XXIII ERAD/RS

Escola Regional de Alto Desempenho da Região Sul

MINICURSOS

ERAD/RS 2023

Porto Alegre-RS - 10 a 12 de maio

Fomento

FAPERGS

Realização

SBC Sociedade Brasileira de Computação

Organização

PUCRS

UNIVERSIDADE FEDERAL DE SANTA CATARINA

Apoio

Universidade Federal de Pelotas RS-Brasil

unipampa Universidade Federal do Pampa

UFRGS Universidade Federal do Rio Grande do Sul

FURG Universidade Federal do Rio Grande

unioeste Universidade Estadual do Oeste do Paraná

UNIJUÍ Universidade Regional

UNISINOS

UDESC

Patrocinadores

Diamante

DELL Technologies

SCHERM

SBC

SEAGATE

Ouro

globo

versatus HPC

Hewlett Packard Enterprise

Edson Luiz Padoin
Guilherme Galante
Rodrigo Righi

MINICURSOS DA XXIII ESCOLA REGIONAL DE ALTO DESEMPENHO DA REGIÃO SUL (ERAD/RS)

Porto Alegre
Sociedade Brasileira de Computação – SBC
2023

Dados Internacionais de Catalogação na Publicação (CIP)

E74 Escola Regional de Alto Desempenho da Região Sul (23. : 10 – 12 maio 2023 : Porto Alegre)

Minicursos da ERAD-RS 2023 [recurso eletrônico] / organização: Edson Luiz Padoin ; Guilherme Galante ; Rodrigo Righi. Dados eletrônicos. – Porto Alegre: Sociedade Brasileira de Computação, 2023.

92 p. : il. : PDF ; 4.10MB

Modo de acesso: World Wide Web.
Inclui bibliografia
ISBN 978-85-7669-538-7 (e-book)

1. Computação – Brasil – Evento. 2. Processamento de Alto Desempenho. 3. Inteligência Artificial. 4. Compressão de dados. 5. Internet das Coisas. 6. Aprendizado de Máquina. I. Padoin, Edson Luiz. II. Galante, Guilherme. III. Righi, Rodrigo. IV. Sociedade Brasileira de Computação. VI. Título.

CDU 004(063)

Ficha catalográfica elaborada por Annie Casali – CRB-10/2339

Biblioteca Digital da SBC – SBC OpenLib

Índices para catálogo sistemático:

1. Ciência e tecnologia dos computadores : Informática – Publicação de conferências, congressos e simpósios etc. ... 004(063)

ERAD/RS 2023

XXIII Escola Regional de Alto Desempenho da Região Sul
10 a 12 de maio de 2023
https://cradrs.github.io/eradrs2023/

A XXIII Escola Regional de Alto Desempenho da Região Sul (ERAD/RS 2023) acontece entre os dias 10 e 12 de maio de 2023, na cidade de Porto Alegre, no campus da Pontifícia Universidade Católica do Rio Grande do Sul (PUCRS). A ERAD/RS é promovida anualmente pela Sociedade Brasileira de Computação (SBC) e pela Comissão Regional de Alto Desempenho da Região Sul (CRAD/RS).

O público alvo da ERAD/RS 2023 são alunos, profissionais e professores/pesquisadores que atuam direta ou indiretamente na computação de alto desempenho e em áreas correlatas. O evento engloba a região sul do Brasil (RS, SC e PR).

Os principais objetivos são:

- Qualificar os profissionais do sul do Brasil nas áreas que compõem o processamento de alto desempenho;
- Prover um fórum regular onde possam ser apresentados os avanços recentes nessas áreas;
- Discutir formas de ensino de processamento de alto desempenho nas universidades.

A programação da ERAD/RS 2023 foi composta por sessões técnicas, com a apresentação de 41 trabalhos nos Fóruns de Iniciação Científica e de Pós-Graduação. Além disso, o evento proporcionou aos participantes 6 minicursos, 2 palestras científicas, 7 palestras industriais, o workshop Women in High Performance Computing, 2 painéis para discussão sobre avanços na área, 1 tutorial sobre redes neurais, Maratona de Programação Paralela e a reunião anual da CRADRS.

A edição da ERAD/RS 2023 foi coordenada pelos professores Tiago Ferreto (PUCRS), Dalvan Griebler (PUCRS/Setrem) e Odorico Mendizabal (UFSC). O Fórum e Iniciação Científica foi coordenado pelos professores André Dubois (UFPEL) e Cláudio Schepke (UNIPAMPA); o Fórum de Pós- Graduação foi coordenado pelos professores Lucas Schnorr (UFRGS) e Vinícius Pinto (FURG); os Minicursos foram coordenados pelos professores Guilherme Galante (UNIOESTE), Edson Padoin (UNIJUI) e Rodrigo Righi (UNISINOS); a Maratona de Programação Paralela foi coordenada pelos professores Mauricio Pilon (UDESC) e Guilherme Koslovski (UDESC); o WHPC foi coordenado pela professora Andriele do Carmo (UNISINOS); o painel foi mediado pelos professores César De Rose (PUCRS) e Adenauer Yamin (UCPEL/UFPEL). Além disso, a equipe de apoio local foi composta pelos discentes Jessé da Silva Rodrigues (PUCRS), Paulo Severo de Souza (PUCRS), Carlos Kayser (PUCRS), Lucas Roges (PUCRS) e Henrique Brum (PUCRS).

Índice

Mensagem da Coordenação Geral

É com grande satisfação que saudamos e damos as boas vindas à vigésima terceira edição da Escola Regional de Alto Desempenho da Região Sul, a ERAD/RS 2023, que neste ano acontece na cidade de Porto Alegre, capital do estado do Rio Grande do Sul, entre os dias 10 e 12 de maio. A ERAD/RS é um evento anual, promovido pela Sociedade Brasileira de Computação (SBC), por meio da Comissão Regional de Alto Desempenho da Região Sul (CRAD-RS), desde 2001.

Esta escola, de caráter essencialmente regional, tem como objetivo qualificar profissionais da região sul do Brasil nas áreas relacionadas ao Processamento de Alto Desempenho (PAD) e proporcionar um fórum regular no qual se possa apresentar os avanços recentes nessas áreas e discutir as formas de ensino de processamento de alto desempenho nas Instituições de Ensino Superior (IES) do sul do Brasil.

Além disso, as atividades da ERAD/RS aproximam academia, sociedade e indústria. A troca de experiências e conhecimento de novidades é estimulada com a intercalação de palestras com pessoas de renome na área, apresentações de trabalhos acadêmicos e relatos do mercado e indústria. Essa sinergia é ampliada pela interação de professores, estudantes e profissionais de diversas regiões do Brasil, a qual possibilita analisar questões por uma visão diferente da sua realidade regional.

Para ampliar o alcance e potencializar seus resultados, a ERAD/RS é itinerante, sendo abrigada, a cada ano, por diferentes instituições. Neste ano, coube à Pontifícia Universidade Católica do Rio Grande do Sul (PUCRS) gerenciar a execução deste encontro com o apoio da Universidade Federal de Santa Catarina (UFSC). A região sul do Brasil é, há muitos anos, referência no país na área de PAD. A ERAD/RS reflete este fato e reforça esta posição de destaque quando se preocupa em formar novos pesquisadores e em manter atualizados os pesquisadores que criaram suas bases nos estados do sul.

Foi um prazer nos envolvermos na organização deste grande evento, que é a ERAD/RS. Reconhecemos a tradição do evento e reforçamos a importância de todas as instituições que acompanham a evolução e a história da ERAD/RS, colaborando de forma efetiva para sua realização.

Neste ano, a ERAD/RS contou com o apoio da Fapergs (Fundação de Amparo à pesquisa do Estado do RS), além de patrocínios das empresas Dell Technologies, Scherm, SDC, Seagate, Globo, Versatus, Hewlett Packard Enterprise e Atos.

Agradecemos, em especial, a todo(a)s o(a)s autore(a)s que submeteram trabalhos às sessões técnicas, ao corpo de revisores que compuseram o comitê de programa, aos patrocinadores e aos convidados que prontamente aceitaram nossos convites e, sem dúvida, engrandecem a ERAD/RS com as suas participações.

Por fim, agradecemos todos os coordenadores dos eventos da ERAD/RS, que tomaram para si diversos encargos e os conduziram com pleno sucesso.

Obrigado a todos pela presença e desejamos que a ERAD/RS 2023 seja muito proveitosa.

Tiago Ferreto (PUCRS), Dalvan Griebler (PUCRS/Setrem) e Odorico Mendizabal (UFSC)
Coordenadores gerais da ERAD/RS 2023

Mensagem da Coordenação dos Minicursos

A Escola Regional de Alto Desempenho da Região Sul (ERAD/RS) é um evento anual promovido pela Sociedade Brasileira de Computação (SBC) e pela Comissão Regional de Alto Desempenho da Região Sul (CRAD/RS). A escola, que neste ano completa os seus vinte e três anos, foi realizada entre os dias 10 e 12 de maio de 2023, na cidade de Porto Alegre/RS, no campus sede da Pontifícia Universidade Católica do Rio Grande do Sul (PUCRS).

Um dos objetivos da ERAD/RS é qualificar profissionais da região sul nas diversas áreas que envolvem e Processamento de Alto Desempenho (PAD). Com este intuito, todo o ano são selecionados minicursos introdutórios e avançados em tópicos estratégicos e de interesse à comunidade. Não diferente, neste ano de 2023 foram selecionados seis minicursos, dos quais cinco estão formatados como capítulo deste livro. Os minicursos aqui representados, apresentam tópicos de ponta da área de PAD, os quais irão certamente agradar os participantes do evento. Em particular, na presente edição, estamos abordando: Aprendizado de Máquina, Inteligência Artificial, Compressão de Dados, Internet das Coisas, Projeto de Aplicações Paralelas, Políticas DevOps, Otimizações no uso de Clusters e Programação Eficiente com OpenMP. Entendemos que a área de PAD está passando por uma transformação, onde sedimentados conhecimentos de Pthreads, MPI e OpenMP são revisitados frente à demandas crescentes nas mais diversas áreas do conhecimento, como saúde, cidades inteligentes, agricultura, automação de sistemas, chats inteligentes, dentre outros.

Os coordenadores dos minicursos agradecem aos autores, por compartilharem seus conhecimentos através da submissão de minicursos de alto nível para esta edição da escola. Agradecemos também aos coordenadores e organizadores da ERAD/RS 2023, pelo apoio dado na seleção dos minicursos e na realização do evento. Por fim, acreditamos que o conhecimento é a mola propulsora de uma nova sociedade com foco em tecnologia e ética e os conteúdos aqui apresentados caminham fortemente nessa direção. Desejamos a todos uma ótima ERAD/RS 2023, com muito aprendizado e troca de conhecimentos.

Edson Luiz Padoin (UNIJUÍ), Guilherme Galante (UNIOESTE), Rodrigo da Rosa Righi (UNISINOS)
Coordenadores dos Minicursos da ERAD/RS 2023

Comitês Organizadores

Coordenação Geral

- Tiago Ferreto (PUCRS)
- Dalvan Griebler (PUCRS/Setrem)
- Odorico Mendizabal (UFSC)

Fórum de Pós-Graduação

- Lucas Schnorr (UFRGS)
- Vinícius Pinto (FURG)

Fórum de Iniciação Científica

- André Dubois (UFPEL)
- Cláudio Schepke (UNIPAMPA)

Minicursos

- Edson Luiz Padoin (UNIJUI)
- Guilherme Galante (UNIOESTE)
- Rodrigo Righi (UNISINOS)

Maratona Paralela

- Mauricio Pilon (UDESC)
- Guilherme Koslovski (UDESC)

Painel

- César De Rose (PUCRS)
- Adenauer Yamin (UCPEL/UFPEL)

Women in HPC

- Andriele do Carmo (UNISINOS)

Minicursos

Capítulo 1

Diretivas Paralelas de OpenMP: Um Estudo de Caso

Claudio Schepke, Natiele Lucca
Universidade Federal do Pampa

João Vicente Ferreira Lima
Universidade Federal de Santa Maria

Resumo

OpenMP é uma interface de programação paralela usada para a geração implícita de threads em CPUs e GPUs, o que possibilita uma computação heterogênea. O padrão é baseado em diretivas de compilação, isto é, linhas de comando são inseridas em um código-fonte e tratadas em tempo de pré-compilação. O paralelismo é gerado pelas diretivas, as quais podem delimitar a região concorrente de código. Apesar da simplicidade de injeção de paralelismo em um determinado programa, a grande maioria dos usuários da interface limita-se a utilizar recursos elementares, como o paralelismo de laços em ambientes multiprocessados em detrimento de outras formas de invocação de operações concorrentes não tão difundidas. Este capítulo tem como objetivo apresentar como diferentes tipos de diretivas de OpenMP podem ser chamadas e seus impactos no desempenho paralelo em uma aplicação. Este estudo de caso busca mensurar especialmente a performance dos recursos disponíveis pelas versões mais recentes do padrão OpenMP.

1.1. Introdução

A interface de programação paralela OpenMP provê um conjunto de diretivas de compilador, bibliotecas de rotinas/funções, variáveis de ambiente e suporte a ferramentas para o gerenciamento, depuração e análise de programas paralelos em memória compartilhada e em aceleradores (OpenMP 2023). As diretivas multiplataforma estendem as linguagens de programação Fortran, C e C++ com construções do tipo: SPMD (*Single Program Multiple Data*), tarefas (*tasks*), dispositivos (*device*), distribuição de trabalho e sincronização. Além disso, as diretivas fornecem suporte para compartilhamento, mapeamento e privatização de dados. O controle do ambiente em tempo de execução é suportado por rotinas/funções da biblioteca e variáveis de ambiente. Os compiladores que suportam a

interface OpenMP geralmente incluem opções para habilitar ou desabilitar a interpretação de algumas ou todas as diretivas OpenMP por linha de comando.

A primeira especificação de OpenMP surgiu, para a escrita de códigos em Fortran, em outubro de 1997, e, para C/C++, em outubro de 1998, em uma tentativa das principais fabricantes de computadores da época de unificar o padrão de diretivas de compartilhamento de memória (Dagum and Menon 1998). Com o passar do tempo, outros fabricantes também passaram a colaborar com a especificação através de um conselho de revisão de arquitetura. Pelo fato da especificação ser um padrão dominante para um ambiente de programação paralela em memória compartilhada, diferentes fornecedores de compiladores suportam implementações da interface. Com isso, a API fornece um modelo de programação paralelo que é portável entre diferentes arquiteturas de computadores.

Em novembro de 2022 foi anunciada uma primeira descrição da versão 6 da especificação da interface OpenMP. Nela foram adicionadas novas funcionalidades e removidos recursos já considerados obsoletos em versões de especificações mais antigas da API. Embora uma apresentação da versão 6 já tenha sido feita, continuam ainda atuais e válidas as versões 5.2 (novembro de 2021) e 5.1 (novembro de 2020) da especificação. Vale lembrar que nem todos os compiladores implementam as modificações que ocorrem em novas versões das especificações imediatamente. Outros compiladores implementam apenas alguns dos recursos de uma determinada versão avançada da especificação.

Além do atraso natural existente entre o lançamento de uma versão da especificação e a implementação dela em software para uma nova versão de um compilador, há também uma aparente demora na utilização dos recursos por parte dos programadores. Por exemplo, a versão 3 introduziu a noção de tarefa (*tasks*) em maio de 2008. A versão 4 incorporou a construção de diretivas para uso de aceleradores (GPUs, coprocessadores, FPGAs, Cell BE, ...) e operações do tipo SIMD (*Single Instruction Multiple Data*) em julho de 2013. Já a versão 5, desde novembro de 2018, provê melhorias como: suporte completo para aceleradores, depuração aprimorada e análise de desempenho, suporte para uma construção de laços que permite que o compilador escolha uma boa implementação para um dispositivo específico e sistemas de memória multinível. Embora as versões mais recentes incorporam um poder de abstração significativamente alto e permitem instanciar rotinas/funções em multi-core, GPUs e outros tipos de hardware, a popularização destas diretivas tem sido lenta.

Diante do que foi previamente exposto, este capítulo apresenta algumas diretivas paralelas fornecidas por OpenMP. Na sequência, como estudo de caso, a paralelização de uma aplicação de dinâmica dos fluidos ((da Silva et al. 2022b)) é acelerada através da adoção de diferentes diretivas que possibilitam executar o código, tanto em arquitetura multi-core como em GPU. Cada uma das versões de código com diretiva paralela distinta tem o seu tempo de execução mensurado para fins de comparação.

As próximas seções apresentam aspectos iniciais que precisam ser observados na paralelização de aplicações (Seção 1.2), uma discriminação das diretivas paralelas de OpenMP que podem ser usadas (Seção 1.3 e uma descrição da aplicação que servirá como estudo de caso (Seção 1.4). Na sequência, são mostrados exemplos de trechos de código implementados com as diretivas paralelas de OpenMP (Seção 1.5), seguido dos resultados experimentais obtidos para a avaliação da performance para cada tipo de diretiva adotada

(Seção 1.6). O capítulo encerra com a conclusão (Seção 1.7).

1.2. Paralelização de Aplicações

Antes de iniciar propriamente a paralelização de qualquer aplicação é necessário um estudo prévio, a fim de identificar os trechos de código que podem ser paralelizados. Um código possui trechos que não são paralelizáveis. Este é o caso de etapas de pré e pós processamento, que incluem a leitura ou escrita de arquivos, a alocação de memória e a inicialização de variáveis. Em outras situações, há trechos de códigos em que o custo de instanciação da execução paralela não compensa o pouco tempo de execução, devido a natureza das operações ou a baixa carga de computação.

Muitos algoritmos têm uma característica iterativa, onde uma etapa depende da computação da etapa anterior e internamente a cada iteração há computações que podem ocorrer concorrentemente. A dependência entre os dados é um fator importante na paralelização de aplicações uma vez que, se não tratado adequadamente, pode gerar resultados incorretos devido ao acesso de posições de memória com valores ainda não atualizados.

Uma característica que deve ser evitada na programação paralela são sincronizações sucessivas em blocos paralelos. Um bloco paralelo realiza o lançamento de *n threads*. Entretanto sucessivas pausas para sincronizar os resultados parciais reduzem significativamente a eficiência gerada pelo paralelismo, uma vez que cada *thread* pode ter um tempo de computação distinto, o que demanda de esperas para sair da barreira (sincronização).

O código que faz uso da GPU deve obter ganho de desempenho superior ao da execução do bloco em CPU. A GPU possibilita a execução de blocos com grande volume de dados em um tempo significativamente inferior ao da CPU. Mas, todos os dados manipulados no bloco paralelo devem ser copiados para a memória da GPU e os dados retornados devem ser copiados para a memória da CPU. Essas cópias podem inviabilizar algumas paralelizações, pois a análise não deve considerar apenas a execução das instruções, mas também a sincronização das memórias.

Uma abordagem objetiva para identificar trechos de código paralelos é fazer uso de ferramentas de perfilamento de aplicações. A ferramenta gprof (Graham et al. 1982), por exemplo, faz coletas estatísticas do tempo de execução demandado por cada rotina que compõe o código e tem sido usado por muitos programadores para identificar inicialmente as funções mais custosas do código.

A performance de um código OpenMP pode ser avaliada pelo *speedup* (S). O *speedup* é definido como a razão entre o tempo de computação do algoritmo serial (T_{serial}) e o tempo de computação do algoritmo paralelo ($T_{paralelo}$), dado pela Equação 1. O *speedup* mostra o ganho efetivo do tempo de processamento do algoritmo paralelo sobre o algoritmo serial.

$$S = \frac{T_{serial}}{T_{paralelo}} \tag{1}$$

Quando o tempo paralelo é exatamente igual ao tempo sequencial, o *speedup* é igual a 1. Neste caso não há ganho de desempenho. Uma outra forma de mensurar o quanto uma versão paralela é melhor que a versão sequencial é considerar o percentual de

```
1 int fibo( int n ){
2  if( n < 2 ) return n;
3  int x = fibo(n-1);
4  int y = fibo(n-2);
5  return x + y;
6 }
```

```
1 for( i = 0; i < n; i++ )
2  compute_job(i);
```

ganho de desempenho apresentado na Equação 2.

$$S = \frac{T_{serial} - T_{paralelo}}{T_{paralelo}} * 100 \tag{2}$$

Algoritmos paralelos dependem da divisão da computação entre as unidades de processamento disponíveis. Desta forma, processos, threads ou fluxos concorrentes em GPU recebem preferencialmente uma quantidade equilibrada de carga de trabalho. Idealmente, também precisa-se coordenar as unidades de computação para sincronização e comunicação. Foster (Foster 1995) descreve um roteiro de quatro passos para desenvolver um programa paralelo: particionamento, comunicação, aglomeração e mapeamento.

Em relação à fase de particionamento, que indica onde deve haver paralelismo, as estratégias possíveis podem ser baseadas na concorrência sobre os dados ou na concorrência entre trechos de código independentes (Lima et al. 2021):

- O *Paralelismo de dados*, também conhecido como *decomposição de dados* ou *decomposição de domínio*, é um método recorrente para expressar concorrência em algoritmos. Nesse modelo de paralelismo, os dados associados ao domínio são particionados e então mapeados para tarefas concorrentes. Os dados dessa decomposição podem ser dados de entrada, saída, ou intermediários, ou seguem *Owner-Computes Rule* (OCR).

- O *Paralelismo de tarefas*, também chamado de *paralelismo funcional* ou *paralelismo de controle*, decompõe a computação ao invés dos dados manipulados. Esse modelo de paralelismo pode ser utilizado em tarefas que realizam computações distintas e independentes. Todavia, o paralelismo de tarefas é utilizado em algoritmos onde as tarefas podem ter dependências que resultam em um Grafo Acíclico Direcionado (*Directed Acyclic Graph - (DAG)* (Gautier et al. 2007). Algoritmos recursivos são um exemplo direto de paralelismo de tarefas em que a chamada recursiva é substituída por uma tarefa e por uma sincronização para esperar os resultados se necessário. Outro exemplo pode ser descrito por laços paralelos em que cada iteração é mapeada para uma tarefa sem dependências.

O exemplo a seguir ilustra os dois tipos de paralelismo, em que se pode substituir cada chamada de função pela criação de uma tarefa concorrente.

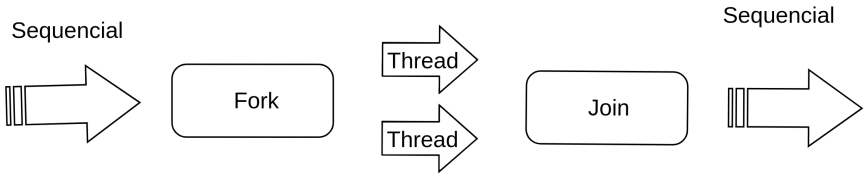


Figura 1.1. Instanciação de novas *threads* (*fork*) e término da execução (*join*).

1.3. Diretivas Paralelas de OpenMP

A API de OpenMP é fundamentada no modelo de execução *fork-join*. Esse modelo possui uma *thread* mestre que inicia a execução e gera *threads* de trabalho para executar as tarefas em paralelo (Chapman et al. 1999). OpenMP aplica o modelo em segmentos do código que são informados pelo programador. Dessa forma, um código sequencial é executado pela *thread* mestre até um bloco ou área de execução paralela, conforme apresentado na Figura 1.1.

O início da área paralela é demarcado por uma diretiva OpenMP que é responsável por sinalizar que as *threads* de trabalho devem ser lançadas (*fork*). Uma diretiva é precedida obrigatoriamente por `#pragma omp` (em C/C++) ou `!$omp` (em FORTRAN) e seguida por atributos opcionais. Todo o código seguinte é executado em paralelo pelas *threads* até o fim da área paralela que pode ser demarcado explicitamente como o símbolo de `}` (em C/C++) ou `!%omp end` (em FORTRAN), ou implícito, como por exemplo, em um laço de repetição `do/for`, onde o fim do laço também é o fim da área paralela. O fim da área paralela implica no encerramento das *threads* de trabalho, sincronização (*fork*) e retorno da *thread* mestre para a execução (*join*).

Seguem algumas diretivas que compõem a API OpenMP e que tradicionalmente são usadas pelos desenvolvedores (OpenMP 2023):

- `parallel`: Essa diretiva especifica que uma área do código será executada por *n threads*.

C / C++

```
1 #pragma omp parallel
2 {
3    // Bloco Paralelo
4 }
```

Fortran

```
1 !$omp parallel
2   ! Bloco Paralelo
3 !$omp end parallel
```

- `for`: Essa diretiva especifica como o trabalho será dividido entre as *threads*, mas não instância nenhuma *thread*. Ou seja, para que o paralelismo de laços ocorra, é necessário que esta chamada esteja precedida por uma chamada `parallel`, como no item anterior.

C / C++

```
1 #pragma omp for
2 for(i=1; i<imax; i++){
3     Bloco
4 }
```

Fortran

```
1 !$omp do
2   do i, imax
3     Bloco
4   enddo
5 !$omp end do
```

- `parallel for`: Especifica a construção de um laço paralelo, sendo que as iterações do laço de repetição serão distribuídas e executadas por *n threads*. Ou em outras palavras, ela é uma combinação dos dois itens anteriores, produzindo o mesmo resultado. A grande maioria dos programadores adota este estilo de chamada, sendo que o paralelismo de laços ainda é um dos modelos mais utilizados em OpenMP.

C / C++

```
1 #pragma omp parallel for
2 for(i=1; i<imax; i++){
3     Bloco Paralelo
4 }
```

Fortran

```
1 !$omp parallel do
2   do i, imax
3     Bloco Paralelo
4   enddo
5 !$omp end parallel do
```

- `simd`: Essa diretiva descreve que algumas iterações de um laço de repetição podem ser executadas simultaneamente por unidades vetoriais.

C / C++

```
1 #pragma omp simd
2 for(i=1; i<imax; i++){
3     Bloco
4 }
```

Fortran

```
1 !$omp simd
2   do i, imax
3     Bloco
4   enddo
5 !$omp end simd
```

- `for simd`: Essa diretiva especifica que um laço pode ser dividido em *n threads* que executam algumas iterações simultaneamente por unidades vetoriais.

C / C++

```
1 #pragma omp do simd
2 for(i=1; i<imax; i++){
3     Bloco
4 }
```

Fortran

```
1 !$omp do simd
2   do i, imax
3     Bloco
4   enddo
5 !$omp end do simd
```

- `target`: Mapeia variáveis para um ambiente de dados de um dispositivo e executa a construção neste dispositivo.

C / C++

```
1 #pragma omp target
2 for(i=1; i<imax; i++){
3     Bloco
4 }
```

Fortran

```
1 !$omp target
2   do i, imax
3     Bloco
4   enddo
5 !$omp end target
```

- `task`: Essa diretiva cria unidades de trabalho independentes. O uso da diretiva pode definir explicitamente uma tarefa. A diretiva identifica um bloco de código a ser executado em paralelo com o código fora da região de tarefa. As diretivas de tarefa podem ser para paralelizar algoritmos irregulares, onde as unidades de trabalho são geradas dinamicamente, como em estruturas recursivas ou laços do tipo *enquanto* (`while`). A programação com tarefas facilita a paralelização das aplicações.

C / C++

```
1 #pragma omp task
2   function(a)
3 }
4 #pragma omp task
5   function(b)
6 }
```

Fortran

```
1 !$omp task
2   function(a)
3 !$omp end task
4 !$omp task
5   function(b)
6 !$omp end task
```

Em OpenMP também há construções de compartilhamento de trabalho, como:

- `section`: A construção de seções é uma construção de compartilhamento de trabalho não iterativo que contém um conjunto de blocos estruturados que devem ser distribuídos e executados pelas *threads* de um grupo. Cada bloco estruturado é executado uma vez por uma das *threads* do grupo, no contexto de sua tarefa implícita.

- `single`: a construção especifica que o bloco estruturado associado é executado por apenas uma das *threads* do grupo (não necessariamente a *thread* principal), no contexto de sua tarefa implícita. As outras *threads* do grupo, que não executam o bloco, esperam em uma barreira implícita no final de uma única região, a menos que uma cláusula `nowait` seja especificada.

- `workshare`: A construção de compartilhamento de trabalho divide a execução do bloco estruturado fechado em unidades de trabalho separadas e faz com que as *threads* do grupo compartilhem o trabalho de modo que cada unidade seja executada apenas uma vez por uma *thread*, no contexto de sua tarefa implícita.

Na sequência são apresentados alguns atributos da API OpenMP, os quais podem ser associados às diretivas previamente apresentadas (OpenMP 2023). Para todos os casos, `lista` representa uma ou mais variáveis.

- `private (lista)`: Esse atributo informa que o bloco paralelo possui variáveis privadas para cada uma das *n threads*. As variáveis do bloco que não são informadas na lista são públicas.

- `shared (lista)`: O atributo especifica que as variáveis são públicas e compartilhadas entre as *n threads*.

- `num_threads(int n)`: Este atributo determina o número *n* de *threads* utilizadas no bloco paralelo. O valor de *n* é válido apenas para o bloco em que foi definido.

- `reduction (operador: lista)`: A redução é utilizada para executar cálculos em paralelos. Cada *thread* tem seu valor parcial. Ao final da região paralela o valor final da variável é atualizado com os cálculos parciais. O operador pode ser, por exemplo $+$, $-$, $*$, *max* e *min*.

- `nowait`: Uma diretiva OpenMP possui uma barreira implícita ao seu fim, com o objetivo de garantir a sincronização. Entretanto a diretiva `nowait` omite a existência dessa barreira. Dessa forma, as *threads* não ficam em espera até que as demais também terminem o trabalho.

- `collapse`: a cláusula associa um ou mais laços à diretiva, com o objetivo de identificar a profundidade *n* dos laços aninados, que podem aplicar a semântica da diretiva.

As variáveis de ambiente do OpenMP especificam características que afetam a execução dos programas. Seguem algumas variáveis (OpenMP 2023):

```
1  #pragma omp parallel
2  {
3  #pragma omp single
4   {
5    node* p = head;
6    while(p) {
7  #pragma omp task firstprivate(p)
8     process(p);
9     p = p->next;
10   }
11 #pragma omp taskwait
12  }
13 }
```

Figura 1.2. Exemplo de tarefas OpenMP para visitar elementos de uma lista encadeada.

- `OMP_NUM_THREADS`: Especifica o número *n* de *threads* utilizados nos blocos paralelos do algoritmo.
- `OMP_SCHEDULE`: A variável de ambiente controla o tipo de programação e o tamanho do bloco de todas as diretivas de *loop* do tipo `runtime` com as opções `static`, `dynamic`, `guided` ou `auto`.
- `OMP_THREAD_LIMIT`: Descreve o número máximo de *threads*.
- `OMP_NESTED`: Permite ativar ou desativar o paralelismo aninhado.
- `OMP_STACKSIZE`: Especifica o tamanho da pilha para as *threads*.

1.3.1. Paralelismo de Tarefas

A partir de sua versão 3.0, o OpenMP suporta o paralelismo de tarefas através da construção `task` para tarefas explícitas e `taskwait` para sincronização. A Figura 1.2 ilustra uma função para percorrer listas com criação de tarefas OpenMP. Note que em relação aos outros programas OpenMP, as tarefas são criadas dentro de uma construção `single` na linha 3. Isso se deve ao fato da região paralela executar o mesmo código em todas as threads, o que não é desejado nesse caso. Aqui a execução inicia com uma única thread apenas para que novas tarefas sejam criadas em seguida. Na linha 7 uma nova tarefa é criada para a função `process`. Note que a cláusula `firstprivate` é usada pois a variável `p` é modificada na próxima linha. Caso contrário, haveria uma condição de corrida entre a thread que cria tarefas e a nova tarefa.

O paralelismo de tarefas desenrola sua execução em um DAG onde as dependências são descritas pela estrutura recursiva do programa. Esse modo de execução, denominado *fully strict mode*, define que as relações de dependências ocorrem somente entre nós raízes e folhas com ligação direta.

```
1 #pragma omp taskgroup
2 {
3 #pragma omp task depend(in:data) depend(out:result)
4  foo(data, result);
5 }
```

Figura 1.3. Exemplo simples de tarefas OpenMP com dependências de dados.

Por outro lado, o paralelismo com dependências de dados controla a execução por meio de um grafo de fluxo de dados ou *Data Flow Graph* (DFG) (Gautier et al. 2007). O controle de execução é feito exclusivamente pelo fluxo de dados da aplicação e depende do modo de acesso descrito pela tarefa.

Os modos de acesso que podem ser listados, de uma forma genérica, são:

- **Read only (RO ou R)** - somente leitura, sem permissão para modificar.
- **Write only (WO ou W)** - somente escrita, sem leitura de dados de entrada.
- **Read and write (RW)** - ou modo exclusivo, com leitura e escrita.

O OpenMP versão 4.0 incluiu o uso de diretivas para expressar dependências de dados em tarefas. A diretiva `depend` de uma construção `task` lista as dependências de dados que podem ser:

- **in** – somente leitura.
- **out** – somente escrita.
- **inout** – leitura e escrita.

Além disso, a API inclui a construção de sincronização *taskgroup* que permite a sincronização implícita ao final do bloco de código a fim de esperar por todas as tarefas criadas recursivamente, o que não era possível com a diretiva **taskwait**. A Figura 1.3 demonstra um exemplo simples da criação de tarefas OpenMP com dependências de dados de entrada (`in`) e saída (`out`), além da sincronização recursiva para este bloco de código (`taskgroup`).

1.3.2. Aceleradores

O padrão OpenMP 4 ou superior inclui diretivas de execução de trechos de código em aceleradores por meio do modelo de execução *host-centric* onde a CPU principal, ou *host*, é o lugar onde a execução do programa inicia e o *device* é o acelerador para execução de trechos de código. O acelerador pode executar iterações de laços paralelos por meio de grupos de threads, chamados *teams*, que cooperam a fim de executar o trabalho.

```
1 int n = 1024;
2 float a = 32.0f, b = 17.0f;
3 float x[1024], y[1024];
4
5 #pragma omp target teams map (to:x[0:n]) map(tofrom:y[0:n])
6 #pragma omp distribute parallel for
7 for(int i= 0; i < n; i++) {
8   y[i] = a*x[i] + b*y[i];
9 }
```

Figura 1.4. Exemplo simples de uso de OpenMP para aceleradores.

A construção `target` muda o controle de execução do *host* para o acelerador e a construção `teams` cria um grupo de threads semelhante à construção `parallel`. Apenas algumas construções podem estar aninhadas a um `teams` como `distribute` e `parallel`. A construção `distribute` distribui as iterações de um laço entre as threads do grupo no acelerador. Outros atributos de `distribute` podem determinar o escalonamento e o grão de trabalho a cada thread (`dist_schedule`).

A diretiva `map` descreve o mapeamento explícito de variáveis ao ambiente de dados do acelerador. O tipo de mapeamento de dados com a diretiva `map` pode ser:

- `alloc` - aloca memória para a variável correspondente;
- `to` - aloca memória e copia o valor original para esta variável na entrada;
- `from` - aloca memória e copia o valor dela para a variável original na saída;
- `tofrom` - é o padrão, onde copia o valor na entrada e saída da região.

A construção `target` pode ser acompanhada da diretiva `nowait`, indicando que a CPU não espera o término do código na região `target`. A diretiva `depend` também pode ser utilizada a fim de sincronizar trechos de código assíncronos com `nowait`.

A Figura 1.4 demonstra um exemplo simples de programa SAXY de um laço executado em um acelerador com a construção `target`. Primeiramente a construção `target` (linha 5), seguida da construção `teams`, define a região a ser acelerada com um grupo de threads juntamente com o mapeamento dos vetores `x` e `y`. O vetor `x` é um dado de entrada e o vetor `y` é um dado de entrada e saída. Cada vetor tem o atributo `[0:n]` que define o tamanho do dado mapeado, sendo o vetor inteiro, em nosso exemplo. Em seguida, a construção `distribute parallel for` permite que o compilador execute um laço paralelo dentro da região acelerada no grupo de threads definido anteriormente.

1.4. Estudo de caso: Simulação de Secagem de Grãos

Uma aplicação de simulação de secagem de grãos foi escolhida como estudo de caso (de Oliveira 2020). Tal aplicação representa os grãos e o espaço entre os grãos como um sistema de escoa-

Algoritmo 1: Etapa iterativa do algoritmo

```
max_iterations ← 20.000 // número máximo de iterações
t0 ← 0 // tempo inicial
t ← 0.04 // tempo final
dt ← 0.01
while t0 < t do
  t0 ← to + dt
  i ← 1
  // calculando a convergência
  while i ≠ max_iterations do
    solve_U()
    solve_V()
    solve_P()
    solve_Z()
    convergence()
  end while
end while
```

mento em meios porosos, um típico problema da área de Dinâmica dos Fluidos (Lucca 2022). Assim, tem-se uma aplicação científica à disposição para a inserção de diferentes diretivas paralelas de OpenMP[1].

1.4.1. Algoritmo da Aplicação

A aplicação simula a secagem dos grãos através da passagem de ar quente, o que faz com que a transferência de temperatura para os grãos tenham como efeito a remoção da umidade (da Silva et al. 2022a). A simulação é baseada na discretização bidimensional usando volumes finitos para o problema, o qual é descrito matematicamente através das equações de Navier-Stokes. A transição do tempo também ocorre de forma discreta.

A aplicação modela as etapas de leitura dos valores de entrada, alocação e inicialização de dados, iteração do passo de tempo discreto, escrita dos resultados de saída em arquivos e desalocação de memória. A etapa que mais demanda de tempo de processamento é a iteração do passo de tempo discreto (da Silva et al. 2022c).

O Algoritmo 1 apresenta as rotinas invocadas na etapa iterativa da simulação. O laço principal itera o passo de tempo discreto. Em cada passo, são computadas rotinas responsáveis pela interação das propriedades físicas calculadas. Isto é feito iterativamente enquanto um critério de parada (`convergence()`) previamente definido ou um limite máximo de iterações não for atingido. Para cada tempo discreto há um número máximo de iterações até que se chegue a convergência dos valores de resíduo das propriedades de continuidade e momentos (U, V e P) do código. Nesta etapa iterativa são calculados as equações de Momento advindas de uma técnica conhecida como *Quick Scheme* (`solve_U` e `solve_V`), além de uma equação de Continuidade (`solve_P`) e de uma equação de Energia (`solve_Z`).

[1]https://github.com/GabrielDT02/porous-media-application

1.4.2. Rotinas e Subrotinas da Etapa Iterativa

As rotinas `solve_U`, `solve_V`, `solve_P` e `solve_Z` invocam sub-rotinas que iteram sobre todo o domínio bidimensional, para cada uma das propriedades físicas computadas. As sub-rotinas também realizam operações distintas para os elementos da borda do domínio. Estes exigem interações distintas entre os pontos. Também é necessário a invocação de sub-rotinas específicas para a aplicação das condições de contorno do problema.

Todas as rotinas e subrotinas responsáveis pela computação das equações de quantidade de movimento na dimensão horizontal e vertical, de continuidade e de energia estão implementadas no arquivo `equations.f90`. As rotinas `solve_U` e `solve_V` representam o tempo de, respectivamente, 43% e 41% do tempo total de execução do código sequencial, restando 7% para `solve_Z` e 1% para `solve_P` (Lucca et al. 2023).

1.5. Paralelização da Aplicação

As próximas subseções apresentam as formas como os trechos de código foram paralelizados.

1.5.1. `parallel do`

O laço externo da aplicação possui dependência temporal entre uma iteração e outra e, portanto, cada passo de tempo discreto não pode ser executado concorrentemente. Ou seja, uma etapa depende da etapa anterior, o que é uma característica de aplicações semelhantes. Situação semelhante acontece com o laço em que as operações continuam se repetindo enquanto a convergência não é atingida. Neste caso, é preciso calcular a convergência primeiro para saber se uma nova etapa iterativa precisa ser computada.

Já as quatro rotinas (`solve_U`, `solve_V`, `solve_P` e `solve_Z`) chamadas na etapa iterativa possuem trechos de código que podem ser executados concorrentemente. Como cada elemento a ser computado é o mesmo, o paralelismo de laços tende a ser uma abordagem eficiente para garantir um bom desempenho paralelo. Essencialmente há um padrão em cada rotina: há 4 laços aninhados que percorrem a representação bidimensional do domínio do problema. Assim, cada chamada (`do`) pode ser paralelizada com `!$omp parallel do`, com as devidas indicações de variáveis privadas especificadas com o atributo `private`. Na implementação do `!$omp parallel do`, foi necessário apenas identificar os laços internos da aplicação e incluir a diretiva apropriada. Foram encontrados 20 laços (ou laços aninhados) de potencial para a aplicação de `parallel do`. Alguns desses laços são chamados mais vezes (sub-rotinas) no mesmo passo discreto. Portanto, o número total de laços paralelos invocados é de 36 para cada etapa iterativa do laço de convergência, sendo 12 para `solve_U`, 12 para `solve_V`, 6 para `solve_P` e 6 para `solve_Z`. O algoritmo 2 mostra um exemplo de paralelismo de laço aplicado a um trecho de código da rotina `solve_U`. Aplicações semelhantes são feitas para os demais laços identificados.

1.5.2. `parallel task`

Uma outra abordagem de paralelização possível é fazer uso da diretiva `!$omp task`, uma vez que existem trechos de computação que podem ser executados concorrentemente. Isto é, existem rotinas que a cada iteração do tempo discreto do código são executados

Algoritmo 2: Implementação paralela usando a diretiva `parallel do` em um laço da função `solve_U`

```
1
1 !$omp parallel do private(i,j)
2 DO j=2,jmax-1
3   DO i=3,imax-1
4     res_u(i,j) =((um(i,j)-um_tau(i,j)) +RU(i,j)*dt)*dtau
5     ui(i,j) = ( um_tau(i,j) + res_u(i,j))
6   ENDDO
7 ENDDO
8 !$omp end parallel do
```

sequencialmente, mas que poderiam ser executados concorrentemente, pois operam sobre conjuntos de dados distintos. Como exemplo tem-se as operações que são feitas em `solve_U` e `solve_V`, pois são de dimensões distintas (em x e em y).

A Figura 1.5 apresenta algumas modificações necessárias para que os trechos de código de ambas as rotinas possam ser executados concorrentemente. Isso acontece porque as condições de contorno operam sobre as estruturas de dados utilizadas tanto em `solve_U`, como em `solve_V`, ou seja, tal operação precisa ser feita por um único fluxo de execução.

O Algoritmo 3 apresenta a invocação de tarefas concorrentes para executar as rotinas `solve_U` e `solve_V`.

1.5.3. `parallel sections`

A diretiva de seções paralelas do OpenMP identifica seções de código para dividir entre todas as threads. De acordo com a especificação do padrão OpenMP, a construção de uma seção é uma construção de compartilhamento de trabalho não iterativa. Ele contém um conjunto de blocos estruturados a serem distribuídos e executados pelas threads em um grupo. A execução de um bloco estruturado é feita uma vez por uma das threads do grupo, considerando o contexto de sua tarefa implícita (OpenMP 2023). Isso permite a definição de concorrência em alto nível.

Na aplicação, pode-se executar simultaneamente `solve_U` e `solve_V`, duas das principais rotinas chamadas em um laço iterativo. Para utilizar de forma eficiente as seções paralelas, foi necessário reestruturar o código para manter a sequência de execução entre os métodos de resolução para execução paralela em eixos diferentes (x e y) da diretiva `teams`. Uma região definida permite que o código seja específico em paralelo. Cada seção cria uma thread que executa uma tarefa independente.

Nesta nova estrutura, foi possível inserir a diretiva `teams` para executar o *solve_U*$_i$ e `solve_V`$_i$ em paralelo, para i=1, 2 e 3, como apresentado na Figura 1.5. As diretivas `task` e *section* são de muitas maneiras semelhantes. As seções incluídas estão dentro da construção `sections` e as threads não sairão dela até a execução de todas as *sections*.

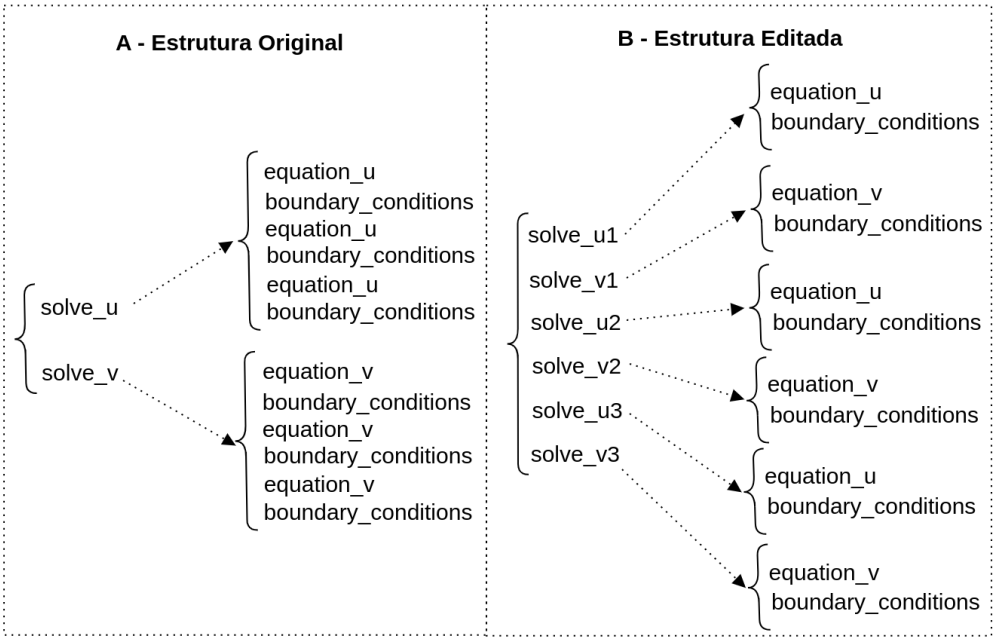


Figura 1.5. Modificações das funções `solve_U` e `solve_V` para a obtenção do paralelismo de tarefas

Algoritmo 3: Uma região de código usando a diretiva `!$omp task`

```
1
1  DO WHILE (time .LT. final_time)
2    time = time + dt
3    DO WHILE(itc.LT.itc_max)
4      !$omp parallel
5
6      !$omp task
7      CALL solve_U1()
8      !$omp end task
9
10     !$omp task
11     CALL solve_V1()
12     !$omp end task
13
14     !$omp taskwait
15
16     !$omp end parallel
17     ...
18     CALL solve_P()
19     CALL solve_Z()
20     CALL convergence()
21   ENDDO
22 ENDDO
```

Algoritmo 4: Trecho de código usando a diretiva `!$omp sections`

```
1
1  DO WHILE (time .LT. final_time)
2   time = time + dt
3   DO WHILE(itc.LT.itc_max)
4    !$omp parallel sections
5    ...
6    !$omp section
7    CALL solve_U1()
8    !$omp section
9    CALL solve_V1()
10   ...
11   !$omp end parallel sections
12   ...
13   CALL solve_P()
14   CALL solve_Z()
15   CALL convergence()
16  ENDDO
17 ENDDO
```

1.5.4. `target teams distribute parallel do`

A diretiva `target` permite a seleção de um tipo de arquitetura (Multi-core ou GPU), onde a execução do segmento de código será sequencial. A combinação com outras diretivas, como a diretiva `teams` apresentada anteriormente, gera paralelismo. A diretiva nos permite explorar o recurso *Streaming Multiprocessor* disponível nas GPUs ou nos núcleos do Multi-core. Também é possível usar a diretiva `distribute` para explorar o paralelismo. Esta diretiva distribui as iterações de um laço do tipo *do-loop* entre as threads mestres. Também é possível combinar com a diretiva *parallel do*.

O Algoritmo 5 mostra a definição de uma região do tipo `target`, onde usou-se a diretiva `teams`, e um trecho de código chama uma operação `distribute parallel do`.

1.6. Resultados Experimentais

Foi definido um caso de teste para uma malha de tamanho de 100×124 pontos para avaliar o desempenho das implementações paralelas. Foram realizadas 30 execuções para obter o tempo médio para cada um dos casos de teste. Consideramos um tempo de simulação de $0,04$. O tempo começa em $0,0$. O intervalo de tempo discreto é $0,01$ (Δt). Em todos os casos, o número máximo de iterações usadas para convergência em cada tempo discreto foi de 20.000.

O código foi compilado com o compilador pgf90 (nvfortran), usando o NVidia HPC_SDK 21.2 *toolkit*, com a adição das flags `-O3`, `-fopenmp` e `Minfo=all`. A composição do ambiente computacional utilizado neste trabalho para execução dos testes é de dois processadores Intel Xeon CPU E5-2650 octa-core e uma GPU Nvidia Quadro

Algoritmo 5: Trecho de código usando a diretiva `!$omp target`

```
1
1  !$OMP TARGET
2  ...
3  !$OMP TEAMS
4  ...
5  !$omp distribute parallel do collapse(2)
6  do i=2,imax-1
7   do j=2,jmax-1
8    pi(i,j) = p(i,j) + 8.d-2 * RP(i,j)*c2
9   enddo
10 enddo
11 !$omp distribute parallel do
12 do i=1,imax
13  pi(i,1 ) = pi(i,2)
14  pi(i,jmax) = pi(i,jmax-1) + 1.d0*(pi(i,jmax-1)-pi(i,jmax-2))
15 enddo
16 !$omp distribute parallel do
17 do j=1,jmax
18  pi(1,j) = pi(2,j)
19  pi(imax,j) = pi(imax-1,j)
20 enddo
21 ...
22 !$OMP END TEAMS
23 ...
24 !$OMP END TARGET
```

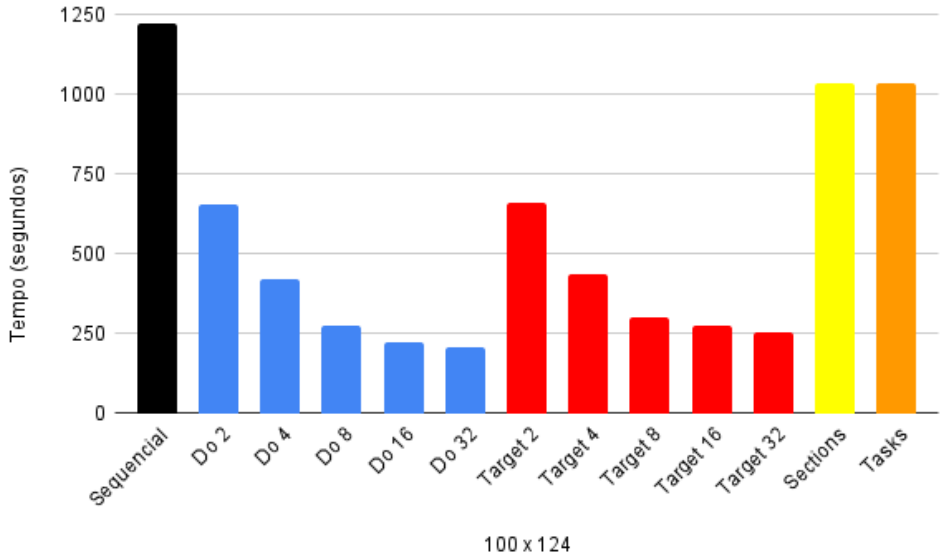


Figura 1.6. Tempo de execução para uma malha de tamanho 100×124**.**

M5000.

A Figura 1.6 apresenta o tempo médio de execução em segundos para uma malha de configuração de tamanho 100×124 pontos. A figura mostra as versões: sequencial, OpenMP `parallel do`, `sections`, `tasks` e `target`. O número de threads utilizado para avaliar as implementações de OpenMP foram 2, 4, 8, 16 e 32. Também foi calculado o desvio padrão para esses tempos médios. O desvio padrão permaneceu baixo e seu valor é inferior a 1% do tempo total de execução. Também foi avaliado e garantido a compatibilidade numérica das versões paralelas em relação à versão sequencial. Ou seja, todos os códigos precisam produzir valores idênticos para os resultados produzidos.

A implementação paralela de OpenMP fornece redução do tempo de execução. Em todos os experimentos, adicionar mais threads resulta em maior redução no tempo de execução. Uma exceção ocorre quando usamos mais threads do que núcleos físicos (teste com 32 threads). Neste caso, o ganho foi menor que no teste com 16 threads. O *speedup* com a diretiva `parallel do` foi de $5,53$, considerando o uso de 16 threads. A execução em cada etapa iterativa consiste em 36 loops paralelos. Só não foram paralelizados os laços das operações de condições de contorno devido à simplicidade dos cálculos, ou seja, apenas operações de atribuição. Esses resultados são compatíveis com os valores encontrados em outros trabalhos relacionados. Os resultados de *speedup* poderiam ser mais expressivos caso uma malha de tamanho maior fosse utilizada.

As diretivas `tasks` e `sections` fornecem resultados semelhantes, até mesmo pelo fato de realizarem chamadas semelhantes. No entanto, esses resultados são apenas um pouco melhores do que a implementação sequencial. A aceleração foi de $1,18$. A aplicação é executada simultaneamente apenas por solve_u e solve_v por um grupo de threads. Foram utilizadas 32 threads como padrão nos experimentos, mas de fato o número

de seções ou tarefas simultâneas é limitado às operações nas dimensões x e y.

Os resultados do OpenMP `target` apresentam resultados próximos ao `parallel do`. O *speedup* foi de $4,50$ usando 16 threads. Na implementação, a versão `target` chama `teams` e `distribute parallel do` para executar os laços simultaneamente.

1.7. Conclusão

Paralelizar uma aplicação possui desafios. Muitas vezes é necessário reescrever ou realizar adaptações no código sequencial, para que o mesmo possa ser executado concorrentemente, ou seja, sem dependência de dados ou de operações. Posteriormente, deve-se partir para a paralelização do código. Algumas técnicas de paralelismo podem ser escolhidas por serem mais adequadas para uma determinada classe de problemas ou devido às características que o domínio do problema possui. Todos os casos de teste apresentados obtiveram *speedup* positivo, ou seja, obtiveram ganho de desempenho. Além do ganho de desempenho também é preciso garantir a equivalência numérica dos resultados, ou seja, uma versão paralela não pode resultar em valores inconsistentes da solução do programa sequencial.

Neste capítulo foi descrita uma aplicação e apresentadas formas de paralelização que puderam ser aplicadas usando a interface de programação OpenMP. As especificações mais recentes de OpenMP permitem tanto a criação de tarefas paralelas quanto o uso de GPUs através de diretivas `target`. Desta forma, OpenMP aparece com uma alternativa para que que uma aplicação possa ser paralelizada tanto em um ambiente multi-core, quanto many-core, deixando de ser utilizado somente o tradicional paralelismo de laços através da combinação das diretivas `parallel` e `for`.

Referências

Chapman et al. 1999 Chapman, B., Mehrotra, P., and Zima, H. (1999). Enhancing OpenMP with features for locality control. In *Proceedings of Eighth ECMWF Workshop on the Use of Parallel Processors in Meteorology*, pages 301–13, Singapore. Towards Teracomputing. World Scientific Publishing.

da Silva et al. 2022a da Silva, H. U., Lucca, N., Schepke, C., de Oliveira, D. P., and da Cruz Cristaldo, C. F. (2022a). Parallel OpenMP and OpenACC Porous Media Simulation. *The Journal of Supercomputing*.

da Silva et al. 2022b da Silva, H. U., Schepke, C., da Cruz Cristaldo, C. F., de Oliveira, D. P., and Lucca, N. (2022b). An Efficient Parallel Model for Coupled Open-Porous Medium Problem Applied to Grain Drying Processing. In Gitler, I., Barrios Hernández, C. J., and Meneses, E., editors, *High Performance Computing*, pages 250–264, Cham. Springer International Publishing.

da Silva et al. 2022c da Silva, H. U., Schepke, C., Lucca, N., da Cruz Cristaldo, C. F., and de Oliveira, D. P. (2022c). Parallel OpenMP and OpenACC Mixing Layer Simulation. In *2022 30th Euromicro International Conference on Parallel, Distributed and Network-based Processing (PDP)*, volume 1, pages 181–188.

Dagum and Menon 1998 Dagum, L. and Menon, R. (1998). OpenMP: An Industry Standard API for Shared-Memory Programming. *IEEE Computational Science and Engineering*, 5(1):46–55.

de Oliveira 2020 de Oliveira, D. P. (2020). Fluid Flow Through Porous Media with the One Domain Approach: A Simple Model for Grains Drying. Dissertação de mestrado, Universidade Federal do Pampa, Alegrete.

Foster 1995 Foster, I. (1995). *Designing and Building Parallel Programs: Concepts and tools for Parallel Software Engineering*. Addison Wesley, Reading, MA.

Gautier et al. 2007 Gautier, T., Besseron, X., and Pigeon, L. (2007). KAAPI: A thread scheduling runtime system for data flow computations on cluster of multi-processors. In *2007 international workshop on Parallel symbolic computation*, pages 15–23, Waterloo, Canada. ACM.

Graham et al. 1982 Graham, S. L., Kessler, P. B., and Mckusick, M. K. (1982). Gprof: A Call Graph Execution Profiler. *SIGPLAN Not.*, 17(6):120–126.

Lima et al. 2021 Lima, J. V. F., Schepke, C., and Lucca, N. (2021). Além de Simplesmente: #pragma omp parallel for. In Charão, A. and da Silva Serpa, M., editors, *Minicursos da XXI Escola Regional de Alto Desempenho da Região Sul*, chapter 4, pages 86–103. Sociedade Brasileira de Computação, Porto Alegre/RS.

Lucca 2022 Lucca, N. (2022). *Avaliação de estratégias de paralelismo em simulação de meios porosos*. PhD thesis, Dissertação (Mestrado Profissional em Engenharia de Software) – Universidade Federal do Pampa, Campus Alegrete, Alegrete/RS.

Lucca et al. 2023 Lucca, N., Schepke, C., and Tremarin, G. D. (2023). Parallel Directives Evaluation in Porous Media Application: A Case Study. In *2023 31th Euromicro International Conference on Parallel, Distributed and Network-based Processing (PDP)*, volume 1.

OpenMP 2023 OpenMP (2023). The OpenMP API Specification for Parallel Programming.

Capítulo 2

Projeto de Aplicações Paralelas

Guilherme Galante
Universidade Estadual do Oeste do Paraná

Resumo

O objetivo deste minicurso é fornecer uma visão geral do processo de projeto de aplicações paralelas. São apresentadas duas abordagens: PCAM e Padrões de Projeto. Como se trata de um minicurso introdutório, espera-se que, ao final, o leitor tenha um entendimento inicial que sirva de base para continuar com seus estudos na área.

2.1. Introdução

Hoje em dia, todos os computadores são essencialmente paralelos. Ensinar e aprender programação paralela tornou-se cada vez mais importante devido à onipresença de processadores com algum grau de paralelismo em dispositivos portáteis, estações de trabalho e clusters de computação. Para utilizar totalmente os recursos de computação das arquiteturas de hardware atuais, é necessário que futuros cientistas e engenheiros da computação escrevam código altamente paralelizado. Além disso, o crescimento de áreas como aprendizagem de máquina e big data exige a adoção de processamento de alto desempenho como parte integrante de seu domínio de conhecimento. Adquirir habilidades de programação paralela é hoje em dia uma parte indispensável de muitos currículos de graduação e pós-graduação (Schmidt et al. 2018).

No entanto, o projeto de aplicações paralelas ainda é considerado uma ordem de magnitude mais difícil do que o projeto de algoritmos sequenciais e o desenvolvimento de programas sequenciais (Trobec et al. 2018). Projetar corretamente aplicações paralelas envolve conhecer bem o problema e como este pode ser trabalhado para se adaptar corretamente à arquitetura será adotada, seja um processador multicore, cluster ou GPU. Enfim, as escolhas na fase de projeto são fundamentais para se obter um bom desempenho e um uso eficiente dos recursos de processamento.

Nesse contexto, o objetivo deste minicurso é fornecer uma visão geral do processo de projeto de aplicações paralelas. São apresentadas duas metodologias: PCAM, proposta por (Foster 1995), e Padrões de Projeto, propostos por (Mattson et al. 2004). Como se

trata de um minicurso introdutório, espera-se que, ao final, o leitor tenha um entendimento inicial que sirva de base para continuar com seus estudos na área.

Este texto está organizado da seguinte forma. Seção 2.2 apresenta as etapas do desenvolvimento de aplicações paralelas. Seção 2.3 aborda a questão da detecção dos principais pontos de paralelismo (*hotspots*) em um programa paralelo. A Seção 2.4 apresenta a metodologia PCAM. A Seção 2.5 introduz alguns padrões de projeto para o desenvolvimento de aplicações paralelas. Por fim, a Seção 2.6 conclui o minicurso.

2.2. Do problema à solução paralela: etapas de desenvolvimento

De acordo com (Czarnul 2018), o desenvolvimento de código paralelo envolve quatro etapas:

- Formulação de um problema com a definição dos dados de entrada, operações necessárias e formato dos resultados de saída.
- Projeto da aplicação. Normalmente, uma aplicação sequencial pode ser paralelizada, ou seja, adaptada para executar mais rápido, dividindo cálculos/dados para executar em vários núcleos com comunicação/sincronização necessária, de modo que a saída correta seja produzida. Outra abordagem é produzir um algoritmo paralelo do zero, o que não é comum (ou trivial).
- Implementação do algoritmo. Nesse caso, uma ou mais interfaces de programação (API) podem ser usadas para codificar o algoritmo. Normalmente, uma API é usada de modo que sua implementação possa ser executada com eficiência em uma classe selecionada de hardware. Por exemplo, OpenMP para processadores multicore, MPI para clusters e CUDA para GPUs.
- Otimização de código. Esta etapa inclui a aplicação de técnicas de otimização no código que podem estar relacionadas a hardware específico. Tais técnicas podem incluir reorganização de dados, escalonamento, sobreposição de comunicações e cálculos, balanceamento de carga entre nós e núcleos de computação.

A paralelização de um código sequencial pode ser fácil, como é o caso, por exemplo, nos chamados problemas *trivialmente paralelizáveis*, caracterizados pelo esforço mínimo necessário para dividir a aplicação em uma coleção de tarefas que podem ser distribuídas entre um conjunto de componentes de computação para processamento em paralelo (Wilkinson and Allen 2005). Por outro lado, o processo é mais difícil quando o fluxo de controle é complexo, o algoritmo contém partes difíceis de paralelizar ou quando a proporção de cálculos para comunicação/sincronização é relativamente pequena. Por exemplo, em um cálculo direto dos números de Fibonacci:

```
f[0] = f[1] = 1;
for (i = 2; i <= n; i++)
     f[i] = f[i-1] + f[i-2];
```

essencialmente não há oportunidade para execução simultânea de instruções, considerando que a computação em uma iteração depende dos resultados de uma ou mais iterações anteriores (dependência de laço) (Pacheco 2011).

2.3. Identificando Hotspots

Um *hotspot* é uma seção de código que leva a maior parte do tempo de execução de um programa. Portanto, identificar os hotspots pode dar pistas de onde o esforço efetivo de paralelização deve ser concentrado. Seções do programa que representam pouco uso da CPU devem ser ignorados inicialmente. A maioria das aplicações científicos e técnicas, por exmeplo, geralmente realiza a maior parte de seu trabalho em alguns trechos bastante restritos. Geralmente laços de repetição e funções com uma alta porcentagem de contagem de instruções são identificados como hotspots.

A detecção de *hotspots* pode ser realizada usando várias ferramentas de análise de desempenho, que podem identificar as seções do código que estão consumindo a maior parte dos recursos computacionais ou do tempo de processamento.

2.3.1. Exemplo: Intel VTune Profiler

O Intel VTune Profiler[1] é uma ferramenta de análise de desempenho amplamente utilizada para detecção de hotspots em aplicações. O VTune é capaz de fornecer informações detalhadas sobre o desempenho da aplicação, permitindo que os desenvolvedores identifiquem áreas críticas do código que estão consumindo a maior parte dos recursos computacionais.

O VTune Profiler também fornece visualizações gráficas das informações coletadas, permitindo que os desenvolvedores compreendam facilmente os resultados e identifiquem os hotspots de maneira eficiente. As visualizações incluem gráficos de tempo de CPU, uso de memória e latência, bem como fluxogramas de chamada de função e tabelas de eventos.

Na Figura 2.1 apresenta-se um exemplo de análise de uma aplicação de multiplicação de matrizes. Observa-se que o tempo total de CPU para a aplicação é igual a 150.993 segundos. A seção *Top Hotspots* fornece dados sobre as funções mais demoradas (funções de *hotspot*) classificadas pelo tempo de CPU gasto em sua execução. Para a aplicação teste, a função *multiply0*, que levou 150.670 segundos para ser executada, aparece no topo da lista como a função mais ”quente”, e portanto, um possível ponto de partida para a paralelização. Inclusive, é possível visualizar no código-fonte quais as linha de código que demandam mais tempo de execução (Figura 2.2).

[1] ⟨https://www.intel.com/content/www/us/en/developer/tools/oneapi/vtune-profiler.html⟩

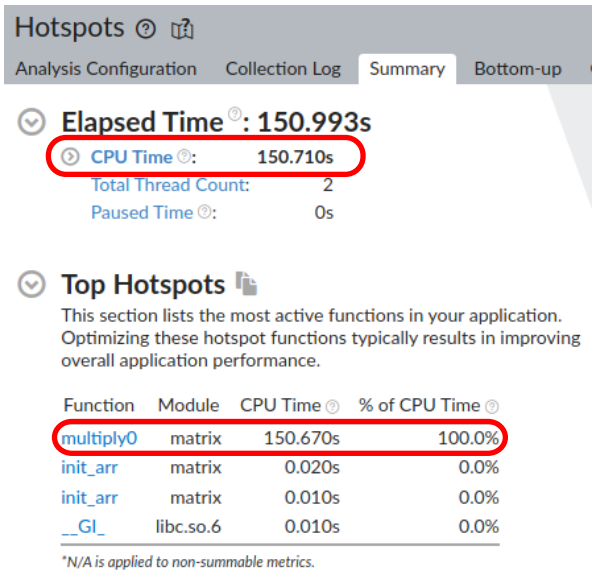


Figura 2.1. Detecção de hotspots usando Intel VTune.

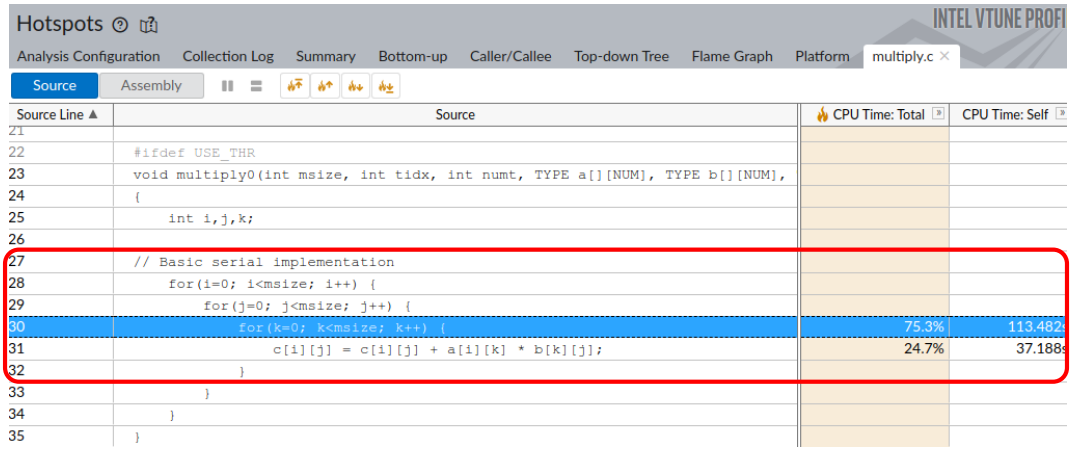


Figura 2.2. Trecho de código referente ao hotspot.

Mais detalhes sobre o uso da ferramenta pode ser obtido em sua documentação oficial[2].

[2] ⟨https://www.intel.com/content/www/us/en/developer/tools/oneapi/vtune-profiler-documentation.html⟩

2.4. Metodologia PCAM

Nessa seção, apresenta-se o método PCAM, apresentada por (Foster 1995). O nome deriva-se das iniciais das quatro fases que a compõem: (1) Particionamento, (2) Comunicação, (3) Agregação e (4) Mapeamento, conforme ilustrado na Figura 2.3.

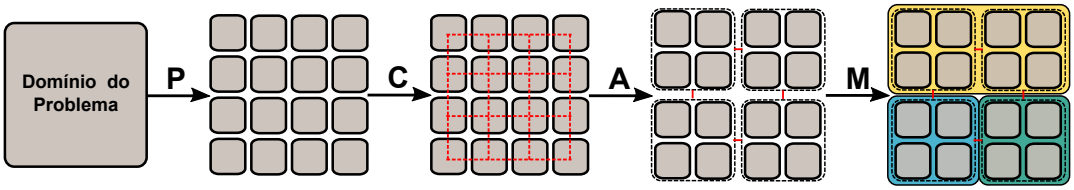


Figura 2.3. Fases da metodologia PCAM.

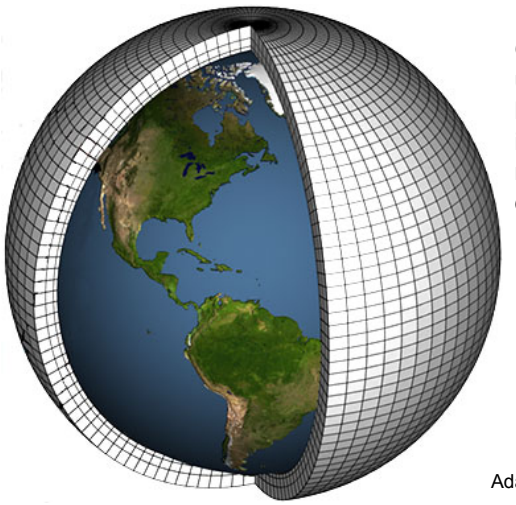
De acordo com o método, o projeto de um programa paralelo deve começar a partir de uma solução algorítmica existente (possivelmente sequencial) para um problema computacional, particionando-o em (muitas) pequenas tarefas e identificando dependências entre elas que podem resultar em comunicação e sincronização, para as quais deve-se selecionar as estruturas mais apropriadas. A granularidade da tarefa deve ser a mais fina possível para não restringir artificialmente as fases posteriores do projeto. O resultado dessas duas fases iniciais é um algoritmo paralelo escalável em um modelo de programação abstrato que é amplamente independente de um determinado computador paralelo e do número de processadores. Em seguida, as tarefas são aglomeradas em macrotarefas (processos) para reduzir a comunicação interna e as relações de sincronização dentro de uma macrotarefa a acessos à memória local. Por fim, mapeia-se as tarefas para os processadores, geralmente com o objetivo de minimizar o tempo total de execução. Técnicas de balanceamento de carga e/ou escalonamento de tarefas podem ser usadas para melhorar a qualidade do mapeamento. As duas últimas etapas, aglomeração e mapeamento, são mais dependentes da máquina porque usam informações sobre o número de processadores disponíveis, a topologia da rede, o custo de comunicação, etc.

2.4.1. Particionamento

Nesta primeira etapa, identificam-se fontes potenciais de paralelismo particionando (ou decompondo) o problema em pequenas tarefas. É a base de toda programação paralela, de uma forma ou de outra (Wilkinson and Allen 2005). O foco está em definir um grande número de pequenas tarefas para produzir uma decomposição refinada de um problema. A decomposição do problem em uma granularidade mais fina fornece flexibilidade em termos de possíveis algoritmos paralelos. Posteriormente, a avaliação dos requisitos de comunicação, a arquitetura da máquina ou questões de engenharia de software nos levam a abrir mão de muitas oportunidades identificadas para execução paralela neste estágio (Stanimirović 2020).

O particionamento pode ser aplicado aos dados do programa, dividindo-os e realizando o processamento sobre as partes simultaneamente. Essa abordagem de particionamento é chamada de *decomposição de domínio*. Diferentes partições podem ser possíveis, com base em diferentes estruturas de dados e modos de particionamento. Uma boa prática é focar na maior estrutura de dados ou naquela que é acessada com frequência.

A Figura 2.4 ilustra a decomposição de domínio em um modelo climático envolvendo uma grade tridimensional, composto por um conjunto de células. No estágio de particionamento, a ideia é decompor da forma mais agressiva possível, por exemplo, definindo uma tarefa para cada celula da grade. Nesse tipo de modelo, a computação é executado repetidamente, sendo que o cálculo referente a cada célula pode (potencialmente) ser resolvido em paralelo.



Os modelos climáticos dividem a Terra em uma grade com intervalos verticais e horizontais. Quanto menores os intervalos, mais fina a grade e melhor a resolução do modelo, ou seja, mais detalhes o modelo pode produzir.

Adaptado de: https://str.llnl.gov/december-2017/bader

Figura 2.4. Decomposição de domínio.

O particionamento também pode ser aplicado às funções de um programa, ou seja, dividindo-o em funções independentes e executando-as simultaneamente. Isso chama-se *decomposição funcional*. A ideia de executar uma tarefa dividindo-a em várias tarefas menores que, quando concluídas, completarão a tarefa geral é, obviamente, bem conhecida e pode ser aplicada em muitas situações, quer as tarefas menores operem em partes dos dados ou sejam funções concorrentes separadas. A Figura 2.5 ilustra um exemplo de decomposição funcional em um modelo climático.

Enquanto cada componente pode ser paralelizado mais naturalmente usando técnicas de decomposição de domínio, o algoritmo paralelo como um todo pode ser decomposto usando técnicas de decomposição funcional, mesmo que esse processo não produza um grande número de tarefas. Observe que o a decomposição de domínio geralmente tem uma granularidade mais fina do que o paralelismo de tarefas.

Embora a decomposição de domínio seja a base para a maioria dos algoritmos paralelos, a decomposição funcional é valiosa como uma maneira diferente de pensar sobre os problemas. Por esse motivo, deve ser considerado ao explorar possíveis algoritmos paralelos. Um foco nos cálculos a serem executados pode, às vezes, revelar a estrutura de um problema e, portanto, oportunidades de otimização, que não seriam óbvias apenas com o estudo dos dados.

Antes de avaliar as necessidades de comunicação, a checklist a seguir pode ser usada para garantir que o projeto não tenha falhas óbvias. Em geral, todas essas perguntas devem ser respondidas afirmativamente:

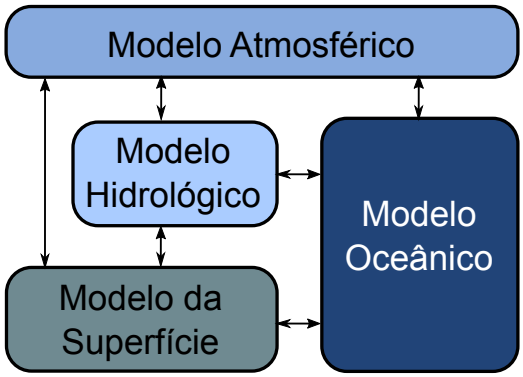


Figura 2.5. Decomposição funcional.

1. O particionamento do problema tem mais tarefas do que o número de processadores disponíveis? Se não tem, há pouca flexibilidade nos estágios de projeto subsequentes.
2. A partição evita computação e armazenamento redundantes?
3. As partições tem tamanho ou custo computacional semelhante? Quanto mais próximas, melhor, evitando problemas de balanceamento de carga.
4. O número de tarefas aumenta com o tamanho do problema? Idealmente, um aumento no tamanho do problema deve aumentar o número de tarefas em vez do tamanho das tarefas.
5. Foi possível identificar várias partições alternativas? Pode-se maximizar a flexibilidade nos estágios de projeto subsequentes considerando as diferentes alternativas. Lembre-se de verificar as decomposições de domínio e funcional.

2.4.2. Comunicação

Nesta etapa, determina-se a comunicação necessária entre as tarefas identificadas na primeira etapa. Especificam-se os dados que devem ser transferidos entre duas tarefas em termos de um canal que liga essas tarefas. Em um canal, uma tarefa pode enviar mensagens (produtor) e a outra pode receber (consumidor) (Schmidt et al. 2018).

Considerando o exemplo do modelo climático da seção anterior, as setas na Figura 2.5 representam trocas de dados entre componentes durante a computação: o modelo atmosférico gera dados de velocidade do vento que são usados pelo modelo oceânico, o modelo oceânico gera dados de temperatura da superfície do mar que são usados pelo modelo atmosférico e assim por diante.

Ainda no modelo climático, outro tipo de comunicação também pode ser necessário no caso da decomposição de domínios. Assumindo uma decomposição refinada na qual cada tarefa encapsula uma única célula da grade, e que o modelo numérico necessita de dados das 8 células vizinhas (estêncil de 9 pontos) na dimensão horizontal e duas células na vertical (estêncil de 3 pontos), o padrão de comunicação entre as células é como ilustrado na Figura 2.6.

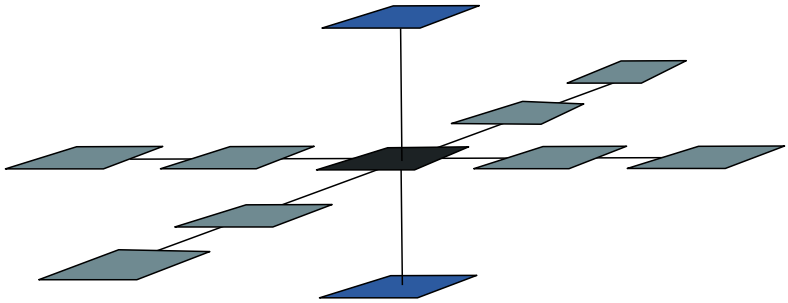


Figura 2.6. Para um único ponto de grade o estêncil de nove pontos é usado para troca de dados horizontal e o estêncil de três pontos usado para trocas de dados na vertical.

Na prática, existem muitos padrões de comunicação diferentes, como local ou global, estruturado ou não estruturado, estático ou dinâmico, síncrono ou assíncrono. Na comunicação local, cada tarefa se comunica com um pequeno conjunto de outras tarefas (seus ”vizinhos”); já a comunicação global exige que cada tarefa se comunique com muitas tarefas. Na comunicação estruturada, uma tarefa e seus vizinhos formam uma estrutura regular; em contraste, as redes de comunicação não estruturadas podem ser grafos arbitrários. Na comunicação estática, a identidade dos parceiros de comunicação não muda com o tempo; em contraste, a identidade dos parceiros de comunicação em estruturas de comunicação dinâmica pode ser determinada por dados computados em tempo de execução e pode ser altamente variável. Na comunicação síncrona, produtores e consumidores executam de forma coordenada, com pares produtor/consumidor cooperando em operações de transferência de dados; em contraste, a comunicação assíncrona pode exigir que um consumidor obtenha dados sem a cooperação do produtor.

A fase de comunicação também contém um checklist:

1. Todas as tarefas executam o mesmo número de operações de comunicação? Comunicação desequilibrada gera baixa escalabilidade.
2. Cada tarefa se comunica apenas com um pequeno número de vizinhos? Em caso necgativo, pode ser necessário reformular a comunicação global em termos de estruturas de comunicação locais.
3. As comunicações podem prosseguir em paralelo?
4. Os cálculos associados a diferentes tarefas podem ocorrer simultaneamente? Não: pode ser necessário reordenar cálculos/comunicações.

2.4.3. Aglomeração

No terceiro estágio, altera-se a granularidade do projeto obtida nos estágios anteriores, combinando uma série de pequenas tarefas em tarefas maiores. A aplicação particionada com granularidade muito fina pode não ser eficiente ao ser implementada diretamente em uma máquina real (em uma arquitetura de memória distribuída, por exemplo). Uma razão para isso é que a execução de um grande número de pequenas tarefas em paralelo usando diferentes processos/threads pode ser altamente ineficiente devido à sobrecarga

de comunicação. Para reduzir essa sobrecarga, pode ser benéfico agrupar várias tarefas pequenas em uma única tarefa maior no mesmo processador. Isso geralmente melhora a *localidade dos dados* e, portanto, reduz a quantidade de dados a serem comunicados entre as tarefas.

Na Figura 2.7, apresenta-se um recorte de 8×8 células da malha horizontal do modelo climático (a) e duas possibilidades de aglomeração (b) e (c). Note que o número de partições é distinto, bem com as necessidades de comunicação entre essas partições. Portanto, para um determinado problema pode haver diversas formas de aglomerar os dados, e cada uma delas pode resultar em diferentes resultados de desempenho.

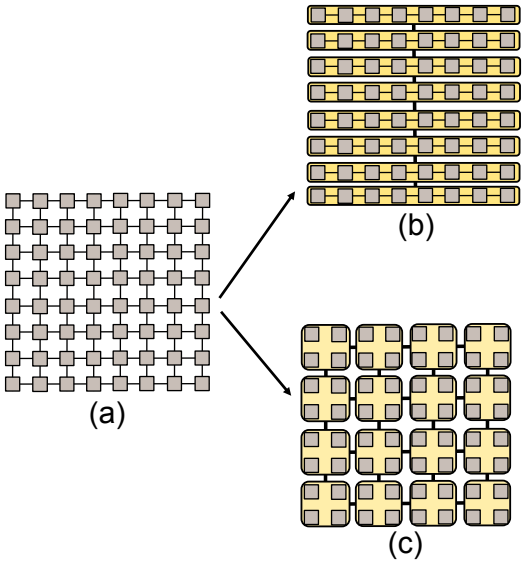


Figura 2.7. Diferentes abordagens de aglomeração. Partindo da mesma partição e padrão de comunicação (a), na primeira abordagem aglomera-se todas as tarefas ao longo da mesma linha (b) e na segunda abordagem aglomera-se uma grade quadrada de tarefas (c).

Nesta seção revisitou-se as decisões de particionamento e comunicação desenvolvidas nos dois primeiros estágios do projeto, aglomerando tarefas e operações de comunicação. A seguir, apresenta-se o checklist par esta fase, no qual as enfatizam a análise quantitativa do desempenho, que se torna mais importante à medida que passamos do abstrato para o concreto:

1. A aglomeração reduziu os custos de comunicação? Em caso negaitvo, deve ser usado uma estratégia alternativa de aglomeração.

2. A aglomeração produziu tarefas com custos de computação e comunicação semelhantes? Quanto maiores as tarefas criadas pela aglomeração, mais importante é que tenham custos semelhantes. Se criamos apenas uma tarefa por processador, essas tarefas devem ter custos quase idênticos.

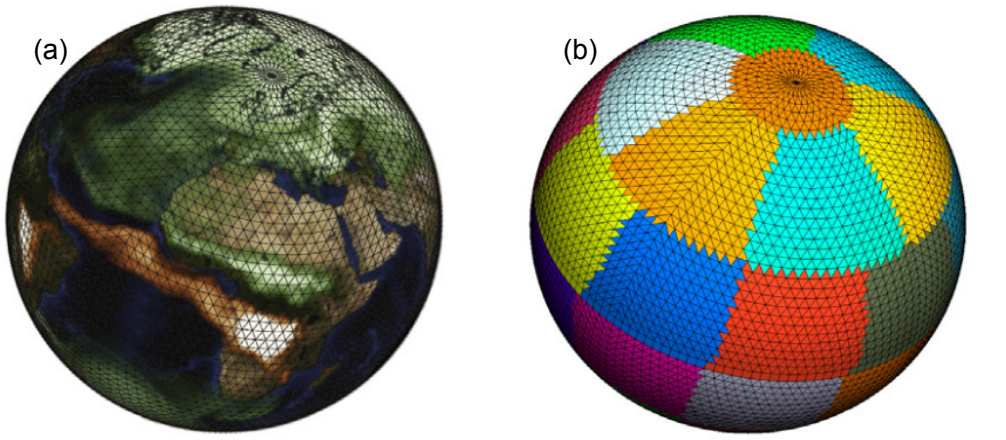
3. O número de tarefas ainda aumenta com o tamanho do problema? Caso contrário, seu algoritmo não será mais capaz de resolver problemas maiores em computadores paralelos maiores.

4. O número de tarefas pode ser reduzido ainda mais, sem introduzir desbalanceamento de carga ou reduzir a escalabilidade? Algoritmos que criam menos tarefas granulares são geralmente mais simples e mais eficientes do que aqueles que criam muitas tarefas refinadas.

2.4.4. Ferramentas de particionamento de grafos

Como visto, escolher o modo de aglomeração mais apropriado não é trivial principalmente quando temos domínios irregulares ou não-estruturados. Por exemplo, simulações numéricas de grande escala em computadores paralelos, requerem a distribuição dos elementos da malha entre os processadores. Essa distribuição deve ser feita de forma a balancear as computações entre os processadores e minimizando a comunicação (Barlas 2014).

Em muitos casos, a aglomeração pode ser mapeada para um problema de particionamento de grafos. O problema de particionamento de grafos consiste na divisão do conjunto de nós de um grafo em k blocos de tamanho igual, de modo que o número de arestas que correm entre os blocos seja minimizado (Buluç et al. 2016). Um exemplo de domínio computacional particionado é apresentado na Figura 2.8. Cada partição (conjunto de células) é representado por uma cor e possuem um número muito próximo de elementos.



Adaptada de Deconinck et al. (2017).

Figura 2.8. Malha esférica contendo 5248 células, agrupadas em 32 partições.

Na decomposição funcional, as tarefas geralmente dependem umas das outras, pois calculam dados que são usados por outras tarefas. As dependências entre as tarefas são descritas por meio de um grafo de dependência acíclico (DAG). Os vértices do grafo representam tarefas, e os arcos representam dependências entre tarefas decorrentes da transferência de dados ou sincronização de seu trabalho garantindo a ordem adequada de execução das tarefas. A aglomeração das tarefas com base na decomposição funcional

consiste em particionar o conjunto de vértices do grafo de dependência em subconjuntos que são atribuídos aos respectivos processadores (Czech 2016). A Figura 2.9 ilustra a aglomeração de um conjunto de tarefas mapeadas como um DAG. Cada cor representa um grupo de tarefas.

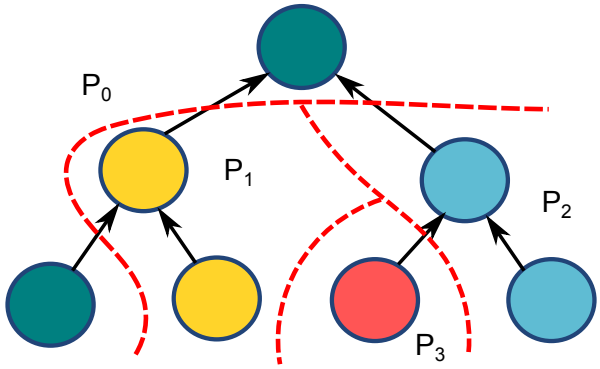


Figura 2.9. Aglomeração de tarefas baseada no particionamento de grafos. Adaptada de (Czech 2016)

Por se tratar de um problema NP-Difícil, ferramentas de particionaemnto que implementam heurísticas podem auxiliar nessa tarefa. Exemplos são os softwares Metis[3] e Zoltan[4] . O funcionamento dessas ferramentas está fora do escopo desse minicurso. Mais detalhes sobre as ferramentas podem ser encontradas em suas documentações oficiais.

2.4.5. Mapeamento

Para que a aplicação seja executada, os grupos de tarefas produzidos pela terceira etapa devem ser atribuídos/mapeados aos nós/processadores/cores disponíveis na arquitetura alvo. Os objetivos que precisam ser alcançados nesta etapa são (a) balancear a carga dos nós, ou seja, todos eles devem ter aproximadamente a mesma quantidade de trabalho medido pelo tempo de execução, e (b) minimizar a comunicação entre processadores atribuindo tarefas com interações frequentes ao mesmo processador (ou mais próximos).

A complexidade do mapeamento pode ser bastante distinta, dependendo da arquitetura e das características da aplicação. Por exemplo, muitos algoritmos desenvolvidos usando técnicas de decomposição de domínio apresentam um número fixo de tarefas de tamanho igual e comunicação local e global estruturada. Nesses casos, um mapeamento eficiente ocorre de modo direto. Por outro lado, em algoritmos baseados em decomposição de domínio mais complexos com quantidades dinâmicas de trabalho por tarefa e/ou padrões de comunicação não estruturados, estratégias eficientes de aglomeração e mapeamento podem não ser óbvias.

A Figura 2.10 mostra dois cenários relacionados ao mapeamento de 16 grupos de tarefas. No primeiro cenário (a), as 16 tarefas são mapeadas em quatro nós de processamento com a mesma configuração de hardware, e portanto, com mesma capacidade de computação. Nesse caso, o grupo é divido igualmente entre os nós. No cenário (b), as

[3] ⟨https://github.com/KarypisLab/ParMETIS⟩
[4] ⟨https://sandialabs.github.io/Zoltan/⟩

tarefas são distribuídas entre 3 nós heterogêneos, sendo que 2 deles possuem mais capacidade de computação que o terceiro. Aqui, as duas máquinas devem receber mais grupos para processamento que a terceira máquina.

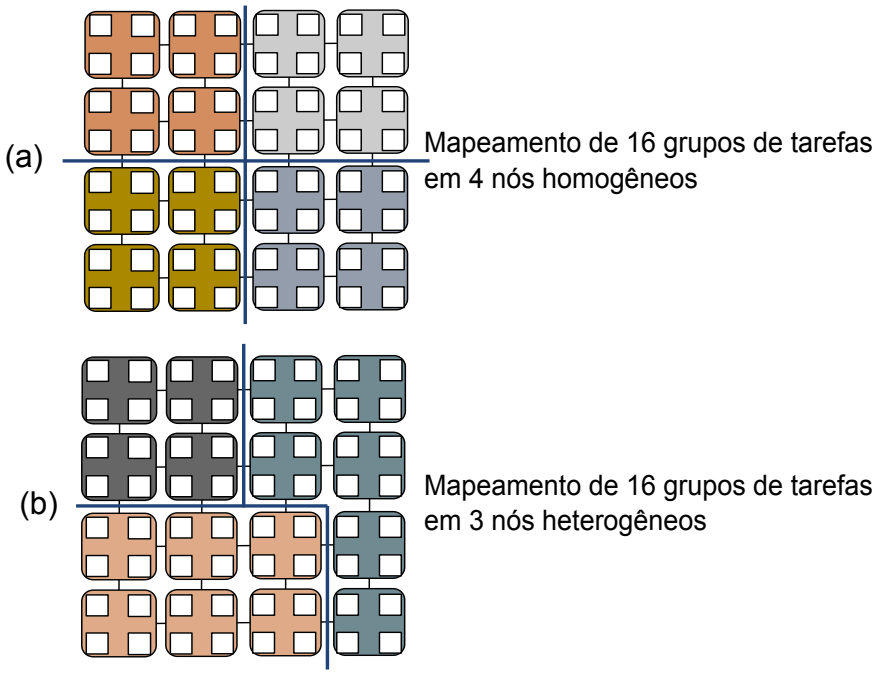


Figura 2.10. Diferentes mapeamentos de tarefas.

O mapeamento em uniprocessadores ou em computadores de memória compartilhada é mais simples, geralmente realizado pelo próprio sistema operacional ou por uma platarforma de execução (runtime). No entanto, o mapeamento em máquinas de memória distribuída deve ser feito por escalonadores e balanceadores de carga apropriados. Os detalhes dos escalonadores e balanceadores de carga está além do contexto desse minicurso. Sugestões e leitura sobre o assunto são as referências (Czech 2016) e (Barlas 2014).

2.5. Padrões de Projeto para Aplicações Paralelas

Um padrão de projeto descreve uma boa solução para um problema recorrente em um contexto particular. A ideia é registrar a experiência dos especialistas de uma forma que possa ser usada por outros que enfrentem um problema semelhante.

(Mattson et al. 2004) propõem um conjunto de padrões organizado em quatro espaços de projeto: (1) Encontrando o Paralelismo, (2) Estrutura do Algoritmo, (3) Estruturas de Apoio e (4) Mecanismos de Implementação. O fluxo descrito na Figura 2.11 sugere que o programador realize a modelagem do sistema seguindo passos que permitem explorar o paralelismo de maneira adequada. Os passos consistem em definir a melhor maneira para encontrar porções paralelizáveis, estruturar o sistema para que as partes paralelizáveis possam ser executadas de maneira concorrente, escolher as estruturas de suporte à execução paralela que mais condizem com o algoritmo definido e por fim, es-

colher quais os mecanismos de implementação que permitem transformar estas estruturas de suporte em programas executáveis.

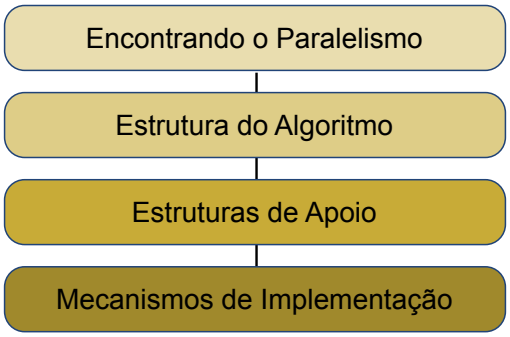


Figura 2.11. Espaços de projeto propostos por (Mattson et al. 2004).

2.5.1. Encontrando o Paralelismo

Nesta primeira etapa, o projetista precisa entender quais partes do problema são mais computacionalmente intensivas, porque o esforço para paralelizar o problema deve ser focado nessas partes (Hotspots, já tratados na Seção 1.3 deste minicurso).

Após a conclusão dessa análise, os padrões no espaço de projeto *Encontrando o Paralelismo* podem ser usados para iniciar o projeto de um algoritmo paralelo. Os padrões neste espaço de design podem ser organizados em dois grupos: (1) Decomposição, (2) Análise de dependências.

Basicamente, os padrões de decomposição *Decomposição de tarefas* (ou funcional) e *Decomposição de dados* (ou de domínio), são usados para decompor o problema em partes que podem ser executadas simultaneamente. Note que essa etapa relaciona-se à etapa de Particionamento da metodologia PCAM (ver Seção 1.4.1).

O grupo de Análise de Dependência contém três padrões que ajudam a agrupar as tarefas e analisar as dependências entre elas: Agrupamento de Tarefas, Ordenação de Tarefas e Compartilhamento de Dados. O ponto de partida em uma análise de dependência é agrupar tarefas com base em restrições entre elas e, em seguida, determinar quais restrições de ordenação se aplicam a grupos de tarefas. O próximo passo é analisar como os dados são compartilhados entre os grupos de tarefas, para que o acesso aos dados compartilhados seja gerenciado corretamente. Esse padrão equivale à fase de Comunicação e Aglomeração da metodologia PCAM, apresentada na Seção 1.4.3.

2.5.2. Estrutura do Algoritmo

Objetivo do espaço de Estrutura do Algoritmo é refinar o projeto e aproximá-lo de um programa que pode executar tarefas simultaneamente mapeando a simultaneidade em várias unidades de execução executando em um computador paralelo.

(Mattson et al. 2004) listam uma série de padrões de decomposição que podem abranger as formas básicas pelas quais uma carga de trabalho pode ser decomposta para eventual distribuição aos nós de uma plataforma paralela/multicore. A Figura 2.12 mostra a árvore de decisão que leva a um dos seis padrões possíveis. Em geral, um problema pode

ser decomposto de várias maneiras diferentes. A variedade de padrões nos permite pensar sobre o problema de paralelização de diferentes perspectivas.

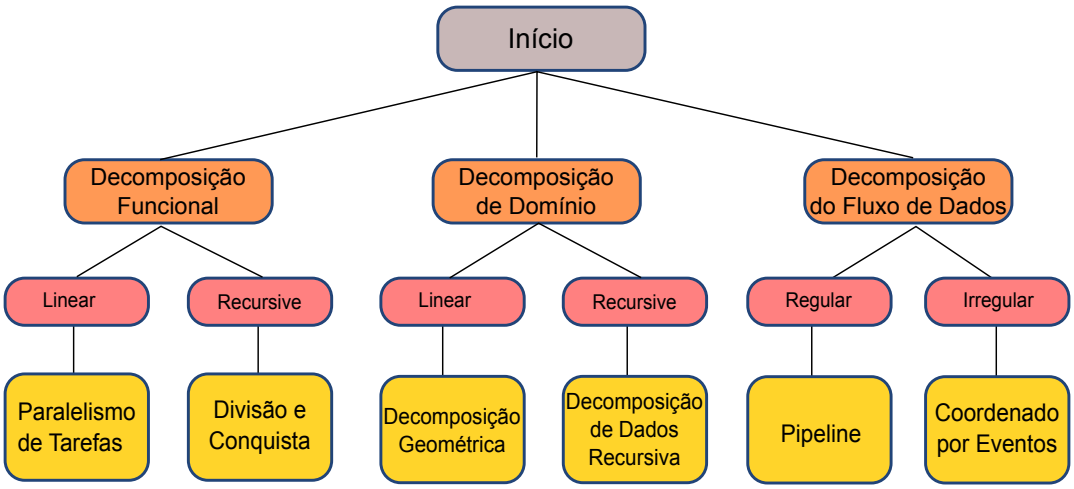


Figura 2.12. Padrões para Estrutuda do Algoritmo.

A Decomposição Funcional deve ser escolhida quando a execução das próprias tarefas for o melhor princípio de organização. Em seguida, determine como as tarefas são enumeradas. Se eles puderem ser reunidos em um conjunto linear em qualquer número de dimensões, escolha o padrão Paralelismo de Tarefas. Este padrão inclui tanto situações em que as tarefas são independentes umas das outras (trivialmente paralelizáveis) quanto situações em que existem algumas dependências entre as tarefas na forma de acesso a dados compartilhados ou necessidade de troca de mensagens. Se as tarefas forem enumeradas por um procedimento recursivo, escolha o padrão Dividir e Conquistar. Nesse padrão, o problema é resolvido dividindo-o recursivamente em subproblemas, resolvendo cada subproblema independentemente e, em seguida, recombinando as subsoluções na solução do problema original.

A Decomposição de Domínio é mais apropriada quando o particionamento dos dados for o principal forma de organizar o paralelismo. O padrão de Decomposição Geométrica pode ser escolhido quando o espaço do problema for decomposto em partições discretas e o problema for resolvido computando soluções para estas partições, com a necessidade de troca de dados com outras partições. A escolha do padrão Dados Recursivos é mais apropriada quando o problema for definido em termos de uma estrutura de dados recursiva (por exemplo, uma árvore binária).

A Organizar por Fluxo de Dados pode ser usada quando o princípio de organização é baseado em como o fluxo de dados impõe uma ordem nos grupos de tarefas. Os dados podem seguir uma sequência predeterminada de etapas que levam ao uso do padrão Pipeline, ou podem seguir padrões irregulares, como no caso de simulações de eventos discretos, levando ao uso do padrão Coordenado por Eventos.

2.5.3. Estruturas de Apoio

Os padrões no espaço de projeto de Estruturas de Apoio abordam a fase do processo de projeto de programa paralelo, representando um estágio intermediário entre os padrões orientados a problemas do espaço de projeto de Estrutura de Algoritmo e os mecanismos de programação específicos descritos em Mecanismos de Implementação. Alguns padrões de estrutura são resumidos a seguir (Barlas 2014).

SPMD Em um programa SPMD (Single Program, Multiple Data) todos as unidades de processamento da plataforma de execução executam o mesmo programa, mas podem aplicar as mesmas operações em dados diferentes e/ou seguir diferentes caminhos de execução dentro do programa. Geralmente, essa diferenciação é baseada em um identificador de processo/thread.

MPDM O padrão MPMD (Multiple-Program, Multiple-Data) permite que diferentes executáveis componham uma aplicação. Cada nó de computação é livre para executar sua própria lógica de programa e processar seu próprio conjunto de dados.

Master/Worker Este padrão distingue dois tipos de atores: um mestre e um conjunto de escravos. O mestre coordena a execução da aplicação atribuindo e escalonando unidades de trabalho a cada um dos escravos para processamento. Os escravos recebem as tarefas, as processam e retornam os resultados. Assume-se que o trabalho pode ser dividido em um conjunto de tarefas independentes que podem ser processadas independentemente pelos escravos. Aqui, o número de tarefas pode ser igual ao número de escravos ou o trabalho pode ser dividido em um número mais significativo de tarefas

Loop Parallelism Este padrão aborda o problema de transformar um programa serial, cujo tempo de execução é dominado por um conjunto de laços, em um programa paralelo onde as diferentes iterações do loop são executadas em paralelo.

Fork/Join O padrão fork/join é empregado quando o algoritmo paralelo solicita a criação dinâmica (fork) de tarefas em tempo de execução. Essas tarefas filhas (processos ou threads) geralmente precisam terminar (join) antes que o processo/thread pai possa retomar a execução.

Qual é a melhor forma de estruturar o programa? A resposta é altamente dependente da aplicação. Alguns padrões podem ser utilizados de forma isolada, mas também podem ser combinados de maneiras diferentes para atender às necessidades de um determinado problema. Várias plataformas de execução podem impor um padrão específico de estrutura de programa aos desenvolvedores. Por exemplo, o MPI usa o padrão SPMD/MPMD, enquanto o OpenMP promove o padrão de paralelismo de laço (Barlas 2014). Além disso, dado um padrão de decomposição específico, certos padrões de estrutura de programa são mais adequados para fins de implementação. A Tabela na Figura 2.5.3 resume essas combinações.

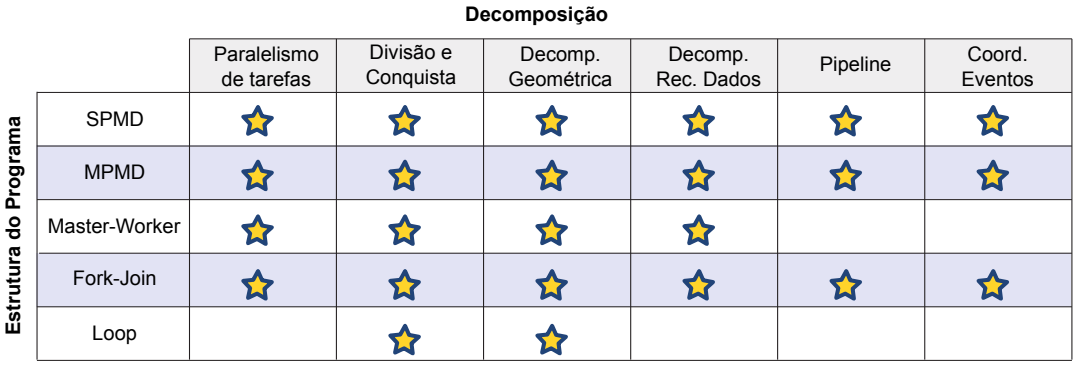


Figura 2.13. Padrões de Decomposição e os Padrões de Estrutura de Programa Mais Adequados para Implementá-los.

2.5.4. Mecanismos de Implementação

Os mecanismos de implementação definem a gestão de unidades de execução, comunicação e sincronização. Neste nível, os padrões são estabelecidos pelo modelo de programação, de forma que cabe ao programador escolher adequadamente este modelo.

Dependendo da arquitetura alvo, diferentes unidades de execução (UE). Para ambientes de software baseados em memória distribuída, como MPI, as UEs são mapeadas em processos. Ambientes baseados em memória compartilhada, como OpenMP, utilizam threads.

Na maioria dos algoritmos paralelos, as UEs precisam se comunicar para trocar informações à medida que a computação prossegue. Os ambientes de memória compartilhada fornecem esse recurso por padrão. Em sistemas de memória distribuída, como não há uma memória comum entre os processadores, a comunicação é feita por troca de mensagens, que pode ser feita de modo coletivo (broadcast, multicast) ou ponto-a-ponto (send/recv).

No gerenciamento de threads e processos, a sincronização ocorre quando se deseja manter uma ordem na execução dos eventos. Para sincronização, os padrões utilizados são: Cercas (Fences), Barreiras e Exclusão Mútua.

2.6. Considerações Finais

O objetivo deste minicurso foi apresentar uma visão introdutória sobre o projeto de aplicações paralelas, de modo a servir como um ponto de partida para os iniciantes na área. Foram duas abordagens diferentes, PCAM e Padrões de Projeto, cada uma com suas particularidades mas que podem ser utilizadas de forma complementar.

Para um aprofundamento deste tópico, recomenda-se a leitura dos livros indicados nas referências.

Referências

Barlas 2014 Barlas, G. (2014). *Multicore and GPU Programming: An Integrated Approach*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

Buluç et al. 2016 Buluç, A., Meyerhenke, H., Safro, I., Sanders, P., and Schulz, C. (2016). *Recent Advances in Graph Partitioning*, pages 117–158. Springer International Publishing, Cham.

Czarnul 2018 Czarnul, P. (2018). *Parallel Programming for Modern High Performance Computing Systems*. Chapman, 1st edition.

Czech 2016 Czech, Z. J. (2016.). *Introduction to parallel computing /*. Cambridge University Press,, Cambridge:. Includes index.

Foster 1995 Foster, I. T. (1995). *Designing and building parallel programs - concepts and tools for parallel software engineering*. Addison-Wesley.

Mattson et al. 2004 Mattson, T., Sanders, B., and Massingill, B. (2004). *Patterns for Parallel Programming*. Addison-Wesley Professional, first edition.

Pacheco 2011 Pacheco, P. (2011). *An Introduction to Parallel Programming*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1st edition.

Schmidt et al. 2018 Schmidt, B., González-Domínguez, J., Hundt, C., and Schlarb, M. (2018). *Parallel Programming*. Morgan Kaufmann.

Stanimirović 2020 Stanimirović, I. (2020). *Parallel Programming*. Arcler Press, first edition.

Trobec et al. 2018 Trobec, R., Slivnik, B., Bulic, P., and Robic, B. (2018). *Introduction to Parallel Computing - From Algorithms to Programming on State-of-the-Art Platforms*. Undergraduate Topics in Computer Science. Springer.

Wilkinson and Allen 2005 Wilkinson, B. and Allen, M. (2005). *Parallel programming - techniques and applications using networked workstations and parallel computers (2. ed.)*. Pearson Education.

Capítulo 3

DevOps para HPC: Como configurar um cluster para uso compartilhado

Lucas Leandro Nesi, Lucas Mello Schnorr
Universidade Federal do Rio Grande do Sul

Resumo

O compartilhamento de recursos para a execução de aplicações de alto desempenho é uma tarefa considerável. Requisitos como controle de usuários, compartilhamento de dados, assim como a reserva e isolamento de recursos são cruciais no sistema. Neste minicurso, apresentaremos um conjunto de softwares que pode ser utilizado para resolver tais desafios, sempre demonstrando as configurações possíveis para adequar o sistema a ser compartilhado para os requisitos dos usuários. A pilha de software é inspirada naquela utilizada no PCAD (Parque Computacional de Alto Desempenho) do Grupo de Processamento Paralelo e Distribuído da Universidade Federal do Rio Grande do Sul.

3.1. Introdução

O agrupamento e compartilhamento de recursos computacionais entre diversos usuários é um passo essencial para garantir a boa utilização dos mesmos, saciando as demandas dos utilizadores. Com um compartilhamento bem feito, pode-se garantir uma justa utilização dos recursos por diversos usuários sem que suas cargas de trabalho interfiram umas nas outras. A agregação de diversos nós computacionais é conhecida como *cluster*. Cargas de trabalho de Computação de Alto Desempenho (CAD) normalmente utilizam um ou mais nós para a execução de uma tarefa finita de um usuário.

Para um bom funcionamento, o ambiente de compartilhamento deve proporcionar algumas características básicas, entre elas, um sistema de autenticação e um sistema de controle de recursos. O minicurso considera **os seguintes objetivos** na construção de uma infraestrutura compartilhada para CAD. (i) A uniformidade dos usuários entre nós com um sistema de autenticação unificado. Isto garante o mesmo perfil (UID, GID, caminho da `$HOME`) em todas as máquinas. (ii) O compartilhamento da `$HOME` via um sistema de arquivos em rede para todos os nós. Desta forma, existe uma unificação virtual da infraestrutura para os usuários de forma transparente. Não sendo necessário realizar cópias de

arquivos manualmente via rede entre máquinas. (iii) Um sistema que controla a utilização dos recursos pelos usuários, como um que ofereça escalonamento de tarefas. (iv) Que a infraestrutura computacional tenha algumas características mínimas de segurança.

Para atingir tais objetivos, a organização da infraestrutura proposta neste minicurso segue uma abordagem centralizada. Ela é estruturada com a presença de um nó controlador, que sustenta a maioria dos serviços essenciais e controla os nós de computação denominados clientes. Considera-se adicionalmente o acesso dos usuários direto ao sistema operacional, sem virtualização. É responsabilidade dos usuários a instalação de bibliotecas necessárias para a compilação de suas aplicações, incluindo a própria execução destas em nível de usuário. Um sistema operacional único já atende a maioria dos usuários e facilita a manutenção, sendo um primeiro passo para o compartilhamento dos recursos.

Este minicurso é inspirado no caso real do Parque Computacional de Alto Desempenho (PCAD)[1], mantido pelo Grupo de Processamento Paralelo e Distribuído (GPPD) do Instituto de Informática da UFRGS. O PCAD está em funcionamento com configurações similares as propostas desde 2018 (Nesi et al. 2019, Nesi et al. 2020).

Este capítulo está organizado da seguinte forma. A Seção 3.2 apresenta um conjunto de softwares que podem ser utilizados para cada objetivo e as razões para estas escolhas. A Seção 3.3 discute os passos iniciais da instalação, estrutura e configuração. As próximas seções discutem a instalação e configuração de softwares específicos no controlador: Seção 3.4 para segurança, Seção 3.5 para autenticação unificada, Seção 3.6 para o sistema de arquivos distribuído, Seção 3.7 para compartilhar os recursos. A Seção 3.8 discute a instalação e configuração de um nó computacional. A Seção 3.9 apresenta outros softwares úteis que podem ser configurados. Finalmente, a Seção 3.10 conclui este documento.

3.2. Softwares para a infraestrutura

Esta seção discute os vários programas e serviços que podem ser utilizados para atingir os objetivos mencionados na introdução. Primeiramente, utiliza-se o sistema operacional Linux com a distribuição Debian 12, por ser *open-source*, possuir uma forte comunidade, e contar com várias aplicações em seu gerenciador padrão de pacotes. Considera-se que vários usuários estão familiarizados com sistemas baseados no Debian.

3.2.1. Sistema de autenticação

Os programas existentes para oferecer um sistema de autenticação unificado possuem diferentes objetivos e sua disponibilidade e estabilidade em diversos sistemas é variada. Uma das soluções mais comuns é o protocolo LDAP para o compartilhamento de informações de diretório. Neste caso, ele pode ser utilizado para armazenar informações de usuários, senhas e chaves ssh. Um cliente LDAP se comunica com um controlador LDAP para recuperar estas informações. Por ser concebido na década de 1990 e amplamente utilizado, diversas implementações do protocolo LDAP existem para diferentes sistemas operacionais e distribuições. No caso do Debian, uma opção é o OpenLDAP, uma implementação aberta. Como alternativa mais recente, pode-se citar FreeIPA, mantida pela Red Hat para seus sistemas. Ela possui um suporte experimental no Debian atualmente.

[1]<https://gppd-hpc.inf.ufrgs.br/>

3.2.2. Sistema de arquivos distribuído

Um dos sistemas de arquivos distribuídos mais simples e utilizado é o *Network File System* (NFS). Ele possui um controlador que exporta um diretório do sistema de arquivos local para clientes remotos de forma transparente. Atualmente o NFS faz parte do kernel do Linux, condição que melhora seu desempenho, disponibilidade e facilidade de uso comparada com outras alternativas. Entretanto, em ambientes CAD é comum a utilização de sistemas de arquivos paralelos para maximizar o desempenho, sendo o Lustre um exemplo. Apesar disto, seus requisitos mínimos (nós dedicados para armazenamento e controle de requisições) e sua complexidade de instalação maior que o NFS são desvantagens em *clusters* de menor porte. Na necessidade destes sistemas de arquivos paralelos, os passos que descrevem a instalação do NFS neste minicurso podem ser evitadas.

3.2.3. Sistema de compartilhamento de recursos

No ambiente de CAD, o gerenciador de recursos e workloads SLURM é bastante utilizado, sendo presente em alguns dos maiores supercomputadores do mundo. Com ele, usuários podem submeter tarefas, que serão escalonadas e isoladas nos recursos pelo SLURM. Outras alternativas existem, como o OAR. Umas das principais diferenças do OAR em relação ao SLURM é a possibilidade da reserva de recursos por usuários e seu possível forte acoplamento com o Kadeploy, permitindo o boot de imagens customizadas. Considerando a grande utilização do SLURM, sua fácil instalação, e para proporcionar aos estudantes do grupo experiência na utilização do mesmo, o PCAD escolheu utilizar o SLURM, estendendo tais razões para este minicurso.

3.3. Prelúdio da Instalação

Esta Seção introduz os passos para a instalação da infraestrutura. Consulte a versão do guia atualizado em: <https://gitlab.com/lnesi/mc-hpc-share>.

3.3.1. Estrutura do sistema de arquivos

O Controlador possui uma `/home` exportada para todas as máquinas via NFS. Cada máquina possui um diretório `/scratch` com um subdiretório por usuário, normalmente `/scratch` é um volume em um disco diferente do `/`, e local à máquina.

3.3.2. Instalação do Debian

O Debian utilizado neste minicurso é o 12. Atualmente o Debian 12 encontra-se em *release candidate*[2]. Durante sua instalação pode-se seguir a estrutura de arquivos sugerida e padrão. Atente-se que o usuário criado durante a instalação (UID 1000) deve ser o mesmo entre controlador e clientes. Assumiremos que trata-se do usuário USER.

3.3.3. Redes do Controlador e dos Clientes

Na configuração do PCAD, as máquinas possuem duas redes. A primeira interface está conectada na rede do Instituto de Informática da UFRGS (com DHCP), permitindo conexão direta com qualquer máquina da instituição e com a internet. A segunda in-

[2]<https://www.debian.org/devel/debian-installer/>

terface de cada máquina é a rede interna da infraestrutura, com IP fixo nos endereços 192.168.30.01/24. Os serviços relacionados a infraestrutura (NFS, LDAP, SLURM) estão somente habilidades na rede interna. Ainda, para as máquinas com IPMI a rede é configurada nos endereços 192.168.10.01/24 quando possível.

A configuração de rede no Debian é feita modificando o arquivo: `</etc/network/interfaces>`. Para habilitar a segunda interface com IP estático é necessário adicionar as seguintes linhas, sendo INTERFACE o nome da segunda interface (verificar com `ip address`) e FINAL o identificar escolhido da máquina.

Adição em `/etc/network/interfaces`

```
auto INTERFACE
iface INTERFACE inet static
  address 192.168.30.FINAL
  network 192.168.30.0
  netmask 255.255.255.0
  broadcast 192.168.30.255
```

É possível ainda uniformizar os domínios na rede, e forçar o uso dos *hostnames* via a segunda interface. É necessário configurar com a rede interna o arquivo `</etc/hosts>`:

Adição em `/etc/hosts`

```
192.168.30.2 controlador
192.168.30.3 client1
```

3.3.4. Pacotes e Configuração Básica

Com a instalação base do Debian é necessário instalar o `sudo` via superusuário (root) e adicionar o usuário não `root` ao `sudoers`. Assumimos que trata-se do USER criado na instalação com UID 1000.

Comandos no terminal do controlador como superusuário (root)

```
apt install sudo git nano
/sbin/adduser USER sudo
```

De volta ao usuário USER, já parte do grupo `sudo`, configuramos o emprego do comando `sudo` sem pedir sua senha, por agilidade, e geramos as chaves internas deste usuário.

Comando no terminal do controlador (usuário normal)

```
echo "%sudo   ALL=(ALL:ALL) NOPASSWD: ALL" | sudo tee -a /etc/sudoers
ssh-keygen -t rsa -f ~/.ssh/id_rsa -q -P ""
cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys
chmod 600 ~/.ssh/authorized_keys
```

Aconselha-se copiar a sua chave pública ssh para o usuário USER criado na instalação do Debian no controlador. Execute o próximo comando em sua máquina pessoal.

Comando no terminal de uma máquina pessoal

```
ssh-copy-id USER@CONTROLADOR
```

3.3.5. Nosso repositório com configurações e scripts

Com os pacotes básicos instalados utilizando o superusário, é possível clonar o repositório que acompanha este minicurso utilizando o usuário USER, criado durante a instalação do Debian no controlador. Para clonar o repositório, usa-se este comando no controlador:

Comando no terminal do controlador (usuário normal)

```
git clone https://gitlab.com/lnesi/mc-hpc-share
```

3.3.6. Variáveis utilizas durante a instalação

Algumas variáveis serão referenciadas durante a instalação e utilizadas no script automático da próxima subseção. Considere as seguintes variáveis: `NOME_INFRA` é nome dado para a infraestrutura, `CONTROLADOR_IP` e `CONTROLADOR_DNS` são o IP e o *hostname* do controlador, `CONTROLADOR_LDAP` é o *hostname* do controlador no formato para o LDAP (adicionando *dc=* para cada ponto), `ARQUIVO_SENHA` é o caminho para o arquivo de senha, `REDE_INTERNA` é a rede interna, `SCRIPTPATH` é o caminho absoluto para a pasta script do repositório, `ADMIN_USER` é o usuário USER criado durante a instalação do Debian (UID 1000).

Comando no terminal do controlador (usuário normal)

```
export NOME_INFRA="gppd-hpc"
export CONTROLADOR_IP="192.168.30.2"
export CONTROLADOR_DNS="controlador"
export CONTROLADOR_LDAP="dc=${CONTROLADOR_DNS}"
export ARQUIVO_SENHA="${SCRIPTPATH}/confs/pass"
export REDE_INTERNA="192.168.30.2/24"
export SCRIPTPATH="${SCRIPTPATH:-$HOME/mc-hpc-share/}"
export ADMIN_USER="parque"
```

3.3.7. Instalação via script

Disponibilizamos o script `<scripts/make_controler.sh>` para realizar a instalação automática do controlador seguindo este tutorial. Basta abrir o script e alterar as variáveis discutidas na seção anterior e o arquivo de senha `<scripts/confs/pass>`. Este arquivo não deve conter o caractere *newline* (utilize a opção -L no nano, `:set binary` no vim). Para executar o script entre no repositório git recém clonado e utilize o bash no arquivo `<scripts/make_controler.sh>`:

Comando no terminal do controlador (usuário normal)

```
cd mc-hpc-share
bash scripts/make_controler.sh
```

Após a conclusão deste script será necessário um reboot manual. Feita a reinicialização manual da máquina, continue a instalação com o segundo script:

Comando no terminal do controlador (usuário normal)

```
bash scripts/make_controler_2.sh
```

3.4. Segurança básica do controlador

Esta Seção apresenta três configurações para melhorar a segurança do controlador.

3.4.1. Firewall

Como possivelmente o controlador estará exposto à internet, é recomendado controlar quais serviços estão realmente expostos. Pode-se instalar o `ufw` para gerenciar o `iptables`, adicionar o comportamento padrão de negar qualquer conexão, adicionar a regra permitindo acesso à porta 22 (ssh) e 80 (http), e liberar tudo para a rede interna.

Comando no terminal do controlador (usuário normal)

```
sudo apt install -y ufw
sudo ufw default deny incoming
sudo ufw allow 22
sudo ufw allow 80
sudo ufw allow from $REDE_INTERNA to any
sudo ufw --force enable
```

3.4.2. Fail2Ban

O `fail2ban` é um software que monitora os arquivos de logs de serviços comuns (ssh, apache) e bane conexões que tentaram e falharam em se autenticar nestes serviços. Um usuário pode personalizar suas configurações criado o arquivo `</etc/fail2ban/jail.local>`. Por exemplo, considere um banimento de 240 horas, se dentro de 6 horas um mesmo IP tentou e falhou três vezes para se conectar em um serviço.

Arquivo `/etc/fail2ban/jail.local`

```
[DEFAULT]
bantime  = 240h
findtime  = 6h
maxretry = 3
```

Os comandos em seguida podem ser utilizados para instalação e configuração.

Comando no terminal do controlador (usuário normal)

```
sudo apt install -y rsyslog fail2ban
sudo cp $SCRIPTPATH/confs/jail.local /etc/fail2ban/jail.local
sudo chown root:root /etc/fail2ban/jail.local
sudo chmod 644 /etc/fail2ban/jail.local
sudo service fail2ban restart
```

3.4.3. Audit

O *Audit* é o sistema de auditoria presente no kernel do Linux. Ele permite o registro de eventos baseados nos arquivos de configuração. Podemos remover alguns eventos muito recorrentes e adicionar todos os comandos executados por um usuário nos registros alterando o arquivo `</etc/audit/rules.d/audit.rules>`. Considere a configuração proposta em `<scripts/confs/audit.rules>`.

Para instalar o serviço de acesso à auditoria, desativar que os registros apareçam no *journal* (para não sobrecarregar de mensagens, para acessá-los utilizar o comando *ausearch*):

Comando no terminal do controlador/cliente (usuário normal)

```
sudo -E apt install -y auditd
cat $SCRIPTPATH/confs/audit.rules | \
   sudo tee -a /etc/audit/rules.d/audit.rules
sudo systemctl restart auditd
sudo systemctl stop systemd-journald-audit.socket
sudo systemctl disable systemd-journald-audit.socket
sudo systemctl mask systemd-journald-audit.socket
```

Para buscar os comandos executados pelos usuários hoje:

Exemplo de comando para controlador/cliente (usuário normal)

```
sudo ausearch -ts today
```

3.5. Sistema unificado de autenticação

Esta Seção descreve a instalação dos pacotes selecionados e discutidos em um Debian 12.

3.5.1. Controlador: Instalação do LDAP - OpenLDAP

O LDAP já se encontra nos repositórios oficiais do Debian. Comando de instalação:

Comando no terminal do controlador (usuário normal)

```
sudo apt install -y slapd ldap-utils
```

Para configurar o controlador do LDAP:

Comando no terminal do controlador (usuário normal)

```
sudo dpkg-reconfigure slapd
```

Utilize as seguintes configurações:

Configuração	Sugestão de Valor
Omit OpenLDAP server configuration?	No
DNS domain name: provavelmente:	**$CONTROLADOR_DNS**
Organization name:	**$NOME_INFRA**
Administrator password:	**$SENHA_SERVICOS**
Database backend?	MDB
Remove the database when slapd is purged?	No
Move old database?	Yes

Considerando algumas questões de segurança[3] é necessário aplicar os comandos LDAP

[3]<https://www.openldap.org/doc/admin23/security.html>

do arquivo <scripts/confs/ldap_disable_bind_anon.ldif> para desativar acesso anônimo e requerer o uso de autenticação:

Comando no terminal do controlador (usuário normal)

```
sudo ldapadd -Y EXTERNAL -H ldapi:/// -f scripts/confs/ldap_anon.ldif
```

3.5.2. Controlador: Informação de chave ssh nos usuários

Primeiramente é necessário adicionar ao LDAP scheme a variável que guardará a chave ssh. Aplicando o arquivo <scripts/confs/openssh-lpk.ldif> utilizando:

Comando no terminal do controlador (usuário normal)

```
sudo ldapadd -Y EXTERNAL -H ldapi:/// -f scripts/confs/openssh-lpk.ldif
```

3.5.3. Controlador: Gerenciamento auxiliar (web) do LDAP - phpLDAPadmin

O phpLDAPadmin é uma interface web para a manipulação dos dados armazenados no LDAP. Para a instalação do pacote:

Comando no terminal do controlador (usuário normal)

```
sudo apt install -y phpldapadmin
```

As próximas configurações são para melhorar a qualidade de vida do administrador no uso do phpLDAPadmin. Conside o patch <scripts/confs/diff_phpldapadmin_config> para ser aplicado no arquivo </etc/phpldapadmin/config.php>. Alterando as variáveis necessárias.

Comando no terminal do controlador (usuário normal)

```
sudo patch -l /etc/phpldapadmin/config.php \
          $SCRIPTPATH/confs/diff_pla_config
sudo sed -i "s/NOME_INFRA/${NOME_INFRA}/g" /etc/phpldapadmin/config.php
sudo sed -i "s/CONTROLADOR_LDAP/${CONTROLADOR_LDAP}/g" \
          /etc/phpldapadmin/config.php
```

Aplicar os seguintes patchs. Primeiro no arquivo </etc/phpldapadmin/templates/creation/posixAccount.xml> para adicionar os campos e-mail, chave ssh, e forçar o uso do bash. Então no arquivo </usr/share/phpldapadmin/htdocs/create_confirm.php> para melhorar a confirmação de adição de usuários. No arquivo </usr/share/phpldapadmin/lib/ds_ldap_pla.php> para mudar o padrão de senha. Por último no arquivo </usr/share/phpldapadmin/lib/PageRender.php> para automaticamente gerar uma senha.

Comando no terminal do controlador (usuário normal)

```
sudo patch -l /etc/phpldapadmin/templates/creation/posixAccount.xml \
           $SCRIPTPATH/confs/diff_pla_posix
sudo patch -l /usr/share/phpldapadmin/htdocs/create_confirm.php \
           $SCRIPTPATH/confs/diff_pla_create
```

```
sudo patch -l /usr/share/phpldapadmin/lib/ds_ldap_pla.php \
           $SCRIPTPATH/confs/diff_pla_pla
sudo patch -l /usr/share/phpldapadmin/lib/PageRender.php \
           $SCRIPTPATH/confs/diff_pla_page
```

3.5.4. Controlador: Primeiro uso do LDAP e do phpLDAPadmin

Com estas configurações, o phpldapadmin está em <http://controlador/phpldapadmin/>.

O login é definido por "cn=admin,CONTROLADOR_LDAP"(deve estar autopreenchido) e a senha do LDAP. É necessário criar as primeiras entidades que vão organizar os diretórios do LDAP. Após logado criar uma entrada “Generic: Organisational Unit”, que será a organização com a qual as contas estarão associadas. Pode-se utilizar o nome da infraestrutura. Então, crie uma *child entry* na organização e adicione um entrada “Generic: Posix Group”, chamado “members”. Este será o grupo principal de membros da infraestrutura. Pode-se agora cadastrar um usuário criando uma “/child entry/ Generic: User Account” no grupo “members”. O “User ID” será o identificador Linux do usuário. Um primeiro usuario a ser criado é o usuário com o “User ID” “slurm”, sem chave ssh.

3.5.5. Controlador: Login de novos usuários: autocriação da HOME e banner

Edite o arquivo `</etc/pam.d/common-session>` para configurar a autocriação da `home`. Adicionando na última linha:

Adição em `/etc/pam.d/common-session`

```
session required    pam_mkhomedir.so skel=/etc/skel umask=0077
```

Desta forma, a pasta `/etc/skel` será copiada para todo novo usuário que entrar no sistema. Por exemplo, podemos alterar `</etc/skel/.bashrc>` para gerar uma chave que será utilizada internamente na infraestrutura:

Comando no terminal do controlador (usuário normal)

```
sudo patch -l /etc/skel/.bashrc $SCRIPTPATH/confs/diff_skel_bashrc
ssh-keygen -t rsa -f ~/.ssh/id_rsa -q -P ""
cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys
chmod 600 ~/.ssh/authorized_keys
```

Para adicionar um Banner de entrada, adicione as seguintes linhas em `</etc/pam.d/common-session>` e criar o arquivo `</etc/motd>` com a mensagem desejada.

Adição em `/etc/pam.d/common-session`

```
session [default=1 success=ignore] pam_succeed_if.so quiet user ingroup
   → members
session optional pam_motd.so motd=/etc/motd
```

3.5.6. Controlador e Cliente: Login de usuários via LDAP e chaves ssh

Esta configuração se aplica ao controlador e aos clientes. Pode-se utilizar o `PAM` (Linux Pluggable Authentication Modules) para consultar os usuários no LDAP. Para isso, utilizamos um sistema de caches dos nomes com o *nscd* e do LDAP *nslcd*. Para instalação:

Comando no terminal do controlador/cliente (usuário normal)

```
sudo apt install libnss-ldapd libpam-ldapd nscd nslcd nslcd-utils
```

Para fazer todas as configurações é necessário utilizar:

Comando no terminal do controlador/cliente (usuário normal)

```
sudo dpkg-reconfigure nslcd
```

Considere as seguintes configurações:

Configuração	Sugestão de Valor
LDAP server Uniform Resource Identifier:	ldap://**$CONTROLADOR_IP/**
Distinguished name of the search base:	**$CONTROLADOR_LDAP**
Name services to configure:	passwd, group and shadow
LDAP authentication to use:	Simple
LDAP database user	cn=admin,**$CONTROLADOR_LDAP**
LDAP root account password:	**$SENHA_SERVICOS**
Use StartTLS	No[4]

Estas configurações habilitam uma base de usuários unificada. Para realizar login com chaves ssh armazenadas no LDAP é necessário criar um script que consulte elas e informe ao serviço do sshd. Primeiramente copiando a senha para um arquivo local:

Comando no terminal do controlador/cliente (usuário normal)

```
sudo cp $ARQUIVO_SENHA /etc/infra_access
sudo chown root:root /etc/infra_access
sudo chmod 700 /etc/infra_access
```

Agora, definindo o script `</usr/bin/authp>` com o script de exemplo em `<scripts/authp>`. Modificando as variáveis corretamente e a permissão do script.

Comando no terminal do controlador/cliente (usuário normal)

```
sudo cp $SCRIPTPATH/authp /usr/bin/authp
sudo -E sed -i "s/CONTROLADOR_IP/${CONTROLADOR_IP}/g" /usr/bin/authp
sudo -E sed -i "s/CONTROLADOR_LDAP/${CONTROLADOR_LDAP}/g" /usr/bin/
   → authp
sudo chown root:root /usr/bin/authp
sudo chmod 700 /usr/bin/authp
```

Modificar para o sshd consultar o script, adicionar o arquivo `<scripts/confs/custom_sshd.conf>` em `</etc/ssh/sshd_config.d>`. Consulte este arquivo para verificar outras configurações do ssh.

[4]Para simplificação do tutorial apenas, em instalações complexas deve-se considerar sua configuração.

Comando no terminal do controlador/cliente (usuário normal)

```
sudo cp $SCRIPTPATH/confs/custom_sshd.conf /etc/ssh/sshd_config.d/
sudo systemctl restart sshd
```

Uma configuração no </etc/pam.d/su> para permitir que somente o usuário base possa utiliza o comando su.

Comando no terminal do controlador/cliente (usuário normal)

```
sudo -E sed -i  "s/# auth       required   pam_wheel.so/auth
   → required   pam_wheel.so group=$ADMIN_USER/" /etc/pam.d/su
```

3.5.7. Checkpoint de progresso

Foram configurados a gestão unificada de usuários. É importante um reboot do sistema:

Comando no terminal do controlador/cliente (usuário normal)

```
sudo reboot
```

3.6. Sistema de arquivos distribuído

Esta seção explica a instalação do controlador NFS.

3.6.1. Controlador: Gerenciador NFS

Para instalar o NFS:

```
sudo apt-get install -y nfs-kernel-server
```

Para compartilhar um diretório é necessário declará-lo para cada máquina cliente no arquivo </etc/exports>. Um exemplo se encontra em seguida. Uma linha define uma exportação, onde o primeiro item, /home, é o diretório local para compartilhar, REDE_INTERNA é o IP ou hostname da máquina que faz o compartilhamento, neste caso sendo permitido o compartilhando para todas as máquinas desta rede. A opção rw é para permitir escrita e leitura; sync para responder a requisições somente quando as alterações forem confirmadas em disco; no_root_squash para manter o UID de root nas requisições; no_subtree_check para desabilitar o check na subtree, opção recomendada para /home e para compartilhamentos que iniciem na raiz do volume. Outras opções estão descritas no manual do exports[5].

Adição em /etc/exports

```
/home       $REDE_INTERNA(rw,sync,no_root_squash,no_subtree_check)
```

Após cara alteração é necessário reiniciar o gerenciador do NFS:

Comando no terminal do controlador (usuário normal)

```
sudo systemctl restart nfs-kernel-server
```

[5]<https://linux.die.net/man/5/exports>

3.6.2. Cliente: Configurando o Cliente NFS

Primeiramente no cliente é necessário instalar o seguinte pacote:

Comando no terminal do Cliente (usuário normal)

```
sudo apt-get install nfs-common
```

Para adicionar o diretório no cliente é necessário adicionar em `/etc/fstab` a entrada do volume. Por exemplo, a seguinte linha, onde CONTROLADOR_DNS é o IP ou hostname do gerenciador, `:/home` é o caminho do gerenciador a ser compartilhado (o mesmo definido em `/etc/exports`) e o segundo `/home` é o caminho de montagem no cliente.

Adição em `/etc/fstab`

```
CONTROLADOR_DNS:/home       /home      nfs auto,nofail,noatime 0 0
```

Para validar a configuração sem reiniciar a máquina pode-se utilizar:

Comando no terminal do Cliente (usuário normal)

```
sudo mount -a
```

3.7. Compartilhamento de Recursos

Esta Seção descreve a instalação do gerenciador de trabalhos SLURM no controlador.

3.7.1. Controlador: Instalação do SLURM

Instalação de dependências no controlador:

Comando no terminal do controlador (usuário normal)

```
sudo apt install -y build-essential pkg-config libpam0g-dev \
     libmunge-dev munge mariadb-server libopenmpi-dev libmariadb-dev \
     libnuma-dev libjson-c-dev libyaml-dev hwloc libhwloc-dev \
     libcurl4-openssl-dev libdbus-1-dev liblz4-dev \
     libhttp-parser-dev libreadline-dev
```

O Slurm utiliza o munge para autenticações. Para isso, é necessário criar uma chave compartilhada entre todas as máquinas. A geração é feita somente uma vez no gerenciador e a chave será copiada para os clientes:

Comando no terminal do controlador (usuário normal)

```
sudo sh -c 'dd if=/dev/urandom bs=1 count=1024 > /etc/munge/munge.key'
sudo chown munge:munge /etc/munge/munge.key
sudo chmod 600 /etc/munge/munge.key
sudo cp /etc/munge/munge.key /home/munge.key
sudo chown root:root /home/munge.key
sudo chmod 600 /home/munge.key
```

Iniciando o serviço do munge:

Comando no terminal do controlador (usuário normal)

```
sudo systemctl restart munge
```

Baixando a última versão do Slurm e descompactando:

Comando no terminal do controlador (usuário normal)

```
wget https://download.schedmd.com/slurm/slurm-23.02.0.tar.bz2
mkdir slurm
tar --bzip -x -f slurm*tar.bz2 -C ./slurm --strip-components=1
```

Realizando a compilação e instalação:

Comando no terminal do controlador (usuário normal)

```
cd slurm
./configure
make -j
sudo make install
```

É necessário **criar um usuário slurm no LDAP** sem chave ssh para este usuário ter o mesmo UID em todas as máquinas. Consulte o procedimento na Seção 3.5.4.

Para configurar o controlador do Slurm, o primeiro passo é configurar o banco de dados do mesmo. Crie o arquivo `</usr/local/etc/slurmdbd.conf>` com o seguinte conteúdo[6], modificando SENHA para uma senha utilizada em seguida no gerenciador do banco de dados.

Arquivo `/usr/local/etc/slurmdbd.conf`

```
AuthType=auth/munge
DbdHost=localhost
DbdPort=6819
LogFile=/usr/local/etc/dbd.log
SlurmUser=slurm
StorageHost=localhost
StorageLoc=slurm_db
StoragePass=SENHA
StoragePort=3306
StorageType=accounting_storage/mysql
StorageUser=slurm
DebugLevel=info
```

É necessário configurar as permissões do arquivo:

Comando no terminal do controlador (usuário normal)

```
sudo chown slurm:root /usr/local/etc/
sudo chown slurm:root /usr/local/etc/slurmdbd.conf
sudo chmod 600 /usr/local/etc/slurmdbd.conf
```

Agora é necessário criar o usuário Slurm no gerenciador do banco, o banco de dados e dar as permissões corretas no Mariadb (*fork* do MySQL). Abra a interface de gerenciamento:

[6]Explicação de configurações extras disponível em <https://slurm.schedmd.com/slurmdbd.conf.html>

Comando no terminal do controlador (usuário normal)

```
sudo mysql -u root
```

E execute os seguintes comandos SQL alterando SENHA para a mesma senha acima:

Comando no terminal do MYSQL no controlador

```
CREATE DATABASE slurm_db;
CREATE USER 'slurm'@localhost IDENTIFIED BY 'SENHA';
GRANT ALL PRIVILEGES ON slurm_db.* TO 'slurm'@localhost;
```

Ainda é necessário alterar algumas variáveis no banco, alterando o arquivo `</etc/mysql/conf.d/mysql.cnf>` e reiniciando o serviço.

Comando no terminal do controlador (usuário normal)

```
echo "[mysqld]
innodb_buffer_pool_size=4096M
innodb_log_file_size=64M
innodb_lock_wait_timeout=900" | sudo tee -a /etc/mysql/conf.d/mysql.cnf
sudo systemctl restart mariadb.service
```

Copiando o serviço do systemd e ligando-o:

Comando no terminal do controlador (usuário normal)

```
sudo cp etc/slurmdbd.service /etc/systemd/system/
sudo systemctl start slurmdbd
sudo systemctl enable slurmdbd
sudo systemctl status slurmdbd
```

O próximo passo é a configuração do gerenciador do Slurm, pelo arquivo `</home/slurm.conf>` que terá um link simbólico com `</usr/local/etc/slurm.conf>`. Um exemplo de configuração básica se encontra em seguida. Algumas explicações de variáveis: `SlurmctldHost` e `ClusterName` são o hostname, ip do controlador e o nome da infraestrutura. As configurações de `Slurmctld` e `Slurmd` são para a execução dos *daemons* dos serviços no controlador e nos clientes. Para a contabilidade de usuários, as variáveis de `AccountingStorage` controlam tanto as permissões de utilização quanto que dados armazenar. Mais ao final do arquivo, existe a entrada `NodeName`, com uma linha descrevendo um nó de computação cliente. Os nós de computação são organizados em partições, explicitamente configurada nas linhas com `PartitionName`. As configurações completas estão disponíveis na documentação do Slurm[7].

Arquivo `/home/slurm.conf`

```
SlurmctldHost=CONTROLADOR_DNS(CONTROLADOR_IP)
ClusterName=NOME_INFRA
MaxJobCount=10000
SlurmUser=slurm
```

[7]<https://slurm.schedmd.com/slurm.conf.html>

```
AuthType=auth/munge
CryptoType=crypto/munge

ProctrackType=proctrack/cgroup

SlurmctldPidFile=/run/slurmctld.pid
SlurmctldPort=6817
SlurmctldDebug=info
SlurmctldLogFile=/usr/local/etc/log.txt
StateSaveLocation=/var/spool/slurm.state

SlurmdPidFile=/run/slurmd.pid
SlurmdPort=6818
SlurmdDebug=info
SlurmdLogFile=/usr/local/etc/node_log.txt
SlurmdSpoolDir=/var/spool/slurmd

TaskPlugin=task/cgroup
TaskPluginParam=None
TrackWCKey=no
SrunPortRange=60001-63000

AccountingStorageEnforce=associations,limits,qos,safe
AccountingStorageHost=localhost
AccountingStoragePort=6819
AccountingStorageType=accounting_storage/slurmdbd
AccountingStoreFlags=job_comment

MinJobAge=2

SelectType=select/cons_res
SelectTypeParameters=CR_CPU
SchedulerType=sched/backfill
SchedulerParameters=sbatch_wait_nodes,salloc_wait_nodes,bf_continue,
    → bf_hetjob_immediate

JobCompType=jobcomp/none

GresTypes=gpu
ReturnToService=1
PrologFlags=contain

#Nodes
NodeName=client1 NodeAddr=192.168.30.3 Sockets=1 CoresPerSocket=1
    → ThreadsPerCore=1 CPUs=1 MemSpecLimit=512 RealMemory=3900 State=
    → UNKNOWN

PartitionName=shared Nodes=client1 Default=YES MaxTime=24:00:00
    → DefaultTime=10:00 State=UP
```

Ainda é necessário definir uma configuração de como o Linux `cgroups` deve ser comportar em cada nó. Definindo o arquivo `</home/cgroup.conf>`.

Arquivo `/home/cgroup.conf`

```
CgroupAutomount=yes
ConstrainCores=yes
ConstrainRAMSpace=yes
```

O próximo passo é mudar as permissões destes arquivos, criar o link simbólico (todas as máquinas vão compartilhar o mesmo arquivo via NFS), copiar o serviço do systemctl do slurmctld, e criar a pasta de spool do daemon do slurm.

Comando no terminal do controlador (usuário normal)

```
sudo chown slurm:root /home/slurm.conf
sudo chmod 644 /home/slurm.conf
sudo ln -s /home/slurm.conf /usr/local/etc/slurm.conf
sudo chown slurm:root /home/cgroup.conf
sudo chmod 644 /home/cgroup.conf
sudo ln -s /home/cgroup.conf /usr/local/etc/cgroup.conf
sudo cp etc/slurmctld.service /etc/systemd/system/
sudo mkdir /var/spool/slurm.state
sudo chown slurm /var/spool/slurm.state
```

Para garantir que o Slurm inicie apenas depois do LDAP e do NFS é necessário adicionar esta configuração no systemd. Criando arquivo `</etc/systemd/system/slurmctld.service.d/override.conf>` com o conteúdo:

Arquivo `/etc/systemd/system/slurmctld.service.d/override.conf`

```
[Unit]
After=network-online.target munge.service remote-fs.target nslcd.
   → service
```

Ou copiando o arquivo do repositório:

Comando no terminal do controlador (usuário normal)

```
sudo mkdir /etc/systemd/system/slurmctld.service.d/
sudo cp $SCRIPTPATH/confs/slurmctld_override.conf /etc/systemd/system/
   → slurmctld.service.d/override.conf
sudo systemctl daemon-reload
```

Após, é necessário inicializar o daemon do controlador do slurm:

Comando no terminal do controlador (usuário normal)

```
sudo systemctl start slurmctld
sudo systemctl enable slurmctld
sudo systemctl restart slurmctld
```

3.7.2. Controlador - Recomendado: Prólogo e Epílogo de trabalhos Slurm

Uma das configurações possíveis no Slurm é a definição de scripts que executaram antes ou depois das tarefas. Uma descrição completa está disponível na página no Slurm[8].

[8]<https://slurm.schedmd.com/prolog_epilog.html>

É recomendado fortemente adicionar as variáveis propostas no módulo PAM do Slurm[9] nos prólogos e epílogos referenciados. Outra possível utilização é a configuração dos nós para o governor "powersave"ou "performance", ou a configuração de algum pacote (como `docker`). A adição de scripts de prólogo e epilogo é feito no `<slurm.conf>` com a adição das seguintes linhas. Lembrando de incluir as permissões de execução.

Adição no arquivo `/home/slurm.conf`

```
Prolog=/usr/local/etc/prolog.sh
Epilog=/usr/local/etc/epilog.sh
```

3.7.3. Controlador - Recomendado: Prioridade dos Jobs

Um dos aspectos fundamentais do Slurm é o escalonamento de tarefas seguindo prioridades. Uma extensiva documentação está disponível numa página do Slurm dedicada[10]. Também incluímos no repositório o arquivo `<scripts/confs/priority.conf>` com os exemplos de configuração de prioridades adotados no PCAD. Estas configurações devem ser adicionadas no `<slurm.conf>`.

3.7.4. Controlador - Opcional: Sistema de e-mail para o Slurm

O Slurm pode alertar os usuários via e-mail em alguns eventos relacionados às tarefas. Para isto, é necessário configurar um script que recebe como segundo parâmetro o endereço de destino (do usuário) e como terceiro parâmetro a mensagem do e-mail. Um exemplo é o script abaixo, configurado em `</usr/bin/smail>` no controlador.

Arquivo `/usr/bin/smail`

```
#!/bin/bash
mail -s "$2" --append="FROM:NOME <infra@instruicao.com>" "$3"
```

É necessária a configuração do `MailProg` no `</home/slurm.conf>`.

3.8. Configurando um nó computacional (Cliente)

A instalação pode ser executada via o script `scripts/make_client.sh`. Atentando-se que este script é um exemplo tendo em vista que no PCAD empregamos Ansible para fazer a instalação obedecendo condições mais complexas. O repositório tem mais informações sobre este assunto. Primeiramente devemos seguir os passos da Seção 3.5.6 para o LDAP, e depois da Seção 3.6.2 para o NFS.

3.8.1. Configurando opções do /scratch

Uma das configurações do PCAD é ter um diretório local onde diretórios reservados para cada usuário são criados. Os usuários recebem a recomendação de usar estes diretórios locais para seus experimentos, evitando o uso do NFS por questões de desempenho. Este é um passo opcional. Para criação e configuração do `</scratch>`:

[9]<https://slurm.schedmd.com/pam_slurm_adopt.html>
[10]<https://slurm.schedmd.com/priority_multifactor.html>

Comando no terminal do Cliente (usuário normal)

```
sudo mkdir -p /scratch
sudo chmod 755 /scratch
sudo mkdir /scratch/$ADMIN_USER
sudo chown $ADMIN_USER:$ADMIN_USER /scratch/$ADMIN_USER
sudo chmod 700 /scratch/$ADMIN_USER
echo 'SCRATCH=/scratch/$(whoami)/' | sudo tee -a /etc/profile
echo 'export SCRATCH' | sudo tee -a /etc/profile
```

Criar o script </usr/bin/scratch.sh> que será executado pelo PAM toda vez que o usuário entrar no nó computacional. Este script criará a pasta do usuário no </scratch> se ela não existir. Uma versão se encontra no repositório em <scripts/scratch.sh>. Em seguida, configurar as permissões e a chamada no PAM, modificando o arquivo </etc/pam.d/common-session>:

Comando no terminal do Cliente (usuário normal)

```
sudo cp  $SCRIPTPATH/scratch.sh /usr/bin/scratch.sh
sudo chmod 755 /usr/bin/scratch.sh
echo "session required    pam_exec.so /usr/bin/scratch.sh" | \
     sudo tee -a /etc/pam.d/common-session
```

3.8.2. Instalação do Slurm

Caso a máquina tenha aceleradores, como GPUs NVIDIA, seus drivers devem ser instalados antes do Slurm, para que este possa usá-los (exemplo CUDA) em sua compilação. Instalação de dependências e procedimentos de instalação do Slurm no cliente:

Comando no terminal do Cliente (usuário normal)

```
sudo apt install -y build-essential pkg-config libpam0g-dev \
    libmunge-dev munge libopenmpi-dev libnuma-dev libjson-c-dev \
    libyaml-dev hwloc libhwloc-dev libcurl4-openssl-dev libdbus-1-dev \
    liblz4-dev libhttp-parser-dev libreadline-dev
cd /scratch/$ADMIN_USER/
wget https://download.schedmd.com/slurm/slurm-23.02.0.tar.bz2
mkdir slurm
tar --bzip -x -f slurm*tar.bz2 -C ./slurm --strip-components=1
cd slurm
./configure
make -j
sudo make install
sudo cp /home/munge.key /etc/munge/munge.key
sudo chown munge:munge /etc/munge/munge.key
sudo chmod 600 /etc/munge/munge.key
sudo systemctl restart munge
pushd contribs/pam_slurm_adopt/
make -j
sudo make install
popd
echo "/usr/local/lib/" | sudo tee -a /etc/ld.so.conf.d/slurm.conf
sudo ldconfig
```

É necessário realizar os links de configuração do slurm com os arquivos do NFS. Ainda é necessário criar o arquivo `</usr/local/etc/gres.conf>` para configurar aceleradores. Caso a máquina não tenha aceleradores pode-se utilizar o exemplo do repositório em `<scripts/confs/gres.conf>`. Caso contrário, verificar a documentação[11].

Comando no terminal do Cliente (usuário normal)

```
sudo ln -s /home/slurm.conf /usr/local/etc/slurm.conf
sudo ln -s /home/cgroup.conf /usr/local/etc/cgroup.conf
sudo cp $SCRIPTPATH/confs/gres.conf /usr/local/etc/gres.conf
sudo chown slurm:root /usr/local/etc/gres.conf
sudo chmod 644 /usr/local/etc/gres.conf
```

Finalmente é necessário copiar o serviço do slurmd, criar os diretórios de spool, e alterar o serviço para esperar o LDAP e o NFS. Assim pode-se ligar o serviço do slurm:

Comando no terminal do Cliente (usuário normal)

```
sudo cp etc/slurmd.service /etc/systemd/system/
sudo mkdir /var/spool/slurmd
sudo chown slurm /var/spool/slurmd
sudo mkdir /etc/systemd/system/slurmd.service.d/
sudo cp $SCRIPTPATH/confs/slurmd_override.conf /etc/systemd/system/
   → slurmd.service.d/override.conf
sudo systemctl daemon-reload
sudo systemctl start slurmd
sudo systemctl enable slurmd
```

3.8.3. Configurações de autenticação para o Cliente

Algumas configurações são necessárias no arquivo `</etc/pam.d/sshd>` para configurar o PAM quando utilizado ssh nos clientes. As modificações se resumem ao patch disponível no repositório em `<scripts/confs/diff_pam_sshd_perm>`. Primeiramente ele adiciona o módulo do slurm para controlar acessos somente com quem tem a máquina reservada, mas adiciona uma exceção para os usuários configurados no `</etc/security/access.conf>`. Depois, ele desabilita o motd para usuários comuns, e a checagem de mail para todos. Para aplicar o patch e adicionar a exceção:

Comando no terminal do Cliente (usuário normal)

```
sudo patch /etc/pam.d/sshd $SCRIPTPATH/confs/diff_pam_sshd_perm
sudo sed -i "s/ADMIN_USER/${ADMIN_USER}/g" /etc/pam.d/sshd
echo -e "+ : $ADMIN_USER : ALL\n- : ALL : ALL" | sudo tee -a /etc/
   → security/access.conf
sudo systemctl restart sshd
```

3.8.4. Aplicação de teste

Para o usuário utilizar a infraestrutura é necessário associá-lo ao um grupo. Pode-se criar um grupo geral com o nome da infraestrutura e adicionar o usuário padrão. O Comando `add user` deverá ser realiza para todos os novos usuários.

[11]<https://slurm.schedmd.com/gres.conf.html>

Comando no terminal do controlador (usuário normal)

```
sudo sacctmgr -i create account $NOME_INFRA
sudo sacctmgr -i add user $ADMIN_USER account=$NOME_INFRA
```

Finalmente, pode-se submeter a primeira tarefa de teste:

Arquivo `job_teste.sbatch`

```
#!/bin/bash
#SBATCH -p shared
#SBATCH -o job_%j.out

echo "Estou executando em:"$HOSTNAME
sleep 60
```

Submeter o job e acompanhar a fila:

Comando no terminal do controlador (usuário normal)

```
sbatch $SCRIPTPATH/job_teste.sbatch
squeue
```

3.9. Outras configurações opcionais

Outras configurações são interessantes mas que este minicurso apenas aponta como próximos passos. Por exemplo, recomendamos (1) o uso de sistemas de arquivos como BTRFS com compressão nativa para a `</home>`, (2) a utilização de softwares de monitoramento de recursos como o Ganglia ou o Netdata, (3) a configuração de sincronia de relógios das máquinas com o NTP e (4) a utilização do ansible para uniformizar as configurações e softwares entre os nós.

3.10. Conclusão

No final deste minicurso, esperamos que seus recursos computacionais estejam aptos para serem compartilhados entre seus usuários. Diversas outras configurações são possíveis e talvez mais adequadas para cada caso.

Referências

Nesi et al. 2019 Nesi, L. L., Serpa, M. S., Schnorr, L. M., and Navaux, P. O. A. (2019). Hpc resources management infraestruture description and 10-month statistics. In *Proceedings do XVIII Workshop de Processamento Paralelo e Distribuído (WSPPD)*. GPPD, Porto Alegre.

Nesi et al. 2020 Nesi, L. L., Serpa, M. S., Schnorr, L. M., and Navaux, P. O. A. (2020). Advances in gppd–pcad management with 12-months analysis and perspectives. In *Proceedings do XVIII Workshop de Processamento Paralelo e Distribuído (WSPPD)*. GPPD, Porto Alegre.

Capítulo 4

Aprendizado de Máquina e Computação de Alto Desempenho

Manuel Binelo, Edson Luiz Padoin
Universidade Regional do Noroeste do Estado do Rio Grande do Sul

Resumo

A Inteligência Artificial é um campo que vem evoluindo radicalmente nos últimos anos. Ela tem se mostrado uma ferramenta com alto poder de capilaridade, sendo incorporada nas mais diversas áreas econômicas e sociais. Sua aplicação em áreas como automação, robótica, classificação de dados, reconhecimento de imagens e em sistemas especialistas já é bastante sólida, mas tem chamado a atenção sua aplicação em áreas relacionadas à subjetividade humana, como a criação de ilustrações, textos e música. A principal metodologia para a criação de agentes inteligentes é baseada no aprendizado máquina, que consiste em apresentar uma série de exemplos a um sistema computacional. Esses exemplos frequentemente compreendem grandes volumes de dados, e o processo de aprendizagem é um método iterativo de otimização que precisa ser repetido muitas vezes sobre esse conjunto de dados. Essas características fazem com que o desenvolvimento de modelos de inteligência artificial dependam de computação de alto desempenho. Modelos de inteligência artificial mais complexos, baseados em redes neurais artificiais com muitas camadas ocultas, empregam modelos de aprendizagem profunda (deep learning). A aprendizagem profunda é um problema de otimização de alta ordem, que exige uma capacidade de processamento ainda maior. Muitos modelos de inteligência artificial são baseados em redes neurais artificiais, que são modelos matemáticos/computacionais inspirados em neurônios biológicos. Assim como sua fonte de inspiração, sua arquitetura é massivamente paralela, o que torna possível a aplicação de diversas técnicas de computação paralela para acelerar o processo de aprendizagem. Neste minicurso serão abordados os fundamentos do aprendizado de máquina, suas implicações quanto à computação de alto desempenho, e as principais técnicas empregadas nesse contexto. Serão explorados modelos computacionais, frameworks mais utilizados, e diversos trabalhos científicos do estado da arte.

4.1. Introdução

A Inteligência Artificial (IA), e em especial o Aprendizado de Máquina (ML do Inglês *Machine Learning*), têm tido um grande crescimento, não apenas quanto aos resultados de pesquisa, mas quanto à sua aplicação na base estrutural de grandes sistemas de informação, como redes sociais e *e-commerce*, e em aplicações de interesse do usuário final, como nos sistemas geradores de conteúdo.

Como será mostrado, o ML depende de computação de alto desempenho (HPC do Inglês *High Performance Computing*), mas por outro lado, pode também contribuir para a evolução da HPC. O presente capítulo tem como objetivo explorar os conceitos relacionados a ML e HPC, as interações entre esses dois campos, e traçar algumas perspectivas.

4.2. Aprendizado de Máquina (ML)

A aprendizagem é um fenômeno complexo que envolve diferentes processos, como adquirir conhecimentos, desenvolver habilidades motoras e cognitivas por meio de instrução ou prática, organizar novos conhecimentos em representações eficazes e descobrir novos fatos e teorias por meio de observação e experimentação. Desde a era dos computadores, os pesquisadores têm buscado replicar essas capacidades em máquinas, um objetivo desafiador e fascinante na área de IA. O estudo e a modelagem computacional dos processos de aprendizagem são o foco do ML (Carbonell et al. 1983).

O ML é um subcampo da IA que envolve treinar algoritmos para realizar tarefas aprendendo padrões a partir de dados, em vez de usar programação explícita. Métodos de ML analisam automaticamente entradas e derivam respostas, ao contrário de técnicas convencionais de IA que usam critérios baseados em manuais. Exemplos de algoritmos de ML incluem k-vizinhos mais próximos, árvores de decisão e algoritmo Naive Bayes. Aprendizado de representação (RL do Iglês *Representative Learning*) é um subcampo de ML que se concentra em aprender representações de dados sem depender de pipelines de pré-processamento. Redes neurais artificiais (RNAs) são um exemplo comum de RL, e o aprendizado profundo (DL do Inglês *deep learning*) se refere a RNAs com muitas camadas, também conhecidas como redes neurais profundas (RNAPs) (Wataya et al. 2020).

De cordo com (Wang et al. 2020), os principais avanços em DL estão relacionados com quatro principais categorias de sistemas. A primeira categoria são RNAPs, que são extensões de redes neurais padrão com múltiplas camadas ocultas, permitindo representações mais complexas dos dados de entrada. As Resed Neurais Convolucionais (RNCs), inspiradas no córtex visual de animais, consistem em camadas de convolução, *pooling* e camadas completamente conectadas. As camadas de convolução e *pooling* reduzem a complexidade e diminuem o tamanho dos dados de entrada, enquanto as camadas completamente conectadas aprendem representações abstratas. Modelos baseados em RNCs têm alcançado resultados impressionantes no processamento de imagens e visão computacional.

A terceira categoria são as redes neurais recorrentes (RNRs), que são algoritmos de aprendizado profundo capazes de mapear dados de entrada sequenciais para sua saída. As RNRs possuem autoconexões entre os nós em cada camada, permitindo que memorizem informações ao longo do tempo a partir de uma sequência de dados. Apesar de

desafios quanto à sua aplicação, especialmente em relações a modelos de memória curta e longa, os modelos baseados em RNNs têm sido amplamente utilizados em problemas de aprendizado sequencial.

A quarta categoria são os modelos generativos, que têm como objetivo gerar novas amostras com variações através do aprendizado da distribuição das amostras de treinamento. Os autoencoders variacionais (VAE) e as redes adversariais generativas (GAN) são dois membros proeminentes dos modelos generativos. Modelos de aprendizado profundo frequentemente requerem uma grande quantidade de amostras rotuladas para o treinamento, o que pode ser difícil e computacionalmente caro de obter em aplicações práticas. Modelos generativos podem ajudar a aliviar esse problema e podem ser usados em várias tarefas, como reconhecimento, aprendizado semi-supervisionado, aprendizado de características não supervisionado e *denoising*.

A partir do trabalho de (Vaswani et al. 2017), modelos transformadores generativos pré-treinados (GPT) ganharam muito impulso. Seu sucesso decorre principalmente do mecanismo de auto-atenção, que permite o treinamento semi-supervisionado com grandes volumes de dados não rotulados. A auto-atenção, é um mecanismo de atenção que relaciona diferentes posições de uma única sequência para calcular uma representação da sequência. A auto-atenção tem sido usada com sucesso em uma variedade de tarefas, incluindo compreensão de leitura, sumarização abstrativa, implicação textual e aprendizado de representações de frases independentes de tarefas.

Os modelos de ML discutidos baseiam-se essencialmente em grande redes neurais artificiais, que para seu treinamento dependem de grandes volumes de dados. As seções seguintes irão discutir a HPC no contexto de ML e os desafios e oportunidades que se apresentam.

4.3. Contribuições da Computação de Alto Desempenho para o Aprendizado de Máquina

Os métodos utilizados para ML, de forma geral, são métodos iterativos, em que a cada iteração são modificados os pesos das conexões neurais (ou parâmetros do modelo), buscando minimizar o erro do aprendizado. Em resumo, o ML é um problema de otimização. Entre os métodos mais utilizados estão aqueles baseados em descida por gradiente (Pang et al. 2020, Netrapalli 2019).

A cada iteração do método, todo o volume de dados usados para o treinamento é acessado e interfaceado com a RNA, portanto existem dois aspectos fundamentais do custo computacional relacionado a ML, o primeiro é em relação à memória, seu tamanho e tempo de acesso aos dados de treinamento; já o segundo aspecto é de processamento matemático, para a solução do problema de otimização. Os algoritmos de aprendizagem podem ser considerados em seus dois aspectos principais, um sequencial, que são as iterações, e outro paralelo, que são as atualizações dos valores dos parâmetros.

4.3.1. Paralelismo de modelo

Para otimizar redes neurais profundas, é necessário analisar os dados de forma pontual, com baixa latência de atualização, devido ao aumento do tamanho dos modelos e dos

dados de treinamento. Uma abordagem escalável é distribuir e paralelizar o processo de treinamento. O paralelismo de modelo é uma ideia que consiste em dividir o modelo entre diferentes unidades de processamento, permitindo treinar modelos muito grandes que não caberiam na memória de um único dispositivo. No entanto, a eficiência do paralelismo de modelo depende da arquitetura e da forma como o modelo é dividido. Redes neurais totalmente conectadas são extremamente difíceis de ter seu modelo paralelizado já que cada camada depende do resultado da camada anterior para computar seus parâmetros, já redes mais esparsas, como redes neurais convolucionais, têm maior facilidade de paralelização do modelo (Jäger et al. 2018).

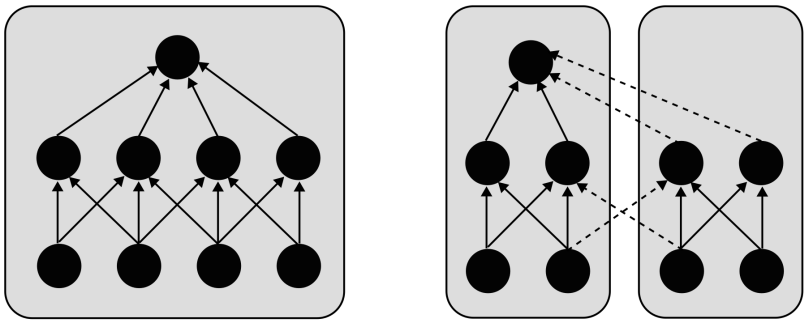


Figura 4.1. Esquema do paralelismo de modelo. Adaptado de (Jäger et al. 2018)

Como ilustrado na Figura 4.1, a rede neural é dividida verticalmente e cada unidade de processamento atualiza um conjunto de parâmetros. Para os nós da rede que possuem conexão com nós que estão em outra unidade de processamento, é necessário que o parâmetro correspondente seja enviado, o que explica o fato de que quanto mais esparsa for a rede, melhor o paralelismo de modelo funciona.

4.3.2. Paralelismo de dados

Algoritmos de paralelismo de dados utilizam vários trabalhadores que processam um subconjunto do conjunto de dados completo para escalonar a computação. Ou seja, a rede neural é replicada em cada nó de processamento que executa os mesmos passos de treinamento com dados diferentes em paralelo. Para que essa abordagem funcione, é necessário que a rede neural caiba na memória, e há duas vantagens interessantes em relação ao paralelismo de modelo. Esse método é independente da arquitetura da rede, podendo ser aplicado com sucesso em redes totalmente conectadas, e tem a possibilidade de ocultar os custos de comunicação quando um modelo assíncrono é adotado (Jäger et al. 2018).

Como mostra a Figura 4.2, o paralelismo de dados funciona da seguinte forma: um ou mais nós, chamados de servidor de parâmetros (PS), são responsáveis por calcular as atualizações de parâmetros e, se solicitado, redistribuí-las. O passo de atualização pode ser feito basicamente de duas maneiras: síncrona e assíncrona, o que tem um alto impacto no desempenho do treinamento. Quando é utilizado um modelo de atualizações síncronas, o PS espera por todas as mensagens dos trabalhadores antes de calcular as atualizações de

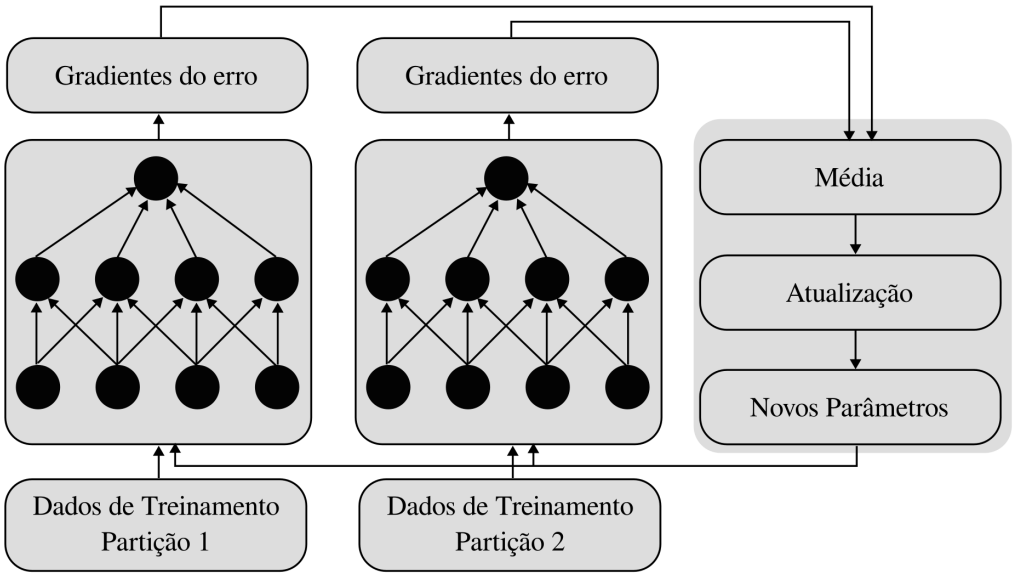


Figura 4.2. Esquema do paralelismo de dados. Adaptado de (Jäger et al. 2018)

parâmetros. Já para o modelo de atualizações assíncronas, para cada mensagem recebida, o PS calcula as atualizações de parâmetros (Jäger et al. 2018, Li et al. 2014).

4.3.3. Frameworks para ML

Diferentes frameworks estão disponíveis para computação de alto desempenho, e cada framework pode definir diferentes estratégias de paralelismo. Existem frameworks estáticos como é o caso do TensorFlow, que são baseados em duas fases. Na fase de cosntrução é definido o grafo de execução, e na fase de computação esse grafo é calculado. Possui como vantagem otimizar mais facilmente a execução, mas como desvantagem possui maior dificuldade de lidar com dados de entrada dinâmicos. Também quanto à estrutura adotada para o caso de modelos de paralelismo de dados existem diferentes abordagens. Como exemplo, o TensorFlow utiliza uma abordagem centralizada, enquanto o MXNet utiliza uma abordagem descentralizada (Jäger et al. 2018). Este capítulo não tem como objetivo trazer uma lista completa dos frameworks utilizados, mas abordará três dos mais utilizados.

4.3.4. TensorFlow

O TensorFlow é uma biblioteca de software flexível e escalável para cálculos numéricos usando grafos de fluxo de dados. Ele permite aos usuários programar e treinar eficientemente modelos de redes neurais e outras técnicas de aprendizado de máquina, e implantá-los em produção. O TensorFlow é escrito em C++ e CUDA (Compute Unified Device Architecture), uma plataforma de computação paralela criada pela NVIDIA. Ele possui APIs disponíveis em várias linguagens, sendo a API Python a mais completa e estável. Outras linguagens oficialmente suportadas incluem JavaScript, C++, Java, Go e Swift. Pacotes de terceiros estão disponíveis para mais linguagens, como C# e Ruby

(Pang et al. 2020, Kunas et al. 2023).

A arquitetura das redes neurais profundas tem tido um progresso significativo, e os framworks Keras e o TensorFlow tê tido bastante sucesso pois possuem diferentes modelos pré-treinados já incluídos em suas bibliotecas, incluindo VGG16, VGG1, ResNet50, Inception V3, Xception e MobileNet. As redes VGG e AlexNet seguem um padrão típico das redes convolucionais clássicas. Existem muitos fatores que explicam a revolução do aprendizado profundo, destacando-se a disponibilidade de conjuntos de dados enormes e de alta qualidade, o uso de computação paralela (GPU) que permite uma ativação eficiente para a retropropagação, novas arquiteturas, novas técnicas de regularização que permitem treinar redes mais extensas com menos risco de overshooting, otimizadores robustos e plataformas de software com grandes comunidades, como é o caso do TensorFlow (Sanchez et al. 2020).

Os modelos de redes neurais e outros modelos de aprendizado de máquina frequentemente envolvem multiplicação de matrizes. A arquitetura de GPU é ideal para esse tipo de computação, pois pode ser várias vezes mais rápida do que a CPU no treinamento de modelos de redes neurais. O código do TensorFlow é otimizado para ser executado em GPU, utilizando CUDA e cuDNN (biblioteca de redes neurais profundas baseada em CUDA), inclusive com o uso de várias GPUs em paralelo. A tf.distribute.Strategy é uma API do TensorFlow para distribuir o treinamento. Com mudanças mínimas no código, os pesquisadores podem distribuir modelos existentes com diferentes estratégias, como a tf.distribute.MirroredStrategy, que cria uma réplica em cada GPU e replica todas as variáveis em todas as réplicas. A tf.distribute.experimental.CentralStorageStrategy coloca todas as variáveis na CPU e replica as operações em todas as GPUs. A tf.distribute.experimental.ParameterServerStrategy designa algumas máquinas como trabalhadores e outras como servidores de parâmetros, replicando a computação em todos os trabalhadores, enquanto cada variável é colocada em um servidor de parâmetros (Pang et al. 2020).

4.3.5. Pytorch

O PyTorch é uma biblioteca em Python baseada no framework Torch, projetada para utilizar tanto GPUs quanto CPUs. Foi lançado em outubro de 2016 pelo grupo de pesquisa em inteligência artificial do Facebook. O PyTorch é amplamente utilizado em aplicações de processamento de linguagem natural e é conhecido por seu software Pyro, uma linguagem de programação probabilística desenvolvida pela Uber Research Labs. Em março de 2018, o PyTorch foi aprimorado com a incorporação do Caffe2, um framework de aprendizado profundo originário da UC Berkeley. Ao contrário de simples wrappers para outras linguagens populares, o PyTorch foi reescrito para ser rápido e ter uma aparência nativa, muitas vezes sendo usado como uma alternativa ao Numpy. O PyTorch oferece duas principais capacidades de alto nível: a computação de tensores com aceleração GPU poderosa e suporte a redes neurais profundas por meio do pacote torch.autograd (Flexa et al. 2019).

O PyTorch é um framework de aprendizado profundo que é amplamente recomendado devido à sua facilidade de uso, extensibilidade, desenvolvimento e depuração. Sua natureza Pythonica o torna popular tanto na comunidade de engenharia de software quanto entre pesquisadores e desenvolvedores. O PyTorch também é valorizado por sua capacidade de colocar modelos em produção, com um runtime C++ de alto desempenho.

É uma escolha dominante em conferências de pesquisa de aprendizado profundo. Em resumo, o PyTorch é uma plataforma avançada e fácil de usar para resolver problemas de aprendizado profundo, permitindo experimentação, depuração e implantação de forma eficiente (Ketkar et al. 2021).

4.3.6. Scikit-learn

O Scikit-learn é um pacote de aprendizado de máquina de código aberto e abrangente para Python, que é amplamente utilizado na comunidade de ciência de dados. Uma das vantagens do Scikit-learn é sua ampla cobertura de métodos de aprendizado de máquina, o que o torna uma escolha versátil para uma variedade de tarefas de aprendizado de máquina. Além disso, o Scikit-learn é conhecido por suas implementações otimizadas para eficiência computacional, uma vez que muitos dos métodos de aprendizado de máquina são baseados em bibliotecas binárias compiladas em Fortran, C ou C++, o que melhora significativamente o desempenho computacional. O Scikit-learn impõe convenções padronizadas de entrada/saída de dados e possui um procedimento de ajuste de modelo fixo, o que facilita a transição entre diferentes métodos de aprendizado de máquina dentro do pacote. Isso permite que os usuários experimentem diferentes modelos e técnicas de forma eficiente e sem problemas de integração (Hao and Ho 2019).

4.3.7. Desenvolvimentos recentes

De acordo com (Gan et al. 2020), devido à flexibilidade de seus recursos de configuração em chip, as plataformas de computação reconfiguráveis, como as FPGAs, são capazes de obter tempos de solução e eficiência energética superiores em comparação aos processadores de propósito geral. Essas plataformas têm sido amplamente adotadas em diversas aplicações importantes, desde o processamento numérico tradicional até os sistemas emergentes de aprendizado profundo. Considerando que as FPGAs têm se mostrado promissoras para a computação de alto desempenho, os autores resumem e analisam os esforços recentes relacionados a FPGAs, incluindo as abordagens industriais mais recentes, as soluções reconfiguráveis de ponta e várias questões relevantes, como o uso dos recursos em chip e a produtividade no desenvolvimento.

O trabalho de (Pyzer-Knapp et al. 2022) aborda como as novas ferramentas possibilitam novas formas de trabalho na área da ciência dos materiais. Tradicionalmente, a descoberta de materiais envolve um trabalho manual, serial e intensivo em recursos humanos, porém, essa abordagem está sendo complementada por processos automatizados, paralelos e iterativos impulsionados pela Inteligência Artificial (IA), simulação, automação experimental e computação de alto desempenho. O artigo descreve como essas novas capacidades permitem acelerar e enriquecer cada estágio do ciclo de descoberta de materiais. Utilizando o exemplo do desenvolvimento de um novo fotoresistor quimicamente amplificado, os autores demonstram como o impacto dessas tecnologias é amplificado quando utilizadas em conjunto como fluxos de trabalho heterogêneos e mais eficientes.

O trabalho de (Liu et al. 2022) aborda o tema de aprendizado federado, uma abordagem eficiente para explorar recursos de dados e computação distribuídos em dispositivos de usuários finais, em regiões ou organizações diferentes, garantindo a conformidade com leis e regulamentos, bem como a segurança e privacidade dos dados. Os autores for-

necem uma revisão abrangente das pesquisas existentes em aprendizado federado. Eles propõem uma arquitetura funcional de sistemas de aprendizado federado e uma taxonomia de técnicas relacionadas. Além disso, eles explicam os sistemas de aprendizado federado em quatro aspectos: tipos diversos de paralelismo, algoritmos de agregação, comunicação de dados e segurança dos sistemas de aprendizado federado. Os autores também apresentam quatro sistemas federados amplamente utilizados com base na arquitetura funcional proposta. Por fim, eles resumem as limitações e propõem direções para futuras pesquisas nessa área.

O artigo de (Kim et al. 2022) discute a questão do deslocamento de dados entre os dispositivos de memória e os elementos de processamento como um gargalo nos sistemas de computação tradicionais de arquitetura von-Neumann, que possuem elementos de processamento e dispositivos de memória separados, o que impacta modelos de aprendizado de máquina. Para tratar desse problema, o artigo apresenta a abordagem de computação centrada em memória, conhecida como Processing-In-Memory (PIM), que consiste em integrar os dispositivos de memória com os elementos de processamento, permitindo que os cálculos sejam realizados no mesmo local sem mover dados. Isso tem o potencial de melhorar substancialmente a eficiência energética dos sistemas de computação centrados em memória, minimizando o movimento de dados. O artigo revisa os trabalhos de pesquisa mais recentes sobre PIM, considerando diferentes tipos de dispositivos de memória, como SRAM, DRAM e ReRAM. Ele apresenta uma visão geral dos designs de PIM em cada tipo de memória, abrangendo desde células de bits até circuitos e arquiteturas. Em seguida, discute os desafios e limitações da integração de PIM com a arquitetura de computação convencional, incluindo a necessidade de novos padrões de pilha de software e os desafios associados a essa integração.

O trabalho de (Wang et al. 2021) aborda os desafios de treinar modelos de ML de grande tamanho em ambientes de Internet das Coisas (IoT) que possuem recursos limitados. Apesar de o ML ser a base para diversas aplicações de inteligência artificial, modelos grandes exigem recursos computacionais significativos para terem um desempenho adequado. A necessidade de treiná-los em ambientes de IoT com recursos limitados implica em dificuldades para desenvolver e aplicar técnicas de IA no futuro. O artigo apresenta uma revisão abrangente dos avanços recentes na redução do custo computacional durante a etapa de treinamento, mantendo a acurácia do modelo comparável. O foco é em algoritmos de otimização que visam melhorar a taxa de convergência, em arquiteturas de aprendizado distribuído que exploram recursos de computação ubíquos e em aceleração de hardware computacional e otimização de comunicação para treinamento colaborativo entre várias entidades de aprendizado.

O artigo de (Wang et al. 2022) aborda o campo em rápido crescimento da aprendizagem de máquina aprimorada por técnicas quânticas, conhecido como *quantum-enhanced machine learning*.As limitações de hardware dos computadores clássicos e o aumento do tamanho dos conjuntos de dados têm levado a comunidade a explorar novas técnicas para aprimorar a aprendizagem de máquina. A aprendizagem de máquina aprimorada por técnicas quânticas refere-se a algoritmos quânticos implementados em computadores quânticos, que podem melhorar a velocidade computacional de tarefas de aprendizagem de máquina clássicas e prometer um aumento exponencial na velocidade de computação. Nos últimos anos, o desenvolvimento de tecnologias quânticas experimentais tem levado

a muitas demonstrações experimentais de aprendizagem de máquina aprimorada por técnicas quânticas em diversos sistemas físicos. O artigo revisa o progresso experimental recente nesse campo e discute as demonstrações experimentais realizadas nessas plataformas e os desafios em andamento associados à implementação de técnicas quânticas na aprendizagem de máquina.

4.4. Contribuições do Aprendizado para Máquina para a Computação de Alto Desempenho

É evidente a contribuição da HPC para o desenvolvimento da IA, em especial do campo de ML. No entanto, algo não tão evidente à primeira vista é a contribuição de ML para a HPC. Nessa seção, algumas dessas contribuições mais recentes serão evidenciadas.

Uma questão fundamental para HPC é o projeto de novos chips. Esse tipo projeto tem inúmeras etapas de alta complexidade, sendo o projeto da máscara de litografia uma das mais demoradas e complicadas. As máscaras fotolitográficas são geralmente finas camadas de metal cromado com padrões, sobre vidro, formando efetivamente uma metassuperfície. Descobrir qual o padrão de litografia que resultará no padrão correto de luz projetada no substrato do chip é um problema complexo de otimização. Esse problema tem o nome de Design Eletromagnético Inverso, mas também é chamado de Tecnologia de Litografia Inversa (ILT) em fotomáscaras (Cecil et al. 2022). Métodos tradicionais levam semanas para calcular a máscara de chip, mas a Nvidia anunciou um novo método baseado em ML que pode acelerar em quarenta vezes esse processo, o que contribuiria para um avanço mais rápido no desenvolvimento de novos chips fundamentais para HPC (IEEESpectrum 2023).

Outra contribuição importante do ML para a HPC é utilização de modelos que aprendem o comportamento de sistemas mais complexos e que exigem um grande volume de cálculo ou tempo de simulação, aumentando a capacidade computacional por meio da substituição do modelo computacional original por outro modelo análogo baseado em ML. Em (Binelo et al. 2021), um método baseado em uma RNA foi criado para fazer a estimação e adaptação de parâmetros de modelos elétricos de descarga de baterias. Inicialmente, um algoritmo genético foi criado para estimar os parâmetros de um modelo de descarga de bateria, mas como esse método exige a execução de um grande número de simulações do modelo, sua execução em um dispositivo móvel é inviável. Ao criar uma RNA que aprende o comportamento do estimador de parâmetros, esse processo passou a ter um custo computacional muito mais baixo, abrindo a possibilidade de ser embarcado em dispositivos móveis, tendo um tempo de resposta muito mais rápido.

Dinâmica dos fluídos é um problema numérico computacional que depende de um grande volume de processamento. O trabalho de (Wang et al. 2018) apresenta um método inovador de redução de modelo baseado em métodos de aprendizado profundo. A abordagem de aprendizado profundo tem a capacidade de aprender um sistema não linear com vários níveis de representação e prever dados. No trabalho, os dados de treinamento são obtidos a partir de soluções de modelos de alta fidelidade em níveis de tempo selecionados. A RNA de memória de curto e longo prazo é usada para construir um conjunto de hipersuperfícies representando o sistema dinâmico de fluidos reduzido. O desempenho do novo modelo de ordem reduzida foi avaliado usando dois exemplos numéricos: um giro

do oceano e um fluxo passado um cilindro. Esses resultados ilustram que o custo da CPU é reduzido por várias ordens de magnitude.

Outro problema interessante de HCP é a crescente demanda por gráficos de alta qualidade em jogos, que vem aumentando o custo computacional e apresentando desafios para o hardware de processamento gráfico. Para superar essa limitação, os cientistas da computação precisam desenvolver novas maneiras criativas de melhorar o desempenho do hardware de processamento gráfico. Uma das soluções de maior sucesso é o uso de técnicas de ML para super-resolução de vídeo, que podem permitir que jogos de vídeo tenham gráficos de alta qualidade sem aumentar tanto o custo computacional. Essas tecnologias emergentes têm o potencial de melhorar o desempenho e a satisfação dos consumidores de jogos e caminham para se tornar padrão na indústria de desenvolvimento de jogos. A técnica consiste na renderização da imagem do jogo em uma resolução baixa, e então uma RNA generativa que recebe essa imagem, gera outra em resolução maior, diminuindo consideravelmente o custo computacional e gerando uma imagem bastante próxima da que seria gerada pela renderização convencional (Watson 2020).

4.5. Conclusões e Perspectivas

Os sistemas de inteligência artificial, especialmente no campo do aprendizado de máquina, vêm se desenvolvendo muito nos últimos anos, não apenas em ambientes de pesquisa ou corporativos, mas chegando até os usuários finais com taxas de adoção inéditas. Esse avanço só é possível por causa da computação de alto desempenho, que permite que sistemas com um gigantesco número de parâmetros possam ser treinados em grande volumes de dados obtidos por meio da Internet.

Outro aspecto muito interessante desse desenvolvimento é a contribuição que o aprendizado de máquina tem dado para a computação de alto desempenho. A contribuição se manifesta em métodos mais inteligentes para gerenciar os recursos computacionais, sistemas inteligentes para projetar novos chips e sistemas de computação, e também sistemas baseados em aprendizado de máquina que resolvem problemas de muita exigência computacional por meio de modelos análogos. O que tem se demonstrado é um ciclo no qual a computação de alto desempenho auxilia no avanço da inteligência artificial, que por sua vez impulsiona o avanço da computação de alto desempenho.

A possibilidade de treinar sistemas cada vez maiores com base de dados cada vez maiores, faz com que capacidades emergentes se manifestem, mostrando habilidades não previstas para esses sistemas. O ritmo com que essa evolução tem acontecido impõe uma série de desafios, não apenas técnicos, mas especialmente sociais e econômicos, que precisam ser abordados pela sociedade de forma ampla. Essa discussão e ação só pode ter lugar em uma sociedade que compreende o que é a inteligência artificial e seus impactos.

Referências

Binelo et al. 2021 Binelo, M. F., Sausen, A. T., Sausen, P. S., Binelo, M. O., and de Campos, M. (2021). Multi-phase method of estimation and adaptation of parameters of electrical battery models. *International Journal of Energy Research*, 45(1):1023–1037.

Carbonell et al. 1983 Carbonell, J. G., Michalski, R. S., and Mitchell, T. M. (1983). An

overview of machine learning. *Machine learning*, pages 3–23.

Cecil et al. 2022 Cecil, T., Peng, D., Abrams, D., Osher, S. J., and Yablonovitch, E. (2022). Advances in inverse lithography. *ACS Photonics*.

Flexa et al. 2019 Flexa, C., Gomes, W., and Viademonte, S. (2019). An exploratory study on machine learning frameworks. In *Anais do XVIII Workshop em Desempenho de Sistemas Computacionais e de Comunicação*. SBC.

Gan et al. 2020 Gan, L., Yuan, M., Yang, J., Zhao, W., Luk, W., and Yang, G. (2020). High performance reconfigurable computing for numerical simulation and deep learning. *CCF Transactions on High Performance Computing*, 2:196–208.

Hao and Ho 2019 Hao, J. and Ho, T. K. (2019). Machine learning made easy: a review of scikit-learn package in python programming language. *Journal of Educational and Behavioral Statistics*, 44(3):348–361.

IEEESpectrum 2023 IEEESpectrum (2023). Nvidia speeds key chipmaking computation by 40x. https://spectrum.ieee.org/inverse-lithography.

Jäger et al. 2018 Jäger, S., Zorn, H.-P., Igel, S., and Zirpins, C. (2018). Parallelized training of deep nn: comparison of current concepts and frameworks. In *Proceedings of the Second Workshop on Distributed Infrastructures for Deep Learning*, pages 15–20.

Ketkar et al. 2021 Ketkar, N., Moolayil, J., Ketkar, N., and Moolayil, J. (2021). Recent advances in deep learning. *Deep Learning with Python: Learn Best Practices of Deep Learning Models with PyTorch*, pages 287–300.

Kim et al. 2022 Kim, D., Yu, C., Xie, S., Chen, Y., Kim, J.-Y., Kim, B., Kulkarni, J. P., and Kim, T. T.-H. (2022). An overview of processing-in-memory circuits for artificial intelligence and machine learning. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 12(2):338–353.

Kunas et al. 2023 Kunas, C. A., Padoin, E. L., and Navaux, P. O. A. (2023). Accelerating deep learning model training on cloud tensor processing unit. In *International Conference on Cloud Computing and Services Science (CLOSER)*, pages 1–8, Prague, Czech Republic. Springer.

Li et al. 2014 Li, M., Andersen, D. G., Park, J. W., Smola, A. J., Ahmed, A., Josifovski, V., Long, J., Shekita, E. J., and Su, B.-Y. (2014). Scaling distributed machine learning with the parameter server. In *11th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 14)*, pages 583–598.

Liu et al. 2022 Liu, J., Huang, J., Zhou, Y., Li, X., Ji, S., Xiong, H., and Dou, D. (2022). From distributed machine learning to federated learning: A survey. *Knowledge and Information Systems*, 64(4):885–917.

Netrapalli 2019 Netrapalli, P. (2019). Stochastic gradient descent and its variants in machine learning. *Journal of the Indian Institute of Science*, 99(2):201–213.

Pang et al. 2020 Pang, B., Nijkamp, E., and Wu, Y. N. (2020). Deep learning with tensorflow: A review. *Journal of Educational and Behavioral Statistics*, 45(2):227–248.

Pyzer-Knapp et al. 2022 Pyzer-Knapp, E. O., Pitera, J. W., Staar, P. W., Takeda, S., Laino, T., Sanders, D. P., Sexton, J., Smith, J. R., and Curioni, A. (2022). Accelerating materials discovery using artificial intelligence, high performance computing and robotics. *npj Computational Materials*, 8(1):84.

Sanchez et al. 2020 Sanchez, S., Romero, H., and Morales, A. (2020). A review: Comparison of performance metrics of pretrained models for object detection using the tensorflow framework. In *IOP Conference Series: Materials Science and Engineering*, volume 844, page 012024. IOP Publishing.

Vaswani et al. 2017 Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. (2017). Attention is all you need. *Advances in neural information processing systems*, 30.

Wang et al. 2021 Wang, H., Qu, Z., Zhou, Q., Zhang, H., Luo, B., Xu, W., Guo, S., and Li, R. (2021). A comprehensive survey on training acceleration for large machine learning models in iot. *IEEE Internet of Things Journal*, 9(2):939–963.

Wang et al. 2022 Wang, X., Lin, Z., Che, L., Chen, H., and Lu, D. (2022). Experimental quantum-enhanced machine learning in spin-based systems. *Advanced Quantum Technologies*, 5(8):2200005.

Wang et al. 2020 Wang, X., Zhao, Y., and Pourpanah, F. (2020). Recent advances in deep learning. *International Journal of Machine Learning and Cybernetics*, 11:747–750.

Wang et al. 2018 Wang, Z., Xiao, D., Fang, F., Govindan, R., Pain, C. C., and Guo, Y. (2018). Model identification of reduced order fluid dynamics systems using deep learning. *International Journal for Numerical Methods in Fluids*, 86(4):255–268.

Wataya et al. 2020 Wataya, T., Nakanishi, K., Suzuki, Y., Kido, S., and Tomiyama, N. (2020). Introduction to deep learning: minimum essence required to launch a research. *Japanese journal of radiology*, 38:907–921.

Watson 2020 Watson, A. (2020). Deep learning techniques for super-resolution in video games. *arXiv preprint arXiv:2012.09810*.

Capítulo 5

Explorando Técnicas de Compressão para Melhorar a Eficiência de Tratamento de Dados IoT

Alexandre Luis de Andrade, Rodrigo da Rosa Righi
Universidade do Vale do Rio dos Sinos

Resumo

Esse capítulo de livro descreve o papel, por vias esquecido, que a compressão de dados pode ter no âmbito de processamento de alto desempenho. Hoje, com a concretização do uso de Internet das Coisas e implantação onipresente de algoritmos de Inteligência Artificial, o tratamento de dados ganham muita ênfase para que consigamos desempenho. Isso porque ainda seguimos com a sentença que eles são manipulados numa ordem de grandeza temporal bem maior que processamentos feitos pela CPU. Diante disso, o capítulo de livro apresenta os cenários que podem usufruir de aspectos de compressão de dados, bem como discute diferentes técnicas, sub-tempos envolvidos e casos de uso. Acreditamos que compressão possa ser muito explorada nos novos cenários de transformação digital, podendo também ser protagonista para a modelagem de sistemas computacionais de alto desempenho.

5.1. Introdução

O processo de transformação digital tem impulsionado a utilização de tecnologias digitais nas mais diversas áreas. Em especial, a adoção de dispositivos e sensores conectados, referidos como Internet das Coisas, ou *Internet of Things* (IoT), vem trazendo novas oportunidades e promovendo o avanço da Industria 4.0 (I4.0), na medida em que possibilita redução de custos, maior eficiência e incremento de qualidade (Marinagi et al. 2023) (Aheleroff et al. 2020). Cabe também destacar o impacto que a COVID-19 trouxe na aceleração dos processos de transformação digital e no fomento da disrupção em vários setores, tais com impulso na adoção da telemedicina (Castiglione et al. 2021) e, em particular, o uso de dispositivos vestíveis, em inglês *wearables*.

Esse cenário, com as companhias investindo cada vez mais em processos de automação e infraestruturas escaláveis, leva a um aumento exponencial de sensores conectados. Além desse fator, a popularização dos dispositivos *wearables* também contribui

para geração de uma grande volumetria de dados. Segundo a International Data Corporation (IDC), em 2025 estima-se 41,6 bilhões de dispositivos IoT conectados, gerando mais de 79 Zettabytes (ZB) de dados (Lu et al. 2020). Contudo, ao lidar com tal volumetria nos deparamos com dois principais desafios, que são a transmissão e o armazenamento destes dados.

Quanto à transmissão, devemos considerar que os dispositivos IoT utilizam basicamente redes sem fio, via radio frequência, para conexão e transmissão de dados na rede, e caso a rede não esteja adequadamente dimensionada, pode resultar em perda de pacotes e necessidade de retransmissão. Para sistemas que são sensíveis ao tempo e à qualidade da informação, tais como *healthcare*, para monitoramento de sinais vitais, a transmissão dos dados representa um fator crítico para um diagnóstico preciso e eficiência no tempo de reação (Idrees and Khlief 2023).

Ao tratar do armazenamento desses dados, vale considerar um dos principais elementos da transformação digital que é a computação em nuvem. A computação em nuvem foi um fator chave para a aceleração da transformação pois trouxe mais agilidade, facilidade de alocação e escalabilidade, atendendo rapidamente às necessidades do negócio (Dwivedi et al. 2022). Nesse contexto, o armazenamento de dados nuvem constitui um grande avanço, devido aos seus vastos recursos de armazenamento e escalabilidade, oferecendo segurança de dados, facilidade de acesso e recuperação em caso de desastres. Contudo existe um custo financeiro associado que precisa ser considerado, demandando estratégias adequadas de armazenamento e recuperação da informação (Aggarwal and Lan 2020).

Desse modo, para impulsionar áreas como a nova *healthcare* e a I4.0, que fazem uso extensivo de sensores e geram grande volume de dados, é necessário encontrar métodos para mitigar tais fatores que podem degradar o desempenho e a qualidade do resultado final. Nesse sentido, surgem algumas abordagens como alternativas para lidar com esses problemas, tais como elasticidade computacional, balanceamento de carga, computação de alto desempenho, ou *High-Performance Computing* (HPC), agregação de mensagens, e compressão de dados que vem ganhando espaço. De fato, a técnicas de compressão de dados não constitui algo novo, contudo ressurge nesse novo contexto como alternativa eficiente e de baixa complexidade de implantação (Gia et al. 2019) (Khlief and Idrees 2022) (Das and Rahman 2021).

Para ilustrar esse contexto, na Figura 5.1 estão apresentados três cenários possíveis de transmissão e armazenamento de dados, sendo em (1) sem aplicação de técnicas de compressão, (2) com compressão de dados e (3) comprimindo e aplicando descompressão antes de armazenar. Nesses cenários estão destacados os tempos envolvidos nas diferentes etapas, sendo o tempo necessário para transmissão (T_t), para compressão (T_c) e o tempo para descompressão de dados (T_d). Com isso, os tempos totais envolvidos nos cenários apresentados são $T_1 = T_t$, $T_2 = T_c + T_t$, e $T_3 = T_c + T_t + T_d$. De modo que, se dimensionado adequadamente, o esperado é $T_3 > T_2 > T_1$.

Dado esse contexto, o objetivo desse trabalho é abordar diferentes técnicas de compressão de dados e de que modo podem contribuir na melhoria de desempenho no processos de compactação e restauração da informação, levando a uma maior eficiência na sua transmissão e armazenamento de dados. O estudo também explora cenários práticos

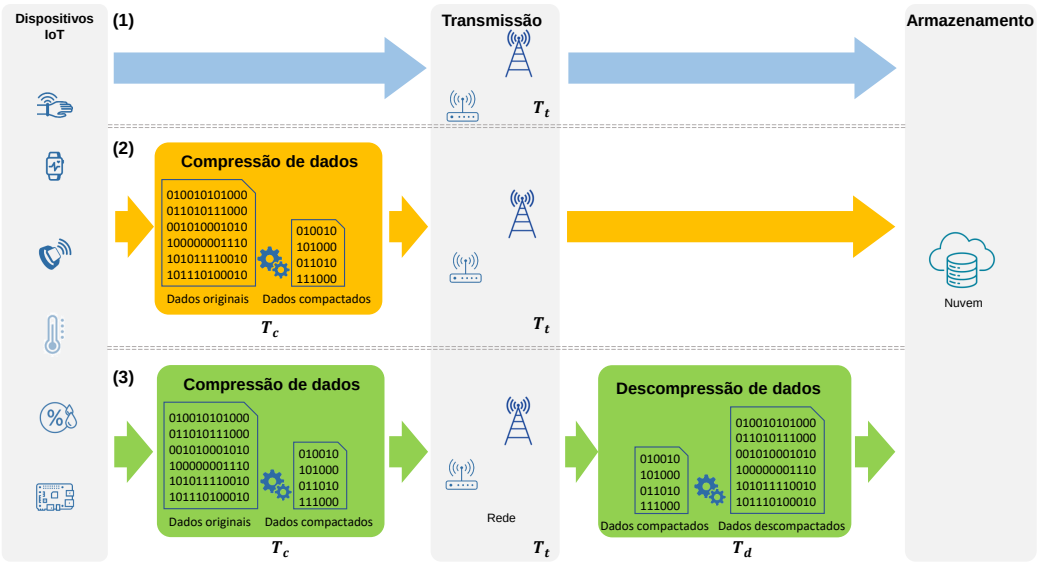


Figura 5.1. Cenários possíveis de transmissão e armazenamento da informação, onde (1) não aplica técnica de compressão; em (2) os dados são compactados e então transmitidos para serem armazenados e em (3) os dados são comprimidos e então transmitidos, contudo, são descompactados antes de serem armazenados.

de monitoramento de sistemas de healthcare e I4.0, nos quais as técnicas de compressão, mesmo que já extensamente estudadas, podem ser adaptadas e combinadas para levar a soluções inovadoras.

O restante do texto está organizado com a Seção 5.2 que aborda o contexto atual de geração de dados e seus desafios, seguido pela Seção 5.3 que analisa o potencial de uso dos métodos de compressão. Na Seção 5.4 serão analisados os diferentes tipos de algoritmos e principais aplicações, enquanto que a Seção 5.5 discute a combinação de diferentes técnicas para atender demandas IoT relacionadas com edge, fog e cloud comouting. Na sequência, tem-se a Seção 5.6, a qual traz as principais métricas adotadas para avaliar desempenho e a qualidade dos algoritmos de compressão. Por fim, na Seção 5.7 será feita uma análise conclusiva do tema de compressão de dados no cenário atual de computação de alto desempenho e alta demanda de dados.

5.2. Demanda de dados IoT e desafios

A presença de dispositivos conectados vem crescendo em um ritmo exponencial nos último anos, cada vez menores e com custos acessíveis, estão revolucionando diversas áreas. A área da saúde, por exemplo, com pacientes assistidos por IoT podem ser supervisionados ininterruptamente através dispositivos vestíveis, permitindo assim que situações de risco sejam detectadas e tratadas em tempo (Fischer et al. 2020). Ainda na área da saúde, o advento do COVID-19 impulsionou o *ehealth* com uso extensivo de dispositivos e trouxe a regulamentação da telemedicina, com sistemas remotos de monitoração.

Do mesmo modo, a I4.0 apoiada na integração com dispositivos IoT em suas instalações e, somada à computação e análise em nuvem, está revolucionando a forma

como as empresas fabricam, melhoram e distribuem seus produtos. A operação de fabricação inteligente, atuando de maneira cruzada com tecnologias automatizadas de aquisição e simulação de dados em tempo real, permite a identificação instantânea de problemas físicos na produção, com agilidade de reação para ações corretivas e otimização do desempenho em todo o sistema de fabricação, conforme Figura5.2 (Elkaseer et al. 2018).

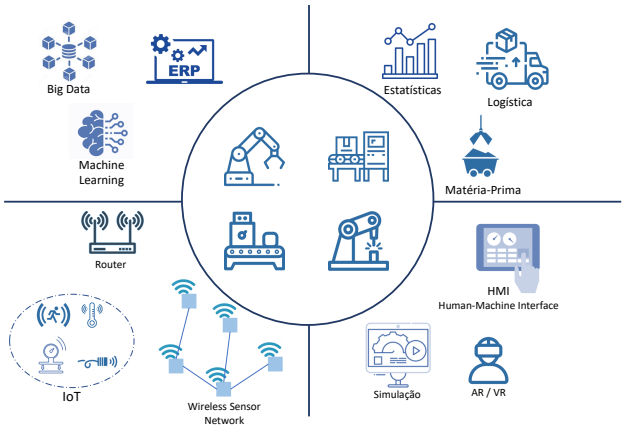


Figura 5.2. Framework de Intustria 4.0, representando a relação cruzada entre tecnologias, integrando diferentes áreas de manufatura, operação e logística.

Posto esse cenário, fica evidente a necessidade crescente de transferência de dados através da rede, seja para monitorar, executar comandos remotos ou mesmo disponibilizar informação em bases para análise e tomada de decisão. Entretanto, para que esses processos atendam às necessidades propostas, alguns fatores críticos de transmissão de dados precisam ser considerados, pois podem influenciar diretamente na qualidade, na eficiência e no desempenho do sistema. Os principais fatores mais citados na literatura são a i) ocupação da largura de banda, ii) taxa de transferência de dados, do inglês *throughput* e iii) latência de rede.

A largura de banda corresponde à quantidade máxima de dados que podem ser transmitidos em um determinado período de tempo em uma conexão de rede, sendo medida em bits por segundo. De modo que, se a transferência de dados excede a capacidade ocorre atraso na transmissão. O *Throughput* dos dados na rede, se refere à taxa em que os dados são transmitidos com êxito de um local para outro em um determinado período. E a latência de rede é o atraso na transmissão de dados da origem ao destino. Embora esses três elementos sejam diferentes, eles estão diretamente relacionados entre si: baixa largura de banda pode resultar em maior latência e menor *throughput*.

Como já citado, se esses fatores estiverem mal dimensionados, podem prejudicar a qualidade do serviço e, portanto, sua confiabilidade e limitar a escalabilidade do sistema. De fato, isso contrasta com o surgimento de novos aplicativos para sistemas e processos críticos, os quais vem se exigindo requisitos de confiabilidade e latência cada vez mais rígidos, conforme Figura 5.3. Na I4.0, em particular, o controle de robôs de alta precisão e, na área automotiva, a operação de veículos autônomos precisam oferecer confiabilidade superior à ordem de cinco noves (99,999%) e latência de milissegundos (Park et al. 2020). A automação na industria com base em comunicação *wireless* deve garantir confiabilidade

na ordem de sete noves (99,99999%) e latência abaixo de 1 ms, semelhante aos padrões de sistema de tempo real baseado em Ethernet (Berardinelli et al. 2018).

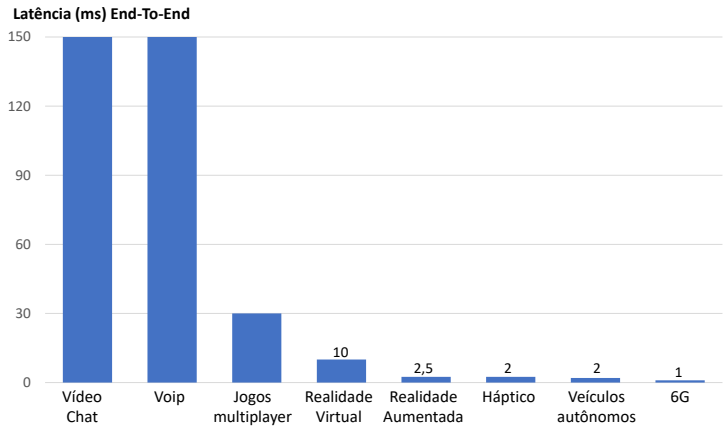


Figura 5.3. Latência esperada por tipo de aplicação (Panwar 2020).

Da mesma forma, em sistemas para processamento de mídia, a latência tem impacto direto sobre a qualidade e a escalabilidade (Abuin et al. 2022). Ou seja, melhorar a qualidade implica em aumento da resolução e aumento do número de quadros por segundo, tornando maior a quantidade de dados transmitidos. Por outro lado, aumentar a escalabilidade, significa levar a mídia para um número maior de usuários e aumentar o número de canais.

Nesse sentido, se procura estabelecer o equilíbrio para o triângulo formado por latência, qualidade e escalabilidade, enquanto se apresentam de novos algoritmos e novos modos de compressão de arquivos (Chen et al. 2020).

5.3. Potencial de compressão de dados

A compressão de dados não é uma ciência nova. Logicamente, existem novas técnicas e soluções aplicadas a tipos de arquivos modernos, mas em geral, as soluções existentes no mercado datam de mais de duas décadas. Apesar de não ser uma novidade, entendemos que compressão de dados ganha relevante importância no cenário atual, onde as áreas de Inteligência Artificial, Big Data, Cidades Inteligentes e IoT ganham destaque. Em particular, vale ressaltar que tais demandas geram muitos dados, os quais muitas vezes precisam ser armazenados, ou ainda transferidos entre uma máquina e outra pela rede de interconexão.

Diante disso, a ideia de compressão de dados é bem simples: diminuir o tamanho de um arquivo (o qual é composto por um fluxo de dados ou mesmo um arquivo tradicional no sistema de arquivos do sistema operacional). Tal diminuição fará, portanto, que se ocupe menos espaço em disco para armazenamento e se ocupe menos largura de banda para a sua transferência pela rede. Apesar de claros os benefícios da técnica, vale atentarmos para uma série de questões. Para explicá-las, vamos usar as notações abaixo:

- $tempo\_comunicacao(dados)$ - Função que torna o tempo para a transferência de

uma região de *dados*. Perceba que *dados* pode estar na sua forma original ou na compra comprimida. Por questões de simplicidade, não estamos levando em conta a tecnologia de rede empregada, tampouco os protocolos usados na comunicação.

- $tempo\_armazenamento(dados)$ - Função que retorna o tempo de armazenamento em disco para uma quantidade de dados denotada por *dados*. Por questão de simplicidade, não estamos considerando a tecnologia de disco rígido empregada.
- $temp\_compressaao(dados, tecnica)$ - Função que apresenta o tempo para aplicar o motor de compressão usando a estratégia *tecnica* sobre uma região de *dados*.
- $tempo\_descompressaao(dados, tecnica)$ - Procedimento que retorna o tempo para aplicar o motor de descompressão usando a estratégia *tecnica* sobre uma região de *dados*.

Com as colocações acima, é possível denotar ao menos dois cenários para explorar o potencial de compressão de dados: (i) cenário de somente transferência de dados; (ii) cenário de transferência e armazenamento. No cenário (i), a ideia é que a compressão seja pertinente para reduzir o tempo de comunicação. Ela é particularmente interessante em redes de larga cobertura ou aquelas que tendem a possuir alto nível de ruido, como algumas transferências por rádio frequência. A Inequação 5.3 apresenta o objetivo final do cenário (i). Nela, *dados* denotam o conjunto de dados original, enquanto $dados'$ representa o resultado após a aplicação de uma determinada *tecnica* de compressão. No cenário (i), temos que a semântica de transferência é exatamente a mesma, ou seja, partimos de dados que devem ser transferidos e chegamos até a transferência deles por completo para a máquina destino. Logicamente, para a compressão há as etapas de comprimir, transferir o material resultante e descomprimi-lo adequadamente.

$$tempo\_compressao(dados, tecnica) + tempo\_comunicacao(dados') + tempo\_descompressao(dados') < tempo\_comunicacao(dados) \quad (1)$$

No cenário (ii), o foco é armazenar os dados no destino na forma comprimida. Esse cenário aparece usualmente quando os dados precisam ser transferidos e agregados em um servidor para posterior análise, a qual não acontece de imediato após a transferência. A Inequação 2 apresenta a estratégia para que valha a pena o uso de compressão nesse cenário. Como é possível perceber, aplica-se a compressão e trabalha-se com a transferência e armazenamento dos dados comprimidos. Essa inequação não leva em consideração a redução do espaço em disco que pode ocorrer no armazenamento da forma comprimida, e sim somente o aspecto temporal do uso da técnica de compressão. Após o armazenamento dos dados na máquina destino, o uso deles pode ou não requisitar a descompressão. Se estivermos trabalhando com compressão do tipo *lossy*, os próprios dados transmitidos já estão no padrão que podem ser trabalhados e manipulados, não requisitando descompressão. Em contra-partida, a transferência e o armazenamento de arquivos que denotam documentos pode ocorrer na forma comprimida, mas o uso necessária

vai incorrer numa descompressão. Essa descompressão pode ser realizada somente em memória ou em espaço de armazenamento temporário, bem como ser realizada de forma assíncrona sem interferir nas atividades críticas de tempo para o usuário final.

$$tempo\_compressao(dados, tecnica) + tempo\_comunicacao(dados') + tempo\_armazenamento(dados') < tempo\_comunicacao(dados) + tempo\_armazenamento(dados) \quad (2)$$

Tanto para o cenário (i) quanto o cenário (ii), a noção de grão é importante para que seja possível entender as inequações apresentadas. Uma vez que estamos analisando o tempo de transferência pela rede de computadores, tempo de armazenamento em memória secundária e os tempos dos motores de compressão e descompressão, o formato e o tamanho dos dados é crucial. Por exemplo, na transferência de 100 bytes, possivelmente tenhamos uma situação onde o uso da técnica de compressão pode ser proibitivo, uma vez que a própria chamada de compressão e descompressão acarretariam em sobrecarga a qual não seria sobrepassada pelos ganhos de comunicação na transferência da forma comprimida. Entretanto, a manipulação de arquivos de 1 Mbyte pode ser profícua com a compressão, desde que o ganho com a transmissão exceda a sobrecarga das traduções de original para comprimido e vice-versa. Aqui vale ressaltar que as CPUs nas máquinas origem e destino desempenho um papel crucial, uma vez que contribuem para a velocidade das execuções dos algoritmos de compressão e descompressão de dados. Por fim, o tipo de dados de um arquivo é importante para denotar o potencial de compressão. Usando o algoritmo empregado no software WinZip, enquanto que em arquivos texto ASCII é possível atingir taxas de compressão de até 73%, bibliotecas DLL ou arqivos EXE possuem taxas de compressão de até 50% (em média)[1].

5.4. Tipos de compressão de dados

Os algoritmos de compressão de dados são tradicionalmente classificados em dois tipos: com perdas (ou não inversíveis), do inglês *lossy*, ou sem perdas (ou inversíveis), do inglês *lossless*. Embora o primeiro algoritmo geralmente forneça um resultado mais significativo no que se refere à taxa de compressão, esse algoritmo não possibilita reconstruir totalmente os dados originais. Por outro lado, os sistemas de compressão sem perdas reduzem o tamanho dos dados transmitidos ou armazenados, focando em reduzir a redundância de informações da origem, preservando a entropia da informação e permitindo a reconstrução integral dos dados originais. Comparativamente, a taxa de compressão dos algoritmos *lossless* são menores que as do algoritmo *lossy*.

Os algoritmos *lossless* comprimem o arquivo geralmente por codificação de repetição, basicamente substituindo uma palavra ou caractere com maior frequência por um código que utiliza menos espaço de armazenamento. Esses algoritmos comumente utilizam estratégias de codificação de redundância mínima ou método de dicionário. A técnica de redundância mínima consiste em representar os caracteres com maior número de ocorrências utilizando menos bits. Um exemplo de algoritmo muito conhecido que

[1]Detalhes em: https://kb.corel.com/en/125890

aplica essa técnica é Codificação de Huffman. Quanto ao método do dicionário, é aplicado um arquivo como sendo um dicionário de expressões com os seus valores correspondentes em bits, como exemplo de algoritmo que aplica essa técnica temos Lempel-Ziv-Welch (LZW). Outros algoritmos *lossless* amplamente conhecidos são ZIP e RAR, e tem maior rendimento, como citado na seção anterior, quando aplicados em arquivos de texto ASCII.

Os algoritmos *lossy*, por outro lado, elimina uma certa quantidade de dados que não é perceptível e, desse modo, não permite que a informação seja restaurada em sua forma original, contudo reduz significativamente o tamanho, conforme ilustrado no Figura 5.4. O esperado deste tipo de algoritmo é que permita uma análise similar entre os dados comprimidos e os originais sem eliminar informações relevantes e com a mais alta taxa de compressão possível. Ou seja, essa técnica é benéfica se a qualidade dos dados não for prioridade mas adequada para enviar ou armazenar os dados. Esse tipo de compactação de dados é usado principalmente para comprimir dados de multimídia como voz, vídeo e imagens.

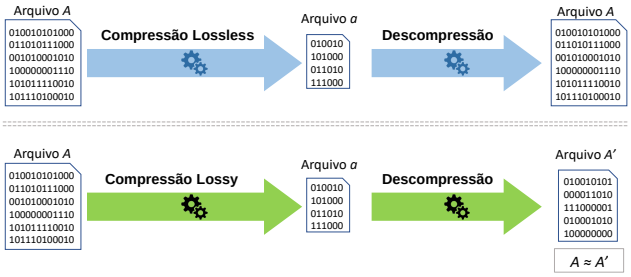


Figura 5.4. Dois principais métodos de compressão, Lossless e Lossy. O método Lossy proporciona uma maior taxa de compressão, contudo os dados restaurados a partir da descompressão não correspondem aos do arquivo original.

Nessa categoria do algoritmo de compressão com perdas, os métodos ”lineares por partes”, em inglês *piecewise linear*, são frequentemente utilizados na indústria, pois demandam baixo poder processamento e podem ser aplicados em tempo real no momento da aquisição dos dados (Roy and Nikolaidis 2022). Estes métodos consideram que o sinal segue uma linha reta enquanto os pontos amostrados estivem contidos dentro de uma tolerância especificada. Os dados considerados redundantes ou mesmo próximos de uma certa tendência linear são descartados e, desse modo, tais dados poderiam ser recuperados futuramente com baixo erro, utilizando técnicas de interpolação. Dentre os algoritmos *lossy* que utilizam esse método que são adotados na I4.0 se destacam o Boxcar/Backslope e o Swinging Door Trending (Khan et al. 2020).

A união de dois algoritmos, Boxcar e BackSlope, resulta numa técnica em que se armazena apenas os pontos que satisfazem as condições de ambos. Os parâmetros do algoritmo são a largura da janela do Boxcar e a largura da janela do Backslope. A partir da largura de janela informada, o algoritmo estabelece um limite de variação horizontal em relação ao último valor armazenado (BoxCar). Conforme apresentado na Figura 5.5a, a cada novo valor reportado são calculados limites de variação diagonal, utilizando a mesma largura de janela (BackSlope). Desse modo, assim que um valor violar duas áreas estabelecidas pelas linhas BoxCar e BackSlope, o valor deve ser armazenado.

O algoritmo Swinging Door Trending, apresentado na Figura 5.5b, utiliza como parâmetros o tempo máximo de compressão e desvio de compressão. Sendo que quanto maior o tempo máximo de compressão e o desvio de compressão, maior será a taxa de compressão. Uma área de cobertura á criada no formato de um paralelogramo com altura igual a duas vezes o desvio de compressão desde o último valor armazenado até o último valor recebido. Se algum dos pontos coletados entre o último valor armazenado e o valor atual estiver fora da área, todos os pontos são descartados, com exceção do penúltimo ponto recebido, que é armazenado. Ainda, sempre que o tempo entre o último valor armazenado for superior ao tempo máximo de compressão último valor deve ser armazenado.

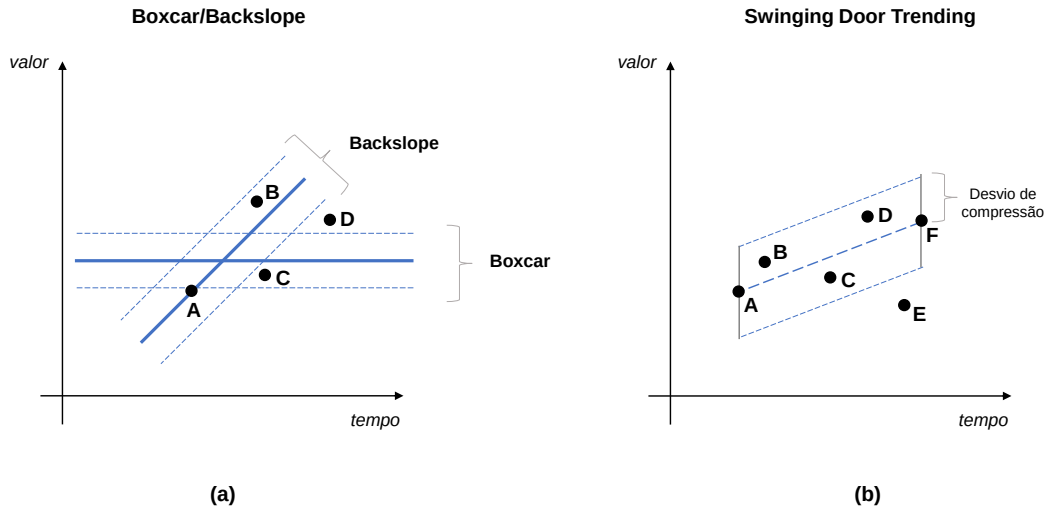


Figura 5.5. Em (a) temos o algoritmo de compressão Boxcar/Backslope, onde o ponto A é o último ponto armazenado, o ponto B viola apenas o Backslope, o ponto C viola apenas o Boxcar e o ponto D viola os dois critérios e portanto é armazenado. Em (b) está o algoritmo Swinging Door Trending, com uma área de cobertura entre o último valor armazenado (A) e o último valor recebido (F), nesse caso os valores B, C e D serão descartados, enquanto o ponto E será armazenado.

5.5. Combinando técnicas de compressão de dados

Diferentes cenários e necessidades de sistemas direcionam para a escolha de um algoritmo de compressão de dados com ou sem perda conforme apresentado acima. A escolha recai sobre a qualidade do dado que se deseja processar, e também no desempenho, custo e eficiência de armazenamento. Geralmente, a compactação com perdas é adequada para dados que não são sensíveis a pequenas alterações ou erros, como arquivos multimídia, conteúdo da Web ou dados analíticos. A compactação com perdas possibilita lidar melhor fatores intrínsecos à transferência de dados, como latência e ocupação de largura de banda, permite economizar mais espaço de armazenamento e acelerar a transferência. No entanto, a compactação com perdas também introduz distorção e degradação dos dados, precisão da análise ou a confiabilidade do backup. A compactação sem perdas é adequada para dados que requerem preservação e reprodução exatas, como arquivos de

texto, arquivos binários ou dados transacionais. A compactação sem perdas pode ajudá-lo a reter todas as informações e a qualidade dos dados e garantir a consistência e a exatidão dos dados. Contudo, a compactação sem perdas também consome mais espaço de armazenamento e recursos e retarda a compactação e descompactação de dados.

Dadas as considerações da Seção 5.4 sobre as técnicas de compressão *lossy* e *lossless* e, retomando os cenários apresentados na Figura 5.1, podemos ponderar sobre quais técnicas são mais aderentes para cada cenário. Para técnica de compressão *lossy*, os cenários 2 e 3 são mais aptos de aplicar, pois alguns tipos de arquivos não necessitam ser restaurados ao tamanho original após terem sido comprimidos. Por exemplo, compressão de imagem padrão JPEG, em que a imagem original tem seu tamanho reduzido e não precisa mais ser restaurado. O mesmo se aplica para arquivos de áudio, no qual o som original pode ser reduzido com baixo impacto ao custo de fidelidade. A compressão do tipo *lossy* elimina apenas sons menos audíveis e menos significativos, reduzindo o espaço necessário para que sejam armazenados ou transmitidos, sem necessidade de restaurar a qualidade original. Contudo para a técnica de compressão *lossless*, apenas o cenário 3 contempla, pois o método de compressão precisa ser revertido através da descompressão, recuperando o arquivo original.

Em alguns casos, pode ser apropriado utilizar os dois tipos de algoritmos de compressão, com e sem perda de dados, como o caso de sistemas de monitoria em *healthcare*, apresentado na Figura 5.6. Nesse contexto, para um diagnóstico confiável é preciso contornar problemas relativos à latência e ocupação da banda, contudo, é um desafio transferir dados coletados periodicamente de diferentes sinais vitais de vários pacientes simultaneamente. Neste cenário, com uma arquitetura em camadas, um algoritmo de compressão *lossy* pode ser aplicado na camada de borda, mais próxima dos sensores, descartando dados redundantes de acordo com cada tipo de sinal vital. Esse estágio de compactação pode ser adaptativo, estabelecendo tempos de aquisição de dados para alguns sinais vitais com pouca variação ao longo do tempo, tais como temperatura e pressão arterial. Por exemplo, a periodicidade da leitura dos sinais pode ser ajustável conforme o estado de saúde da pessoa monitorada. Uma pessoa saudável pode ter uma periodicidade de leitura maior que uma que necessite de acompanhamento com algum tipo de doença. Isso leva a um menor envio de dados mantendo, contudo, o padrão de detecção de falhas. O ajuste adequado desse processo é chave nesse tipo de sistema, um período de leitura muito pequeno leva ao envio de muitos dados, sobrecarregando a rede, gera *overhead*. Contudo, um período muito grande, gera problema de reatividade, no qual se toma uma ação tardia, degradando o resultado esperado.

Passado o estágio adaptativo, os dados dos sinais vitais são então transferidos para o próximo estágio, que aplica o segundo algoritmo, agora *lossless*, comprimindo os dados com métodos tradicionais para reduzir o tamanho total do arquivo antes de enviar para o nuvem, evitando a ocupação desnecessária da largura de banda. Em (Melchiades et al. 2021), os autores apresentam um método de compressão de dados voltado para a I4.0 denominado FastIoT. Nesse método, a partir de um grande volume de dados gerados em dispositivos IoT, a técnica de compressão reduz significativamente o tamanho original possibilitando o envio otimizado das informações, demandando baixa largura de banda de rede, para serem visualizadas de modo ajustado exatamente à região visual do dispositivo de destino. FastIoT funciona dividindo um intervalo de tempo de-

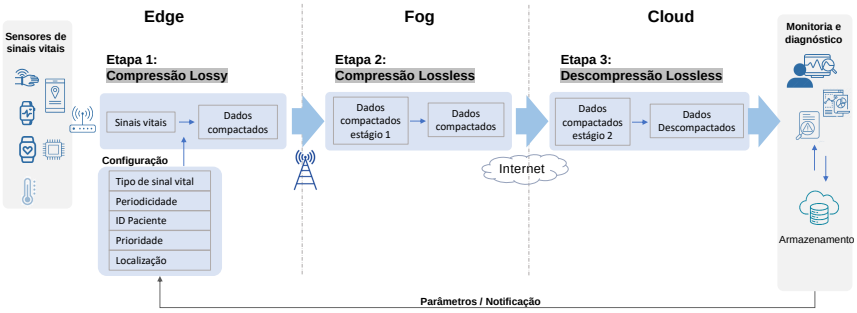


Figura 5.6. Sistema de monitoramento de saúde com duas etapas de compressão. Na camada de borda ocorre a Etapa 1 de compressão, do tipo Lossy. O algoritmo adaptável recebe parâmetros do módulo de configuração e define a cadência da aquisição de dados por tipo de sinal vital e paciente monitorado. Os dados compactados são enviados Etapa 2 de compressão, Lossless. Só então o pacote compactado é enviado pela internet para a nuvem, para ser restaurado, armazenado e utilizado no sistema de monitoração.

finido pelo usuário em períodos iguais, de modo que os valores máximo e mínimo são obtidos e exibidos em um gráfico de barras. Para cada intervalo, o método também retorna sua data inicial, data final e o número de pontos que ele representa. Assim, em termos de comportamento gráfico, o método entrega uma saída composta pelo conjunto de dados original consultado ao servidor. Quanto maior o número de intervalos, melhor a definição do gráfico exibido no lado do cliente.

Desse modo se, por exemplo, dispomos de 1.000 valores IoT armazenados e desejamos plotar em um dispositivo móvel, com uma resolução horizontal de até 500 pontos, não tem necessidade de enviar 1.000 pontos, pois só é possível plotar 500. O método FastIoT se adapta ao destino e entrega de modo eficiente o resultado esperado, ou seja, eficiente na medida em que a janela de transferência de dados se ajusta exatamente à região visual do dispositivo de destino, sem lentidão ou sobrecarga de tráfego de rede. Nestes dois casos analisados, FastIoT e monitoração em sistemas de *healthcare*, os métodos têm como característica principal um processo de compressão adaptável, ou seja, ajustam o nível de compressão através de um mecanismo que considera os principais componentes envolvidos no processo: tipo de dado, sistema de origem, meio de transmissão e sistema de destino. Desse modo, envia apenas o necessário, permitindo um uso eficiente da infraestrutura.

5.6. Métricas para análise de desempenho

A métrica mais básica para avaliar um algoritmo de compressão é a sua taxa de compressão, que consiste na proporção entre o tamanho dos dados compactados e o tamanho dos dados originais, de modo que quanto maior a taxa de compactação, maior o espaço poupado. Entretanto, uma alta taxa de compressão pode resultar alta perda de informações, portanto, a taxa de compactação deve ser considerada juntamente com outras métricas que medem a qualidade e a precisão dos dados compactados. Além disso, outros elementos de análise podem ser uma ferramenta bastante útil para avaliar eficiência e também possibilitar uma melhor comparação entre diferentes métodos. Para medir a

Taxa de Compressão (TC), temos a divisão do tamanho não compactado pelo tamanho compactado (veja Equação 3).

$$TC = \frac{amanho\ dados\ descompactados}{Tamanho\ dados\ compactados} \tag{3}$$

A métrica de Taxa de Erro de Bits, do inglês *Bit Error Rate* (BER), é a medida da integridade dos dados após o processo de descompactação. Este índice baixo indica uma alta fidelidade dos dados compactados, com o algoritmo de compactação preservando a integridade dos bits originais. Contudo, BER alto indica uma baixa precisão dos dados compactados, com o algoritmo de compactação gerando um resultado que contém distorções. Esse índice faz sentido quando aplicado aos métodos de compressão sem perdas, *lossless*, na medida em que os métodos de compressão com perdas, naturalmente os dados descompactados divergem dos originais. O cálculo de BER é dado pelo número de bits do arquivo descompactado com erro dividido pelo total de bits do arquivo original.

Outra métrica útil para avaliação de compressão com perdas é a distorção, que quantifica a qualidade do sinal reconstruído. A medida de distorção mais utilizada é a Diferença da Raiz Quadrática Média, do inglês *Root-Meal-Square Deviation* (RMSD), dado percentualmente pela Equação 4. Aqui, temos a distorção $d(x,\hat{x})$, entre os vetores x e $\hat{x} \in \mathbf{R}$, sendo x o sinal original e $\hat{x}$ o sinal restaurado após a descompressão. Além dessas medidas para avaliar o desempenho do algoritmo de compressão, a solução fi-a-fim precisa ser analisada considerando também fatores que podem afetar o resultado final do processo de compressão. Conforme explorado na Seção 5.3 os fatores tempo de compressão, descompressão e armazenamento influenciam diretamente no desempenho e qualidade final em um processo de compressão.

$$PRMSD = \sqrt{\sum_{n=1}^{N} \frac{(x[n] - \hat{x}[n])^2}{x[n]^2}} \times 100 \tag{4}$$

5.7. Conclusão

Esse capítulo de livro trouxe a tona o tema de compressão de dados, que pode ser explorado em sistemas computacionais que demandam alto desempenho. Mais precisamente, hoje dados são vistos como a grande moeda do século XXI. Tecnologias como sensores, atuadores, GPU (Graphical Processing Unit), bem como algoritmos de inteligência artificial fazem com que tenhamos uma curva exponencial positiva na geração de dados. Para tratá-los de forma efetiva, são necessárias estratégias que garantam a escalabilidade de sistemas de computação, oferecendo qualidades de serviço aceitáveis para usuários finais independente do número deles e da carga de trabalho em voga.

A compressão, portanto, atua como um meio de diminuir o tamanho dos arquivos que serão trafegados pela rede, podendo também fazer com que dados de tamanho menor sejam armazenados. Isso propicia uma melhor utilização da largura de banda da rede, principalmente no tocante a redes com alta flutuação como pode ser a Internet, bem como o armazenamento de uma quantidade maior de dados em memória secundária. Diante disso, esse capítulo de livro apresentou um formalismo matemático para modelar o uso de

compressão de dados, bem como diferentes técnicas e seu casamento para diferentes tipos de arquivos e demandas. Como mencionado ao longo do documento, compressão não é um assunto novo, mas ganha momento frente as direções de foco em que vivemos. Além disso, questões arquiteturais de memória cache, pipeline, TLP (Thread Level Parallelism), coprocessadores matemáticos parecem estar bem resolvidas e sedimentadas e as técnicas aqui apresentadas se mostram como uma alternativa para gerir sistemas escaláveis e com alta disponibilidade.

Agradecimentos

Os autores gostariam de agradecer aos seguintes órgãos de fomento: Fundação de Amparo a Pesquisa do Estado do Rio Grande do Sul (FAPERGS, processo 21/2551-0000118-6); Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq, processo 305263/2021-8).

Referências

Abuin et al. 2022 Abuin, A., Iradier, E., Fanari, L., Montalban, J., and Angueira, P. (2022). High efficiency wireless-noma solutions for industry 4.0. In *2022 IEEE 18th International Conference on Factory Communication Systems (WFCS)*, pages 1–8. IEEE.

Aggarwal and Lan 2020 Aggarwal, V. and Lan, T. (2020). Modeling and optimization of latency in erasure-coded storage systems. *arXiv preprint arXiv:2005.10855*.

Aheleroff et al. 2020 Aheleroff, S., Xu, X., Lu, Y., Aristizabal, M., Velásquez, J. P., Joa, B., and Valencia, Y. (2020). Iot-enabled smart appliances under industry 4.0: A case study. *Advanced engineering informatics*, 43:101043.

Berardinelli et al. 2018 Berardinelli, G., Mahmood, N. H., Rodriguez, I., and Mogensen, P. (2018). Beyond 5g wireless irt for industry 4.0: Design principles and spectrum aspects. In *2018 IEEE Globecom Workshops (GC Wkshps)*, pages 1–6. IEEE.

Castiglione et al. 2021 Castiglione, A., Umer, M., Sadiq, S., Obaidat, M. S., and Vijayakumar, P. (2021). The role of internet of things to control the outbreak of covid-19 pandemic. *IEEE Internet of Things Journal*, 8(21):16072–16082.

Chen et al. 2020 Chen, W.-Y., Chou, P.-Y., Wang, C.-Y., Hwang, R.-H., and Chen, W.-T. (2020). Live video streaming with joint user association and caching placement in mobile edge computing. In *2020 International Conference on Computing, Networking and Communications (ICNC)*, pages 796–801. IEEE.

Das and Rahman 2021 Das, S. K. and Rahman, M. Z. (2021). A compression technique for electronic health data through encoding. In *2021 International Conference on Electrical, Communication, and Computer Engineering (ICECCE)*, pages 1–6. IEEE.

Dwivedi et al. 2022 Dwivedi, S. K., Yadav, J., Ansar, S. A., Khan, M. W., Pandey, D., and Khan, R. A. (2022). A novel paradigm: Cloud-fog integrated iot approach. In *2022 3rd International Conference on Computation, Automation and Knowledge Management (ICCAKM)*, pages 1–5. IEEE.

Elkaseer et al. 2018 Elkaseer, A., Salama, M., Ali, H., and Scholz, S. (2018). Approaches to a practical implementation of industry 4.0. *Resource*, 3:5.

Fischer et al. 2020 Fischer, G. S., da Rosa Righi, R., de Oliveira Ramos, G., da Costa, C. A., and Rodrigues, J. J. (2020). Elhealth: Using internet of things and data prediction for elastic management of human resources in smart hospitals. *Engineering Applications of Artificial Intelligence*, 87:103285.

Gia et al. 2019 Gia, T. N., Qingqing, L., Queralta, J. P., Tenhunen, H., Zou, Z., and Westerlund, T. (2019). Lossless compression techniques in edge computing for mission-critical applications in the iot. In *2019 Twelfth International Conference on Mobile Computing and Ubiquitous Network (ICMU)*, pages 1–2. IEEE.

Idrees and Khlief 2023 Idrees, A. K. and Khlief, M. S. (2023). Efficient compression technique for reducing transmitted eeg data without loss in iomt networks based on fog computing. *The Journal of Supercomputing*, pages 1–26.

Khan et al. 2020 Khan, M. A., Pierre, J. W., Wold, J. I., Trudnowski, D. J., and Donnelly, M. K. (2020). Impacts of swinging door lossy compression of synchrophasor data. *International Journal of Electrical Power & Energy Systems*, 123:106182.

Khlief and Idrees 2022 Khlief, M. S. and Idrees, A. K. (2022). Efficient eeg data compression technique for internet of health things networks. In *2022 IEEE World Conference on Applied Intelligence and Computing (AIC)*, pages 403–409. IEEE.

Lu et al. 2020 Lu, T., Xia, W., Zou, X., and Xia, Q. (2020). Adaptively compressing iot data on the resource-constrained edge. In *HotEdge*.

Marinagi et al. 2023 Marinagi, C., Reklitis, P., Trivellas, P., and Sakas, D. (2023). The impact of industry 4.0 technologies on key performance indicators for a resilient supply chain 4.0. *Sustainability*, 15(6):5185.

Melchiades et al. 2021 Melchiades, M. B., Crovato, C. D. P., Nedel, E., Schreiber, L. V., and Righi, R. D. R. (2021). Fastiot: an efficient and very fast compression model for displaying a huge volume of iot data in web environments. *International Journal of Grid and Utility Computing*, 12(5-6):605–617.

Panwar 2020 Panwar, S. (2020). Breaking the millisecond barrier: Robots and self-driving cars will need completely reengineered networks. *IEEE Spectrum*, 57(11):44–49.

Park et al. 2020 Park, J., Samarakoon, S., Shiri, H., Abdel-Aziz, M. K., Nishio, T., Elgabli, A., and Bennis, M. (2020). Extreme urllc: Vision, challenges, and key enablers. *arXiv preprint arXiv:2001.09683*.

Roy and Nikolaidis 2022 Roy, S. K. and Nikolaidis, I. (2022). Limited size lossy compression for wsns. In *2022 IEEE 47th Conference on Local Computer Networks (LCN)*, pages 359–362. IEEE.