Avaliação de Desempenho em Programas Concorrentes

Paulo Sérgio Lopes de Souza
pssouza@icmc.usp.br

Universidade de São Paulo / ICMC / SSC – São Carlos
Laboratório de Sistemas Distribuídos e Programação Concorrente

CNPq Conselho Nacional de Desenvolvimento Científico e Tecnológico
CAPES
FAPESP

Avaliação de Desempenho: fatores e objetivos

- Computação de Alto Desempenho
  - Deve maximizar o que se entende como “desempenho”
  - Devemos ser capazes de aferir o ganho de desempenho obtido

- **Fatores afetam o desempenho** de aplicações concorrentes
  - Plataformas computacionais distintas
  - Diferentes modelos de programação
  - Diferentes linguagens/bibliotecas para programação concorrente
  - Paradigmas de interação (*comm* e *sync*)
  - Heterogeneidades de arquiteturas, SOs, desempenho, ...

- **Objetivos** distintos mudam o entendimento de “desempenho”
  - Redução do tempo de resposta
  - Otimizar uso de memória para viabilizar execução de grandes volumes de dados
  - Minimizar consumo de energia
  - Aumentar a vazão de tarefas realizadas
  - Tolerância a falhas & Alta disponibilidade
  - ...

- Objetivos distintos têm **métricas distintas** para quantificar o desempenho

Avaliação de Desempenho: métricas

- Exemplos de **métricas** para avaliar o desempenho
  - Tempo de resposta, tempo de execução (sistema e usuário)
  - Vazão (*throughput*) de instruções/processos/requisições/...
  - Speedup & eficiência & escalabilidade
  - Demanda por memória
  - Latência de requisições,
  - Taxas de E/S (throughput de rede, disco, processamento gráfico, ...)
  - Custos de: projeto, implementação, (V, V & T), e manutenção
  - Reusabilidade e portabilidade

- Escolha da métrica para avaliar o desempenho **deve ir ao encontro dos objetivos** da aplicação e/ou do que se deseja avaliar na aplicação

Avaliação de Desempenho: modelos

- Além da métrica, deve-se determinar componentes da aplicação que afetam o desempenho
  - Formam um **modelo de desempenho** que permitem analisar a eficiência dos algoritmos em relação aos objetivos
    - Ressaltam gargalos, limitações e possibilidade de escalabilidade
    - Modelos **guiam o projeto** e mostram a qualidade dele
      - Ajudam a focar onde as otimizações devem ser feitas

- Modelos de desempenho caracterizam aplicações concorrentes em
  - Porções sequenciais e paralelas
  - Custos com criação e gerência de processos
  - Custos com comunicação e sincronização
  - Custos com outras E/S relevantes à aplicação

Avaliação de Desempenho: um exemplo

- Um exemplo:
  - Programa Concorrente X tem três soluções computacionais
    - Para uma carga de trabalho N = 100 e 12 CPUs há
      - Sp = 10,8 e E = 90%.

  - **Algoritmo sequencial**:
    - $N + N^2$

  - As **três versões do algoritmo concorrente** são:
    - 1) $N + N^2 / P$
    - 2) $(N+N^2) / P + 100$
    - 3) $(N+N^2) / P + 0.6P^2$

  - Os três algoritmos têm desempenhos iguais na configuração acima
    - Qual seria o SP e a E com:
      - 1000 processadores?
      - Carga de trabalho N = 10 ou N=1000?

Avaliação de Desempenho: um exemplo

- Algoritmo sequencial do exemplo: $N + N^2$

  N = 10 => 10+[(10*10)] = 110

  N = 100 => 100+[(100*100)] = 10100

  N = 1000 = > 1000+[(1000*1000)] = 1001000

- Considerando a primeira versão do algoritmo paralelo:

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

Foster (1994)

Avaliação de Desempenho: um exemplo

- Algoritmo sequencial do exemplo: $N + N^2$

  N = 10 => 10+[(10*10)] = 110

  N = 100 => 100+[(100*100)] = 10100

  N = 1000 = > 1000+[(1000*1000)] = 1001000

- Considerando a segunda versão do algoritmo paralelo:

2) (N+N2)/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 = 1101,0

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

Foster (1994)

Avaliação de Desempenho: um exemplo

- Algoritmo sequencial do exemplo: $N + N^2$

  N = 10 => 10+[(10*10)] = 110

  N = 100 => 100+[(100*100)] = 10100

  N = 1000 = > 1000+[(1000*1000)] = 1001000

- Considerando a terceira versão do algoritmo paralelo:

3) $(N+N2)/P + 0.6P^2$ (Divide ambos os componentes, com 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

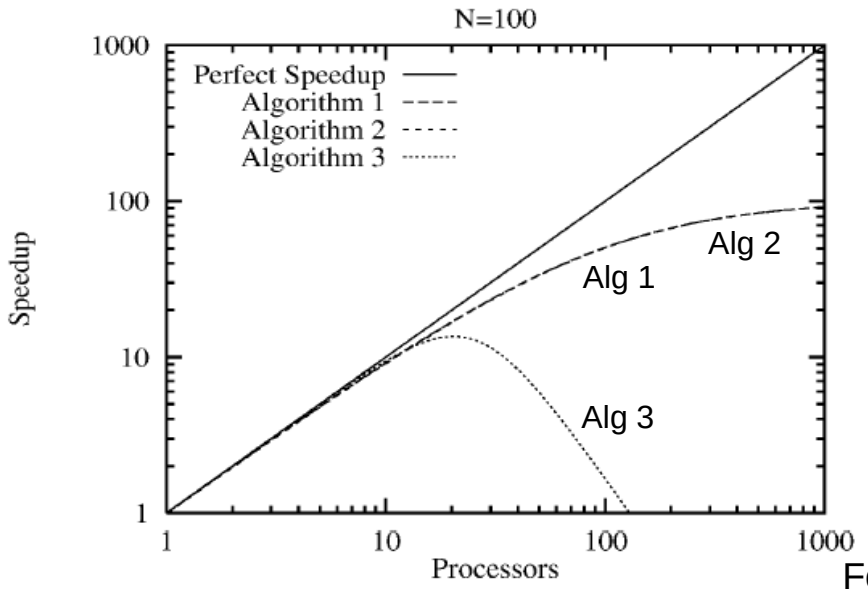
Sp = 1,7 E = 0,2%

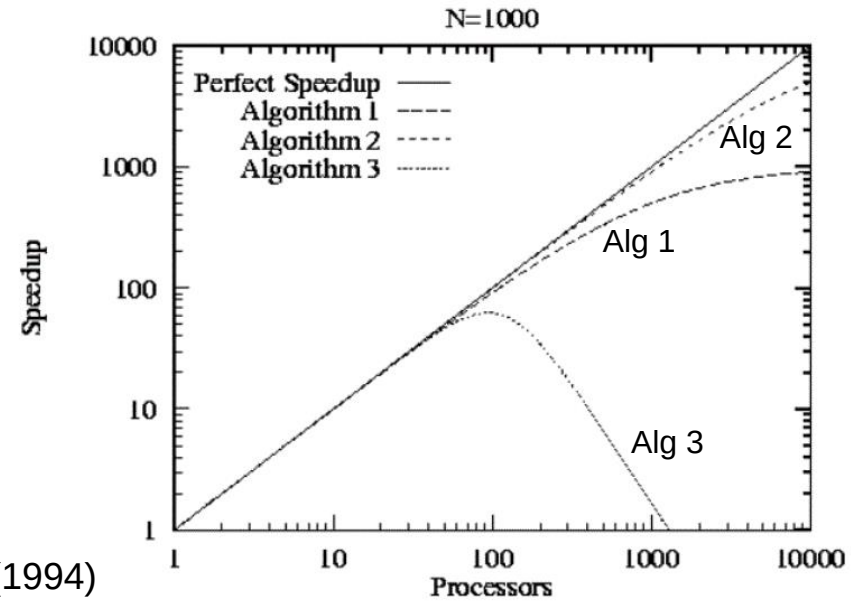
N	P	Tempo	Sp	Ef(%)
100	12	928,1	10,9	90,7
10	12	95,6	1,2	9,6
100	100	6101,0	1,7	1,7
1000	1000	601001,0	1,7	0,2

Foster (1994)

Avaliação de Desempenho: um exemplo

- 1) $N + N^2 / P$
  - Não paraleliza todo o código
  - Porção sequencial torna-se significativa conforme P aumenta para uma mesma carga
  - Desempenho menor que o algoritmo 2 para P=1000

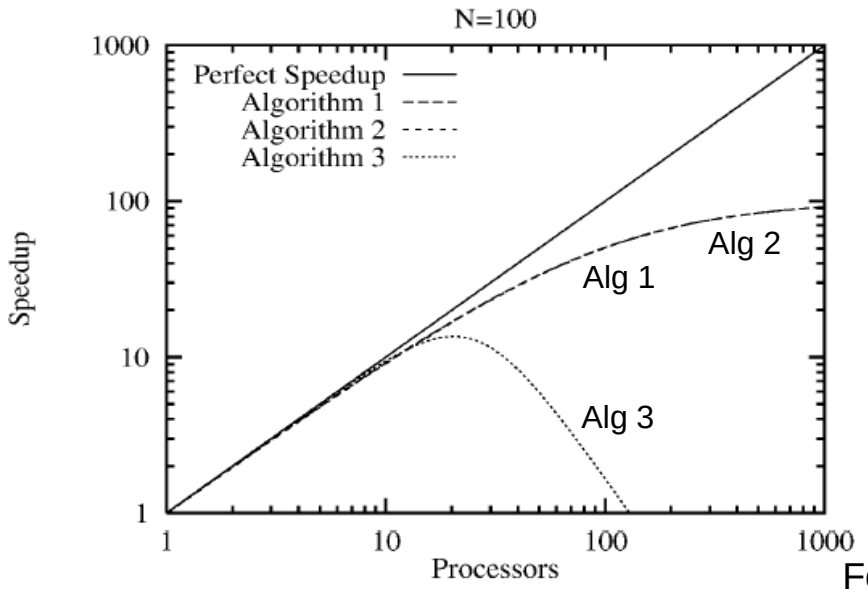


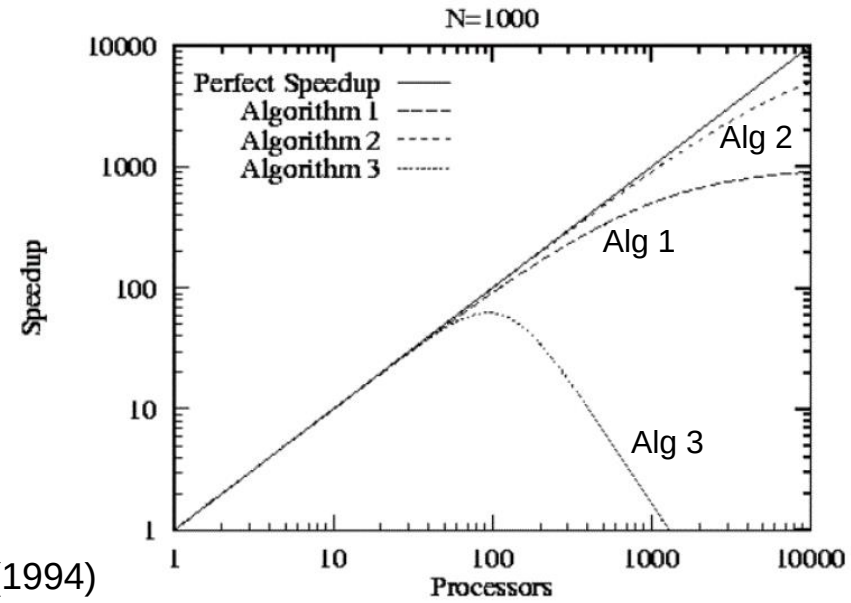


Foster (1994)

Avaliação de Desempenho: um exemplo

- 2) $(N+N^2) / P + 100$
  - Paraleliza 100% do algoritmo, comum custo fixo de 100 unidades
    - Custo fixo impacta mais o desempenho com carga menor
    - Apresentou o melhor desempenho para valores maiores de P e N



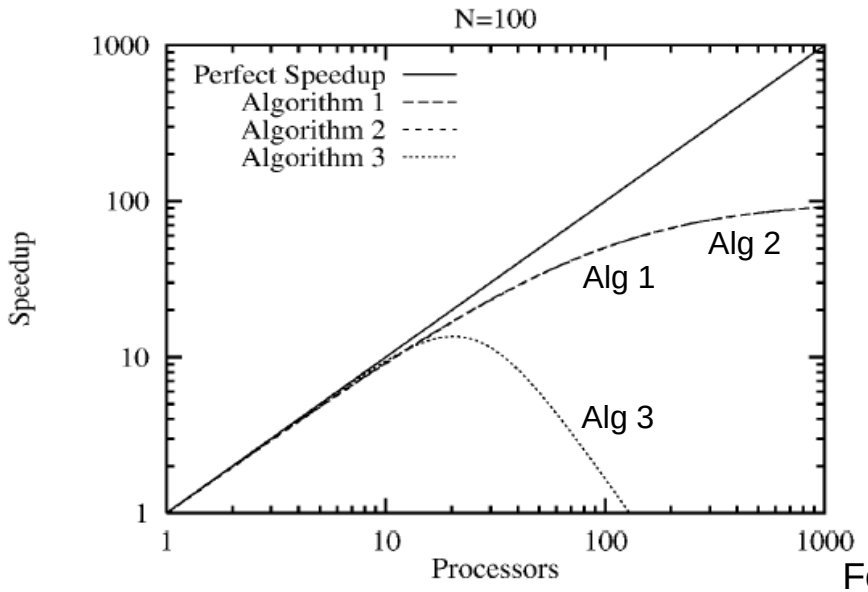


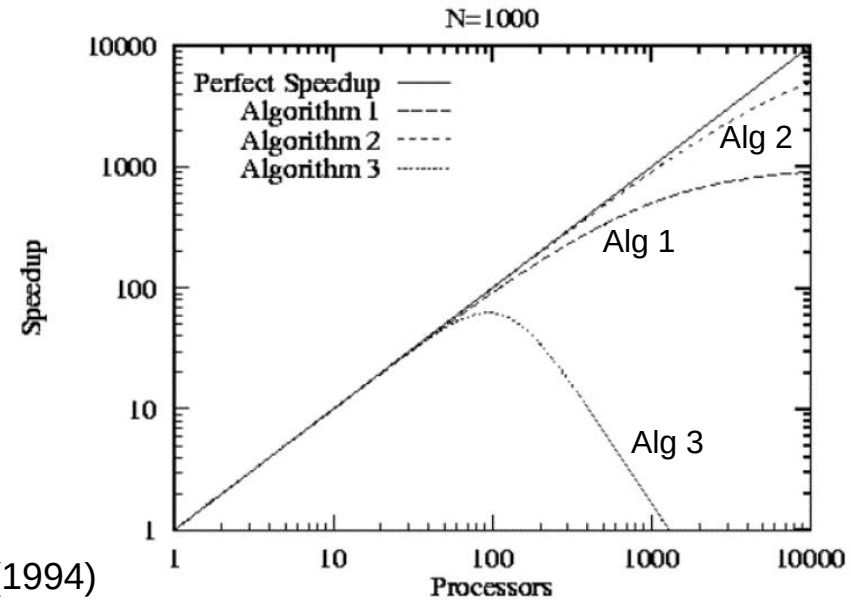
Foster (1994)

Avaliação de Desempenho: um exemplo

- 3) $(N+N^2) / P + 0.6P^2$
  - Paraleliza 100% do algoritmo com um alto custo associado a P^2
  - Apresenta o pior desempenho dos três nos testes realizados

- Conclusões
  - Considere sempre o tempo de variando:
    - custos extras associados ao algoritmo paralelo,
    - tamanho da plataforma (P) e
    - carga de trabalho (N)





Foster (1994)

Avaliação de Desempenho: métricas

- **Custo Total (CT)** da Execução Concorrente
  - $CT = p.T_p$
  - p é o nr de processadores e TP é o custo em um processador
    - Custo normalmente é o tempo de resposta, mas pode ser outra métrica

- **Sobrecarga** (*overhead*) da Versão Concorrente
  - $T_o = CT - T_{seq}$
  - T_{seq} é o custo da versão sequencial mais eficiente conhecida

- Origens da sobrecarga:
  - Interação (comunicação e sincronização)
  - Ociosidade
    - Desbalanceamento
    - Sincronização
    - Porções seriais
  - Computação Extra
    - Algoritmos sequenciais originais podem não ser paralelizáveis
    - Versões paralelas podem agregar novas computações para permitir um grau maior de concorrência

Avaliação de Desempenho: métricas

- **Tempo de Resposta**
  - Tempo decorrido desde que o 1º processo é submetido até o momento que o último processo termina de executar
  - Tempo verificado pelo usuário final
    - Engloba chaveamentos de contexto, computação, comunicação e ociosidade

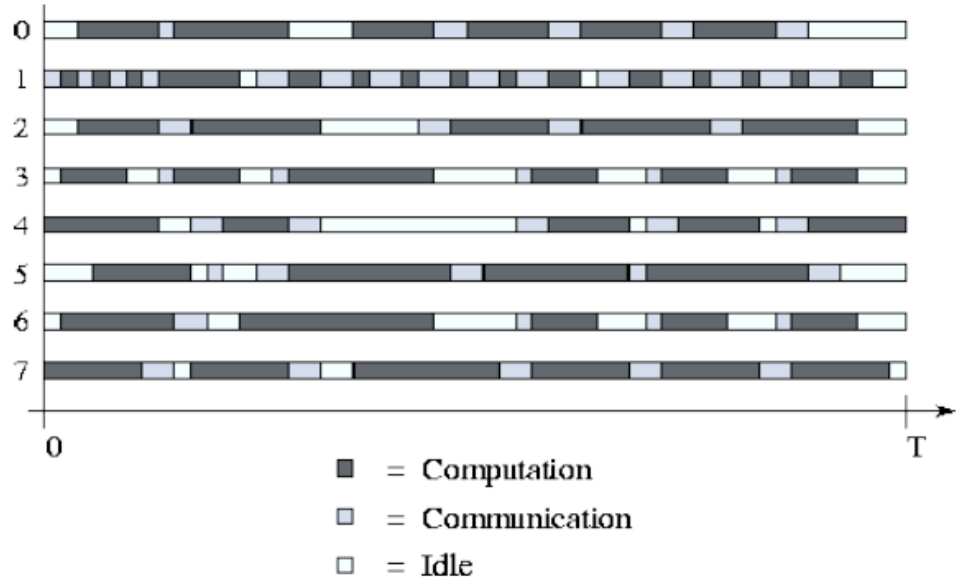


Figure 3.2: *Activity plot during execution of a parallel program on eight processors. Each processor spends its time computing, communicating, or idling. T is the total execution time.*

Foster (1994)

Avaliação de Desempenho: métricas

- Tempo de Resposta, uma generalização

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

Foster (1994)

- T_{comp} é o tempo computando
  - Depende da carga de trabalho N representada por um ou mais parâmetros:
    - Número de operações básicas sobre N
      - Multiplicação de matrizes sequencial seria N^3
      - Soma de dois vetores seria N
    - Considerar porções sequenciais e paralelas dos códigos
  - Determinar o tempo de cada operação básica:
    - Empiricamente em função do hardware usado
  - Devem ser consideradas:
    - Replicações de computação,
    - Plataformas heterogêneas,
    - Custo de acesso às memórias (quando remotas)

Avaliação de Desempenho: métricas

- **T_{comm}** é o somatório do custo **T_{msg}** de todas as mensagens enviadas

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

  - T_s Tempo de *Startup*
    - Tempo do protocolo de comunicação e início da transferência
  - T_w Tempo de transferência de uma palavra
  - Q Tamanho Msg em número de palavras

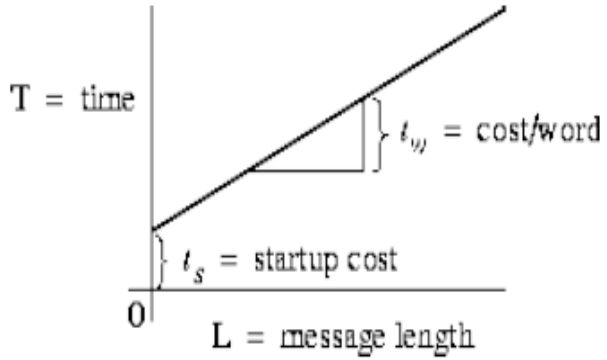


Figure 3.3: *Simple communication cost model:* $T_{msg} = t_s + t_w L$. *In this plot of time versus message length, the slope of the line corresponds to the cost per word transferred and the y-intercept to the message startup cost.*

Foster (1994)

- O **custo da comunicação em memória compartilhada** é o custo da busca/escrita de dados da/para memória

Avaliação de Desempenho: métricas

- Medição do desempenho de uma rede local com máquinas assíncronas
  - ***Round-Trip Time*** (RTT) ou ***Ping-Pong***
    - Processo 0 transmite uma msg para uma máquina remota (*ping*)
    - Processo 1 na máquina remota recebe a msg e a retransmite de volta (*pong*)
  - Transmite com diferentes tamanhos de mensagem (0 Bytes em diante)

- Saltos em diferentes redes também podem ser computados no caminho
  - Enviadas msgs para destinos na própria rede e para máquinas fora dela

- Medições muito curtas precisam **reduzir sobrecarga** da própria medição

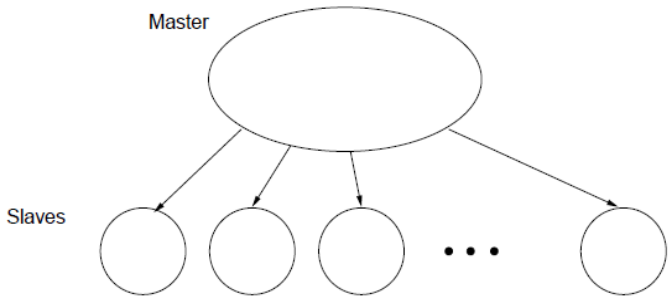
```
...
tmp = tempo();
start = tempo();
/* computação a medir tempo */
end = tempo();
overhead = start – tmp;
total-time = end – start – overhead;
...
```

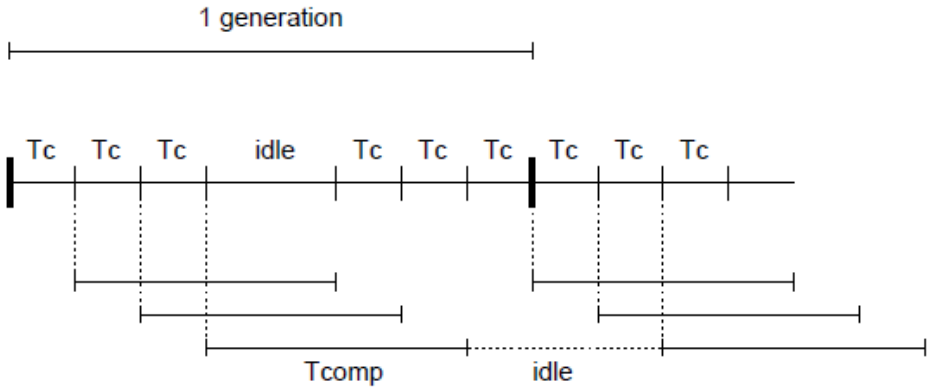
Avaliação de Desempenho: métricas

- Ociosidade T_{idle} é o tempo do processo parado, esperando algo
  - Por dados ou pelo término do processamento remoto
- Falta de balanceamento causa espera por término de processamento remoto
- Falta de dados ocorrem quando precisa sincronização ou comunicação

- Para evitar: sobreposição de computação e comunicação
  - Geração de múltiplas threads mantém CPU processando
  - Primitivas não bloqueantes

- T_{idle} pode ser estimado com base em T_{comp} e T_{comm}





$$idle = T_{comp} - (S-1)T_c$$

Cantú-Paz (1997)

Avaliação de Desempenho: métricas

- *Speed up* (Sp) e Eficiência (E) ... **relembrando**...
  - *Speedup*

    Absoluto (considera melhor algoritmo sequencial conhecido):

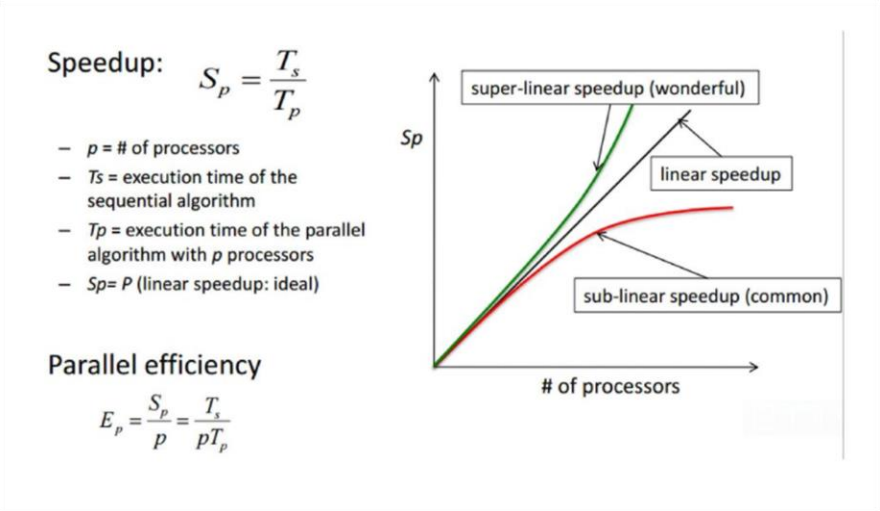
    $Sp = T_{seq} / T_{par\_p}$ (sobre p processadores)

    Relativo (considera programa paralelo em apenas um processador):

    $Sp = T_{par\_1} / T_{par\_p}$

  - Eficiência

    $E = Sp / p$ (p é o número de processadores)



Fonte Figura: https://slideplayer.com/slide/13453241/

Avaliação de Desempenho: métricas

- Lei de Amdahl
  - Com uma **carga W fixa**, o limite do Sp é 1 / α, onde **α é o percentual sequencial** da carga de trabalho (0 >= α <= 1)

    Sp = (W / (αW+(1-α)(W/n))

    …

    n / (1 + (n-1) α)

    …

    1 / α, quando n tende ao infinito
  - Com *overheads* (comunicação, ociosidade, replicação da computação)

    Sp = (W / (αW+(1-α)(W/n) + T0)

    …

    1 / (α + (T0/W))

    …

    1 / α, quando n tende ao infinito

- Para Amdahl maiores limitações vêm de custos de
  - comunicação, ociosidade, replicações e porções sequenciais
- Algoritmo paralelo deve maximizar porções paralelas enquanto diminui a sobrecarga gerada pela computação paralela

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.

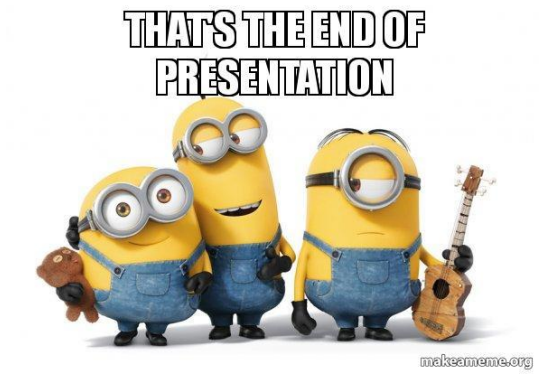
Cantú-Paz, E.; *Designing Efficient Master-Slave Parallel Genetic Algorithms*. University of Illinois at Urbana-Champaign, Report Nr. 97004, 1997.

Grama et al. (2003)

Avaliação de Desempenho em Programas Concorrentes

Paulo Sérgio Lopes de Souza
pssouza@icmc.usp.br

Universidade de São Paulo / ICMC / SSC – São Carlos
Laboratório de Sistemas Distribuídos e Programação Concorrente



CNPq Conselho Nacional de Desenvolvimento Científico e Tecnológico
CAPES
FAPESP