Universidade de São Paulo
Instituto de Ciências Matemáticas e de Computação
Departamento de Sistemas de Computação
Laboratório de Sistemas Distribuídos e Programação Concorrente

Notas de Aulas da Disciplina
SSC0903 – Computação de Alto Desempenho

Módulo 1 – Introdução

por Paulo Sérgio Lopes de Souza

São Carlos/SP – Brasil – 2020

Conteúdo

1 Introdução à Computação de Alto Desempenho

1.1 Motivação para a Computação de Alto Desempenho (CAD)

A motivação histórica para a CAD é a busca por execuções mais eficientes do ponto de vista de desempenho, uso de recursos e consumo energético, onde suas soluções envolvem aspectos de hardware e software. Alguns exemplos de hardware que tentam oferecer melhores desempenho são: paralelismo abaixo da arquitetura do conjunto de instruções (ISA), cache, aumento da frequência de processadores, aumento da vazão de redes de conexão, diminuição da latência de memória RAM, uso de mais processadores em paralelo, entre vários outros. Alguns exemplos de software que visam às melhorias de desempenho são: programação sequencial com uma menor complexidade para o problema, busca por estruturas de programação e de dados mais eficientes, programação paralela ou concorrente (Hauber & Rünger, 2010), (Pacheco, 2011).

A Computação Paralela é uma das ferramentas mais importantes para se buscar melhores desempenhos de aplicações e até mesmo viabilizar a execução de algumas outras que não poderiam ser executadas sequencialmente. A Computação de Alto Desempenho, embora represente um escopo maior, muitas vezes se confunde com a própria Computação Paralela, pois esta última visa justamente oferecer altos desempenhos à computação com o uso de hardware replicado e software paralelo.

Focando mais nos aspectos do software paralelo, percebe-se que o mesmo necessita de uma programação paralela, cuja origem remonta a criação dos sistemas operacionais multiprogramados. De fato, a programação paralela tem sua origem na programação concorrente, esta motivada pela necessidade de se desenvolver sistemas operacionais multiprogramados confiáveis (Hansen, 2001). Os primeiros sistemas multiprogramados eram desenvolvidos em linguagem de montagem sem a fundamentação teórica necessária. No final dos anos 60 esses sistemas operacionais tornaram-se imensos e pouco confiáveis, pois erros simples de programação provocavam a parada dos sistemas operacionais, e eram difíceis de serem encontrados. Testar tais sistemas era praticamente impossível, graças ao não determinismo dos eventos de comunicação e sincronização que ocorriam entre os processos em execução. Esse cenário gerou uma crise de software, sendo que a programação concorrente veio para resolvê-la, tornando-se uma das maiores revoluções na programação de computadores. Com o surgimento de novas estruturas de comunicação e sincronização, logo percebeu-se que o potencial das mesmas estava além dos sistemas operacionais, pois graças às abstrações de processos, memória e E/S, as estruturas de comunicação e sincronização poderiam ser aplicadas em qualquer forma de Computação Paralela (Hansen, 2001).

1.2 Objetivos da Computação Paralela

Permitir o desenvolvimento de soluções que indiquem explicitamente como que as diferentes porções da computação podem ser executadas paralelamente pelos recursos de hardware, como processadores, memória e demais dispositivos.

Através da programação concorrente espera-se:

- Obter ganhos de desempenho com mais transistores: multicore e manycore
- Reduzir o GAP entre memória e CPU: acessos concorrentes à memória

1.3 Por que construir soluções baseadas na Computação Paralela?

Melhores desempenhos podem vir do hardware, apesar dos seus limites físicos (frequência, calor, ...). No entanto, o desenvolvimento de software para máquinas paralelas é fundamental, pois por intermédio do software que podemos personalizar soluções eficientes para problemas específicos e computacionalmente difíceis de serem executados. Esses softwares podem ser desenvolvidos automaticamente por compiladores, apesar destas soluções apresentarem um desempenho limitado à versão sequencial do algoritmo e não apresentarem, necessariamente, bons resultados quando o algoritmo sequencial é paralelizado automaticamente. As dificuldades com dependências de dados, por exemplo, podem acarretam grandes perdas de desempenho.

1.4 Evolução da Computação Paralela

Além dos aspectos de software, há também toda uma evolução do hardware, onde máquinas monoprocessadas (ditas) baratas e de propósito geral formam computadores paralelos virtuais conectados por simples redes de conexão. Tais arquiteturas são conhecidas como clusters de computadores e permitiram o acesso da computação de alto desempenho às pessoas ou suas instituições. Computadores com múltiplos e/ou muitos núcleos povoam o mercado de computadores pessoais, e dispositivos móveis (notebooks, tablets e/ou celulares), caracterizando máquinas multiprocessadas. Diferentes sistemas computacionais embarcados interagem, permitindo que a computação distribuída alavanque a computação ubíqua entre nós. O uso de processadores heterogêneos é um fato comum na atualidade e isso requer o aprendizado do uso de novas técnicas e ferramentas de programação (Quinn, 2003).

1.5 O que Difere as Soluções Sequenciais das Paralelas?

Diferentemente dos sistemas computacionais sequenciais, o desenvolvimento das soluções deve considerar muito mais detalhes de software e hardware. Isso resulta em diferentes soluções para um mesmo problema a depender do hardware, sistema operacional e linguagem de programação usada. Os principais aspectos adicionais que deem ser considerados são (Grama et al., 2003) (Foster, 1994):

- Códigos, no mínimo, bidimensionais devido à interação;
- Identificação do paralelismo na aplicação;
- Maior impacto da plataforma computacional usada (arquitetura + software básico):
  - Tanto no projeto quanto no desempenho da aplicação;
  - Considerar o sistema computacional é essencial.
- Uso de novas ferramentas de software para o desenvolvimento das aplicações;

Uso de novas estruturas de programação;
Iniciação/finalização e comunicação/sincronização de processos;
Distribuição da carga de trabalho entre os processos;
Gerência dados compartilhados e locais (localidade espacial e temporal);
Sincronização dos diferentes estágios durante a execução;
Teste & depuração.

1.6 Exercício Prático

Ordenar as 52 cartas de um baralho sequencialmente e em paralelo, pelo número da carta (A, 2, 3, 4, 5, 6, 7, 8, 9, 10, J, Q e K), e também pelos naipes (ouros, espadas, copas e paus). As cartas devem ser exibidas ao final da ordenação nesta sequência: A-ouros, A-espadas, A-copas, A-paus, 2-ouros, 2-espadas, 2-copas, 2-paus, 3-ouros, 3-espadas, 3-copas, 3-paus, e assim por diante até K-paus. Cronometrar os tempos da execução sequencial e paralela. As escolhas dos algoritmos a serem seguidos pelos alunos são livres. Após as execuções, analisar desempenho de ambas e as opções de projeto escolhidas. Encontrar justificativas para os resultados obtidos

1.7 Conceitos Básicos

1.7.1 Computação Paralela

Uso do computador paralelo para reduzir o tempo necessário para resolver um problema computacional de forma paralela, usando software normalmente desenvolvido com a programação paralela. A programação sequencial também pode ser usada, sendo que neste caso diferentes programas são carregados para execução com seus respectivos dados e executam em processadores distintos simultaneamente (Quinn, 2003).

1.7.2 Computador Paralelo

Sistema Computacional com múltiplos-processadores que dá suporte à Computação Paralela. Podem ser organizados de diferentes formas. Algumas importantes categorias são: multicomputadores, multiprocessadores, processadores vetoriais, processadores heterogêneos. Podem ser referenciados na literatura como plataforma paralela, plataforma distribuída, entre outros (Quinn, 2003).

1.7.3 Programação Paralela ou Concorrente

Atividades de programar em uma linguagem de programação concorrente para determinar quando e como tarefas concorrentes podem ser executadas. A programação concorrente determina como tarefas interagem para solucionar um problema específico. Graças à abstração de processo, os termos programação concorrente e programação paralela são sinônimos (Quinn, 2003).

1.7.4 Programa

Código fonte escrito em uma linguagem de programação. Define uma sequência lógica de passos capaz de solucionar algum problema através da execução futura em um computador.

1.7.5 Processo

Basicamente é um programa em execução, formado pelo programa executável, os dados deste programa e sua pilha, registradores (PC, SP e outros), além das demais informações necessárias à execução do programa.

1.7.6 Threads

São processos leves (ou *lightweight processes*), as quais executam sequencialmente com seu próprio PC, pilha, registradores de dados, threads filhas e estado atual de cada thread.
Compartilham CPU em um modelo *timeshared*, como os processos fazem. Threads compartilham um mesmo espaço de endereçamento de memória global, arquivos abertos, processos filhos e sinais de interrupção.

1.7.7 Processos Concorrentes

São dois processos que começaram e ainda não finalizaram a sua execução, concorrendo por recursos do sistema, como processadores, memórias e E/S. Podem estar executando em um processador ou em diferentes processadores.

1.7.8 Processos Paralelos

São processos concorrentes executando em processadores distintos, i.e., realmente em paralelo. Processos paralelos representam um tipo de processo concorrente.

1.7.9 Processos Distribuídos

De uma forma geral e do ponto de vista conceitual, podem ser sinônimos de processos paralelos, pois precisam ser iniciados, sincronizados e se comunicar. Dados os seus objetivos, os processos distribuídos podem ter seus próprios modelos de programação,

os quais encapsulam mais os detalhes de implementação e os diferenciam, de ordem prática, dos processos paralelos. Não há um consenso na literatura entre as diferenças ou similaridades entre processos paralelos, distribuídos e concorrentes.

1.7.10 Interação: Comunicação & Sincronização

Interação é uma generalização da comunicação ou sincronização.

Comunicação: troca de dados. Pode ser explícita com passagem de mensagens ou implícita através de memória compartilhada.

Sincronização: atividade que garante uma ordem de execução dos processos concorrentes afim de garantir semântica correta na execução

1.7.11 Granulação

Também conhecida como granularidade, a granulação é a relação entre o peso da computação e da comunicação de um processo concorrente. A granulação é grossa quando o processo faz muita computação sequencial até precisar interagir (p.ex. um processo inteiro), e fina quando faz pouca computação até precisar interagir (p.ex. uma operação aritmética). A granulação média fica no meio termo das duas, como p.ex., na execução de blocos de comandos de um processo. Quanto mais fina for a granulação, supõe-se que mais eficiente será a rede de comunicação para que o custo da interação não imponha grandes perdas de desempenho à versão paralela implementada (Rauber & Rünger, 2010).

1.7.12 Pipeline

Sobreposição de múltiplas instâncias de um problema, pelo paralelismo temporal de diferentes estágios extraídos a partir do problema a ser resolvido. Por exemplo, considerando a execução de um ciclo de instrução os estágios podem ser: busca, decodificação, execução e escrita do resultado. Após 04 ciclos de execução, haverá potencialmente 04 instruções sendo executadas ao mesmo tempo, cada uma em um estágio diferente. O número de estágios determina o grau do paralelismo no pipeline. Uma instância do problema é executada sequencialmente, mas várias são executadas em paralelo. Estágios podem ser vistos como uma sequência de produtores e consumidores, onde um estágio i consume a produção de um estágio i-1 e produz algum resultado a ser usado no estágio i+1.

1.7.13 Tempos Usados como Métricas de Desempenho

Há vários tempos importantes na execução de processos em um computador. O tempo de execução é o tempo usado pelo processador para executar o processo. Pode ser tempo de sistema ou tempo de usuário. O tempo de resposta é o tempo gasto entre a submissão e a finalização do processo. Inclui o tempo de execução e o tempo que o processo ficou parado (nas filas de bloqueado ou pronto). O tempo ocioso é o tempo em que o processo ficou parado em uma primitiva de comunicação e/ou sincronização aguardando para continuar. O tempo sequencial normalmente é o tempo de resposta, mas também pode ser o tempo de execução, de um processo sequencial que foi submetido para execução. O tempo paralelo normalmente é o tempo de resposta, mas também pode ser o tempo de execução, de uma aplicação paralela/concorrente que foi submetida para execução. Considera o tempo decorrido da submissão até a finalização do último processo da aplicação.

1.7.14 Métricas para Avaliar o Desempenho: Speedup e Eficiência

Speedup:

Sp absoluto (em relação ao melhor algoritmo sequencial conhecido):

$$Sp = Tseq / Tpar\_p\_processadores$$

Sp relativo (em relação ao programa paralelo rodando com apenas um processador):

$$Sp = Tpar\_1\_processador / Tpar\_p\_processadores$$

Eficiência:

$$E = Sp / p \quad \text{(onde p é o número de processadores)}$$

O Sp pode ser linear (Sp = p) e representa o caso ótimo, onde p é o número de processadores. No entanto, o caso comum é quando Sp < p, em função das sobrecargas esperadas para a versão paralela que não ocorrem na versão sequencial. O Sp também pode ser superlinear (Sp > p), sendo que estes casos acontecem quando a execução de uma operação básica é muito mais lenta na versão sequencial que na versão paralela, em função de alguma característica pontual como a eliminação do uso de swap e/ou favorecimento ao uso de caches na versão paralela (Grama et al., 2003) (Vide Figura 1)

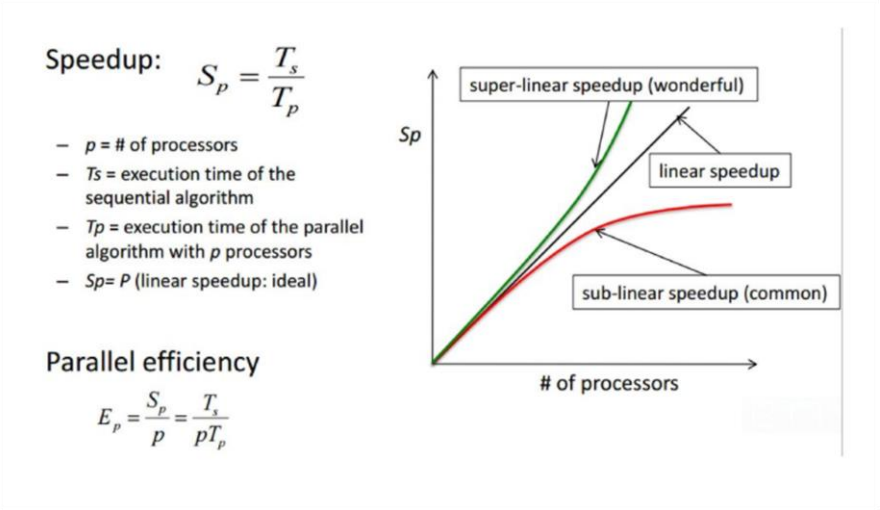

Figura 1. Speedup & Eficiência.
Figura desenvolvida por Sydney Newman e disponível em https://slideplayer.com/slide/13453241/

1.7.15 Escalabilidade

Escalabilidade de uma aplicação paralela determina se a eficiência da mesma permanece constante quando o número de processos aumenta, desde que a carga de trabalho também aumente.

Para a escalabilidade determina-se a taxa que a carga de trabalho deve aumentar frente ao crescimento do número de processadores. Esta taxa determina o grau de escalabilidade do sistema paralelo. Para se dizer que uma aplicação paralela tem escalabilidade, a taxa de crescimento da carga de trabalho frente ao aumento do número de processadores não deve ser alta.

Uma função que determine qual é a taxa de aumento da carga de trabalho necessária para manter a eficiência fixa conforme p aumenta é chamada função de isoeficiência (Grama et al., 2003).

1.8 Modelos para Sistemas Paralelos

O termo modelo aparece em diferentes contextos e com diferentes sentidos na literatura. Precisa-se tomar cuidado ao usá-lo. (Rauber & Rünger 2010) propõe o uso destes modelos, associados a diferentes níveis da solução computacional empregada:

Modelo de Máquina:

Nível de abstração mais baixo que corresponde à descrição do hardware e SO: regs ou buffers de E/S;
Linguagem Assembly é baseada neste nível de modelo.

Modelos Arquiteturais:

Descrevem: rede de conexão, organização de memória, sincronia ou assincronia de processadores;
Arquiteturas SIMD ou MIMD.

Modelo Computacional

Modelo arquitetural mais formal com funções de custo refletindo o tempo necessário para a execução de um algoritmo sobre os recursos dados pelo modelo arquitetural;

RAM (sequencial), PRAM, PHASE, BSP e LogP são exemplos desses modelos;

Modelo de Programação:

Descreve as semânticas da linguagem de programação;
Especifica a visão do programador e como ele pode codificar o algoritmo;
Influenciam: projeto arquitetural, linguagem, compilador e bibliotecas:

Assim, há diferentes modelos de programação para uma mesma arquitetura.

Alguns itens que podem variar:

Nível do paralelismo: instrução, comandos, procedimentos, loops, processos;
Implícito: (compiladores) ou explícito (programador);
Especificar partes concorrentes de um programa: tarefas concorrentes;
Distribuir as tarefas em processos e processadores (balanceamento da carga de trabalho);
Modo de execução de tarefas (*SIMD*, *SPMD*, *MPMD*, *Sync vs assync*);
Padrões de comunicação: passagem de mensagens ou variáveis compartilhadas;
Mecanismos de comunicação e sincronização: organizam computação e comunicação entre processos.

1.9 Considerações Finais

A Computação Paralela é uma das principais formas de se aumentar o desempenho de soluções computacionais, i.e., de se realizar Computação de Alto Desempenho.

No que tange ao software, a programação paralela está no núcleo de diferentes tecnologias de desenvolvimento, viabilizando a construção de aplicações paralelas ou distribuídas (não centralizadas) cada vez mais abrangentes e impactantes à sociedade. A Programação Paralela (ou Concorrente) é amplamente utilizada atualmente em diferentes contextos: sistemas operacionais, sistemas distribuídos, banco de dados, HPC, telefonia móvel, sistemas embarcados, redes sociais e em muitos outros contextos.

Do ponto de vista de hardware, a alta integração de transistores alavancou o uso de processadores com múltiplos núcleos, homogêneos e heterogêneos, capazes de explorar o paralelismo no nível de threads e processos. Tais tecnologias são comuns não apenas aos atuais servidores e desktops, mas também podem ser encontradas em *smartphones*, *tablets* e notebooks.

Este tópico abordou a introdução à Computação de Alto Desempenho associada à Computação Paralela, destacando inicialmente seu contexto, motivação e objetivos. Também foram discutidos alguns desafios encontrados pelos desenvolvedores de aplicações paralelas e alguns dos principais conceitos associados.

Referências Bibliográficas

FOSTER, I. Designing and Building Parallel Programs, Addison-Wesley Publishing Company, 1994.

GRAMA, A.; KUMAR, U.; GUPTA, A.; KARYPIS, G. Introduction to Parallel Computing, 2nd Edition, 2003.

HANSEN, P. B.; The Invention of Concurrent Programming: from Semaphores to Remote Procedure Calls, Springer Verlag, New York, ISBN 978-1-4757-3472-0, 2001.

PACHECO, P. S. An introduction to parallel programming. Morgan Kaufmann. Elsevier Science, 2011.

QUINN, M. J. Parallel Programming in C with MPI and OpenMP, McGraw-Hill,Published 2003.

RAUBER, T.; RÜNGER, G. Parallel programming: for multicore and cluster systems. Springer, 2010.