Universidade Estadual de MS
Curso de Ciência da Computação
Disciplina de Programação Paralela e Distribuída

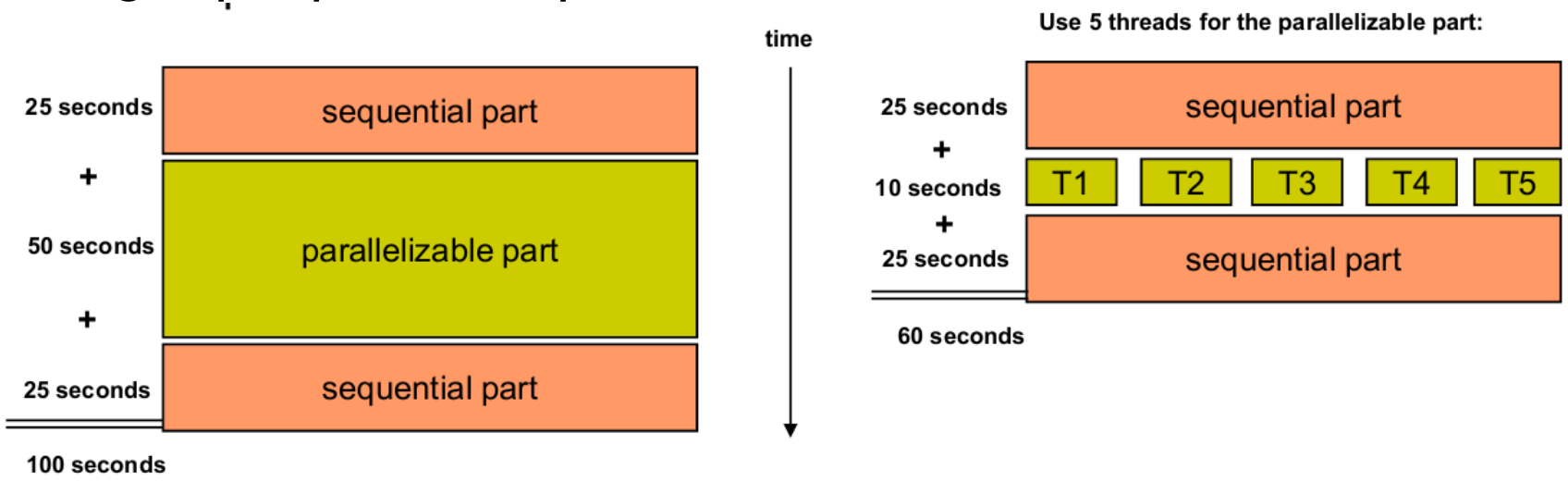
Limite do Desempenho na Escalabilidade

- Nem todos os programas são paralelos.
- Programas possuem partes sequenciais e paralelas.
- Parte sequencial:
  - Não pode ser paralelizada porque há uma dependência de dados.(parte vermelha)
- Parte paralela:
  - Não há dependência de dados (parte verde)

```
1   a = b + c;
2   d = a + 1;
3   e = d + a;
4   for (k=0; k < e; k++)
5       M[k] = 1;
```

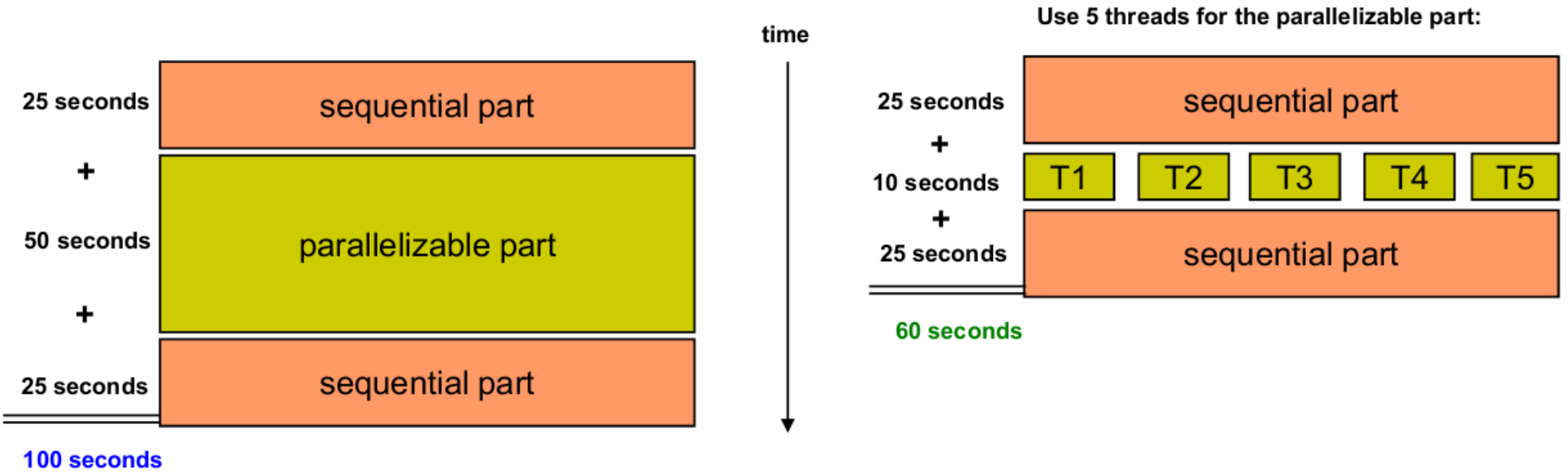
Lei de Amdahl

- A melhoria de desempenho adquirida usando algum modo mais rápido de execução é limitado pela fração de tempo que o modo mais rápido pode ser usado.
- Modo mais rápido de execução significa “paralelização do programa”
- O potencial de ganho (speedup) é definido pela fração do código que pode ser paralelizado.



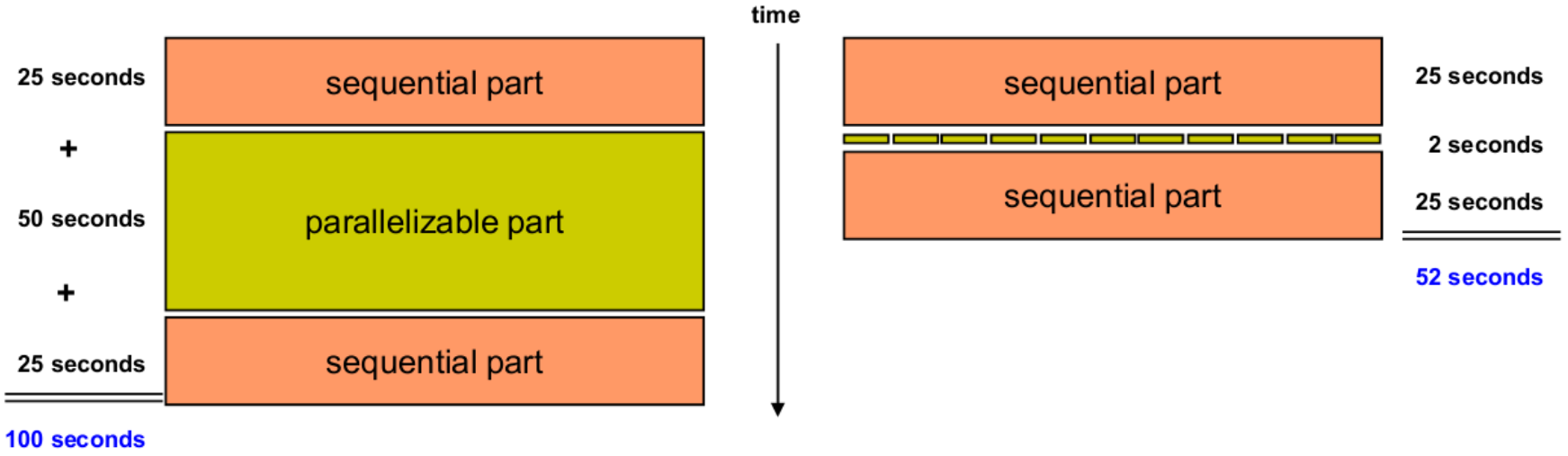
Lei de Amdahl

- Speedup = tempo de execução antigo / novo tempo de execução
- Speedup = 100 segundos / 60 segundos = 1,67
- A versão paralela é 1,67 mais rápido que a versão sequencial.



Lei de Amdahl

- Mesmo que se use mais threads na parte paralela, a parte sequencial permanecerá a mesma.
  - A parte sequencial do programa limita o speedup que se quer alcançar.
- Mesmo se teoricamente fosse possível reduzir a parte paralela para 0 (zero) segundos, o melhor speedup possível seria:
  - Speedup = 100 segundos / 50 segundos = 2



Exemplos

- Seja p a fração que pode ser paralelizada.
- Seja N o número de threads executando em paralelo.
- Speedup = $\frac{1}{(1 - p) + \frac{p}{N}}$
- Exemplos:
  - Para p = 0 (só parte sequencial).
    - $\text{speedup} = \frac{1}{1+\frac{0}{N}} = 1$
      - Não há speedup
  - Para p = 1
    - $\text{speedup} = \frac{1}{0+\frac{1}{N}} = N$
      - O número de processadores será o speedup
  - Para p = 0,75 e N = 8
    - $\text{speedup} = \frac{1}{0.25+\frac{0.75}{8}} = 2.91$
  - Para p = 0,75 e N infinito
    - $\text{speedup} = \frac{1}{0.25+\frac{0.75}{\infty}} = 4$

Gustafson-Barsis

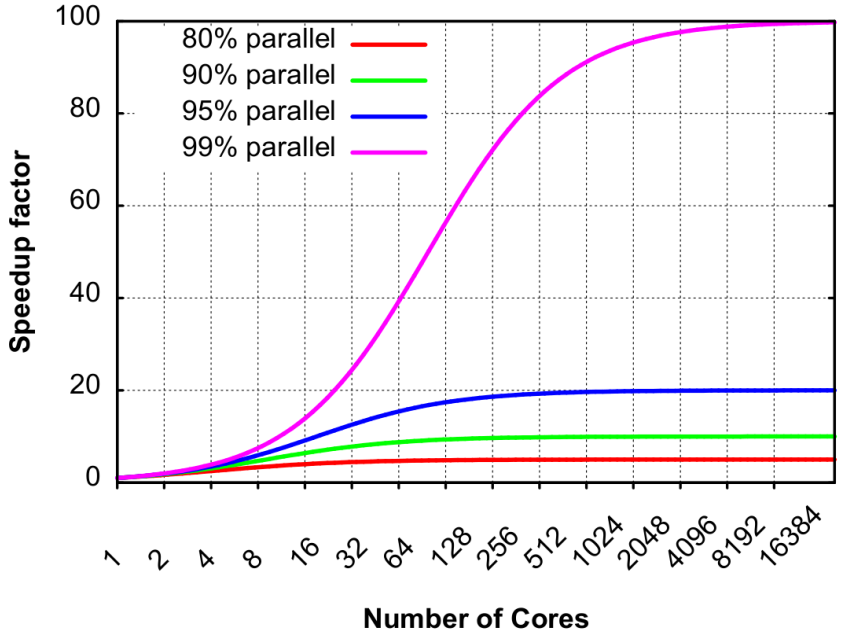
- N é o número de threads executando em paralelo
- np é a fração sequencial
- $$\text{speedup} = N - (N-1) \cdot np$$
- Para N = 256, p = 99% e np = 1%
-

$$\text{speedup}_{\text{Amdahl}} = \frac{1}{0.01 + \frac{99}{25600}} \approx 72$$

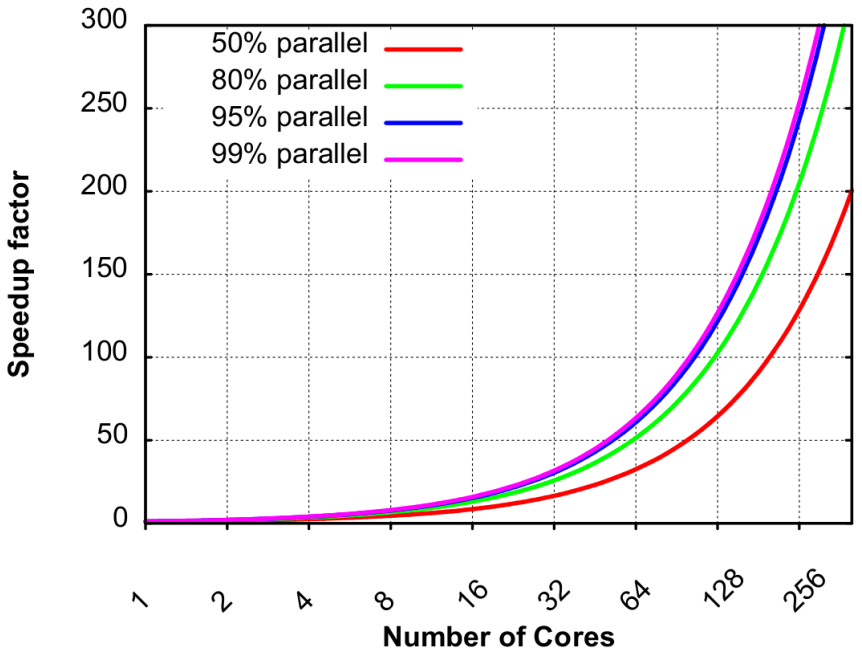
$$\text{speedup}_{\text{Gustafson}} = 256 - \frac{255}{100} \approx 254$$

Gustafson-Barsis

- Speedup Amdahl



- Speedup Gustfson-Barsis

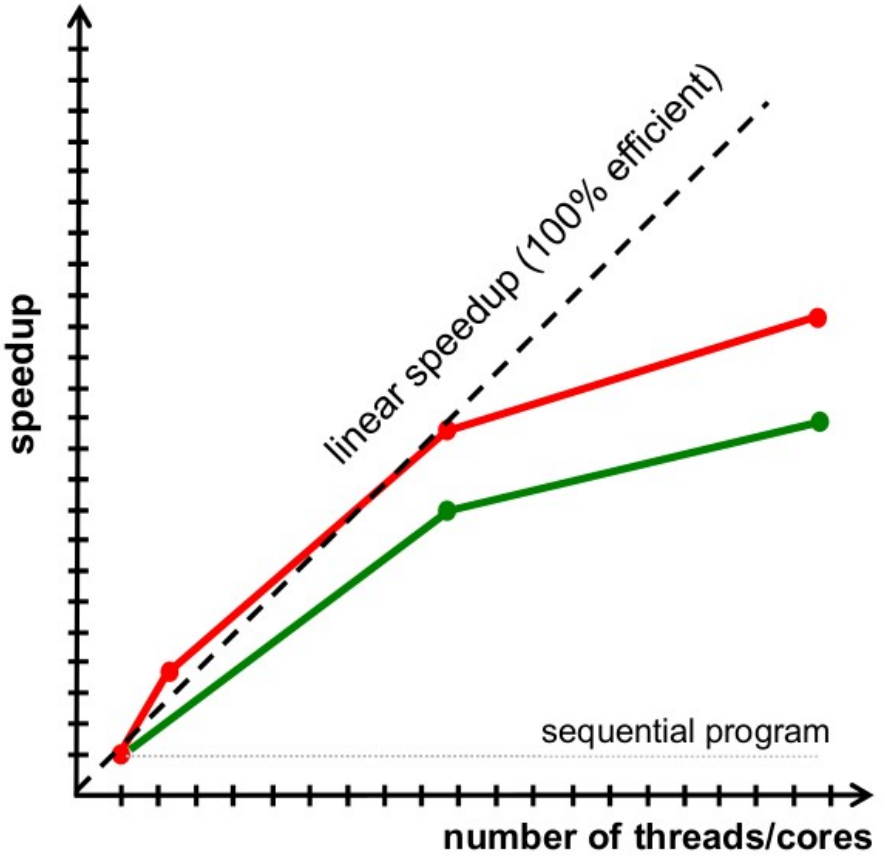


Amdahl e Gustafson-Barsis

- Questão fundamental: Dada uma aplicação, o speedup potencial pode ser limitado ou não?
  - A resposta está na parte sequencial da aplicação.
  - Se a fração da parte sequencial permanece constante conforme o número de núcleos disponíveis aumenta → a lei de Amdahl impõe na aplicação um limite superior possível no speedup.
  - Se a fração da parte sequencial da aplicação diminui com o acréscimo de mais núcleos → lei de Gustafson-Barsis.
- Em ambos os casos, a fração sequencial da aplicação precisa ser reduzida para se conseguir speedup maiores.

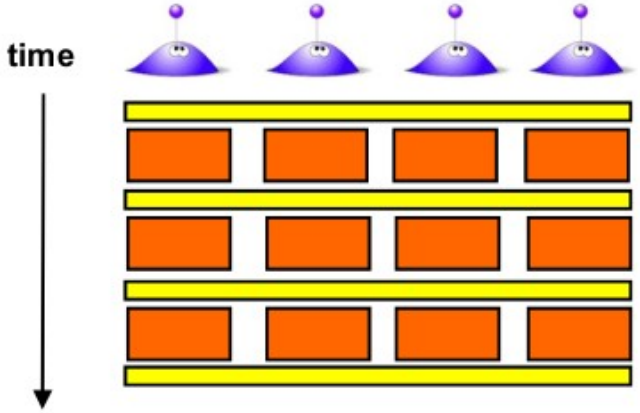
Escalabilidade do Desempenho

- O desempenho e a escala estão diretamente relacionados com o número N de threads/núcleos usados na parte paralelizável p.
- Ex.: Se dobrarmos o número de threads/núcleos, espera-se que o tempo de execução caia pela metade. Isto é chamado speedup linear.
- Entretanto, o speedup linear é um conceito ideal que dificilmente é encontrado.
- ----------
- O gráfico mostra as curvas desniveladas à medida que aumentamos o número de núcleos.
- Resultado de se manter o tamanho do problema constante:
  - qtde de trabalho por thread diminui
  - sobrecarga na criação de thread

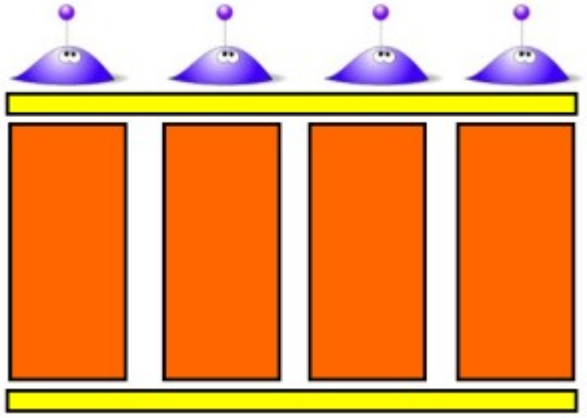


Balanceamento de Carga
Granularidade do Paralelismo

- Granularidade do paralelismo fino
- Pequena qtde de trabalho computacional entre comunicação / sincronização
- Baixa taxa de comunicação
- Alta comunicação / sobrecarga de sincronização
- Menor oportunidade de melhorar o desempenho.

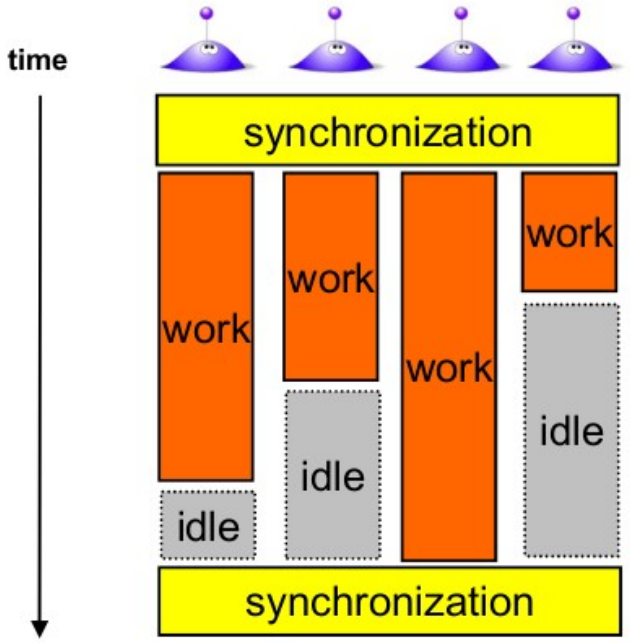


- Granularidade do paralelismo grosso
- Grande qtde de trabalho entre comunicação / sincronização
- Alta taxa de comunicação
- Baixa comunicação / sobrecarga de sincronização
- Melhor oportunidade de melhorar o desempenho.
- Difícil e balancear a carga.



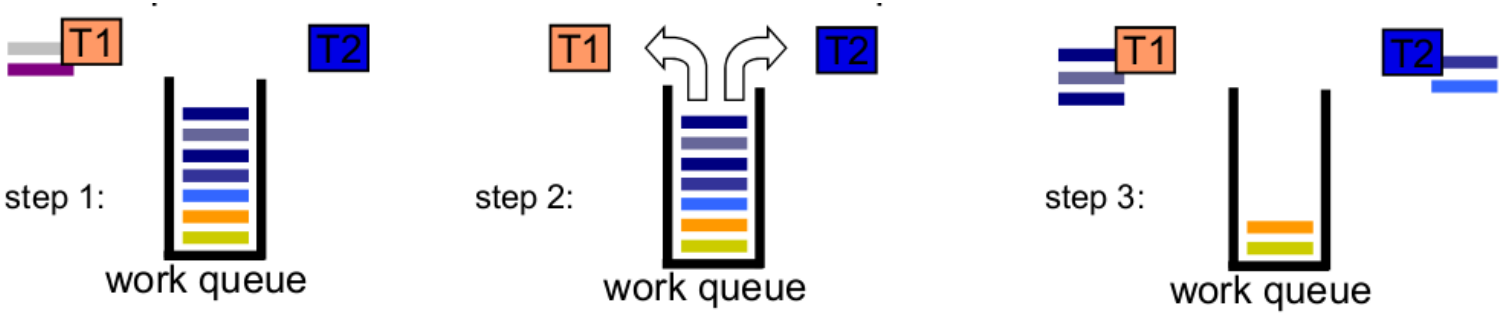
Problema no Balanceamento de Carga

- As threads que terminam antecipadamente precisam esperar pela thread com a maior carga de trabalho.
- Esse fenômeno leva a períodos de ociosidade e baixa utilização do processador.



Balanceamento de Carga

- Carga desbalanceada: o trabalho não fica uniformemente distribuído entre os núcleos. Subaproveitamento do paralelismo.
- Uma saída é pensar na atribuição de tarefas:
  - Atribuição estática: atribuição no momento da escrita do programa.
    - Não pode ser mudada em tempo de execução.
    - Mais propensa a desequilíbrios.
  - Atribuição dinâmica: atribuição em tempo de execução.
    - A quantidade de trabalho deve ser grande o suficiente para amortizar a sobrecarga.
    - Exemplo: Thread que buscam tarefes de uma fila de trabalho.



  - Com alocações flexíveis, o balanço de carga pode ser resolvido no design do programa.

4 origens de perda de desempenho

- A aceleração linear com processamento paralelo frequentemente não é alcançável pelos seguintes motivos:
  - Sobrecarga com computação sequencial desnecessária
  - Computação não paralelizável
  - Processadores ociosos
  - Contenção por recursos
- Qualquer outro motivo é um caso especial de um destes 4

Exemplo: perda de desempenho

- Tarefa: Pintar as quatro paredes de uma sala de aula.
  - Um único pintor levaria um dia inteiro para pintar todas as paredes e, levaria mais um dia para secar as paredes.
- Adicionando mais pintores permitirá um ganho no speedup, mas temos os 4 motivos de perda:
  - Sobrecarga na computação sequencial desnecessária.
    - Decidir sobre quais pintores irá pintar quais paredes da sala de aula.
  - Computação não paralelizável
    - Após o término da pintura, levará um dia para secar a tinta.
  - Processadores ociosos
    - Se utilizarmos 100 homens, eles terão que fazer fila para entrar na sala de aula para fazer o seu trabalho.
  - Contenção por recursos
    - Suponha que você tenha apenas um balde de tinta para ser compartilhado pelos pintores.

Perda 1: sobrecarga

- Qualquer custo causado na solução paralela mas não requerida pela solução sequencial é chamada sobrecarga.
- Exemplos: criação de threads, sincronização
- Sobrecarga 1: comunicação
  - Como soluções sequenciais não precisam se comunicar com outras threads, toda a comunicação é sobrecarregada.
  - Uma função de soma com escopo local.
- Sobrecarga 2: sincronização
  - Acontece quando uma thread deve esperar por um evento ou outra thread
  - Uma thread esperando pela liberação de um recurso, um mutex por exemplo.

Perda 1: sobrecarga

- Sobrecarga 3: computação
  - Computações paralelas quase sempre realizam cálculos extras não necessários na computação sequencial.
  - Exemplo: inicialização de uma variável contador em cada uma das threads paralelas.
- Sobrecarga 4: memória
  - Computações paralelas frequentemente requerem mais memória que a computação sequencial.
  -

Perda 2: computações não paralelas

- Computações não paralelizáveis limitam os benefícios potenciais do paralelismo.
- Segundo Amdahl se 1/S de um programa é sequencial, então o speedup máximo está limitado a um fator de S.
- Computações redundantes:
  - N threads executam o mesmo laço k vezes
    - para(i=0; i<k; i++) {…}
- Threads ociosas esperando pelo cálculo de uma thread.
  - Exemplo: leitura de um dado em disco

Perda 3: Tempo ocioso

- Em um contexto ideal, todos os processadores trabalham ao mesmo tempo.
- Entretanto, algum pode ficar ocioso devido a:
  - Falta de trabalho
  - Aguardar um evento externo, como um dado a ser processado ainda. Ex.: produtor - consumidor
  - Carga desbalanceada
  - Uso excessivo da memória
    - Lentidão, latência...

Perda 4: contenção de recursos

- Degradação do desempenho do sistema devido a competição por recursos compartilhados.
- Compartilhamento de recursos pode criar um fluxo de carga excessiva no barramento da memória.
  - Pode afetar inclusive outros endereçamentos de memória que não sejam o do compartilhamento.

Alternativas de Compensação

- Há 4 fontes de perda. Limitar uma fonte pode aumentar outra.
- Compensação 1: Comunicação versus computação
  - Reduzir a sobrecarga de comunicação adicionando desempenho na computação.
  - Computações redundantes
  - Sobrepor comunicação e computação
    - A comunicação que é independente da computação pode ser realizada enquanto a computação está em andamento (ocultando a comunicação por trás da computação)

Alternativas de Compensação

- Compensação 2: Memória versus Paralelismo
  - O paralelismo pode ser incrementado com o custo do aumento da memória utilizada.
- Compensação 3: Sobrecarga versus Paralelismo
  - Balanceamento de carga versus sobrecarga
    - Aumentar o paralelismo pode aumentar a sobrecarga
  - Compensação de granularidade
    - Aumenta a granularidade da interação. Exemplo:
      - Uma thread pode pegar vários itens de trabalho de uma vez da fila de trabalho.
      - Enviar toda a linha / coluna da matriz em vez de elementos individuais.

Sumário

- Parte sequencial do programa muito grande
  - Segundo a lei de Amdahl não se pode ter muito ganho nesse caso.
- Bloqueio altamente contido
  - Todas as threads, por exemplo, precisam ser colocados na fila para entrar em sua seção crítica
- Alta sobrecarga de comunicação, pouca computação
  - O objetivo é a computação. Comunicação e sincronização é a sobrecarga devido a múltiplos eventos.
- Balanceamento de carga pobre
  - Enquanto alguns processos trabalham com sobrecarga, outros estão ociosos. Os processadores não são bem utilizados.
- Códigos escalares