 2.79 para esse experimento.

Também analisou-se o comportamento da grade executando *jobs* simultâneos nos nodos disponíveis. A tabela 6.4 mostra o tempo total da grade acumulado executando *jobs* simultaneamente em 3 e 2 clusters, e também *jobs* executados em um único cluster. A percentagem significa a média entre o tempo de acumulado em um único cluster e o tempo total da grade. Pode-se concluir, a partir da tabela 6.4, que a política de escalonamento do OurGrid mantém 2 ou 3 clusters ocupados a maioria do tempo. Entretanto, não utiliza a capacidade **total** de todos os clusters.

Tabela 6.4. Uso simultâneo dos clusters.

Número de clusters executando simultanemente	% of tempo acumulado
3	47 %
2	49 %
1	4 %
média: 2.43	

Finalmente, na tabela 6.5, apresenta-se a estatística de uso dos clusters calculada a partir da média entre T_P (para cada cluster) e T_g. Essa estatística confirma que cada cluster estava ocupado a maior parte do tempo. Nota-se, entretanto, que a capacidade da grade pode ser explorada um pouco mais.

Tabela 6.5. Estatística de uso dos clusters.

Cluster	Uso do cluster
Computação-UFSM	95.0%
II-UFRGS	88.1%
INPE/LAC	96.3%

6.5. Considerações Finais

Neste minicurso descreveu-se dois casos práticos de utilização de grades computacionais. Ambos os casos foram aplicações computacionalmente intensivas que foram paralelizadas para execução em clusters, usando uma metodologia que compreendeu o particionamento do trabalho e o estabelecimento de comunicações entre tarefas. Nos dois casos considerados, eram necessárias múltiplas execuções em cluster para obter os resultados desejados, utilizando diferentes dados de entrada. Assim, essas execuções foram configuradas como *jobs* que foram distribuídos nas grades especialmente montadas para este fim.

Os resultados mostraram os benefícios das grades computacionais, bem como as possibilidades práticas de uso integrado de diferentes ferramentas para clusters e grades.

Referências

[Barney 2009] Barney, B. (2009). Message Passing Interface (MPI) Tutorial. https://computing.llnl.gov/tutorials/mpi/.

[Chandra et al. 2001] Chandra, R., Dagum, L., Kohr, D., Maydan, D., McDonald, J., and Menon, R. (2001). *Parallel programming in OpenMP*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

[Chapman et al. 2007] Chapman, B., Jost, G., and Pas, R. v. d. (2007). *Using OpenMP: Portable Shared Memory Parallel Programming (Scientific and Engineering Computation)*. The MIT Press.

[Dongarra et al. 2009] Dongarra, J. J., Meuer, H. W., and Strohmaier, E. (2009). TOP500 supercomputer sites. Technical report.

[Dorigo et al. 1996] Dorigo, M., Maniezzo, V., and Colorni, A. (1996). Ant system: Optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man, and Cybernetics*, 1(26):29–41.

[Fazenda et al. 2007] Fazenda, A. L., Moreira, D. S., Enari, E. H., Panetta, J., and Rodrigues, L. F. (2007). *First Time User's Guide (BRAMS Version 4.0)*.

[Flynn 1972] Flynn, M. J. (1972). Some computer organizations and their effectiveness. *IEEE Transactions on Computers*, 21(9):948–960.

[Forum 1994] Forum, M. P. I. (1994). Mpi: A message-passing interface standard. Technical report, Knoxville, TN, USA.

[Forum 1997] Forum, M. P. I. (1997). MPI-2: Extensions to the message passing interface. Technical report, MPI Forum.

[Foster 1995] Foster, I. (1995). *Designing and Building Parallel Programs: Concepts and Tools for Parallel Software Engineering*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

[Foster 2002] Foster, I. (2002). What is the grid? a three point checklist. *Grid Today*, 1(6):22–25.

[Foster and Kesselman 1999] Foster, I. and Kesselman, C. (1999). The grid: Blueprint for a new computing infrastructure. *Morgan Kauffmann, San Francisco, CA*, 211.

[Foster et al. 2001] Foster, I., Kesselman, C., and Tuecke, S. (2001). The anatomy of the grid - enabling scalable virtual organizations. *International Journal of Supercomputer Applications*, 15:2001.

[Gabriel et al. 2004] Gabriel, E., Fagg, G. E., Bosilca, G., Angskun, T., Dongarra, J. J., Squyres, J. M., Sahay, V., Kambadur, P., Barrett, B., Lumsdaine, A., Castain, R. H., Daniel, D. J., Graham, R. L., and Woodall, T. S. (2004). Open MPI: Goals, concept, and design of a next generation MPI implementation. In *Proceedings, 11th European PVM/MPI Users' Group Meeting*, pages 97–104, Budapest, Hungary.

[Gamma et al. 1994] Gamma, E., Helm, R., Johnson, R., and Vlissides, J. M. (1994). *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional.

[GridSphere Project 2007] GridSphere Project (2007). Gridsphere portlet reference guide. http://docs.gridsphere.org/display/gs30/Portal+Reference+Guide.

[Gropp et al. 1996] Gropp, W., Lusk, E., Doss, N., and Skjellum, A. (1996). A high-performance, portable implementation of the MPI message passing interface standard. *Parallel Computing*, 22(6):789–828.

[Gropp and Lusk 1996] Gropp, W. D. and Lusk, E. (1996). *User's Guide for* `mpich`*, a Portable Implementation of MPI*. Mathematics and Computer Science Division, Argonne National Laboratory. ANL-96/6.

[I. Foster and C. Kesselman 1997] I. Foster and C. Kesselman (1997). Globus: A metacomputing infrastructure toolkit. *International Journal of Supercomputer Applications and High Performance Computing*, 11(2):115–128. DisponÃvel em: ftp://ftp.globus.org/pub/globus/papers/globus.pdf.

[Karypis and Kumar 1995] Karypis, G. and Kumar, V. (1995). METIS - unstructured graph partitioning and sparse matrix ordering system, version 2.0. Technical report.

[Kaw and Kalu 2008] Kaw, A. K. and Kalu, E. E. (2008). *Numerical Methods with Applications*. Florida A&M University.

[Mattson et al. 2004] Mattson, T., Sanders, B., and Massingill, B. (2004). *Patterns for parallel programming*. Addison-Wesley Professional.

[Nagel et al. 1996] Nagel, W. E., Arnold, A., Weber, M., Hoppe, H.-C., and Solchenbach, K. (1996). Vampir: Visualization and analysis of mpi resources.

[Novotny et al. 2004] Novotny, J., Russell, M., and Wehrens, O. (2004). GridSphere: an advanced portal framework. *Euromicro Conference, 2004. Proceedings. 30th*, pages 412–419.

[OurGrid 2008] OurGrid (2008). Ourgrid website. http://www.ourgrid.org/.

[Pacheco 1996] Pacheco, P. S. (1996). *Parallel programming with MPI*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

[Panetta et al. 2008a] Panetta, J., Navaux, P., Fazenda, A., and et al (2008a). Relatório técnico i: Projeto cnpq atmosfera massiva (275/07). Technical report, Instituto Nacional de Pesquisas Espaciais.

[Panetta et al. 2008b] Panetta, J., Navaux, P., Fazenda, A., and et al (2008b). Relatório técnico ii: Projeto cnpq atmosfera massiva (275/07). Technical report, Instituto Nacional de Pesquisas Espaciais.

[Pielke 1984] Pielke, R. A. (1984). *Mesoscale meteorological modeling*. Academic Press, Orlando,Florida.

[Quinn 2003] Quinn, M. J. (2003). *Parallel Programming in C with MPI and OpenMP*. McGraw-Hill Education Group.

[Shende and Malony 2006] Shende, S. S. and Malony, A. D. (2006). The tau parallel performance system. *Int. J. High Perform. Comput. Appl.*, 20(2):287–311.

[Skillicorn 1988] Skillicorn, D. B. (1988). A taxonomy for computer architectures. *Computer*, 21(11):46–57.

[Skillicorn 2001] Skillicorn, D. B. (2001). Motivating computational grids. In *Proceedings of the 2nd IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid2002), IEEE Computer Society*, pages 401–406.

[Souto et al. 2004a] Souto, R. P., CamposVelho, H. F., Stephany, S., and Sandri, S. (2004a). Reconstruction of chlorophyll concentration profile in offshore ocean water using a parallel ant colony code. In *Anais...*, pages 19–24, São Paulo. European Conference on Artificial Intelligence / Hybrid Metaheuristics.

[Souto et al. 2004b] Souto, R. P., de Campos Velho, H. F., Stephany, S., and Sandri, S. A. (2004b). Reconstruction of chlorophyll concentration profile in offshore ocean water using a parallel ant colony code. In *Hybrid Metaheuristics*, pages 19–24.

[Watson 2010] Watson, G. (2010). PTP - Eclipse Parallel Tools Platform. http://www.eclipse.org/ptp/.

[Watson et al. 2006] Watson, G., Rasmussen, C., and Tibbitts, B. (2006). Application development using eclipse and the parallel tools platform. In *SC '06: Proceedings of the 2006 ACM/IEEE conference on Supercomputing*, page 204, New York, NY, USA. ACM.

[Wilkinson and Allen 2004] Wilkinson, B. and Allen, M. (2004). *Parallel Programming: Techniques and Applications Using Networked Workstations and Parallel Computers (2nd Edition)*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA.

[Zaki et al. 1999] Zaki, O., Lusk, E., and Swider, D. (1999). Toward scalable performance visualization with jumpshot. *High Performance Computing Applications*, 13:277–288.