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 3 – Introdução à Avaliação de Desempenho para Computação Paralela

por Paulo Sérgio Lopes de Souza

São Carlos/SP – Brasil – 2020

Conteúdo

1 Avaliação de Desempenho de Aplicações paralelas:

1.1 Considerações Iniciais

Programas paralelos, assim como sequenciais, devem estar corretos e ter bom desempenho. Saber conduzir uma avaliação de desempenho de programas paralelos é essencial e é considerada não trivial, pois há vários aspectos que acarretam um desempenho bom ou um ruim de uma aplicação paralela. Alguns fatores a considerar são: diferentes plataformas computacionais, modelos e ferramentas de programação paralela, paradigmas de comunicação, heterogeneidade em diferentes aspectos.

Além dos diferentes fatores que afetam as aplicações paralelas, estas podem ter **diferentes objetivos**. Uma aplicação numérica, por exemplo, é baseada em grandes matrizes de ponto-flutuante densas, as quais necessitam otimizar o uso da memória para poderem ser executadas. Se puderem ser executadas, então visam executar com eficiência. A **quantidade de memória (cache e principal) são pontos críticos neste contexto**, além de se desejar também o menor tempo de execução. Sistemas embarcados em áreas remotas e com localização de difícil acesso, exigem que o **consumo de energia seja minimizado**, além de se desejar que haja respostas dentro de certos limites de tempo. Grandes sistemas de banco de dados distribuídos preocupam-se mais com a **vazão de requisições atendidas** com sucesso, que com a redução do tempo de resposta de uma requisição em particular.

Projetos não devem minimizar simplesmente o menor "tempo de resposta". **Foco deve ser na otimização de uma função específica do problema, em função de fatores que influenciam no seu desempenho**, como: demanda por memória, consumo de energia, custos de implementação/manutenção, tempo de processamento e outros. Deve-se ter em mente que objetivos distintos podem ser conflitantes e envolvem objetivos contraditórios entre: simplicidade, desempenho, portabilidade, legibilidade e outros.

Alguns **exemplos de métricas** usadas para avaliar o desempenho final de uma aplicação paralela são: tempo de execução, speedup, eficiência, demanda por memória, throughput, latência de requisições, taxas de E/S, throughput de rede, custos de projeto / implementação / V&V, reusabilidade, demanda por hardware, custos de hardware, custos de manutenção, portabilidade e escalabilidade. A escolha correta da métrica de avaliação de desempenho é essencial e está relacionada diretamente aos objetivos da aplicação paralela.

Para tomar suas decisões de projeto, os projetistas de aplicações paralelas usam **Modelos Computacionais Paralelos (também conhecidos como Modelos de Desempenho) que refletem aspectos importantes do comportamento da aplicação.**

Modelos de desempenho **comparam a eficiência dos algoritmos em relação aos objetivos**, ressaltando aspectos como escalabilidade, gargalos e limitações. Eles permitem **guiar a implementação mostrando onde otimizações devem ser feitas**.

De uma forma geral, o modelo **deve caracterizar as aplicações paralelas** considerando aspectos como: **porções sequenciais e paralelas do código, custos com comunicação e sincronização**, entre outros aspectos relevantes.

Este tópico aborda a atividade de avaliar o desempenho de programas paralelos, apresentando motivações, métricas, modelos e exemplos de uso de tais recursos. Nesse sentido, a seção 2 mostra um exemplo relacionado à extrapolação do desempenho de uma aplicação paralela executada. A seguir são apresentadas métricas comumente usadas. Exercícios são propostos no final.

1.2 Aplicando na prática: extrapolando observações

Exemplo para destacar a importância de métricas e modelos (Foster, 1994- seção 3.2.2)
Um problema X tem três soluções computacionais usando programação paralela. Para uma carga de trabalho N = 100, todos têm um Speedup aproximado de 10,8 com 12 processadores e Eficiência de 90%. O Speedup (Sp) = Tseq / Tpar e a Eficiência(E) = Sp / P (P é o nr de Processadores).

O algoritmo sequencial é dado por $N + N^2$.

As três versões do algoritmo paralelo são:
1) $N + N^2 / P$
Divide-se a porção N^2 do problema entre os P processadores, deixando sequencial ainda a parte N.

2) $(N+N^2)/P + 100$
Divide-se o problema todo entre os P processadores, mas traz uma sobrecarga fixa de 100.

3) $(N+N^2)/P + 0.6P^2$
Divide-se o problema todo entre os P processadores, incluindo uma sobrecarga de 0.6P^2.

Estes algoritmos são iguais?
O que acontece com o desempenho se houver 1000 processadores e se N = 10 ou 1000?

O desempenho do **algoritmo sequencial** ($N + N^2$):
N = 10 => 10+[(10*10)] = 110
N = 100 => 100+[(100*100)] = 10100
N = 1000 = > 1000+[(1000*1000)] = 1001000

1) $N + N^2 / P$ (Divide 2º componente, mas replica o 1º).
N = 10 e P = 12 10+(10*10)/12= 18,3 (Sp = 6,0 E=50,0%)
N = 100 e P = 100 100+(100*100)/100 =200 (Sp = 50,5 E=50,5%)
N = 1000 e P =1000 1000+(1000*1000)/1000 =2000 (Sp = 500,5 E=50,1%)

N	P	Tempo	Sp	Ef(%)
100	12	933,3	10,8	90,2
10	12	18,3	6,0	50,0
100	100	200,0	50,5	50,5
1000	1000	2000,0	500,5	50,1

2) $(N+N^2)/P + 100$ (Divide ambos os componentes, mas tem sobrecarga fixa)
N = 10 e P = 12 (10+(10*10))/12 + 100 = 109,2 (Sp = 1,0 E = 8,4%)
N = 100 e P = 100 {100+(100*100)}/100 + 100 =201,0 (Sp = 50,2 E=50,2%)
N = 1000 e P =1000 {1000+(1000*1000)}/1000 + 100 = 1101 (Sp = 909,2 E=90,9%)

N	P	Tempo	Sp	Ef(%)
100	12	941,7	10,7	89,4
10	12	109,2	1,0	8,4
100	100	201,0	50,2	50,2
1000	1000	1101,0	909,2	90,9

3) $(N+N2)/P + 0.6P^2$ (Divide ambos os componentes, mas tem sobrecarga variável)
N = 10 e P = 12 (10+(10*10))/12 + 0.6*(12*12) = 95,6 (Sp=1,2 E=9,6%)
N = 100 e P = 100 [100+(100*100)]/100 + 0,6*(100*100) =6101 (Sp = 1,7 E=1,7%)
N = 1000 e P =1000 [1000+(1000*1000)]/1000 + 0,6*(1000*1000) = 601001
(Sp = 1,7 E=0,2%)

N	P	Tempo	Sp	Ef(%)
100	12	928,1	10,9	90,7
10	12	95,6	1,2	9,6
100	100	6101,0	1,7	1,7
1000	1000	601001,0	1,7	0,2

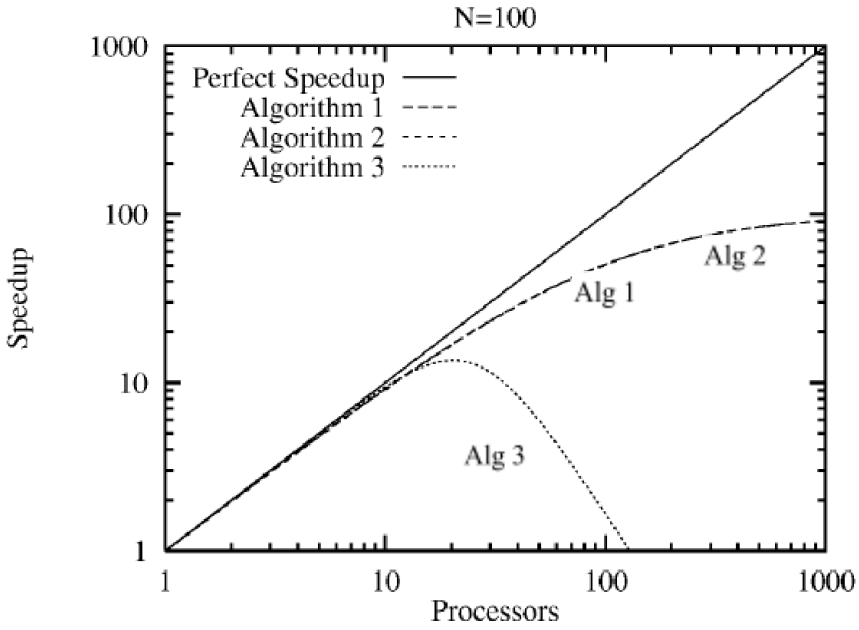

Figura 3.01(a) – Resultados para N = 100.

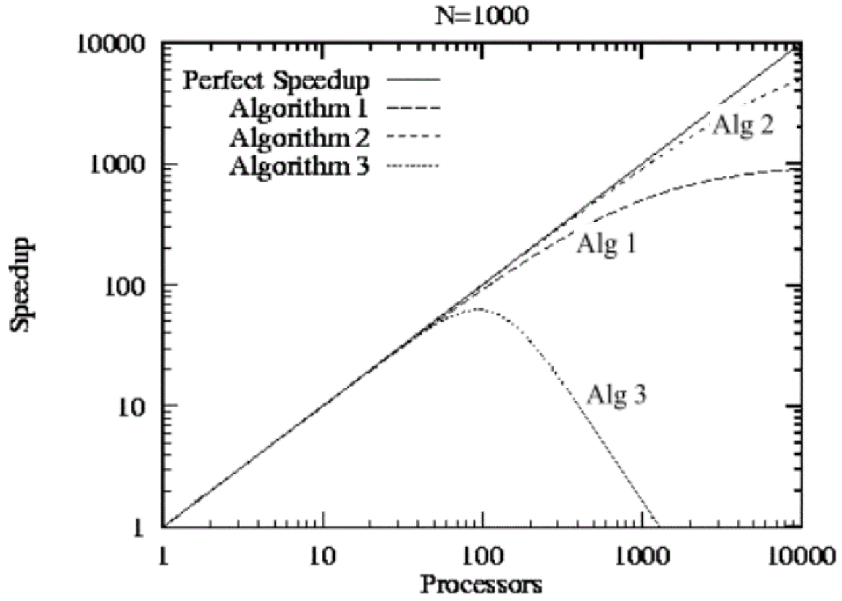

Figura 3.01(b) - Resultados para N = 1000.

Os resultados acima permitem ponderar que:
- O primeiro algoritmo não paraleliza todo o algoritmo inteiro e a porção sequencial é significativa para o desempenho final, conforme P aumenta para uma mesma carga. Apresentou desempenho significativamente menor que o algoritmo 2 para P=1000.
- O segundo algoritmo paraleliza 100% do algoritmo, mas acrescenta um custo fixo de 100 unidades. Esse custo tem um impacto maior para uma carga menor. Apresentou o melhor desempenho para valores maiores de P e N.
- O terceiro algoritmo paraleliza 100% do algoritmo, mas acrescenta um alto custo variável em função do crescimento do número de processadores. Apresenta o pior desempenho dos três nos testes realizados.

Conclusão: não se pode considerar apenas o tempo de resposta de um algoritmo paralelo sem considerar fatores como: custos extra associados, tamanho da plataforma e carga de trabalho.

1.3 Métricas de desempenho

Tempo de execução, tempo de resposta (como resultado do tempo de computação, tempo de comunicação e tempo de ociosidade), Speedup (absoluto e relativo), Speedup Superlinear, Eficiência, Lei de Amdahl;

Custo Total da Versão paralela

Custo = pT_p

Na vdd faz um somatório das execuções de todos os processadores. Pode-se usar os tempos de resposta ou execução, a depender do objetivo da análise (qual tempo se deseja verificar de fato).

Sobrecarga da Versão Paralela (Grama 5.1)

Como quantificar?

$T_0 = pT_p - T_s$ ou $T_0 = \text{Custo} - T_{seq}$ (Grama 5.2.2)

T_{seq} representa o custo da versão sequencial mais rápida conhecida

Quais são as origens da sobrecarga? (Grama 5.1)

- Interação (bem comum),
- Ociosidade
  - falta de balanceamento da carga de trabalho (bem comum),
  - esperas por sincronização (bem comum),
  - porções seriais fazem com que outros processos não tenham o que fazer
- Computação Extra
  - Algoritmos sequenciais podem não ser paralelizáveis
  - Versões paralelas podem agregar novas computações para permitir alto grau de concorrência

Tempos de Resposta da Versão Paralela

Tempo de resposta de um programa paralelo T é o tempo decorrido desde que o primeiro processo inicia a sua execução até o momento que o último processo termina sua execução.

Este tempo é o aquele verificado pelo usuário final para a execução da aplicação e engloba chaveamento de contextos, computação, comunicação e ociosidade.

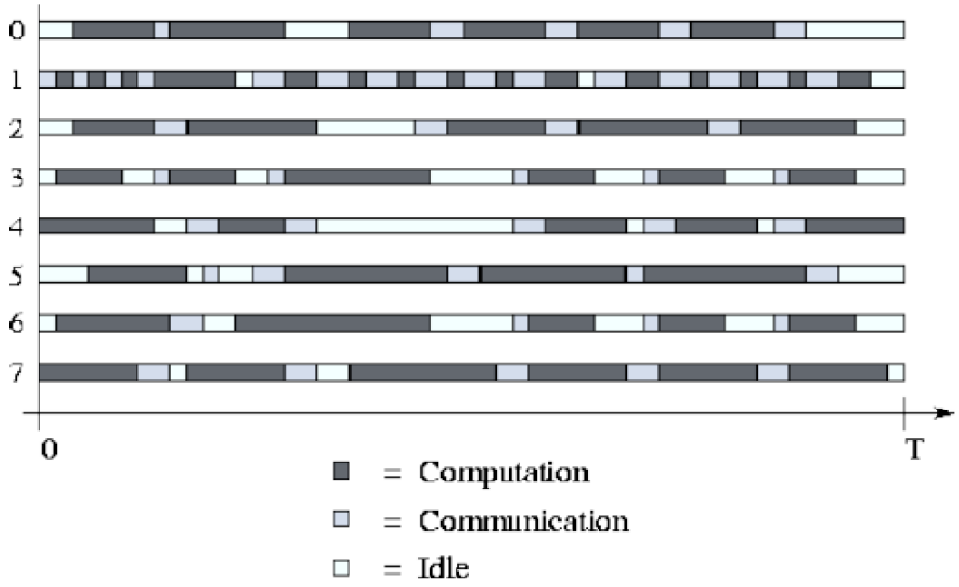

Figure 3.2: *Activity plot during execution of a parallel program on eight processors. Each processor spends its time computing, communicating, or idling. T is the total execution time.*

(Foster, 1994)

O tempo total para a finalização da aplicação paralela pode ser aproximado, generalizando-se com a soma do tempo de todos os processos, considerando computação, comunicação e tempo ocioso.

$$T = T^j_{\text{comp}} + T^j_{\text{comm}} + T^j_{\text{idle}}$$

onde j é um processador arbitrário da plataforma (Foster, 1994)

$$\begin{aligned} T &= \frac{1}{P}(T_{\text{comp}} + T_{\text{comm}} + T_{\text{idle}}) \\ &= \frac{1}{P}\left(\sum_{i=0}^{P-1} T^i_{\text{comp}} + \sum_{i=0}^{P-1} T^i_{\text{comm}} + \sum_{i=0}^{P-1} T^i_{\text{idle}}\right) \end{aligned}$$

(Foster, 1994)

Computação T_{comp} (tempo computando) (Foster 3.3.1)

Dependente do tamanho da carga de trabalho, representado por um parâmetro N

Número de operações básicas feitas sobre N.

Um algoritmo para multiplicação de matrizes teria uma computação N^3 e um algoritmo para soma de dois vetores teria uma computação N.

No exemplo do primeiro algoritmo da seção 1.2 a aplicação tinha um T_{comp} (N + N^2)

T_{comp} estima o tempo para cada operação básica, esta determinada empiricamente em função do hardware usado.

Considerar

replicações de computação, plataformas heterogêneas, custo de acesso memória (quando remotas).

O tempo de computação deve considerar porções sequenciais e porções paralelas, comuns nos algoritmos desenvolvidos.

Comunicação T_{comm} (+ Algoritmos Hwang pp 120, 121)

Tempo enviando e recebendo mensagens

Tempo de Startup T_s (tempo gasto pelo protocolo e início da transferência)

Tempo de Transferência de uma Palavra T_w (normalmente Byte)

$T_{msg} = T_s + T_w * Qtde$

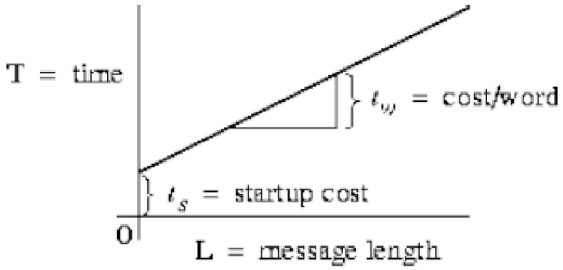

Figure 3.3: *Simple communication cost model: $T_{msg} = t_s + t_w L$. In this plot of time versus message length, the slope of the line corresponds to the cost per word transferred and the y-intercept to the message startup cost.*

(Foster, 1994)

Como medir o desempenho da rede local?

Round-Trip Time (RTT) ou Ping-pong

Parte de 0 bytes e sobe a quantidade de bytes.

Pode-se considerar saltos nas diferentes redes ao longo do caminho

Diminuindo sobrecarga da medição de tempo

A diminuição é importante quando os tempos medidos são muito pequenos, pois a coleta dos tempos inicial e final alteram o desempenho da aplicação. Eliminar esta sobrecarga produz resultados mais confiáveis.

```
...
Tmp = tempo();
Start = tempo();
/* computação a se medir o tempo de resposta*/
End = tempo();
Overhead = Start – Tmp;
Totaltime = End – Start – Overhead;
...
```

Ociosidade (T^j_{idle})

Processo parado por falta de computação ou dado

Computação: Falta balanceamento da carga de cada processador

Dado: Aguardando sincronização/comunicação

Para resolver pode-se, por exemplo:

Sobrepor computação e/ou comunicações

Gerar múltiplas threads

Usar primitivas não bloqueantes

Como Calcular? Normalmente estima-se esse tempo, aproximando seu valor.

Exemplo (Cantú-Paz, 1997): uma aplicação paralela tem 01 processo master e S processos slaves, os quais executam um processo iterativo. O master envia porções de dados para os slaves com um custo Tc a cada mensagem. Os slaves iniciam a sua computação assim que recebem sua porção dos dados, consumindo o tempo Tcomp. Quando finalizam, os slaves retornam o resultado da iteração para o master com o tempo Tc para a mensagem. O master recebe todos os resultados, computa uma consolidação (processamento com tempo desconsiderável) e dispara nova iteração ao enviar novas mensagens aos slaves. O tempo idle do master pode ser estimado calculando a diferença entre tempo que o primeiro slave utiliza para fazer a sua tarefa (Tc + Tcomp) e o tempo que o master utiliza para enviar todas as mensagens aos slaves. Em outras palavras o tempo idle no master é Tidle = Tcomp - (S - 1)*Tc.

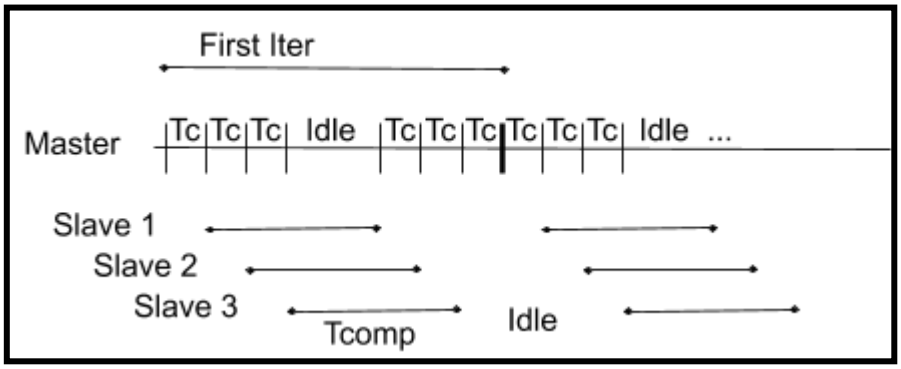

Figura 1 - Esquema para a execução de uma aplicação paralela com 01 master e 03 slaves.

Speedup Sp (Foster 3.3.2)

Razão que determina o ganho de desempenho entre duas versões de uma aplicação. No nosso caso, o Speedup Absoluto é a razão entre o tempo sequencial entre o tempo paralelo com P processadores

Speedup Absoluto Tseq/Tpar_p

O Speedup Relativo considera o tempo do algoritmo paralelo com 1 processador, pelo tempo do mesmo algoritmo paralelo com P processadores, onde P >= 1.

Speedup Relativo Tpar_1/Tpar_p

Casos especiais: Sp Linear (Sp == P) , Sp Super Linear (Sp > P) e Caso Comum (Sp < P)

Speedup Superlinear

Influência da cache ou outros recursos de hardware

Influência em algoritmos de busca (Foster pp109)

Lei de Amdahl (Hwang pp134)

Determina que, com uma carga fixa, o limite do Sp é 1 / α, onde α é o percentual sequencial da carga de trabalho (0 >= α <= 1). Aqui W é a Workload e n é o número de processadores.

Sp = Tseq / Tpar = (W / (αW+(1-α)(W/n)) … n / (1 + (n-1) α) …

1 / α, quando n tende ao infinito

Colocando overheads (T_0) (comunicação, ociosidade, replicação da computação)

Sp = (W / (αW+(1-α)(W/n) + T_0)) … 1 / (α + (T_0/W)) …

quando n tende ao infinito

Ex: em se mantendo um W fixo e um nr infinito de processadores, para Amdahl:
com 20% (α = 0,2) de porção sequencial, o Sp máximo será 5
com 2% (α = 0,02) de porção sequencial, o Sp máximo será 50
com 0,2% (α = 0,002) de porção sequencial, o Sp máximo será 500

De ordem prática: A Lei de Amdahl considera que as maiores limitações vêm de custos de comunicação, ociosidade, computação replicada e de "porções sequenciais" da aplicação. Em outras palavras, o projeto de algoritmo deve aumentar ao máximo o tamanho das porções paralelas do código, enquanto diminui a sobrecarga gerada pela computação paralela.

A Lei de Amdahl considera uma carga de trabalho fixa ao se aumentar o número de processos.
Aumentando-se o número de processos, o impacto da porção sequencial aumenta no tempo final e o Sp diminui a partir de um certo número de processos.

Aumentando-se a carga de trabalho o percentual da parte sequencial fica menos significativa e o Sp aumenta, aumentando o número ótimo de processos que faz com que o Sp seja maior (aproxima-se do ótimo)

Lei de Gustafson
https://www.d.umn.edu/~tkwon/course/5315/HW/MultiprocessorLaws.pdf
Remove a restrição de uma carga de tamanho fixo. Ao se aumentar o número de processos aumenta-se também a carga de trabalho.
Propõe o conceito de tempo fixo que atinge um speedup melhorado por escalando o tamanho do problema com o aumento do número de processos. Isso permite aumentar a precisão da computação e não a redução do tempo de execução.

Deseja-se solucionar o maior problema possível com mais processos com o mesmo tempo de execução usado para solucionar um problema com menos processos. Aumentando a máquina aumenta-se a carga de trabalho W. Gustafson determina quanto o tempo da carga de trabalho W deveria aumentar para garantir o mesmo tempo de execução com mais processadores na plataforma paralela.

O percentual da parte sequencial da carga de trabalho é dada por α e a parte executada em paralelo é (1- α). Para simplificar, o tempo de resposta da aplicação paralela é dado por W (Workload), o qual é formado pela porção sequencial (αW) e uma porção paralela ((1-α)W), capazes de atingir o tempo de execução desejado para a W.
Com n processadores pode-se determinar a carga sequencial de trabalho como W′= αW+ (1-α)nW. O componente nW na versão sequencial ocorre porque a carga de trabalho aumenta conforme aumenta o número de processadores, caso fossem utilizados vários processadores. Reforçando, o tempo para a resposta da execução sequencial passa a ser W′= αW+ (1-α)nW, pois todas as n cargas de trabalho deverão ser executadas sequencialmente.
Embora a carga de trabalho aumente para nW, o tempo de resposta da versão paralela continua sendo W, pois há a sobreposição dos tempos das porções paralelas ((1-α)W).
Com isso tem-se que:

$$Sp = Tseq / Tpar = (\alpha W + (1-\alpha)nW) / W = \alpha + (1 - \alpha)\, n$$

i.e., o Sp é uma função linear de n, se a carga de trabalho é determinada para manter um tempo de resposta fixo.

Ex: para um W variável em função do aumento do número de processadores, o Sp para Gustafson é:

com 20% (α = 0,2) de porção sequencial

- o Sp para n = 10 será: 0,2 + (1- 0,2).10 = 8,2
- o Sp para n = 100 será: 0,2 + (1- 0,2).100 = 80,2
- o Sp para n = 1000 será: 0,2 + (1-0,02)1000 = 800,2

com 5% (α = 0,05) de porção sequencial

- o Sp para n = 10 será: 0,05 + (1 - 0,05).10 = 9,5
- o Sp para n = 100 será: 0,05 + (1 - 0,05).100 = 95
- o SP para n = 1000 será: 0,05 + (1 - 0,05).1000 = 950

Escalabilidade e Isoeficiência

O caso comum na computação paralela:

1) para um dado tamanho de problema, conforme o número de elementos de processamento aumenta, a eficiência diminui.

2) a eficiência aumenta se o tamanho do problema é aumentado enquanto o número de elementos de processamento permanece constante, até certo limite máximo de carga de trabalho.

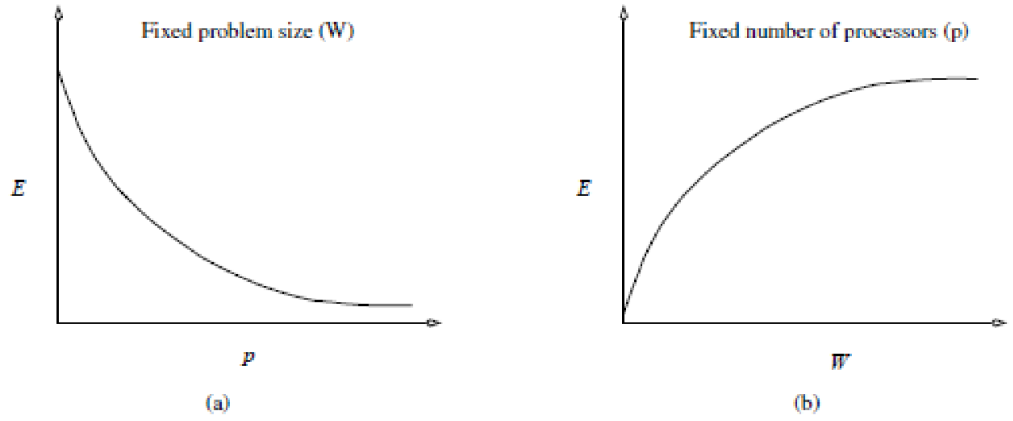

Figure 5.10 Variation of efficiency: (a) as the number of processing elements is increased for a given problem size; and (b) as the problem size is increased for a given number of processing elements. The phenomenon illustrated in graph (b) is not common to all parallel systems.

(Grama et al. 2003)

Escalabilidade de uma aplicação paralela é uma medida da sua capacidade de aumentar o Speedup proporcionalmente ao número de elementos de processamento. Reflete a habilidade do sistema de utilizar os recursos de processamento eficientemente.

Em outras palavras, a Eficiência permanece constante quando o número de processos aumenta, desde que a carga de trabalho também aumente. A questão é determinar 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. Taxas menores são mais desejáveis que uma taxa de crescimento alta do tamanho do problema.

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**.

1.4 Considerações Finais

Pode-se considerar que um dos mais importantes objetivos de se usar programas paralelos é a otimização de uma aplicação. Embora a otimização possa ser vista sob diferentes perspectivas, uma forma simples de quantificar essa otimização é através da redução do tempo de execução.

Tempo de execução depende de vários fatores:

Arquitetura, compilador e SO, ambiente de programação paralela e o modelo de programação onde o ambiente de programação é baseado e da características da aplicação, tais como: localidade dos dados e dependências.

Todos esses fatores devem ser considerados. Tarefa não trivial devido à interações complexas entre fatores.

Medidas de desempenho abstraem complexidade e tornam a análise do desempenho obtido viável.

Este tópico abordou diferentes aspectos neste contexto.

Métricas de desempenho

- Tempos de computação e comunicação
- Origens da sobrecarga
- Speedup e Eficiência
- Ponderações feitas por Amdahal e Gustafson
- Escalabilidade e isoeficiência

Referências

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.

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

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

CANTÚ-PAZ, E. Designing Efficient Master-Slave Parallel Genetic Algorithms. IlliGAL Report No. 97004, University of Illinois at Urbana-Champaign, May 1997.