Paralelização
Introdução a vetorização, OpenMP e MPI

1 – Conceitos

Paulo Penteado
IAG / USP

pp.penteado@gmail.com

Esta apresentação: http://www.ppenteado.net/ast/pp_para_on/pp_para_on_1.pdf
Arquivos do curso: http://www.ppenteado.net/ast/pp_para_on/
Artigo relacionado: http://www.ppenteado.net/papers/iwcca/iwcca_pfp.pdf

Programa

1 – Conceitos

- Motivação
- Formas de paralelização
  - Paralelismo de dados
  - Paralelismo de tarefas
- Principais arquiteturas paralelas atuais
  - Com recursos compartilhados
  - Independentes
- Paradigmas discutidos neste curso:
  - Vetorização
  - OpenMP
  - MPI
- Escolhas de forma de vetorização
- Algumas referências
- Exercícios – testes de software a usar no curso

Slides em http://www.ppenteado.net/ast/pp_para_on_1.pdf

Exemplos em http://www.ppenteado.net/ast/pp_para_on/
Artigo relacionado: http://www.ppenteado.net/papers/iwcca/iwcca_pfp.pdf
pp.penteado@gmail.com

Programa

2 – Vetorização

- Motivação
- Arrays – conceitos
- Organização multidimensional
- Arrays – uso básico
- Arrays – row major x column major
- Operações vetoriais
- Vetorização avançada
  - Operações multidimensionais
  - Redimensionamento
  - Buscas
  - Inversão de índices
- Algumas referências
- Exercícios - vetorização

Slides em http://www.ppenteado.net/ast/pp_para_on_2.pdf

Exemplos em http://www.ppenteado.net/ast/pp_para_on/
Artigo relacionado: http://www.ppenteado.net/papers/iwcca/iwcca_pfp.pdf
pp.penteado@gmail.com

Programa

3 – OpenMP

- Motivação
- Características
  - Diretrizes
  - Estruturação
- Construções
  - parallel
  - loop
  - section
  - workshare
- Cláusulas
  - Acesso a dados
  - Controle de execução
- Sincronização
  - Condições de corrida
- Exercícios - OpenMP

Slides em http://www.ppenteado.net/ast/pp_para_on_3.pdf

Exemplos em http://www.ppenteado.net/ast/pp_para_on/
Artigo relacionado: http://www.ppenteado.net/papers/iwcca/iwcca_pfp.pdf
pp.penteado@gmail.com

Programa

4 – MPI

- Motivação
- Características
- Estruturação
- Formas de comunicação
- Principais funções
  - Controle
  - Informações
  - Comunicação
- Boost.MPI
- Sincronização
  - Deadlocks
- Exercícios - MPI

Slides em http://www.ppenteado.net/ast/pp_para_on_4.pdf

Exemplos em http://www.ppenteado.net/ast/pp_para_on/
Artigo relacionado: http://www.ppenteado.net/papers/iwcca/iwcca_pfp.pdf
pp.penteado@gmail.com

Motivação

Como tornar programas mais rápidos?

Motivação

Como tornar programas mais rápidos?

É só esperar ter computadores mais rápidos?

Motivação

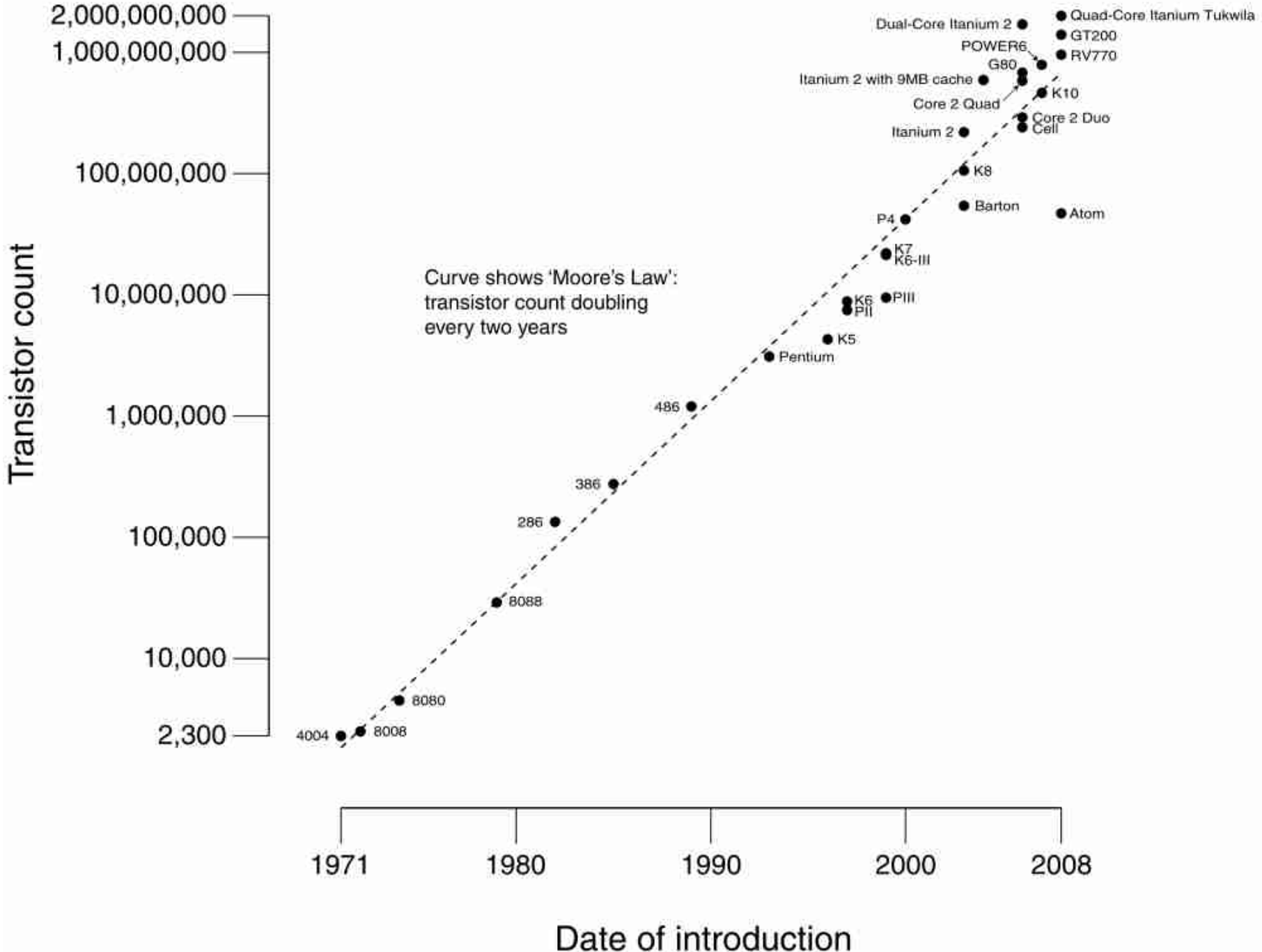
Como tornar programas mais rápidos?

É só esperar ter computadores mais rápidos?

Lei de Moore: *o número de componentes em circuitos integrados tem aumentado exponencialmente, dobrando a cada dois anos*.

- Derivada em 1965, com os dados de 1958 a 1965, prevendo continuar por >10 anos.
- Continua válida, mais de 50 anos depois.

CPU Transistor Counts 1971-2008 & Moore's Law



Motivação

Como tornar programas mais rápidos?

Lei de Moore: *o número de componentes em circuitos integrados tem aumentado exponencialmente, dobrando a cada dois anos*.

- Derivada em 1965, com os dados de 1958 a 1965, prevendo continuar por >10 anos.
- Continua válida, mais de 50 anos depois.

Até 2005 (~ Pentium 4) o aumento de rapidez dos programas era automático: **CPUs mais rápidas tornavam os programas mais rápidos**.

Com o teto em ~3GHz, por motivos térmicos, CPUs tiveram que ser redesenhadas para usar mais núcleos.

- O aumento de capacidade passou a ser predominantemente pelo aumento do número de núcleos.

Além de computadores individuais, muitos (até milhares) de núcleos (CPUs ou GPUs), em computadores interligados ou independentes (clusters, grids, nuvens) formam os supercomputadores.

Motivação

O problema: software que só faz uma coisa de cada vez (***serial, não-paralelizado, sequencial***) não faz uso de múltiplos núcleos.

Portanto, software serial não ganha em tempo com a disponibilidade de vários núcleos.

Não existe um supercomputador feito com um núcleo extremamente rápido.

A única forma de ganhar com o uso de muitos núcleos é fazer mais de uma coisa ao mesmo tempo: **paralelização**.

Paralelização ganhou muita importância recentemente.

Há muitas formas de paralelização, apropriadas a diferentes problemas:

- Algumas delas são mais intuitivas. Outras adicionam muita complexidade para dividir o trabalho entre as unidades e as coordenar.

- Este curso tem apenas algumas das formas mais comuns.

Como fazer mais de uma coisa ao mesmo tempo?

Tradicionalmente, programadores pensam em uma seqüência única (não paralela) de ações.

Identificar como dividir o trabalho em partes que podem ser feitas simultaneamente costuma ser o primeiro obstáculo.

Alguns exemplos da variação de situações:

- Tarefas seriais, a ser executadas para vários conjuntos de dados:
  - Realizar o mesmo processamento em várias observações.
  - Calcular o mesmo modelo para vários parâmetros diferentes.

- Tarefas que contém partes calculadas independentemente:
  - Operações vetoriais (operações sobre arrays, álgebra linear, etc.)
  - Processamento de cada pixel de uma imagem.
  - Cada comprimento de onda em um modelo de transferência radiativa.
  - Cada célula / partícula de um modelo dinâmico.

Como fazer mais de uma coisa ao mesmo tempo?

Tarefas independentes: Ex: um modelo MHD para um objeto astrofísico, após terminar de calcular o estado do sistema em um passo:

Código serial: uma coisa feita de cada vez

(cada ítem numerado sendo uma rotina chamada, cada uma após terminar a anterior)

(fim do cálculo do estado)

1) Escrever arquivos com o estado do sistema

2) Gerar visualizações do estado do sistema (ex: imagens mostrando temperatura, densidade, etc.)

3) Calcular o espectro observado, para diferentes posições no objeto (para comparar com observações)

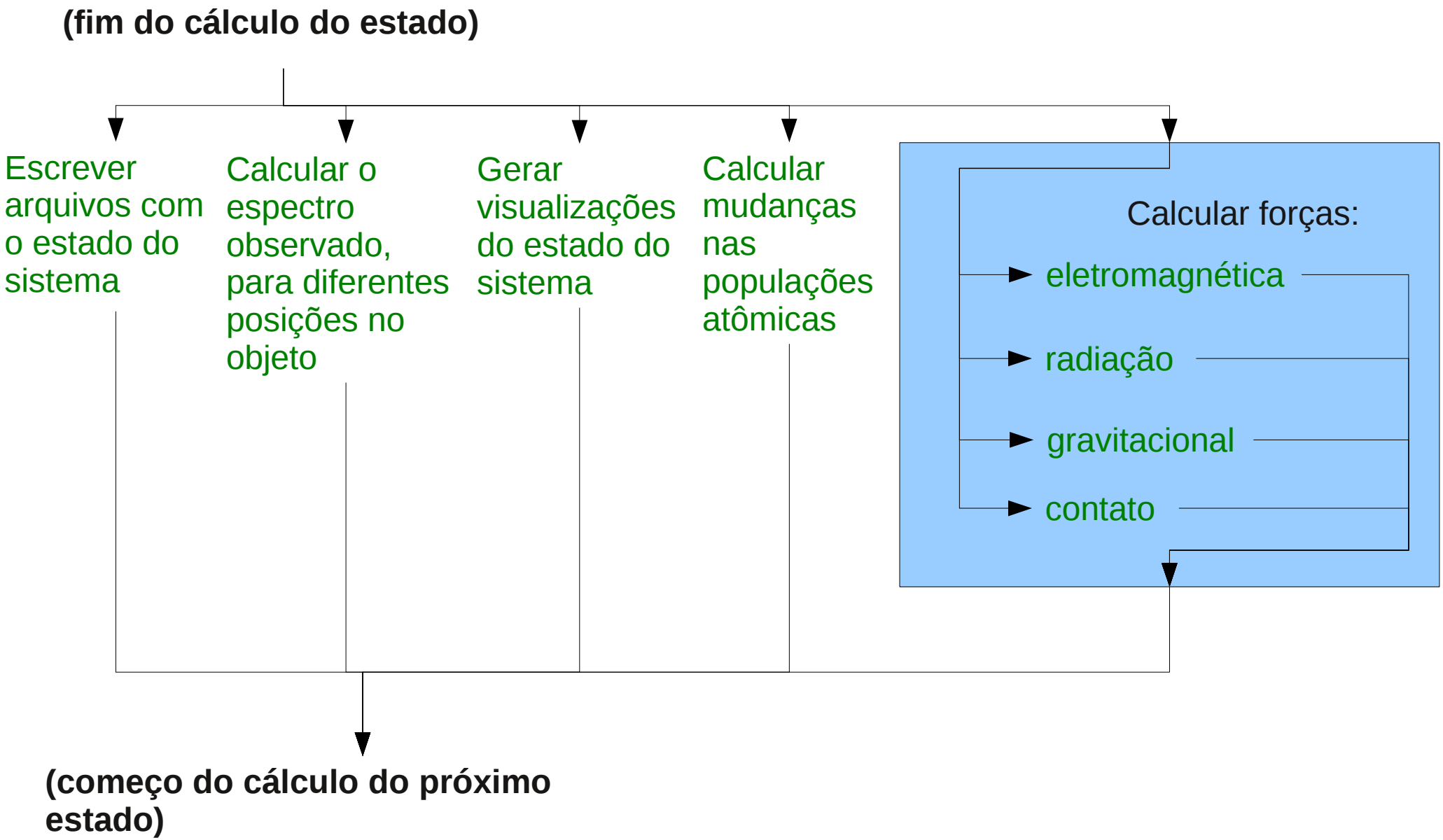
4) Calcular as mudanças para o próximo passo:
   - 4a) Forças de contato (pressão, viscosidade, etc.)
   - 4b) Forças à distância:
     - Gravitacional
     - Eletromagnética
   - 4c) Forças de radiação
   - 4d) Mudanças nas populações atômicas

(começo do cálculo do próximo estado)

Como fazer mais de uma coisa ao mesmo tempo?

Tarefas independentes: Ex: um modelo MHD para um objeto astrofísico, após terminar de calcular o estado do sistema em um passo:

Código paralelizado: todas as 8 tarefas em verde são feitas ao mesmo tempo



Formas de paralelização - classificação

A maior parte dos programadores está habituada em pensar em algorítimos seriais.

Há dificuldade em identificar como paralelizar o programa.

Pode-se pensar em como a mesma tarefa seria realizada manualmente, havendo várias pessoas no lugar de uma só.

Fundamentalmente, a divisão de trabalho pode ser feita no nível de **dados** ou de **tarefas**.

Formas de paralelização - classificação

Paralelismo de dados

Cada unidade processa uma parte dos dados.

Exs:

- Tarefas seriais, a ser executadas para vários conjuntos de dados*:
  - Realizar o mesmo processamento em várias observações.
  - Calcular o mesmo modelo para vários parâmetros diferentes.

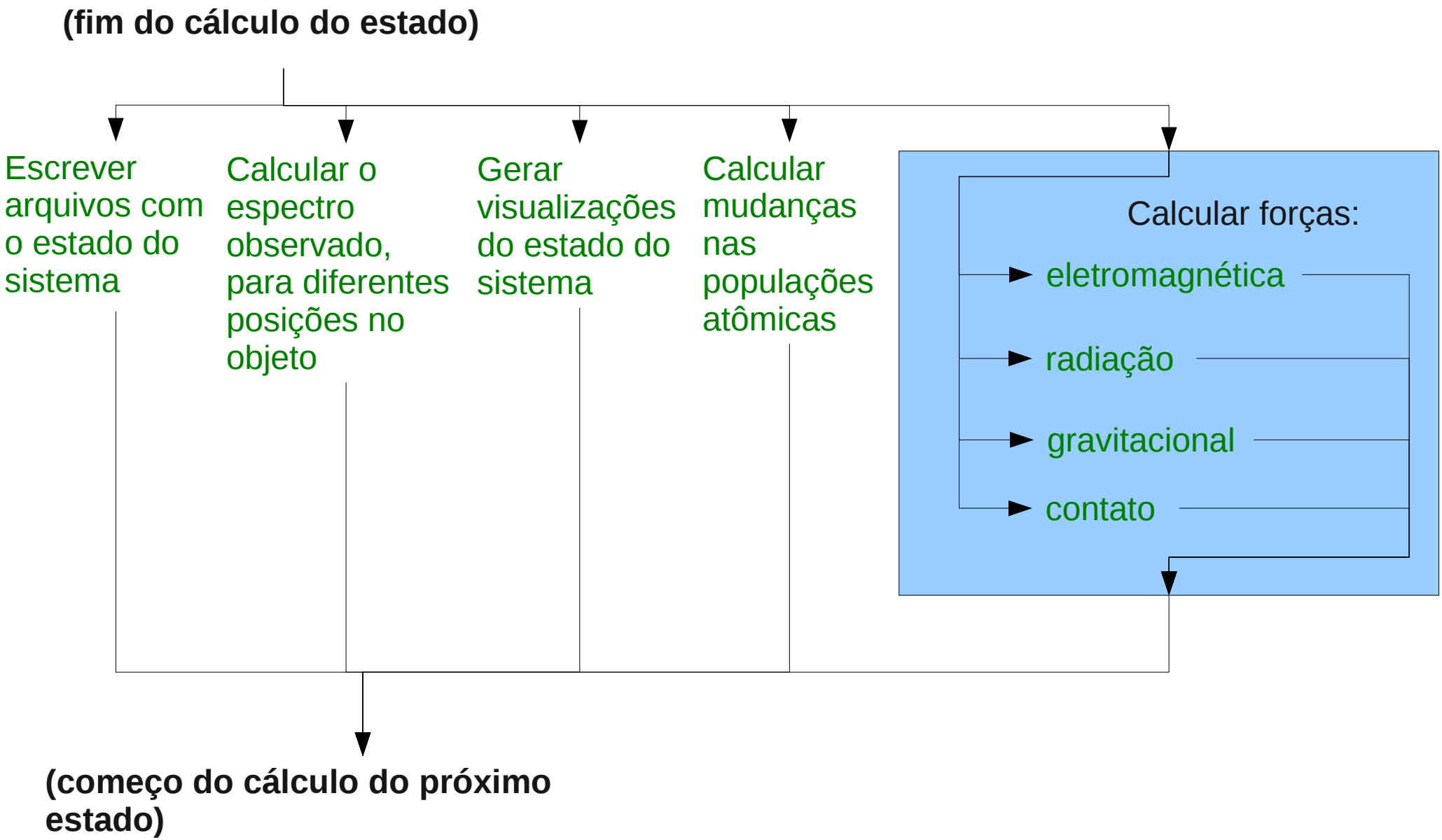
- Tarefas que contém partes calculadas independentemente:
  - Operações vetoriais (operações sobre arrays, álgebra linear, etc.)
  - Processamento de cada pixel de uma imagem.
  - Cada comprimento de onda em um modelo de transferência radiativa.
  - Cada célula / partícula de um modelo dinâmico.

*Este caso pode ser feito por **paralelização extrínseca**: apenas executando ao mesmo tempo várias instâncias de um programa serial, cada uma usando dados diferentes.

Paralelismo de tarefas

Cada unidade realiza uma tarefa independente (sobre dados distintos).

Ex: um modelo MHD para um objeto astrofísico, após terminar de calcular o estado do sistema em um passo:



Formas de paralelização - classificação

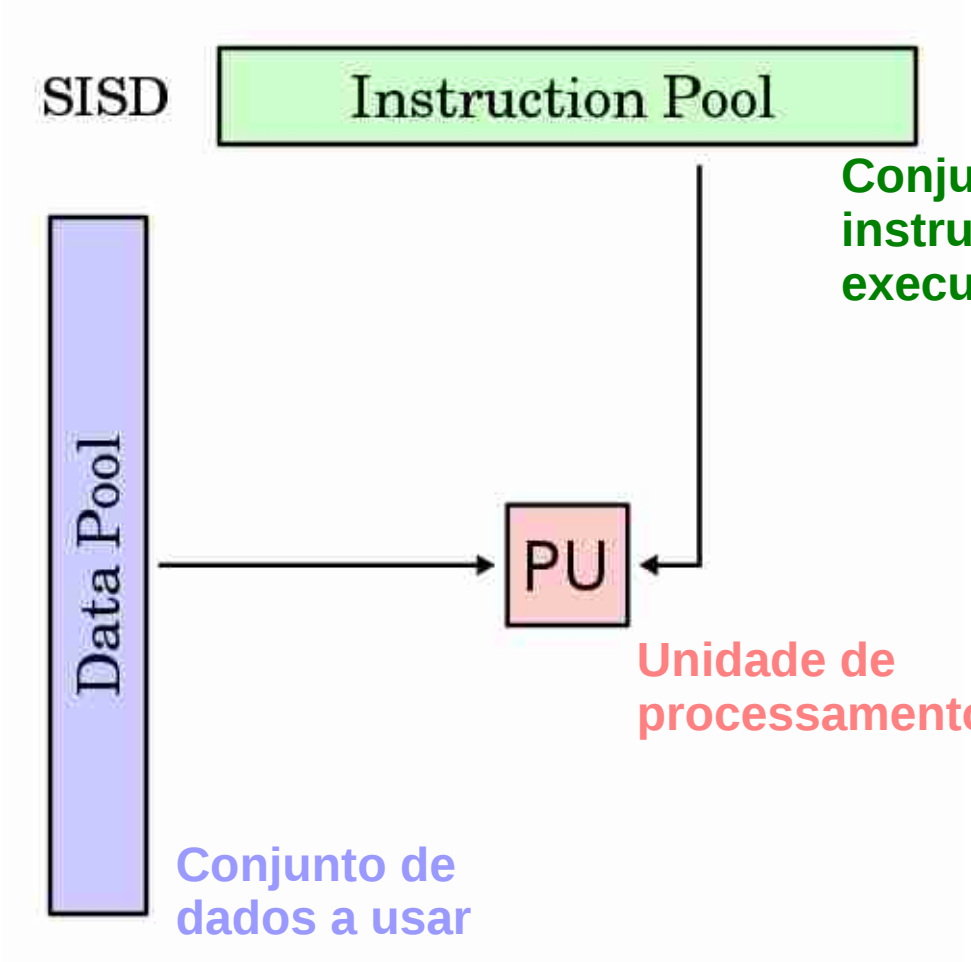
Taxonomia de Flynn (1966):

(Single | Multiple) **Instruction**, (Single | Multiple) **Data**

1) SISD – Single Instruction, Single Data

Programas seriais.

A cada momento, apenas uma instrução é executada, em apenas um elemento dos dados.



Formas de paralelização - classificação

Taxonomia de Flynn (1966):

(Single | Multiple) **Instruction**, (Single | Multiple) **Data**

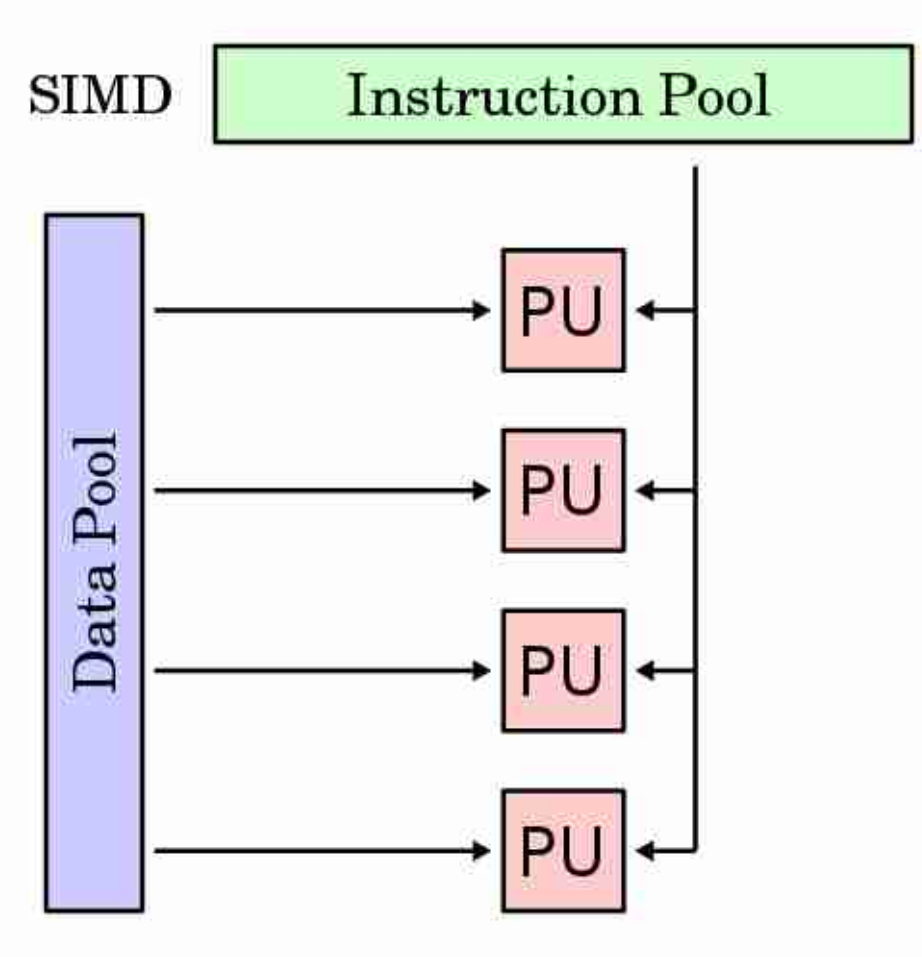
2) SIMD – Single Instruction, Multiple Data

Paralelismo de dados.

A cada momento, apenas uma instrução é executada, em vários elementos do conjunto de dados, simultaneamente.

Exs:

- Operações vetorizadas (adiante)
- Alguns usos de OpenMP e MPI (próxima aula)
- GPUs (em outro curso)



Formas de paralelização - classificação

Taxonomia de Flynn (1966):

(Single | Multiple) **Instruction**, (Single | Multiple) **Data**

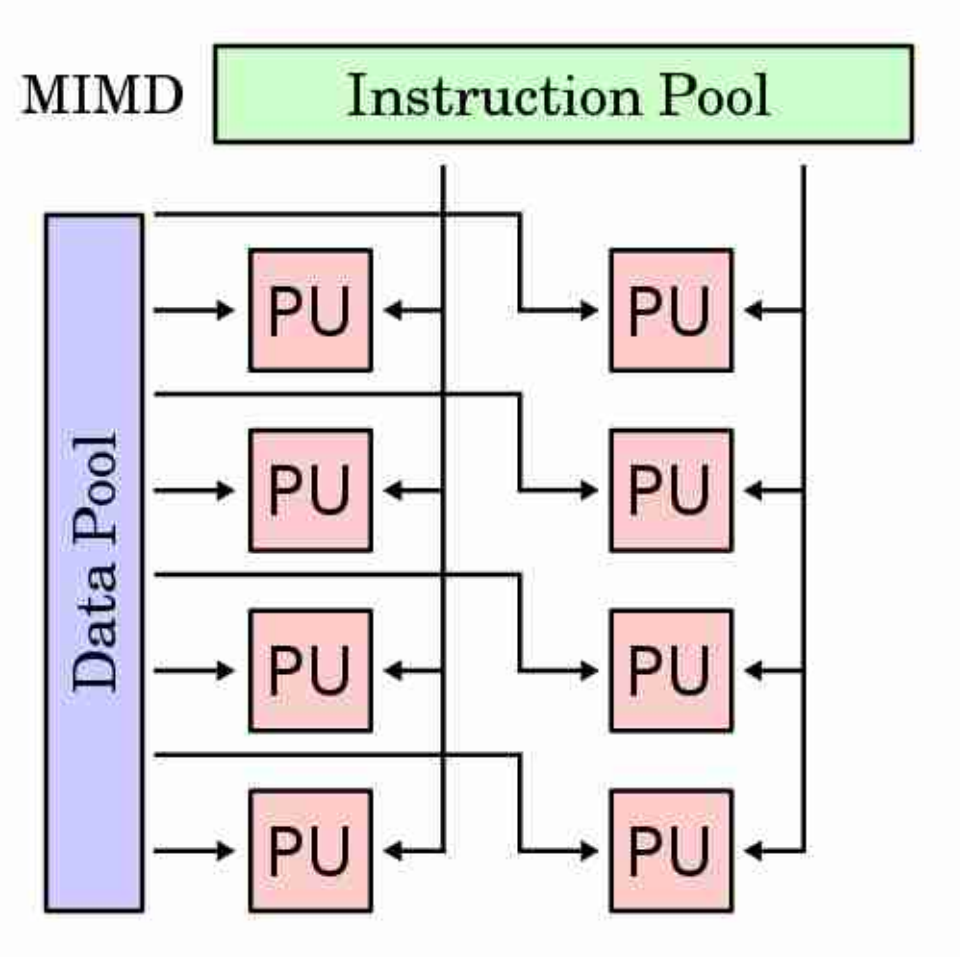
3) MIMD – Multiple Instruction, Multiple Data

Paralelismo de tarefas.

A cada momento, várias instruções são executadas, em vários elementos do conjunto de dados, simultaneamente.

Exs:

- Alguns usos de OpenMP e MPI (próxima aula)
- Paralelização extrínseca (vários programas executados simultaneamente, cada um processando dados diferentes).



Limites à paralelização

Algumas tarefas, intrinsecamente, não são paralelizáveis: cada parte depende das anteriores, não sendo possível as fazer simultaneamente.

Nestes casos, a única possibilidade de paralelização é produzir várias unidades independentes (processar várias imagens, calcular vários modelos) simultaneamente (paralelismo extrínseco).

Exemplo clássico de problema não paralelizável:

- *Não importa quantas pessoas possam ajudar, não vai levar menos que ~9 meses para produzir um bebê*.
- Mas é possível **N** mulheres, simultaneamente, produzirem um bebê cada: ainda leva 9 meses para terminar, mas **N** unidades serão produzidas neste tempo; o tempo médio por unidade é **N** vezes menor.

A maior parte dos problemas em ciências computacionais não tem uma característica única:

- Tipicamente, têm partes não paralelizáveis, e partes de tipos diferentes de paralelização.

É importante **medir** que partes são mais relevantes ao tempo de execução:

- **Sem medir, é fácil se enganar sobre que parte é mais pesada.**
- **A medição deve ser feita com casos representativos**: os gargalos podem mudar entre um caso simples de teste e um caso pesado de uso real.

Limites à paralelização

Paralelizar em N unidades não necessariamente diminui em N vezes o tempo médio para obter cada resultado.

Paralelizar tem um custo (*overhead*), de tarefas adicionais geradas pela paralelização. Exs:

- Dividir o trabalho entre unidades
- Iniciar cada unidade
- Comunicação entre as unidades
- Juntar os resultados ao final
- Finalizar cada unidade

Para que haja um ganho, estes trabalhos adicionais devem ser pequenos em comparação ao trabalho paralelizado.

Em paralelização de tarefas, o número máximo de unidades é determinado pelo algorítimo:

- Não adianta ter 10 mil núcleos, se só há 8 tarefas a fazer simultaneamente.
- Mas é possível (e comum) que cada tarefa possa usar, internamente, paralelização de dados, fazendo uso de várias unidades por tarefa.

Limites à paralelização

Em paralelização de dados a princípio é possível usar qualquer número de unidades.

Mas em casos não completamente paralelizados o ganho não é linear com o número de unidades.

Lei de Amdahl

O modelo mais simples para o ganho de tempo com paralelização.

O ganho de tempo (*speedup*) de um programa com fração **P** paralelizada, executado com **N** unidades:

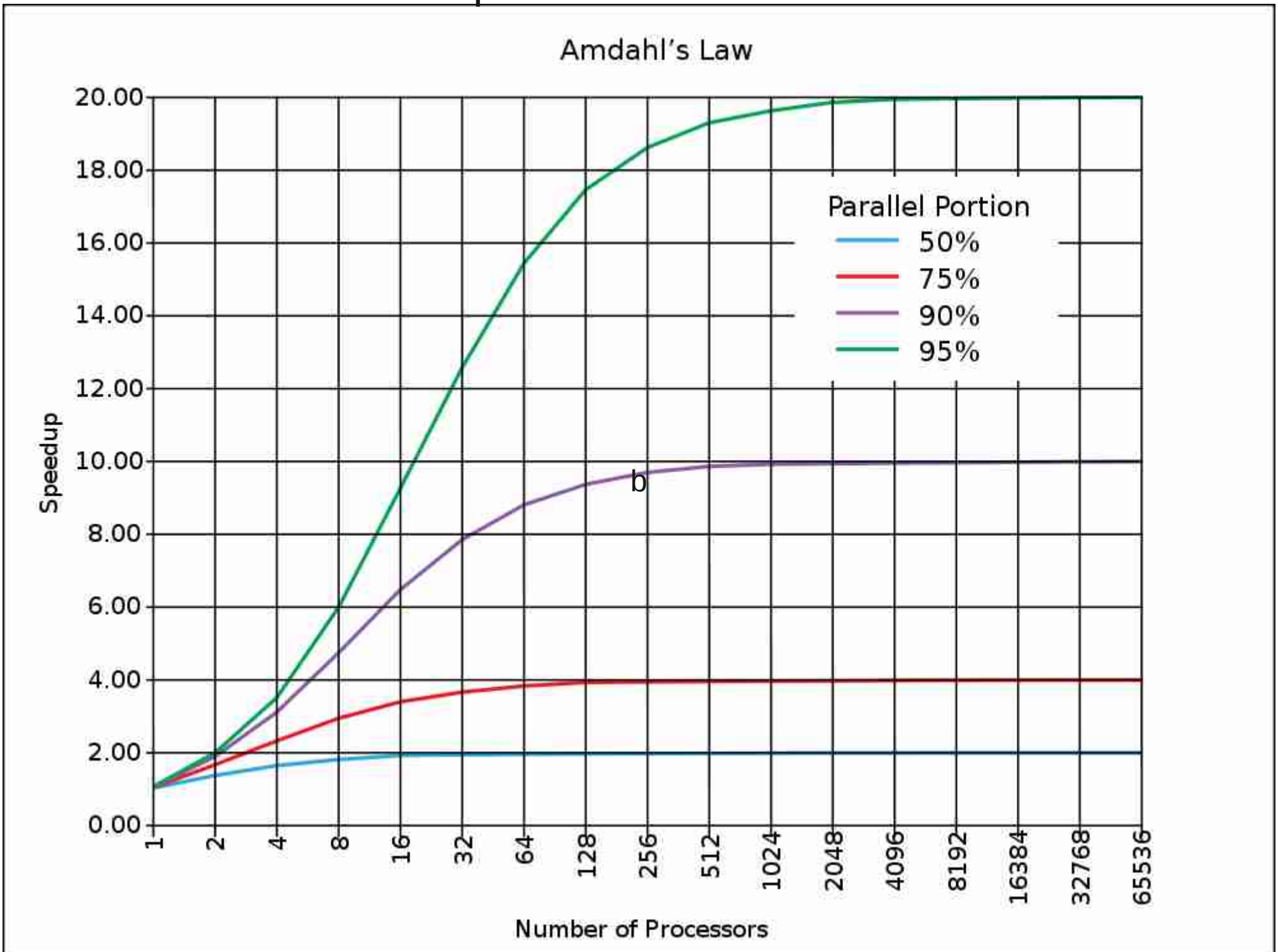
$$S = \frac{1}{(1-P) + \frac{P}{N}}$$

Só considera a relação entre o tempo gasto na parte serial e o tempo gasto na parte paralela.

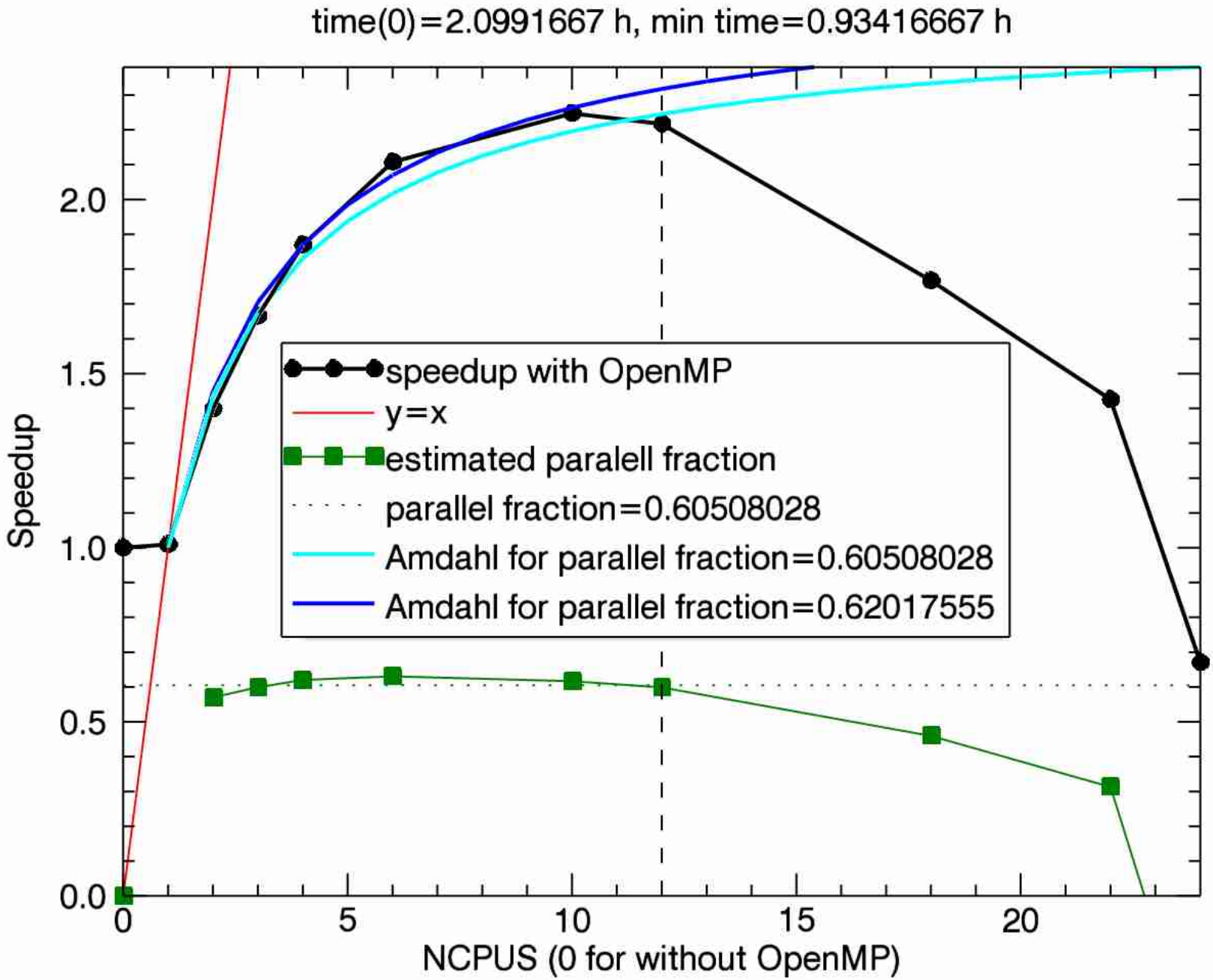
Não considera custos adicionais (*overheads*) de paralelização.

Útil para prever ganhos, e medir a fração paralelizada de um programa.

Lei de Amdahl – exemplos teóricos



Lei de Amdahl – exemplo real de medida



O pico em ~10 (11 não foi testado) se deve a o computador ter 12 núcleos: com mais que um processos por unidade de execução, o *overhead* tende a aumentar.

Principais arquiteturas paralelas atuais

A classificação mais fundamental para as arquiteturas de paralelização está no compartilhamento de recursos.

Em alguns casos, sistemas de arquivos (discos, etc.) podem ser compartilhados: todas as unidades têm acesso aos mesmos arquivos.

A memória (RAM) normalmente é o mais importante recurso:

- É muito mais rápida (ordens de magnitude) que discos.
- Normalmente é onde ficam todas as variáveis que um programa usa.
- É muito mais cara (por isso mais escassa) que discos.
- Só raros (e caros) sistemas conseguem compartilhar memória entre nós (computadores) diferentes. Mesmo quando é possível, o acesso à memória entre nós é mais lento que dentro de um nó (por limitações físicas).

Quando há compartilhamento, **o mesmo recurso está disponível a todas as unidades de execução**. Todas podem (a princípio) acessar os mesmos dados, ao mesmo tempo:

Principais arquiteturas paralelas atuais

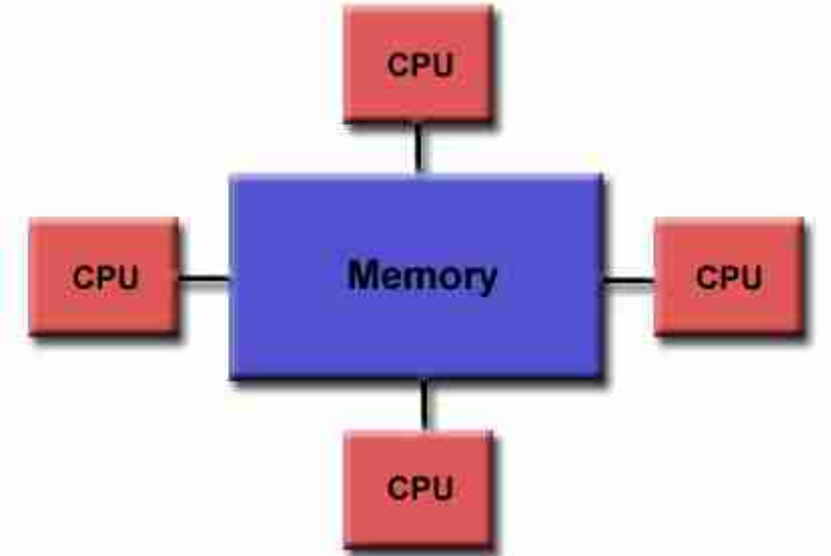
A classificação mais fundamental para as arquiteturas de paralelização está no compartilhamento de recursos.

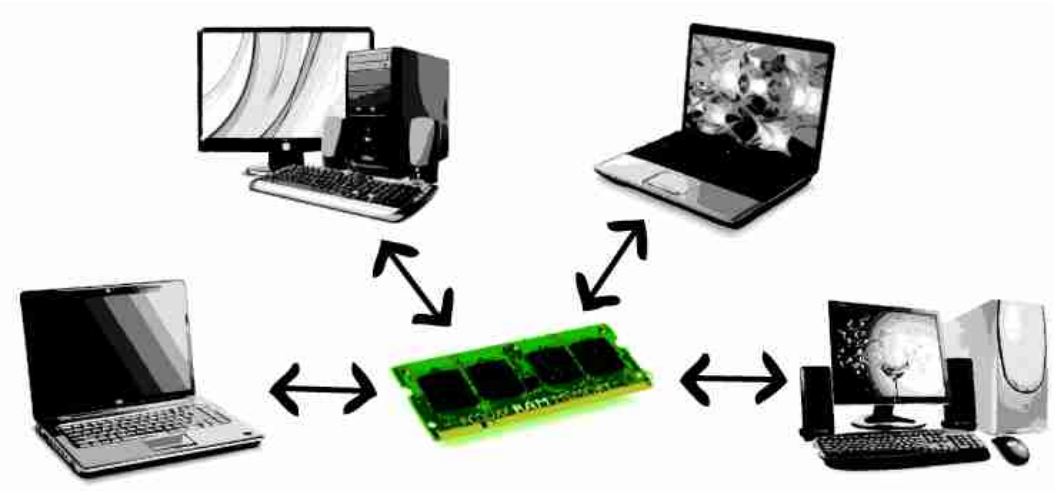
Em alguns casos, sistemas de arquivos (discos, etc.) podem ser compartilhados: todas as unidades têm acesso aos mesmos arquivos.

A memória (RAM) normalmente é o mais importante recurso:

- É muito mais rápida (ordens de magnitude) que discos.
- Normalmente é onde ficam todas as variáveis que um programa usa.
- É muito mais cara (por isso mais escassa) que discos.
- Só raros (e caros) sistemas conseguem compartilhar memória entre nós (computadores) diferentes. Mesmo quando é possível, o acesso à memória entre nós é mais lento que dentro de um nó (por limitações físicas).

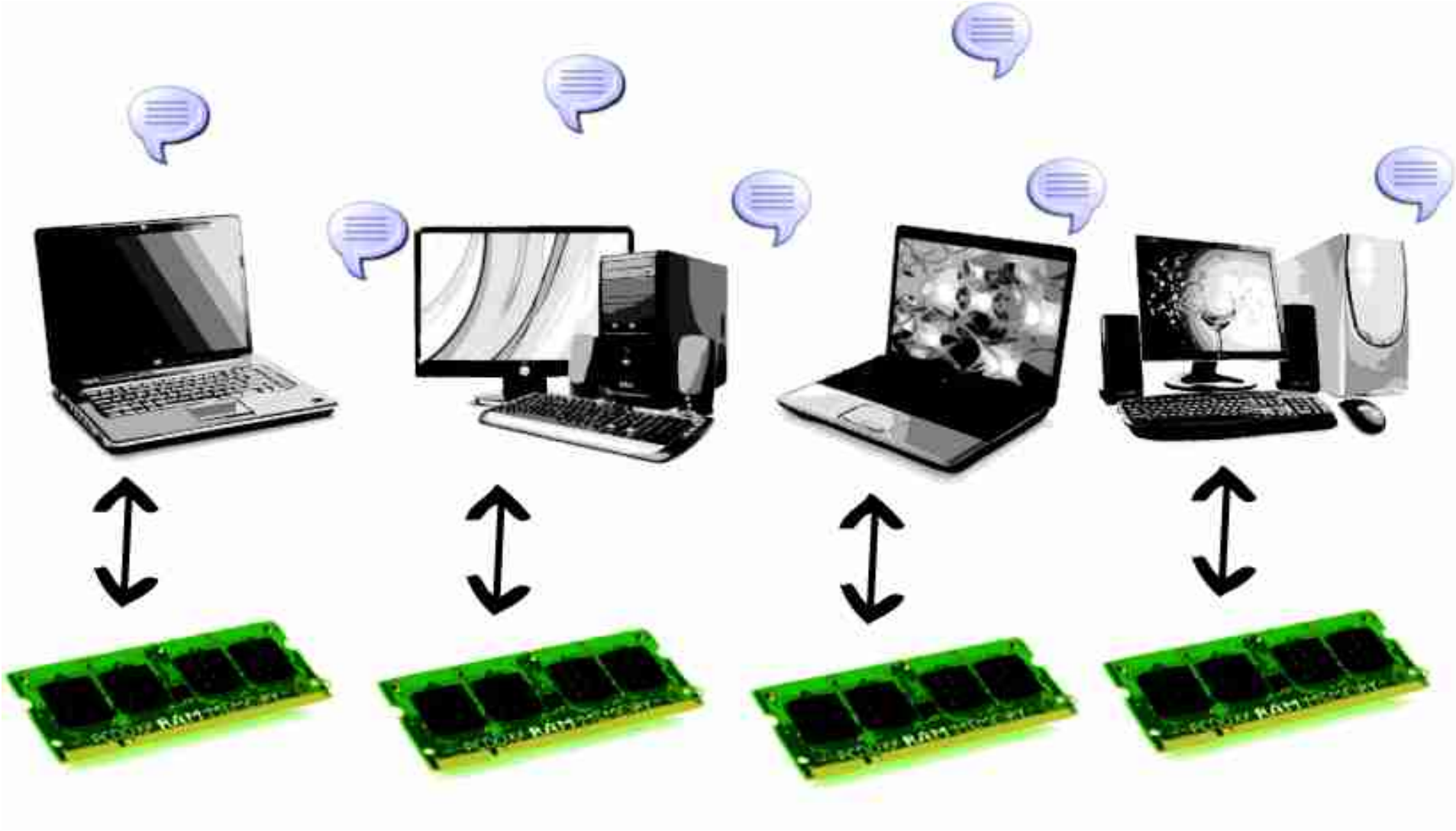
Quando há compartilhamento, **o mesmo recurso está disponível a todas as unidades de execução**. Todas podem (a princípio) acessar os mesmos dados, ao mesmo tempo:





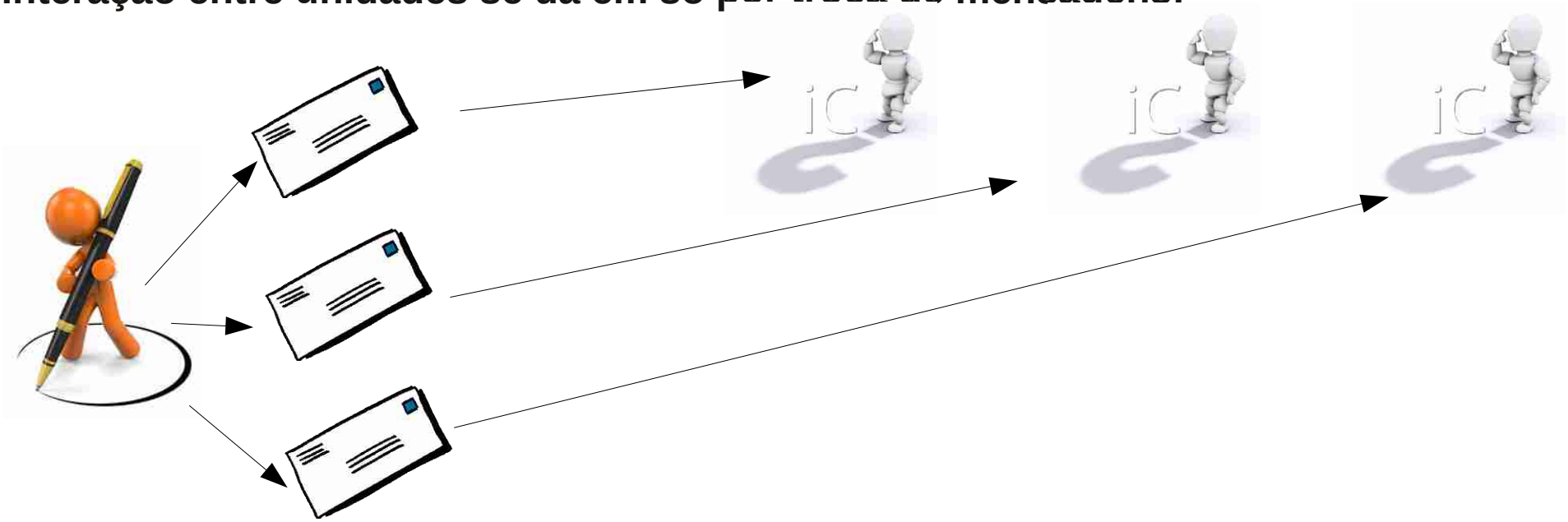
Principais arquiteturas paralelas atuais

Sem compartilhamento, **cada unidade tem uma cópia** dos recursos necessários a todas.
Interação entre unidades se dá em só por troca de mensagens:



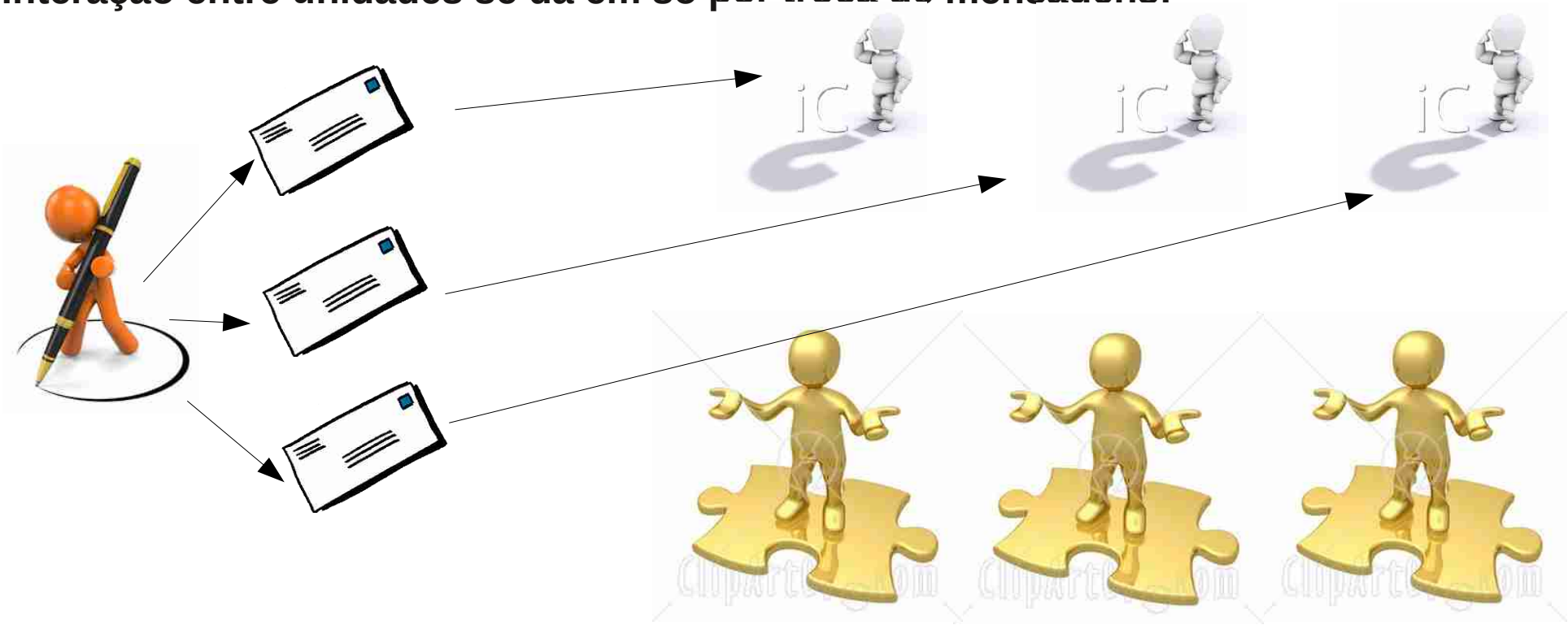
Principais arquiteturas paralelas atuais

Sem compartilhamento, **cada unidade tem uma cópia** dos recursos necessários a todas.
Interação entre unidades se dá em só por troca de mensagens:



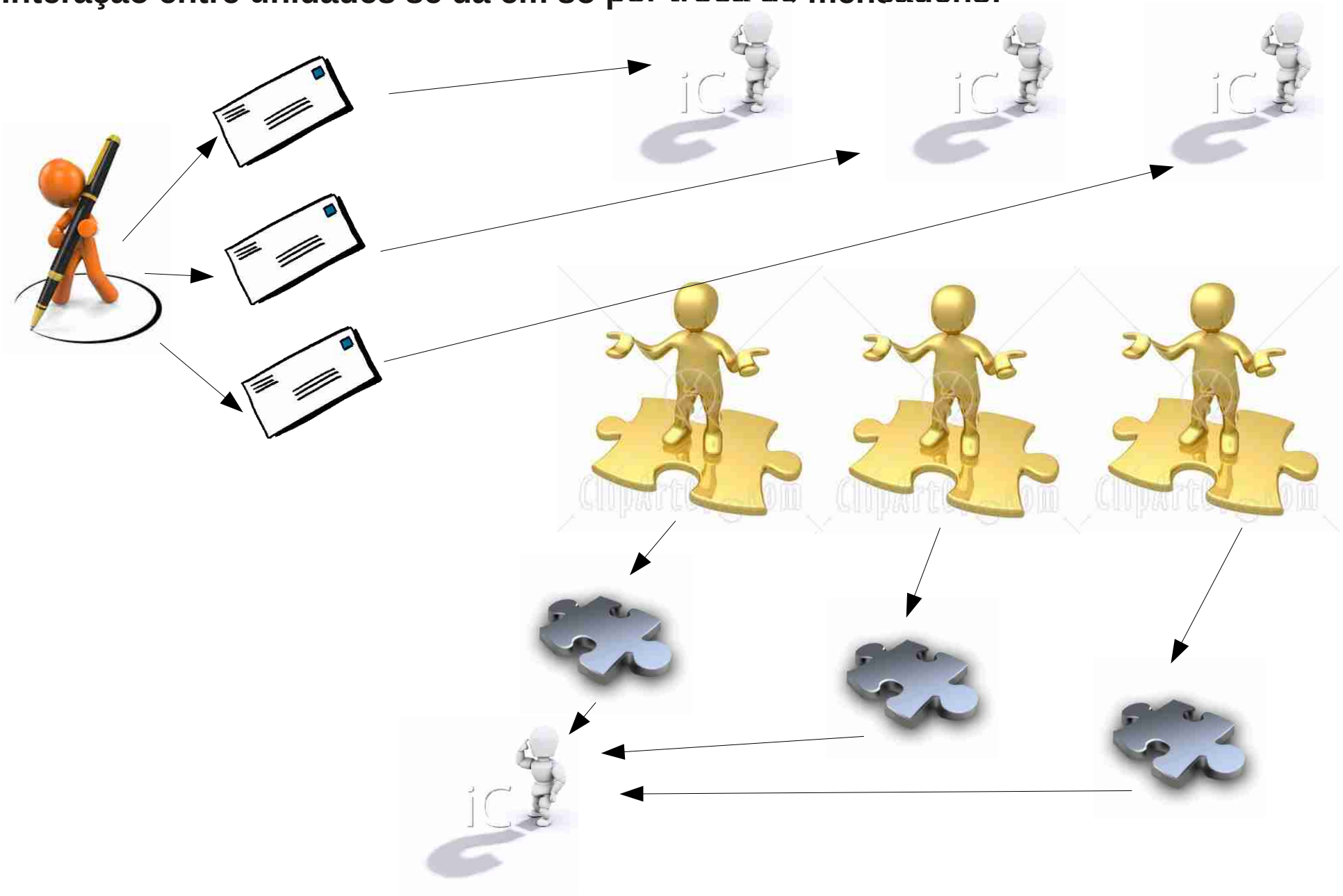
Principais arquiteturas paralelas atuais

Sem compartilhamento, **cada unidade tem uma cópia** dos recursos necessários a todas.
Interação entre unidades se dá em só por troca de mensagens:



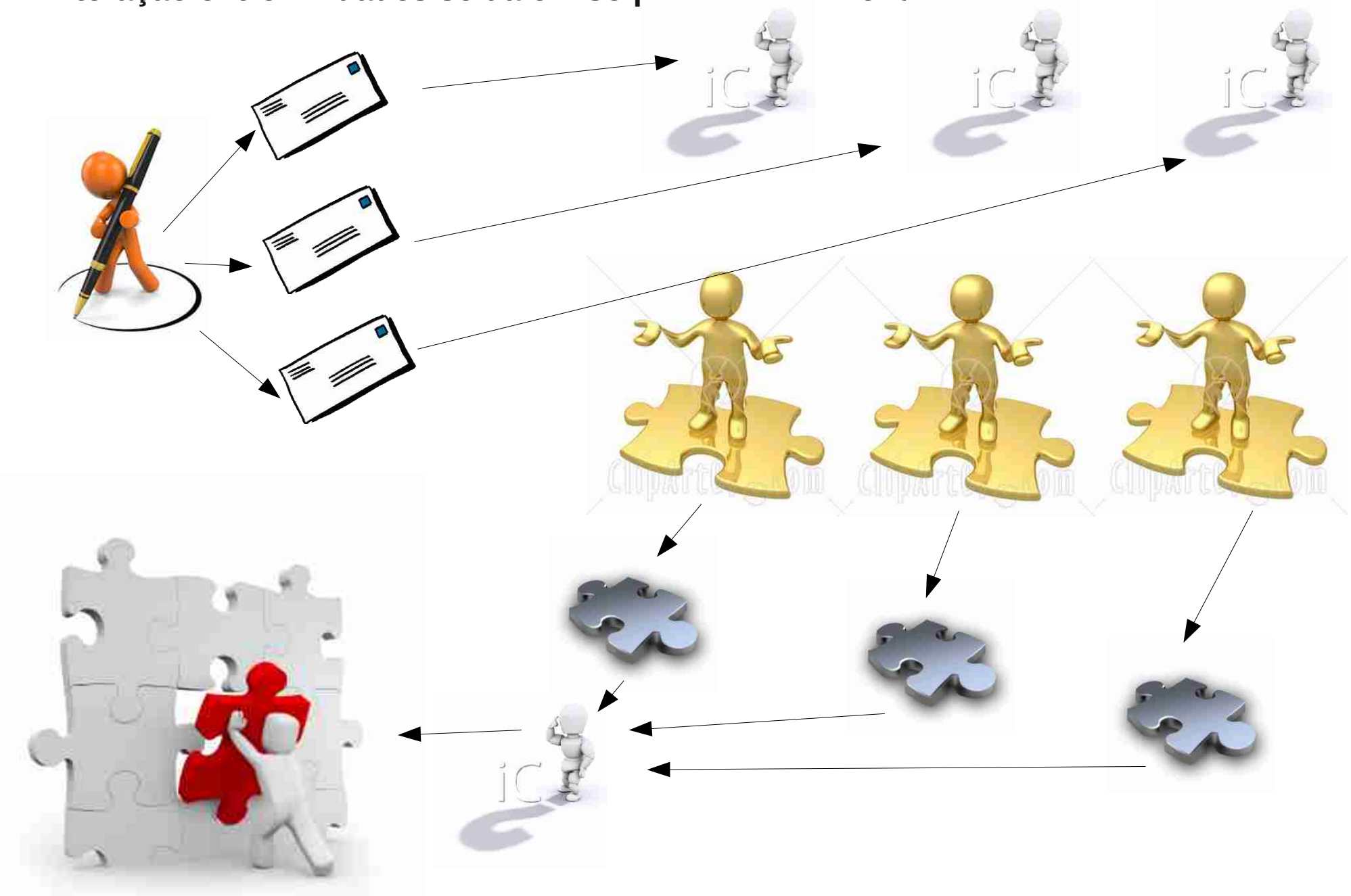
Principais arquiteturas paralelas atuais

Sem compartilhamento, **cada unidade tem uma cópia** dos recursos necessários a todas.
Interação entre unidades se dá em só por troca de mensagens:



Principais arquiteturas paralelas atuais

Sem compartilhamento, **cada unidade tem uma cópia** dos recursos necessários a todas.
Interação entre unidades se dá em só por troca de mensagens:



Principais arquiteturas paralelas atuais

Com memória compartilhada*:

Cada unidade de execução é um ***thread***.

- Vetorização (adiante)
- OpenMP (próxima aula)
- GPUs (em outro curso)
- Gerencialmento manual de *threads* (dependente de linguagem e sistema operacional)

Sem memória compartilhada*:

Cada unidade de execução é um **processo**.

- MPI (próxima aula)
- Grids, clouds (outro curso)
- Gerencialmento manual de *processos* (dependente de linguagem e sistema operacional)

*Mesmo sem memória compartilhada é possível (e comum) compartilhar arquivos.

Paradigmas discutidos neste curso

Vetorização

- **Operações sobre arrays** são expressas com a semântica da linguagem.
- O programador não especifica como dividir o trabalho entre as unidades, as coordenar ou juntar os resultados ao final.
- **A paralelização é feita pelo compilador / interpretador.**
- Traz vantagens (de organização de código e eficiência) mesmo se não houver paralelização.

OpenMP

- **Usa memória compartilhada.**
- O mais comum e melhor padronizado sistema para paralelismo de dados.
- Pode ser usado para paralelismo de tarefas.
- Simples de usar, inclusive paralelizar gradualmente.
- Muito mais simples que gerenciar threads e sua comunicação diretamente.

MPI

- **Usa processos (memória distribuída)**, mesmo que dentro do mesmo computador.
- O sistema mais comum para uso de clusters (vários computadores interligados) e paralelismo de tarefas.
- Em geral o mais complexo de usar destes 3.
- Muito mais simples que gerenciar processos e sua comunicação diretamente.

Escolhas

Não existe uma “melhor solução”.

Algumas são mais adequadas a algumas situações.

Muitos problemas podem combinar vários paradigmas (soluções híbridas).

- Ex: um programa MPI onde cada processo use OpenMP e vetorização.

Há também paralelização indireta:

- O programador usa bibliotecas paralelizadas prontas.
- As bibliotecas podem ter sido implementadas por especialistas (cientistas da computação) para fazer seu trabalho da forma mais eficiente possível.
- Casos comuns de estarem implementados:
  - Álgebra linear
  - FFT
  - Operações de processamento de imagens
  - Algoritmos comuns (ordenação, contêiners, etc.)
  - Problemas científicos comuns: visualização, CFD, vizinhos (KNN), etc.

Escolhas - comparação

Vetorização

- Vantagens:
  - Paralelização, se disponível, feita pelo compilador / interpretador.
  - Torna o programa mais organizado e robusto.
  - Permite identificar trechos que podem ser paralelizados de outras formas, explicitamente.
  - Muito mais fácil que as outras formas.
  - Compiladores podem gerar código vetorial para um só núcleo (MMX, SSE, AVX), ou para vários (em vários threads).
- Desvantagens:
  - Em geral limitada a memória compartilhada.
  - Limitada a paralelismo de dados.
  - Só se aplica aos casos mais simples de paralelização.
  - Abrangência muito variável entre linguagens.

Escolhas - comparação

OpenMP

- Vantagens:
  - O padrão melhor estabelecido para paralelização.
  - Padrão ainda ativamente desenvolvido.
  - Independente de linguagem, compilador e plataforma.
  - Permite paralelismo de dados e tarefas.
  - Possui tanto construções de baixo nível como de alto nível.
  - Mantém compatibilidade com código serial.
  - Mais fácil que MPI.
- Desvantagens:
  - Limitado a memória compartilhada.
  - Só implementado em Fortran, C, C++.

Escolhas - comparação

MPI

- Vantagens:
  - O padrão melhor estabelecido para memória distribuída.
  - Pode ser usado tanto em sistemas de memória compartilhada como de memória distribuída.
  - O padrão mais comum para clusters.
  - Padrão ainda ativamente desenvolvido.
  - Permite paralelismo de dados e tarefas.
  - Implementado em várias linguagens (Fortran, C, C++, Python, Java, IDL, etc.).
- Desvantagens:
  - Mais difícil de usar que OpenMP e vetorização.
  - Sem usar Boost.MPI, desajeitado para transmitir dados mais complexos que arrays de tipos simples.
  - Exige reescrever (possivelmente reestruturar) o código serial.
  - Padrão ainda menos maduro que OpenMP.
  - Ainda varia bastante entre implementações, principalmente na execução (`mpirun`, `mpiexec`).

Algumas referências

Gerais (incluindo OpenMP e MPI e outros)

- *Parallel Programming: for Multicore and Cluster Systems* (2010)
  Rauber e Rünger
  http://www.amazon.com/Parallel-Programming-Multicore-Cluster-Systems/dp/364204817X/

- *An Introduction to Parallel Programming* (2011)
  Peter Pacheco
  http://www.amazon.com/Introduction-Parallel-Programming-Peter-Pacheco/dp/0123742609/

- *An Introduction to Parallel Programming with OpenMP, PThreads and MPI* (2011)
  Robert Cook
  http://www.amazon.com/Introduction-Parallel-Programming-PThreads-ebook/dp/B004I6D6BM/

- *Introduction to High Performance Computing for Scientists and Engineers* (2010)
  Hager e Wellein
  http://www.amazon.com/Introduction-Performance-Computing-Scientists-Computational/dp/143981192X/

Algumas referências

Cloud / computação distribuída

- *The Application of Cloud Computing to Astronomy: A Study of Cost and Perfomance* (2010)
  Berriman et al.
  http://dx.doi.org/10.1109/eScienceW.2010.10
  E outras referências de um dos autores em
  http://www.isi.edu/~gideon/

- *Using Java for distributed computing in the Gaia satellite data processing* (2011)
  O' Mullane et al.
  http://dx.doi.org/10.1007/s10686-011-9241-6

- *Amazon Elastic Compute Cloud (Amazon EC2)*
  http://aws.amazon.com/ec2

Algumas referências

The Application of Cloud Computing to Astronomy: A Study of Cost and Perfomance (2010)
Berriman et al.
http://dx.doi.org/10.1109/eScienceW.2010.10

Table IX summarizes the results of a production run on the cloud. All 210,664 public light curves were processed with 128 processors working in parallel. Each algorithm was run with period sampling ranges of 0.04 days to 16.75 days and a fixed period increment of 0.001 days. The processing was performed in 26.8 hours, for a total cost of $303.06, with processing the major cost item at $291. The transfer cost is, however, significant because the code produced outputs of 76 GB—some four times the size of the input data.

TABLE IX. SUMMARY OF PERIODOGRAM CALCULATIONS ON THE AMAZON EC2 CLOUD

		Result
Runtimes	Tasks	631,992
	Mean Task Runtime	6.34 sec
	Jobs	25,401
	Mean Job Runtime	2.62 min
	Total CPU Time	1,113 hr
	Total Wall Time	26.8 hr
Inputs	Input Files	210,664
	Mean Input Size	0.084 MB
	Total Input Size	17.3 GB
Outputs	Output Files	1,263,984
	Mean Output Size	0.124 MB
	Total Output Size	76.52 GB
Cost	Compute Cost	$291.58
	Transfer Cost	$11.48
	Total Cost	**$303.06**

Programa

1 – Conceitos

- Motivação
- Formas de paralelização
  - Paralelismo de dados
  - Paralelismo de tarefas
- Principais arquiteturas paralelas atuais
  - Com recursos compartilhados
  - Independentes
- Paradigmas discutidos neste curso:
  - Vetorização
  - OpenMP
  - MPI
- Escolhas de forma de vetorização
- Algumas referências
- Exercícios – testes de software a usar no curso

Slides em http://www.ppenteado.net/ast/pp_para_on_1.pdf

Exemplos em http://www.ppenteado.net/ast/pp_para_on/
Artigo relacionado: http://www.ppenteado.net/papers/iwcca/iwcca_pfp.pdf
pp.penteado@gmail.com

Programa

2 – Vetorização

- Motivação
- Arrays – conceitos
- Organização multidimensional
- Arrays – uso básico
- Arrays – row major x column major
- Operações vetoriais
- Vetorização avançada
  - Operações multidimensionais
  - Redimensionamento
  - Buscas
  - Inversão de índices
- Algumas referências
- Exercícios - vetorização

Slides em http://www.ppenteado.net/ast/pp_para_on_2.pdf

Exemplos em http://www.ppenteado.net/ast/pp_para_on/
Artigo relacionado: http://www.ppenteado.net/papers/iwcca/iwcca_pfp.pdf
pp.penteado@gmail.com

Programa

3 – OpenMP

- Motivação
- Características
  - Diretrizes
  - Estruturação
- Construções
  - parallel
  - loop
  - section
  - workshare
- Cláusulas
  - Acesso a dados
  - Controle de execução
- Sincronização
  - Condições de corrida
- Exercícios - OpenMP

Slides em http://www.ppenteado.net/ast/pp_para_on_3.pdf

Exemplos em http://www.ppenteado.net/ast/pp_para_on/
Artigo relacionado: http://www.ppenteado.net/papers/iwcca/iwcca_pfp.pdf
pp.penteado@gmail.com

Programa

4 – MPI

- Motivação
- Características
- Estruturação
- Formas de comunicação
- Principais funções
  - Controle
  - Informações
  - Comunicação
- Boost.MPI
- Sincronização
  - Deadlocks
- Exercícios - MPI

Slides em http://www.ppenteado.net/ast/pp_para_on_4.pdf

Exemplos em http://www.ppenteado.net/ast/pp_para_on/
Artigo relacionado: http://www.ppenteado.net/papers/iwcca/iwcca_pfp.pdf

pp.penteado@gmail.com