Desempenho de Aplicações Paralelas

Métricas, Modelo Work-Span e Armadilhas do Paralelismo

Desempenho

Paralelismo é importante somente se o desempenho é um requisito fundamental de uma aplicação.

Métricas

- Tempo de Resposta
  - Reduzir o tempo total para computar um único resultado (latência).
- Vazão
  - Aumentar a taxa na qual uma série de resultados podem ser computados.

Métricas

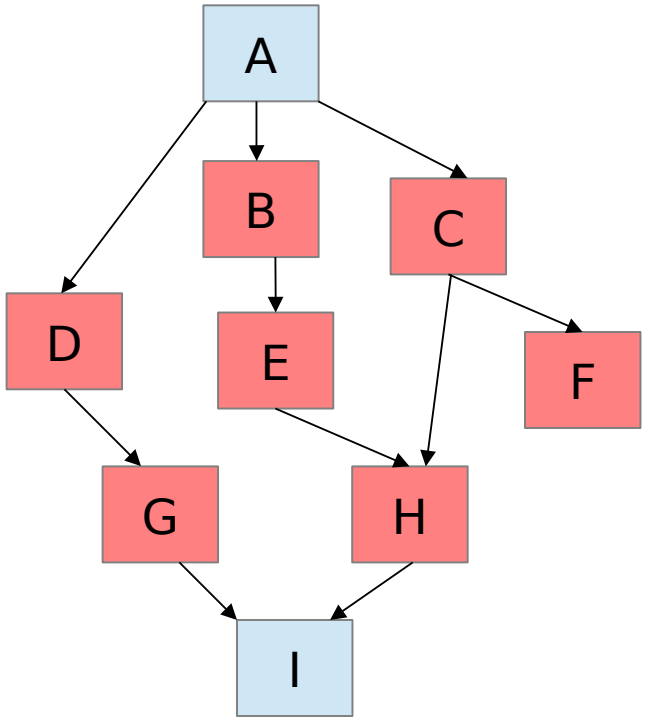
Speedup

- Compara o tempo para resolver um problema computacional em 1 unidade de processamento (T_{seq}) versus em P unidades (T_{par}), ou seja T_{seq} / T_{par}

Eficiência

- Speedup dividido pelo número *P* de unidades de processamento, ou seja, S_p / P.
- É uma medida do retorno de investimento em HW.

Exemplo



$P = 3$

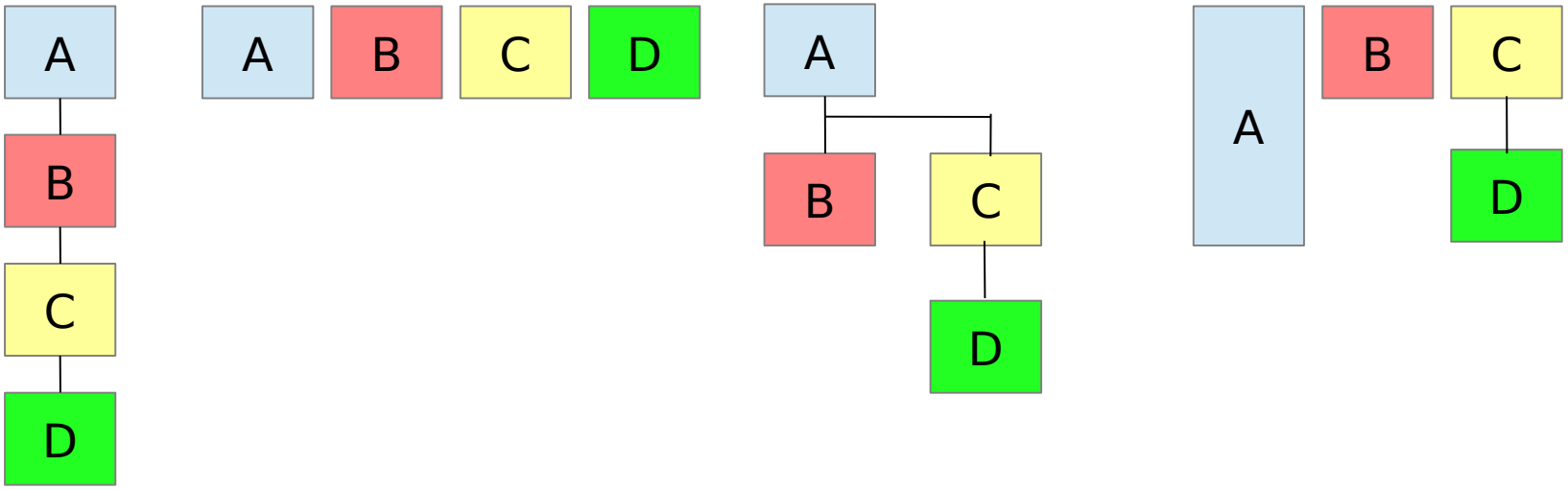
$T_{seq} = 9$

$T_{par} = 6$

*Considere que cada tarefa gaste uma unidade de tempo.

- Speedup (S_p) = T_{seq} / T_{par} = 9/6 = 1.5
- Eficiência = S_p / P = 1.5/3 = 0.5

Escalabilidade



- O desempenho melhora a medida em que se aumenta o número de elementos de processamento?
- Qual o número máximo de elementos de processamento necessários para atingir o maior speedup possível nas aplicações acima?

Limitações do Paralelismo

Limitações do Paralelismo

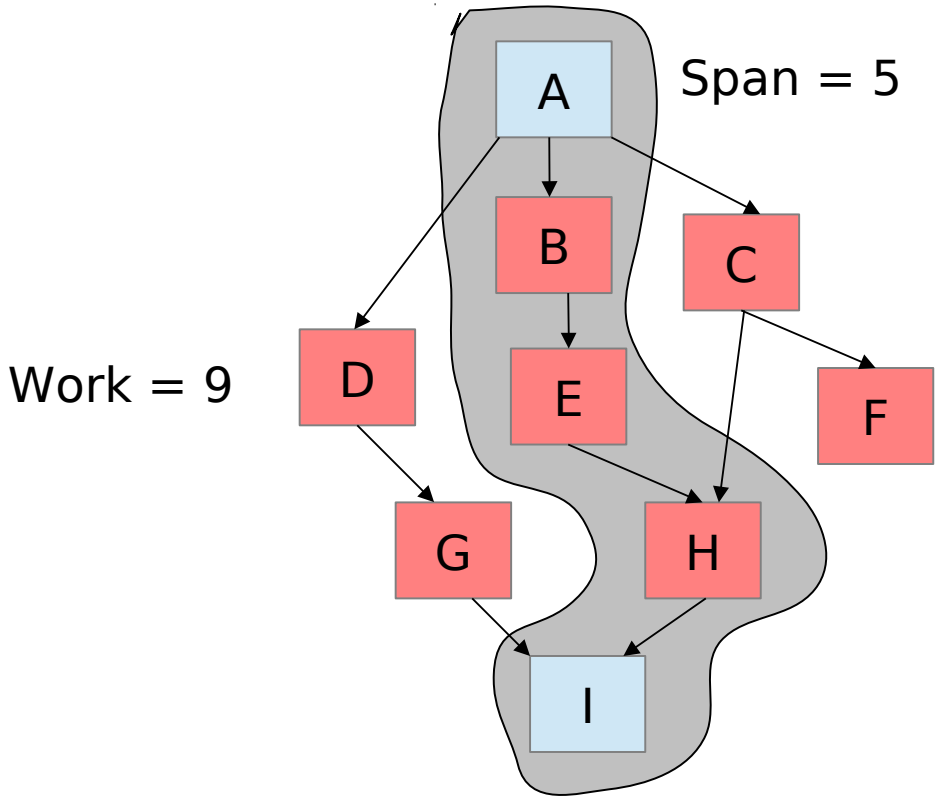
- Nem sempre a computação é um gargalo de uma aplicação (ex: memória).
- Escalabilidade depende do caminho crítico (span*) da aplicação.

Exemplo: Preparar uma pizza

- Fazer a massa, fatiar queijo, cortar cebola, assar a pizza etc.
- Qual a tarefa mais longa? Pode ser feita em paralelo?

*Span = tempo para executar a maior cadeia de tarefas que devem ser sequenciais.

Modelo Work-Span



- Tempo ≥ max(span, work/P) ≥ span
- Speedup ≤ work / span = 9/5 = 1.8
- E se Work = 21 e P = 3, qual o S_p e T_{par}? E P = ∞?

Modelo Work-Span

Observações

- Para um número infinito de processadores, o tempo de execução é no mínimo o próprio span.
- Por outro lado, ao aumentar-se o tamanho do “work”, o speedup aumenta, caso o span (parte sequencial) se mantenha a mesmo.
- O modelo Work-Span escala com o aumento da quantidade de trabalho.

Lei de Amdahl

A Lei de Amdahl diz que:

- $T_{seq} = W_{seq} + W_{par}$
- $T_{par} \geq W_{seq} + W_{par}/P$

Ou seja, a fração sequencial da aplicação limita o máximo desempenho.

Está ultrapassada porque considera que o work é estático, mas na prática o W_{par} aumenta com o aumento do poder computacional, enquanto W_{seq} em menor escala, se tornando desprezível em algum momento.

Lei de Amdahl

No exemplo anterior,

- $T_{seq} = W_{seq} + W_{par} = 2 + 7 = 9$
- $T_{par} \geq W_{seq} + W_{par}/P = 2 + 7/3 = 4.3$

Ou seja, Amdahl é otimista porque não considera o span.

- $S_d \leq T_{seq} / T_{par} = 9/4.3 = 2.1$
- Speedup maior que os 1.8 obtidos na prática.

Dependência de Tarefas

Dependências entre tarefas paralelas limitam o desempenho, pois exigem sincronização.

- Pouca sincronização, aumenta o não determinismo da aplicação.
- Muita sincronização diminui a escalabilidade da aplicação ou pode causar deadlock.

Programação paralela leva a novas “armadilhas” não existentes na programação sequencial.

Armadilhas do Paralelismo

Condição de Disputa

Exclusão Mútua

Deadlock

Contenção por Locks

Localidade de Dados

Balanceamento de Carga

Sobrecarga

Condição de Disputa

Ocorre quando tarefas concorrentes acessam a mesma posição de memória sem a devida sincronização, com pelo menos um dos acessos sendo uma escrita.

```
X = 0;
```

Tarefa 1

```
a = X;
a += 1;
X = a;
```

Tarefa 2

```
b = X;
b +=  2;
X = b;
```

Exemplo

Situação 1: (ordem correta)

Tarefa 1

a = 0;
A = 0 + 1;
X = 1;

Tarefa 2

b = 1;
b = 1 + 2;
X = 3;

Tempo

Situação 2: (ordem errada)

Tarefa 1

a = 0;
A = 0 + 1;
X = 1;

Tarefa 2

b = 0;
b = 0 + 2;
X = 2;

Tempo

Condição de Disputa

Não pode assumir ordem global, ou seja, operações são executadas em uma determinada ordem.

- HW e compilador podem re-ordenar instruções.

Erros não são facilmente detectados.

- Bug pode não aparecer na fase de testes, mas aparecer na versão do cliente.

Exclusão Mútua

Evita que duas tarefas acessem um mesmo dado em paralelo.

É geralmente implementada com um *lock ou mutex*.

Um *lock* possui duas operações atômicas:

- Lock: trava o lock, se ele estiver destravado. Caso contrário, a tarefa espera até que seja destravado.
- Unlock: destrava o lock.

Exemplo

```
Mutex m;
X = 0;
```

Tarefa 1

```
m.lock();
a = X;
a += 1;
X = a;
m.unlock();
```

Tarefa 2

```
m.lock();
b = X;
b +=  2;
X = b;
m.unlock();
```

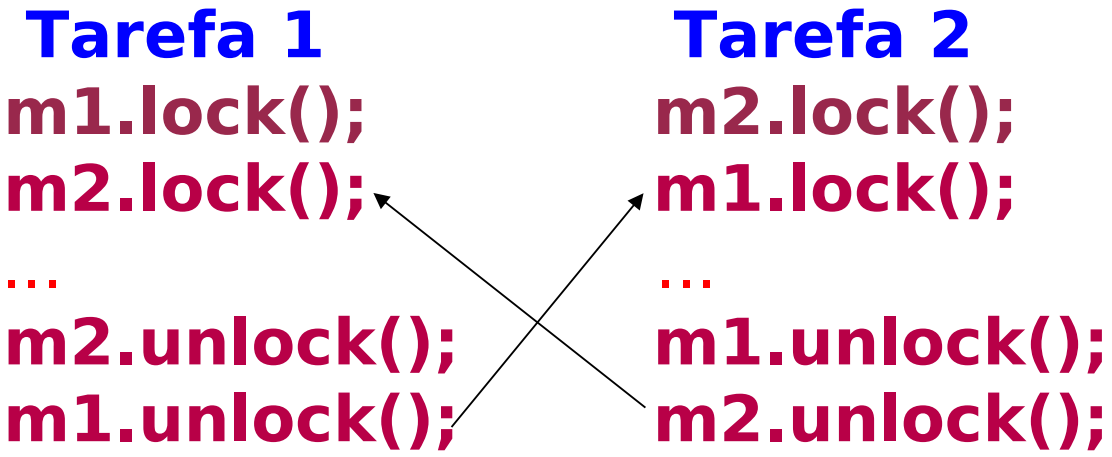
Apenas uma das tarefas atualiza a variável X de cada vez.

Mutexes serializam partes do código, por isso, devem ser usados como último recurso.

Deadlock

Ocorre quando pelo menos duas tarefas esperam pela outra e não pode continuar enquanto a outra não terminar.

Mutex m1, m2;



Como Evitar o Deadlock?

Adquirir os locks sempre na mesma ordem (ex: m1 → m2 → m3 ...).

Adquirir apenas um lock ao mesmo tempo.

Backoff: Ao adquirir um conjunto de locks, se um dos locks está travado, libere todos os outros imediatamente.

Evitar locks, utilizar padrões paralelos que tornem o uso de locks transparente.

Contenção por Locks

Um lock é um potencial gargalo se tarefas acessam-no frequentemente. Ao aumentar-se o número de tarefas, o lock impede que a aplicação escale.

Como evitar? Fine-grain locking.

- Divide uma área de memória em sub-áreas menores (ex: linhas e uma matriz)
- Cada sub-área é protegida por um lock ou invés de um lock para toda a áerea.

Localidade de Dados

Exploração da Localidade de Dados Espacial e Temporal é essencial para a escalabilidade de aplicações paralelas.

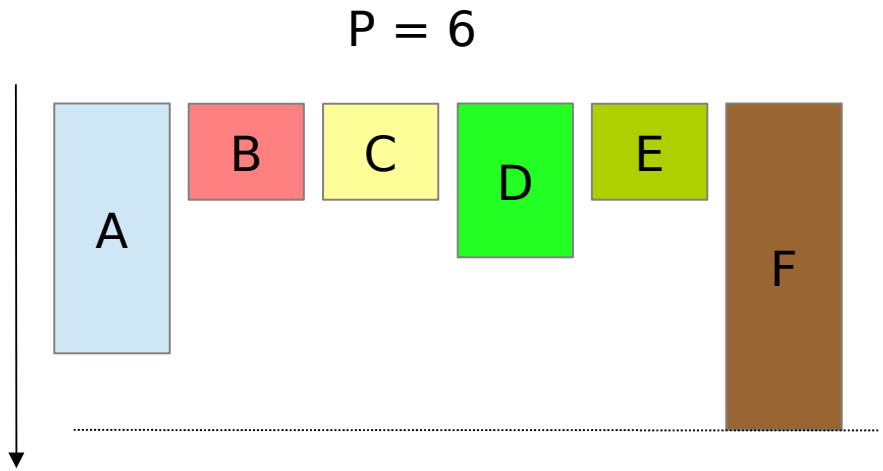
Como aumentar a localidade?

- Processar dados exaustivamente antes de carregar um novo conjunto de dados.
  - Aumenta acerto de caches.
  - Evita que os dados saiam do chip muitas vezes.

Balanceamento de Carga

As tarefas precisam ser distribuídas de forma que os elementos de processamento não fiquem ociosos.

Exemplo de desbalanceamento.



Como Balancear a Carga?

Decomposição de tarefas: dividir as tarefas em tarefas menores, maior que o número de processadores disponíveis.

Distribuí-las para os demais processadores ociosos.