Speedup e Eficiência

Imagem tomada de http://www.deviantart.com/tag/bikedrawing?offset=34

Speedup e Eficiência

- $T_{Serial} = T_1$ = tempo para executar em um processador
- T_P = tempo para executar em P processadores
- *Speedup* = $S = T_S/T_P$
  - A redução relativa no tempo para completar determinada tarefa
  - No caso ideal S = P (SPEEDUP LINEAR)
    - Ex. 4 processos => speedup no melhor caso=4.
  - Usualmente S < P
  - Mas é possível encontrar S > P!!
- Eficiência = $E = S/P = T_S/(PT_P)$
  - E perfeita = 1
  - Difícil de conseguir!!

Efficiency of a parallel program

$$E = \frac{S}{p} = \frac{\left(\frac{T_{serial}}{T_{parallel}}\right)}{p} = \frac{T_{serial}}{p \bullet T_{parallel}}$$

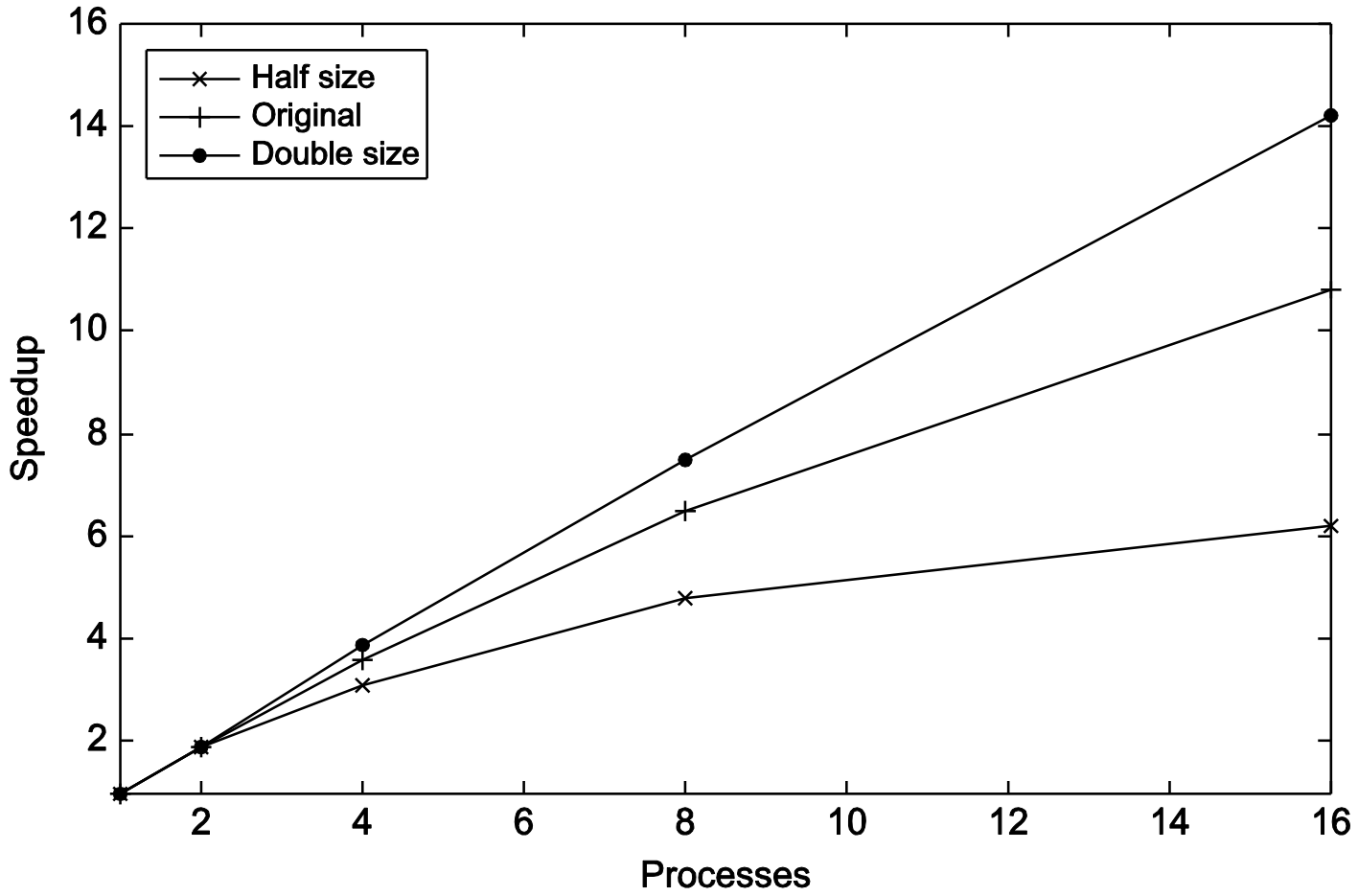
Speedups and efficiencies of a parallel program

p	1	2	4	8	16
S	1.0	1.9	3.6	6.5	10.8
$E = S/p$	1.0	0.95	0.90	0.81	0.68

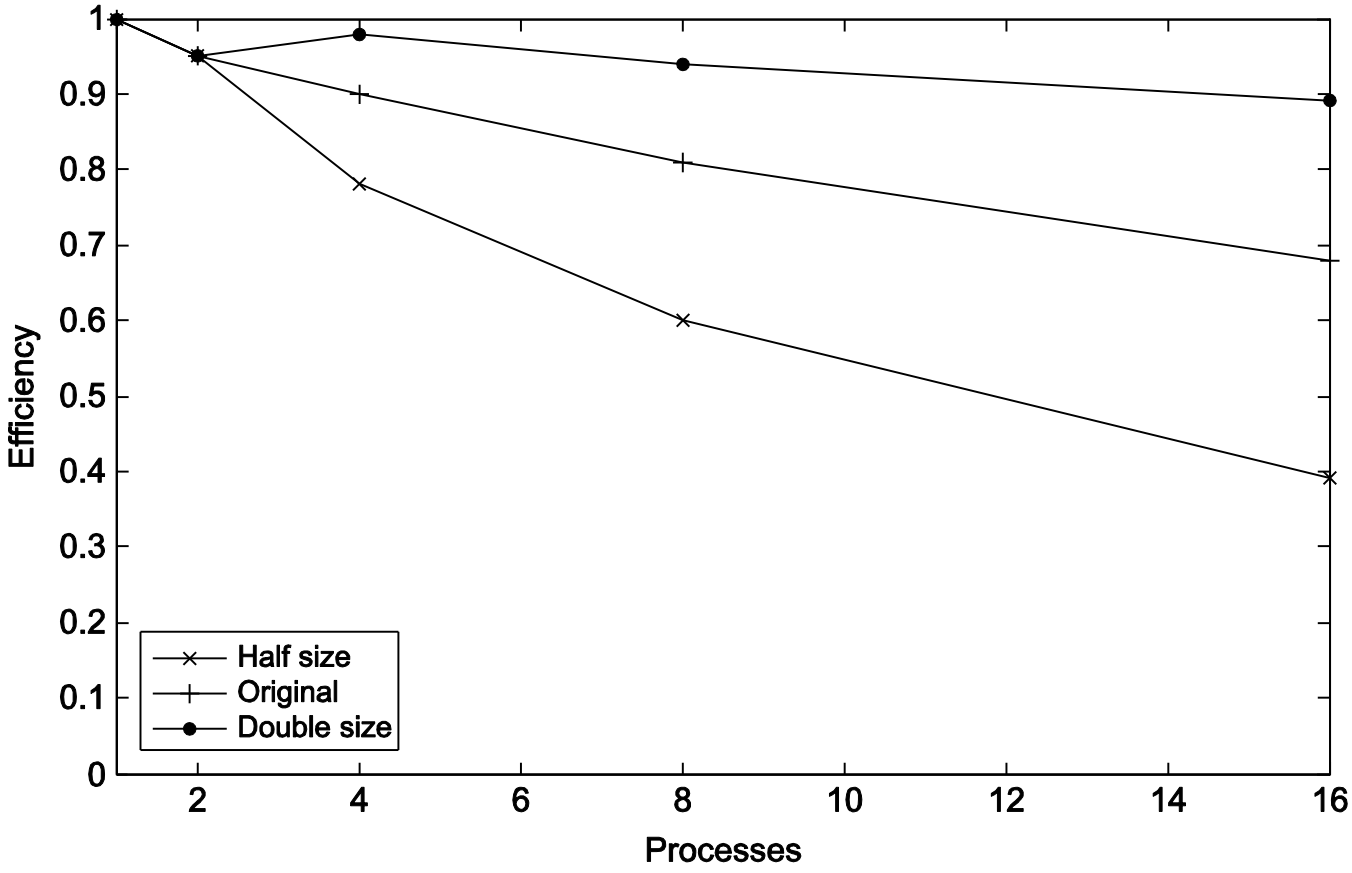
Speedups and efficiencies of parallel program on different problem sizes

	p	1	2	4	8	16
Half	S	1.0	1.9	3.1	4.8	6.2
	E	1.0	0.95	0.78	0.60	0.39
Original	S	1.0	1.9	3.6	6.5	10.8
	E	1.0	0.95	0.90	0.81	0.68
Double	S	1.0	1.9	3.9	7.5	14.2
	E	1.0	0.95	0.98	0.94	0.89

Speedup



Efficiency



Effect of overhead

$$T_{parallel} = T_{serial} / p + T_{overhead}$$

Lei de Amdahl

- A menos que virtualmente todo um programa serial esteja paralelizado, a aceleração possível será muito limitada - independentemente do número de núcleos disponíveis.

Exemplo

- Assuma que:
  - conseguimos paralelizar 90% de um programa serial.
  - A paralelização é “perfeita” independente do número de cores *p* usados.
  - $T_{serial} = 20$ segundos
- Então:
  - O tempo de execução da fração paralelizável é:

$$0.9 \times T_{serial} / p = 18 / p$$

Exemplo (cont.)

- Tempo de execução da fração não paralelizável é

$$0.1 \times T_{serial} = 2$$

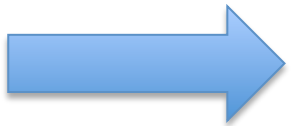
- Tempo de execução paralelo total é:

$$T_{parallel} = 0.9 \times T_{serial} / p + 0.1 \times T_{serial} = 18 / p + 2$$

Exemplo (cont.)

- Speed up

$$S = \frac{T_{serial}}{0.9 \times T_{serial} / p + 0.1 \times T_{serial}} = \frac{20}{18 / p + 2}$$



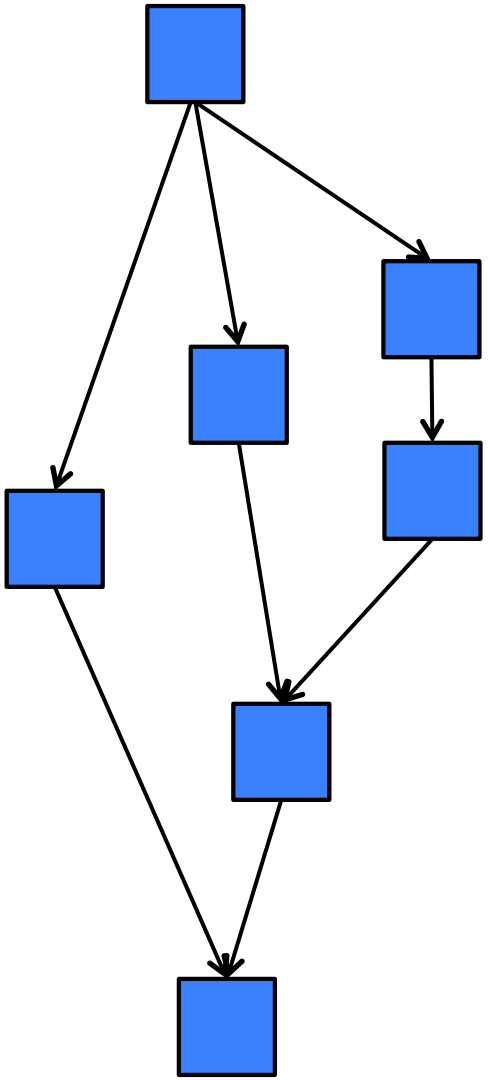
P -> ∞ => s-> ??

Conseqüências

- Devemos nos esforçar para expor tanto paralelismo quanto possível dentro do aplicativo
  - pode envolver reescrever / refatorar
- Se frações significativas do código permanecerem em série, a eficácia da aceleração (devida ao paralelismo) será limitada (lei de Amdahl)

DAG: Modelo de Computação

- Program is a directed acyclic graph (DAG) of tasks
- The hardware consists of workers
- Scheduling is *greedy*
  - No worker idles while there is a task available.



14

Departures from Greedy Scheduling

- Contended mutexes.
  - Blocked worker could be doing another task

Avoid mutexes, use wait-free atomics instead.

- One linear stack per worker
  - Caller blocked until callee completes

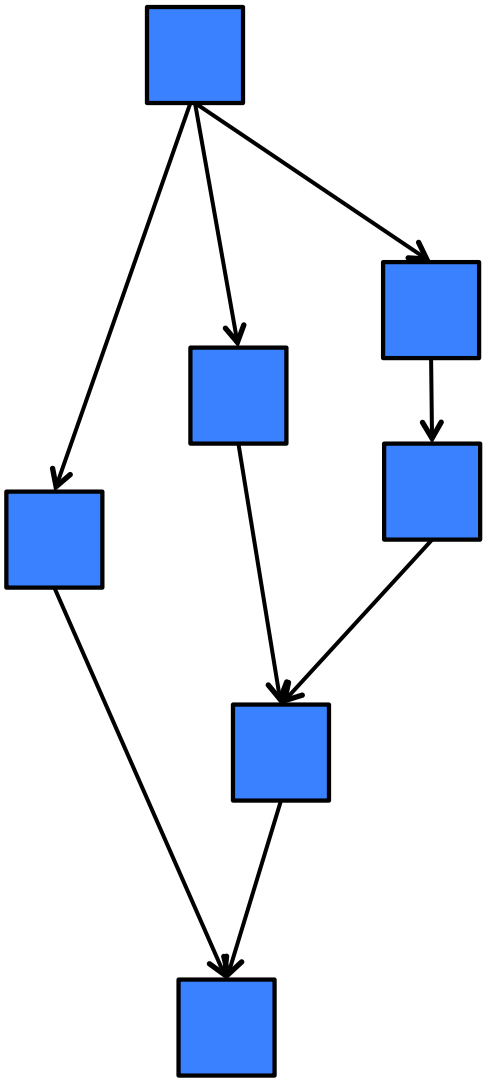
Intel® Cilk Plus has cactus stack.

Intel® TBB uses continuation-passing style inside algorithm templates.

Modelo Work-Span

- Assuma uma unidade de trabalho por nó.

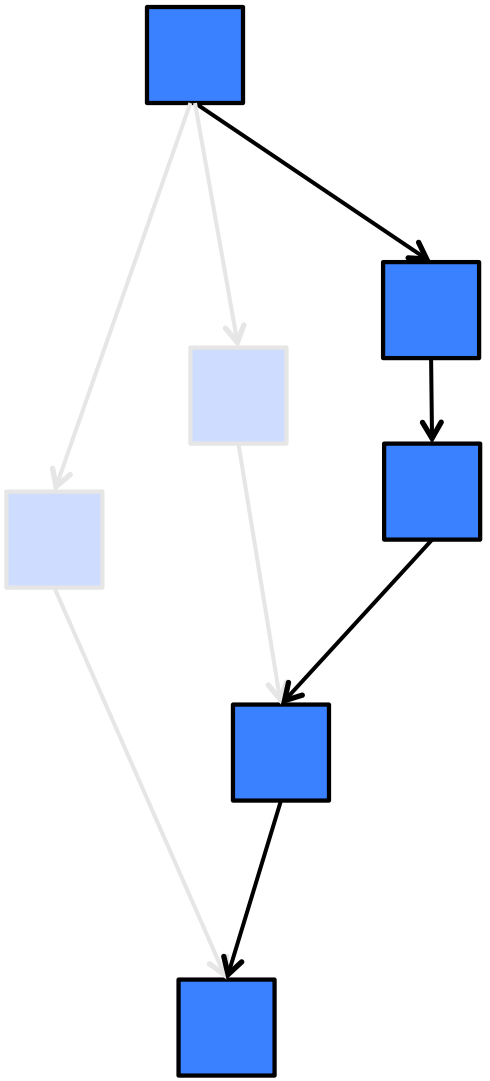
- T_P = tempo para executar com P trabalhadores
- T_1 = *work (trabalho)*
  - =tempo para execução serial
  - Soma de todo o trabalho
- T_∞ = *span*
  - Tempo para caminho crítico (maior caminho)



16

Exemplo Work-Span

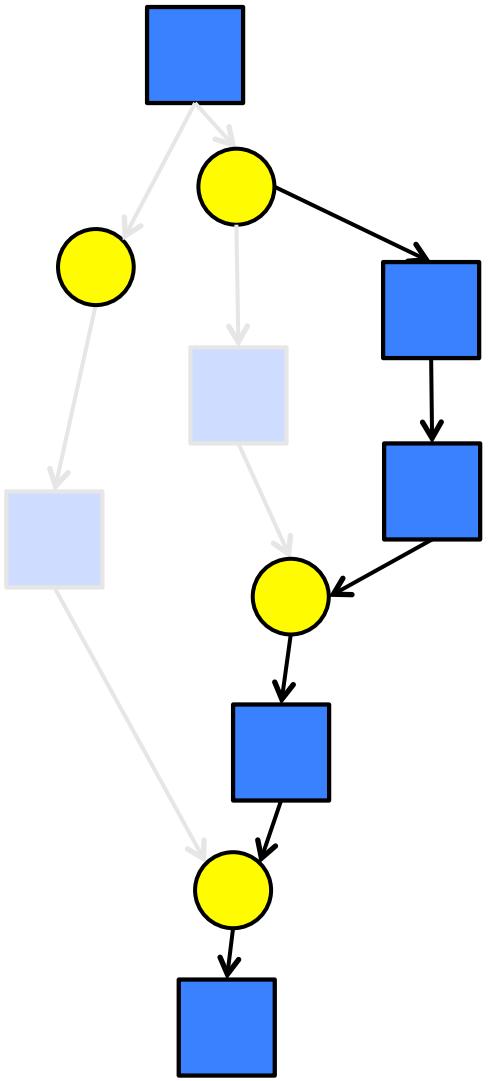
T_1 = *work* = 7
T_∞ = *span* = 5



17

Burdened Span

- Includes extra cost for synchronization
- Often dominated by cache line transfers.

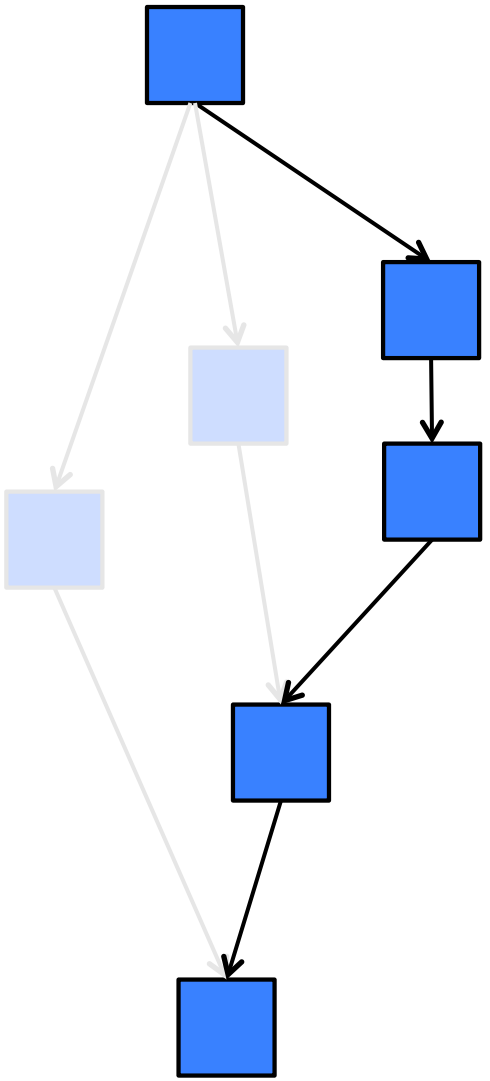


18

Lower Bound on Greedy Scheduling

Work-Span Limit

$$\max(T_1/P, T_\infty) \leq T_P$$

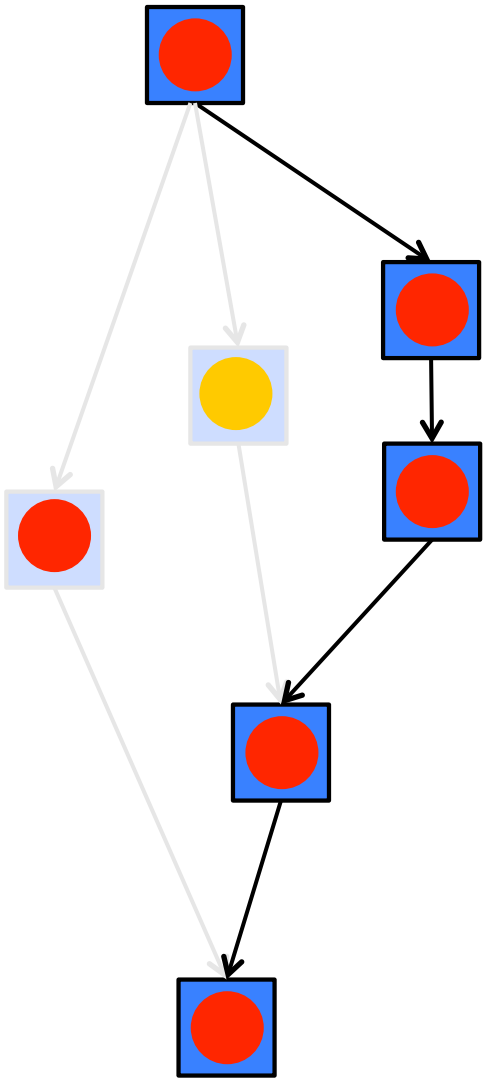


19

Upper Bound on Greedy Scheduling

Brent's Lemma

$$T_P \leq (T_1 - T_\infty)/P + T_\infty$$



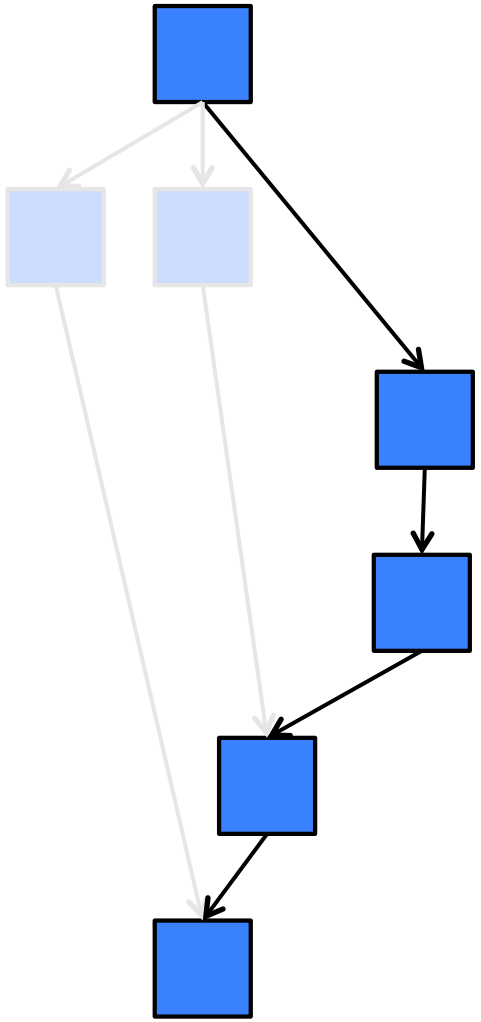
20

Applying Brent's Lemma to 2 Processors

$T_1 = 7$

$T_\infty = 5$

$T_2 \leq (T_1 - T_\infty)/P + T_\infty$
$\leq (7-5)/2 + 5$
≤ 6



21

Exemplos de dependências que forçam a execução serial

Dependência Verdadeira ou Dependência de Fluxo (RAW - Read- After-Write):

No código do exemplo as instruções presentes na linha 1 e 2 não podem ser executadas ao mesmo tempo uma vez que a instrução 2 precisa do valor de A computado na instrução 1.

1 A = B + C
2 D = A + 2
3 E = A * 3

Anti-dependência (WAR - Write After Read):

- No código apresentado a instrução presente na linha 1 usa o valor de B antes da instrução da linha 2 atribuir um novo valor a B. Devido a isso essa ordem deve ser mantida para que o valor de B utilizado seja o valor antigo, não o computado na linha 2.
    - **1**A = B + C
    - **2** B = D *2

Dependência de Saída (WAW - Write After Write)

- No exemplo a instrução 1 e a instrução 3 estão atribuindo valor a variável A. Dependendo da ordem de execução das instruções, o valor resultante de A pode ser errado.

  - **1** A = B + C
  - **2** D = A + 2
  - **3** A = E * F

Dependência de Controle ou Dependência Procedural

- Quando ocorrem desvios condicionais em um código, como pode ser visto no exemplo o valor de A utilizado pela instrução presente na linha 3 tanto pode ser o que foi gerado pela instrução da linha 1 ou pela instrução 2, dependendo do valor de X.

```
1 A = B + C
  if (X >= 0) then
2    A= A+ 2
  end if
3 D = A * 2.1
```

(a)

- Considerando as dependências de dados dentro de laços, como pode ser visto no exemplo na figura (b) existe uma dependência de dados entre a instrução da linha 2 e a instrução da linha 1. Porém essa dependência ocorre na mesma iteração do laço, visto que o valor do elemento A produzido na instrução 1 será utilizado na instrução 2.

```
do i = 2,N
1    A(i) = B(i) + C(i)
2    D(i) = A(i)
end do
```

(b)

- O contrário pode ser visto no exemplo da figura (c) a dependência entre a instrução 2 e 1 permanece, porém, para qualquer iteração i a instrução 2 utilizará o valor do elemento de A produzido na iteração anterior. O que impedirá a paralelização desse laço.

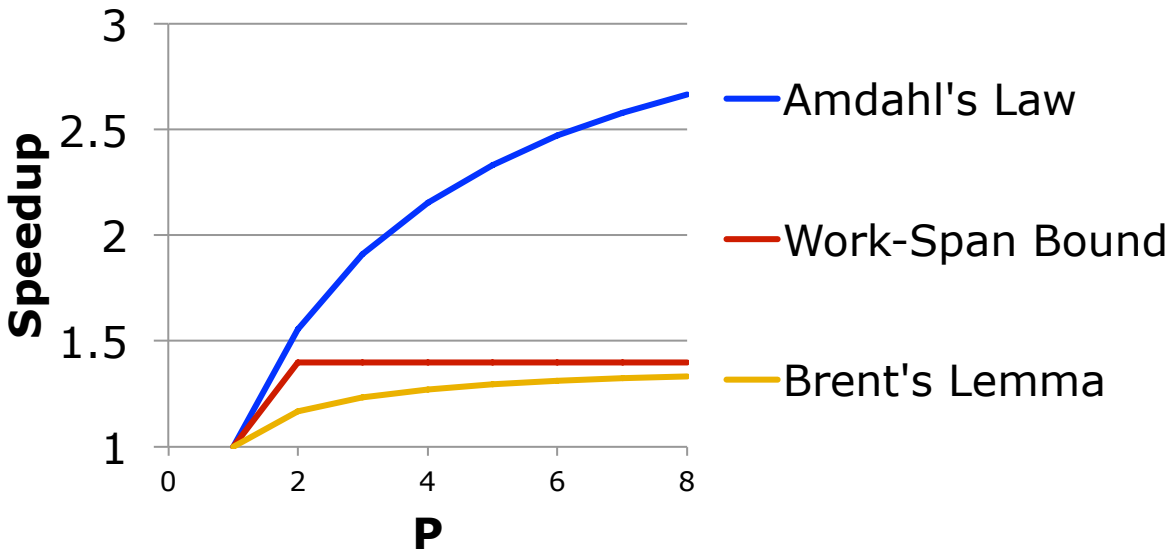
```
do i = 2,N
1  A(i) = B(i) + C(i)
2  D(i)= A(i-1)
  end do
```

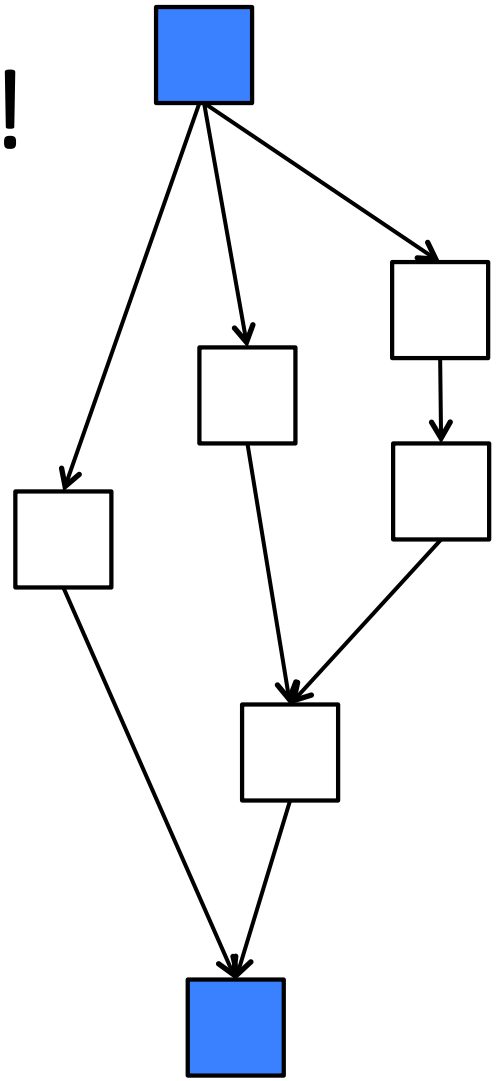
(c)

Amdahl foi um Otimista!

Amdahl's Law

$$T_{serial} + T_{parallel}/P \leq T_P$$





Escalabilidade

- Em geral, um problema é **escalável** se ele pode lidar com tamanhos cada vez maiores de problemas.
- Se ao aumentarmos o número de processos / threads se mantém a eficiência fixa sem aumentar o tamanho do problema, o problema é **fortemente escalável**.
- Se mantivermos a eficiência fixada aumentando o tamanho do problema na mesma taxa que aumentamos o número de processos / threads, o problema é **fracamente escalável**.

Estimando o tempo de execução

- Escalabilidade requer que T_∞ seja dominado por T_1.

$$T_P \approx T_1/P + T_\infty \quad \text{if} \quad T_\infty \ll T_1$$

- Aumentar o trabalho danifica a execução paralela proporcionalmente.
- O span impacta a escalabilidade, ainda para P finito.

Slack (folga) Paralelo

- Paralelismo potencial supera o paralelismo do hardware-> slack paralelo (folga)
- Aumentar o paralelismo faz com que cada tarefa faça menos trabalho
- Em um ponto reduzir o trabalho por tarefa->aumentar o slack paralelo-> reduz o desempenho
- Suficiente paralelismo implica em speedup linear

$$T_P \approx T_1/P \quad \text{if} \quad T_1/T_\infty \gg P$$

Linear speedup — *Parallel slack*

Definitions for Asymptotic Notation

- $T(N) = O(f(N)) \equiv T(N) \leq c \cdot f(N)$ for some constant *c*.
- $T(N) = \Omega(f(N)) \equiv T(N) \geq c \cdot f(N)$ for some constant *c*.
- $T(N) = \Theta(f(N)) \equiv c_1 \cdot f(N) \leq T(N) \leq c_2 \cdot f(N)$ for some constants *c*1 and *c*2.

Quiz: If $T_1(N) = O(N^2)$ and $T_\infty(N) = O(N)$, then T_1/T_∞ = ?

a. O(N)

b. O(1)

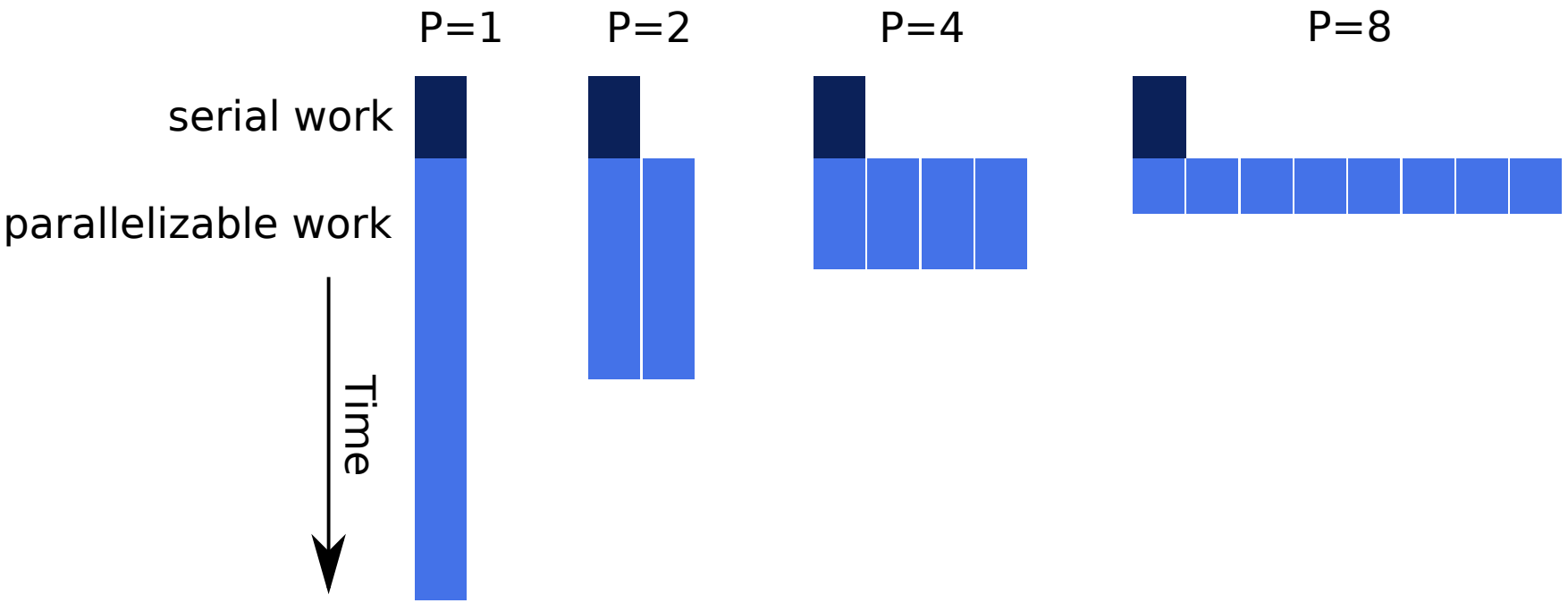
c. O(1/N)

d. all of the above

e. need more information

Amdahl vs. Gustafson-Barsis

- Amdahl



Amdahl vs. Gustafson-Barsis

- Gustafson-Barsis