Projecto e Construção de Programas Paralelos

António M. Pina

Departamento de Informática

Universidade do Minho

Campus de Gualtar

Braga, Portugal

Tel (053)[60]4472

Email: pina@di.uminho.pt

Paralelismo e Computação

Computador paralelo – conjunto de processadores capazes de cooperar para resolver um problema computacional.

- Super-computadores com centenas ou milhares de processadores;
- Redes de estações de trabalho;
- Multiprocessadores.

Capacidade de concentrar recursos para resolver problemas complexos

- Processadores;
- Memória;
- Dispositivos de E/S.

Tendências nas Aplicações

Processamento paralelo – uso de múltiplos processadores que executam partes de um mesmo programa simultaneamente.

- Tradicionalmente – Simulação de sistemas complexos
  - previsão climatérica
  - circuitos electrónicos
  - processos industriais
  - reacções químicas.

À medida que uma tecnologia satisfaz as aplicações existentes, novas aplicações aparecerão que exigirão o desenvolvimento de novas tecnologias

- Presentemente – Aplicações comerciais
  - vídeo-conferências
  - auxiliares ao diagnóstico médico
  - realidade virtual

Tendências no Projecto de Computadores

A capacidade computacional dos mais rápidos computadores têm vindo a crescer exponencialmente desde 1945, com um factor de 10 vezes por quinquénio.

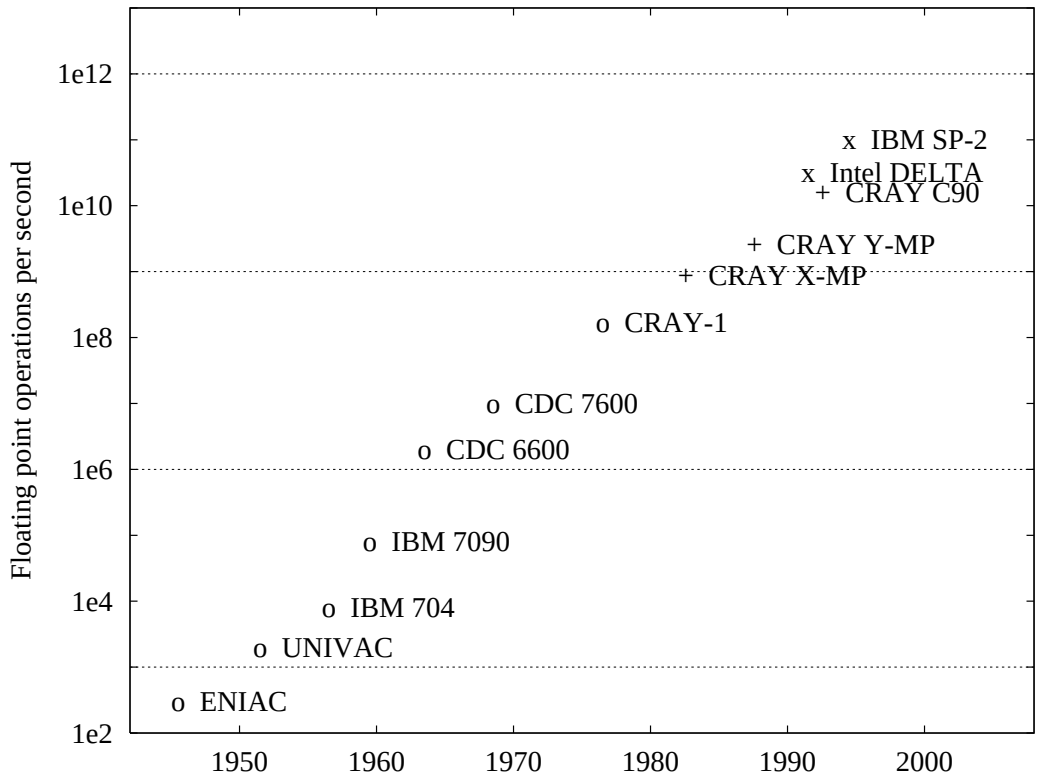

Figura 1: Rendimento de alguns supercomputadores, 1945-1995. "o"Uniprocessadores,"x" MPP

A capacidade de um computador depende directamente do tempo necessário para executar uma operação básica e do número de operações que podem ser executadas concorrentemente.

- Limitações nos uni-processadores
  - Ciclo de relógio do processador
  - Limite físico da velocidade da luz
- Soluções
  - Concorrência interna *pipeline*
  - Múltiplas unidades funcionais
- VLSI
  - Área **A** e o tempo **T** requerido para executar um determinado cálculo relacionados por $\mathbf{AT^2}$
  - Para diminuir de k, o tempo necessário para mover a informação, a área tem de aumentar k^2.

Perspectivas

Não podemos continuar a depender da existência de processadores mais rápidos para elevar o desempenho global de um sistema computacional.

- Difícil construir componentes individuais muito rápidos
- Pode ser mais económico usar um maior número de componentes mais baratos

- Custo de um computador é $\simeq$ proporcional ao número de componentes
- Aumento do número de processadores por integrado

Com uma área $n^2 * A$ de silício, podemos ter n^2 componentes de tamanho A, com tempos de execução T, ou um só componente com tempos de execução T/n.

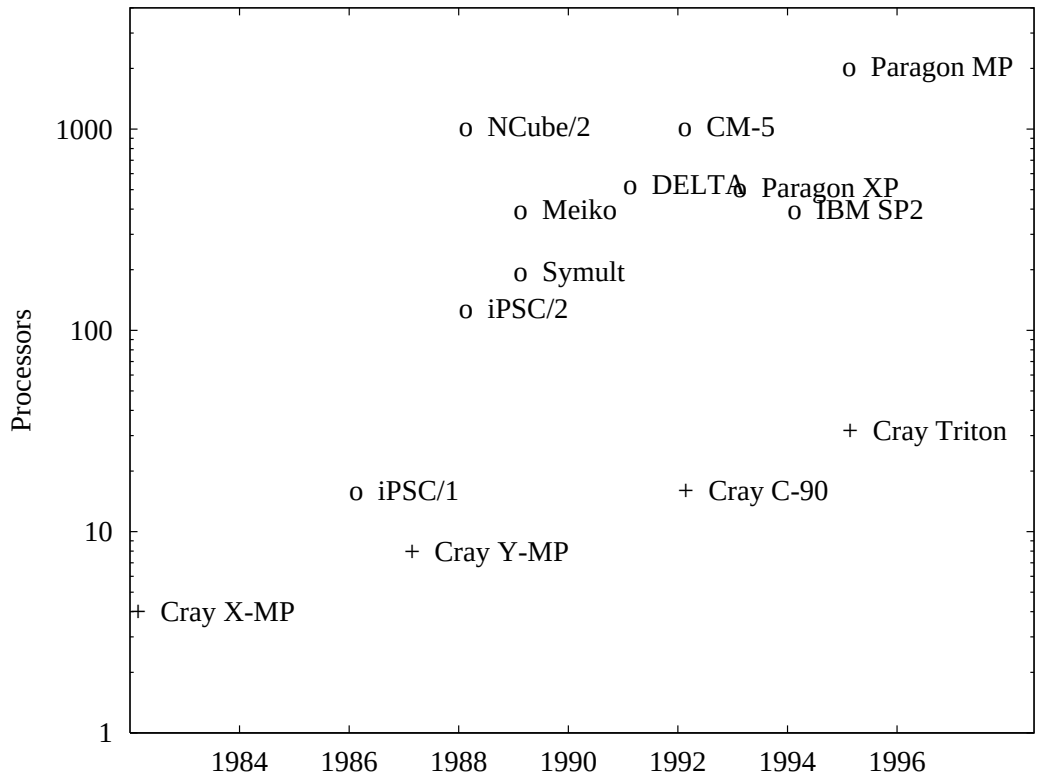

Figura 2: Número de processadores. "+"processadores vectoriais,"o" MPP

Tendências em Interconexão

Num sistema distribuído a falha de um computador, do qual se desconhece a existência, pode ter como consequência a inoperabilidade do nosso próprio computador.

- Prevêm-se para os finais dos anos 90
  - larguras de banda superiores a 1000 Mbits/s
  - aumento substancial do nível da fiabilidade

- Computação Paralela e Distribuída
  - aplicações que usam recurso físicos distribuídos
  - problemas: fiabilidade, segurança e heterogeneidade

Resumo de Tendências

um programa que é apenas capaz de usar um número fixo de processadores é um **mau programa.**

- Programas deverão explorar...
  - múltiplos processadores locais
  - processadores adicionais disponíveis em rede
- Assegurando ...
  - a *concorrência* dos algoritmos e programas
  - a *escalabilidade*, isto é, a portabilidade e a rentabilização do investimento

Modelo de Máquina Paralela

A rápida introdução do computador no comércio, ciência e educação deve muito à rápida divulgação do modelo de Von Neumann.

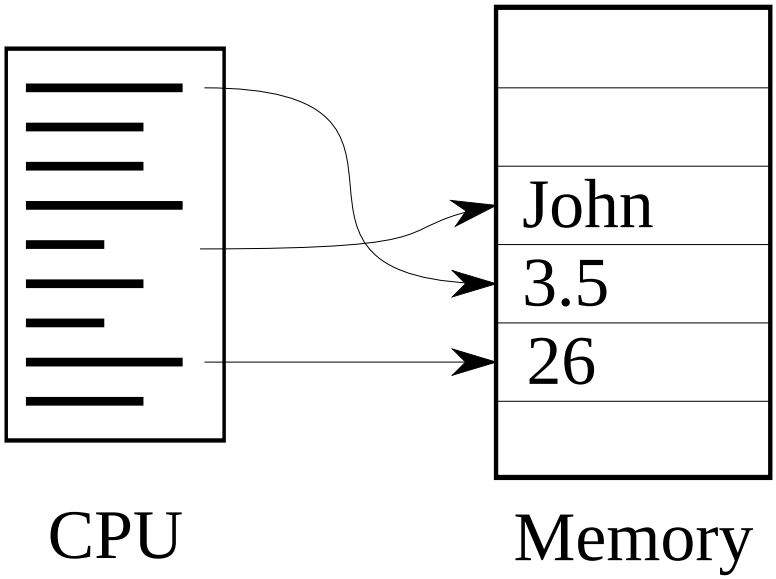

Figura 3: Modelo Von Neumann

- Modelo simples que persiste há mais de quarenta anos
- Independente da evolução das arquitecturas específicas
- Algoritmos desenvolvidos para um modelo abstracto
- Execução com razoável eficiência

O modelo de programação paralela deverá ser ao mesmo tempo geral, simples e realista

simples – para facilitar a compreensão e a programação

realista – para que os programas possam ser executados com razoável eficiência nos computadores reais

O Multicomputador

Um **Multicomputador** – conjunto de computadores ou nodos interligados por uma rede de interconexão

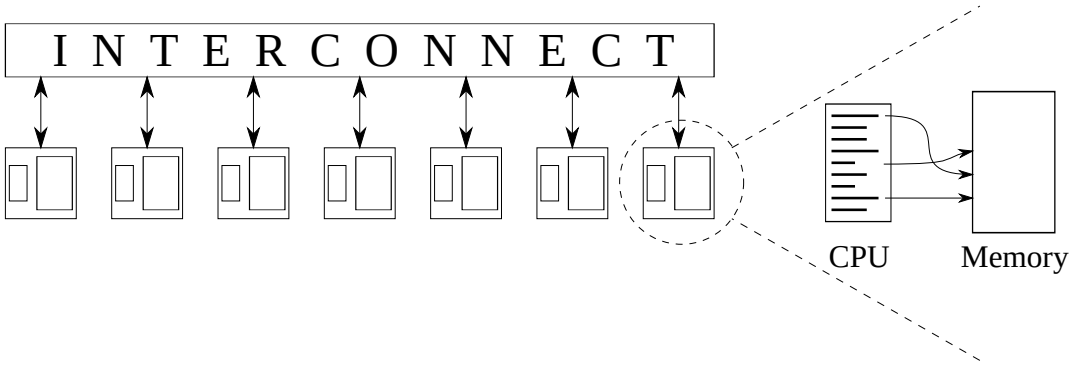

Figura 4: Multicomputador

- Programa próprio
- Acesso a memória local
- Envio e recepção de mensagens através da rede
- Custo de comunicação independente da localização dos nodos e do tráfego na rede. Depende apenas do tamanho da mensagem.

O modelo prevê que o acesso à memória local seja menos oneroso que o acesso à memória nos noutros nodos.

- Localidade
  - Propriedade fundamental do *software* paralelo.
  - Ler e escrever na memória é menos oneroso que enviar e receber mensagens
  - Privilegiar os acessos locais

- Rácio local/remoto depende...
  - Velocidade do computador
  - Velocidade da rede e da tecnologia
  - Protocolos de comunicação

Outros Modelos

MIMD – computadores paralelos que aplicam simultaneamente, múltiplas instruções a múltiplos conjuntos de dados

Memória distribuída

A memória esta espalhada por diferentes processadores ao invés de estar concentrada num local central.

- É no essencial idêntico ao modelo do multicomputador
- Os custos de comunicação dependem da localização física dos computadores e das condições de carga na rede

Memória Partilhada

Todos os processadores têm igual acesso a uma memória central através de um barramento

- Vulgarmente conhecido por multiprocessador
- Os processadores acedem a qualquer célula de memória em tempo fixo
- Usam-se muitas vezes hierarquias de barramentos e *caches*
- Realização eficiente do modelo de comunicação por passagem de mensagens

SIMD – computadores paralelos em que todos os processadores executam a mesma linha de códigos de instruções, sobre diferentes pedaços de dados.

- O modelo pode reduzir a complexidade tanto do hardware como do software
- Apropriado para problemas caracterizados por elevados níveis de regularidade
- Processamento de imagem e simulações numéricas

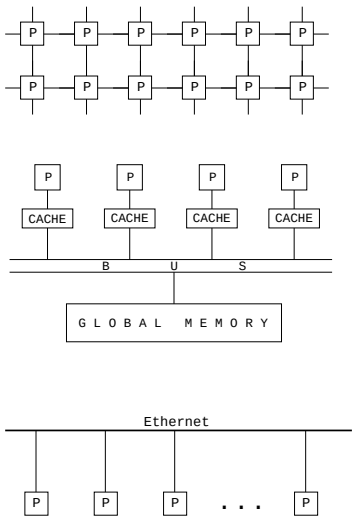

Figura 5: Classes de Computadores Paralelos

Um Modelo de Programação Paralela

Programação paralela – necessita de mecanismos que favoreçam a discussão explícita da concorrência, da localidade, da escalabilidade e da modularidade

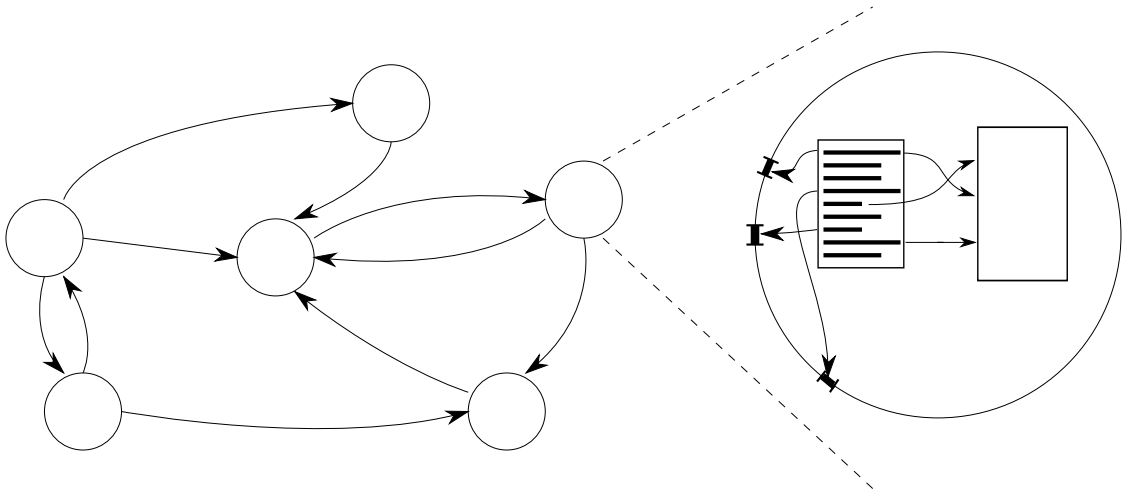

Figura 6: Tarefas e Canais

Tarefas e Canais

- Tarefas
  - Executam concorrentemente podendo o seu número variar dinamicamente
  - Programa sequencial e uma memória local
  - Portas para a entrada e a saída de dados
  - Ler e escrever na sua memória local (máquina Von Neumann virtual)
  - enviar ou receber mensagens, pelas portas correspondentes
  - criar novas tarefas e terminar.

- Mensagens
  - Envio de mensagens é uma operação assíncrona que termina imediatamente após o processamento dos dados.
  - A recepção de mensagens é uma operação assíncrona que bloqueia a tarefa até receber os dados
  - Canais são pares de portas de entrada e saída

- Processadores
  - As tarefas podem ser alocadas a processadores físicos, de várias maneiras
  - Em particular, várias tarefas podem ser alocadas a um mesmo processador

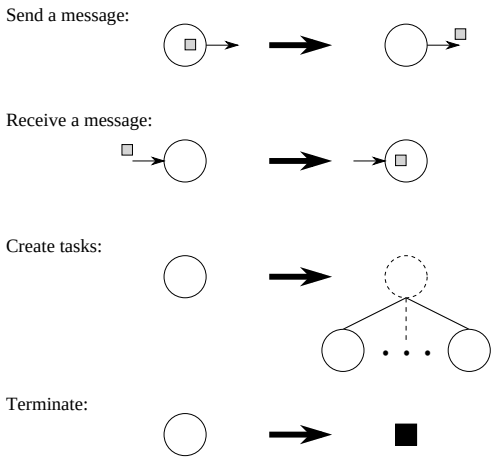

Figura 7: Acções das Tarefas

Propriedades do Modelo

Dependência dos dados – A noção de tarefa fornece os mecanismo necessários para a abordagem da localidade

Rendimento

Uma tarefa representa uma linha sequencial de instruções que está a ser executada num único processador.

A comunicação entre duas tarefas é directamente realizada como uma comunicação entre processadores.

Independência

O mecanismo uniforme de interacção permite que o resultado de um programa não dependa da topologia de localização das tarefas.

Há noção de escalabilidade porque, se o número de processadores aumentar o algoritmo não necessita de ser modificado.

Modularidade

A tarefa é um bloco de modularização natural que usado convenientemente pode reduzir a complexidade da programação e facilitar a reutilização de código.

Determinismo

É o próprio modelo de interacção que facilita a escrita de programas determinísticos.

A tarefa receptora tem de ficar bloqueada até que a operação e cada canal só pode ter um emissor e um receptor.

Outros Modelos de Programação

O modelo tarefa/canais não é certamente o único disponível para representar a computação paralela.

Passagem de Mensagens

- O mais utilizado actualmente
- Não difere muito do modelo tarefa/canais
- interacção utilizando identificadores de tarefas
- Modelo de programação **SPMD** quando cada tarefa executa o mesmo programa mas opera em dados diferentes

Memória Partilhada

- As tarefas partilham um espaço de memória comum que pode ser lido e escrito assincronamente
- O acesso à memória é controlado por mecanismos do tipo: bloqueamento e semáforos
- Difícil compreender e manipular a localidade
- Difícil de escrever programas determinísticos
- Simplifica o desenvolvimento de programas

Paralelismo nos Dados

- Aplicação da mesma operação a múltiplos elementos de uma dada estrutura
- Cada operação, num determinado elemento, pode ser pensada como uma tarefa independente
- A granularidade dos dados é naturalmente pequena, não se põe a questão da localidade dos dados
- Por exemplo – somar o valor 2 a todos os elementos de uma lista
- Frequentemente, os compiladores traduzem programas paralelo para uma descrição tipo SPMD, gerando o código automaticamente

Resumo

O modelo tarefas/canais simplifica a programação porque recorre a abstracções que são a base para a construção de aplicações paralelas modulares.

Concorrência
: Propriedade essencial dos programa que deverão ser executado simultaneamente em mais que um processador.

Escalabilidade
: Adaptabilidade ao aumento do aumento de processadores. Importante se considerarmos a tendência para o aumento do número de processadores, num multicomputador.

Localidade

Um rácio local/remoto muito elevado no acesso à memória.

É a chave para a obtenção de elevados rendimentos em arquitecturas do tipo multicomputador.

Modularidade

Decomposição de um problema complexo em componentes mais simples. Essencial, tanto na programação paralela, como na programação sequencial.

Desenho de Algoritmos Paralelos

O desenho de algoritmos paralelos não é o resultado da aplicação da mesma receita a qualquer tipo de problema.

Metodologia

- criatividade
- mais que uma solução paralela
- independência da máquina específica
- avaliação das alternativas
- reduzir os custos de recuperação de uma má solução.

Metodologia

O desenho de algoritmos paralelos pode ser visto como um processo de elevado nível de concorrência, em que mais que um factor tem de ser pesado simultaneamente.

1. *Partição* – procuram-se reconhecer as oportunidades para a execução paralela
2. *Comunicação* – determinam-se os requisitos de comunicação necessários à coordenação entre tarefas
3. *Agrupamento* – combinam-se várias tarefas numa única grande tarefa
4. *Arranjo* – alocação das tarefas pelos processadores físicos

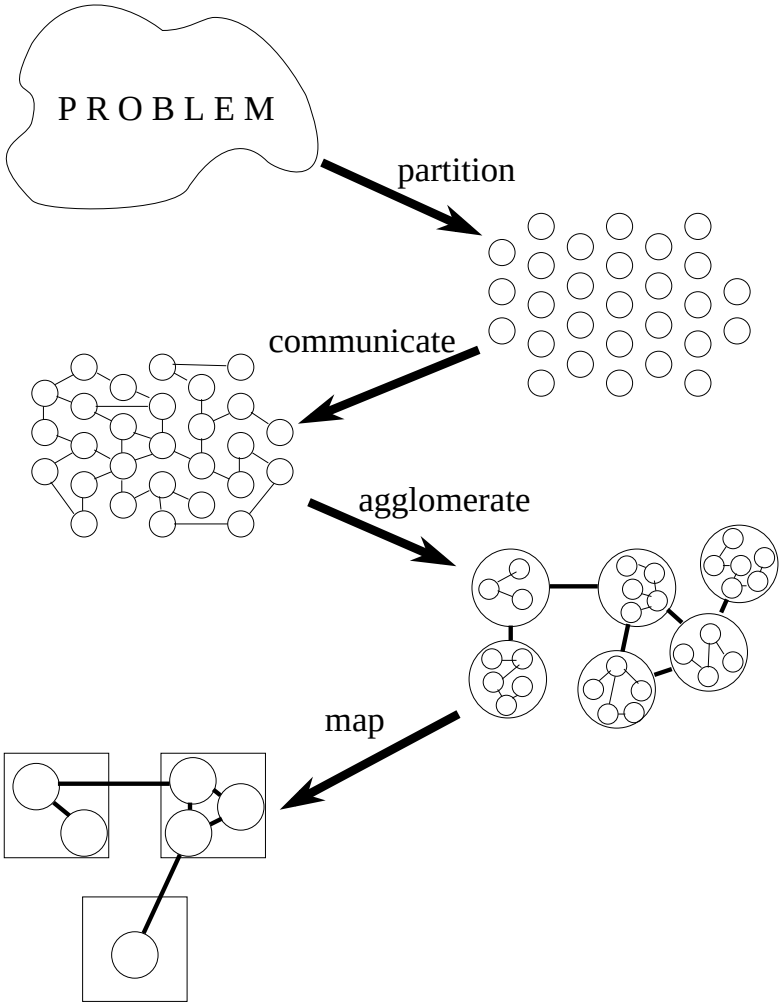

Figura 8: Método de desenho de algoritmos paralelos

Partição

Encontrar o maior número possível de partes, susceptíveis de serem executadas em paralelo – definir o grão mais fino de decomposição.

Evitar replicação de cálculos e de dados

- Decomposição em Funções

  1. decompor em partes computacionais
  2. tratar os dados

- Decomposição em domínios
    1. determinar a partição apropriada para os dados
    2. integrar dados e computação

Decomposição em Domínios

Dirigir a atenção para as estruturas de dados maiores ou para as que são mais frequentemente acedidas.

Dados de entrada, de saída e intermédios

- decompor em pequenas partes com tamanhos aproximadamente iguais
- dividir os cálculos em tarefas a que se associam alguns dados e um conjunto de operações

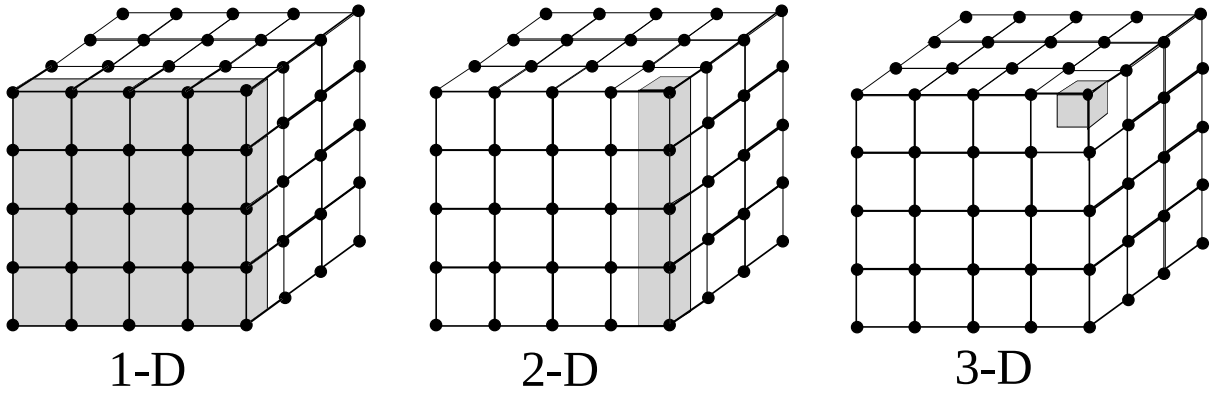

Figura 9: Ilsutração da técnica de decomposição em domínios. A decomposição tridimensional é a mais flexível

Decomposição em Funções

A decomposição em funções pode revelar a estrutura de um problema fornecendo a base para uma optimização que de outra forma permaneceria escondida.

Ênfase nos cálculos

- dividir os cálculos em grupos disjuntos de tarefas
- simplifica o desenvolvimento dos programas é associada à decomposição de código

Decomposição Mista

A decomposição em funções permite pensar cada componente de um problema complexo como uma tarefa separada. Os componentes podem, por sua vez, ser naturalmente paralelizáveis usando decomposição em domínios

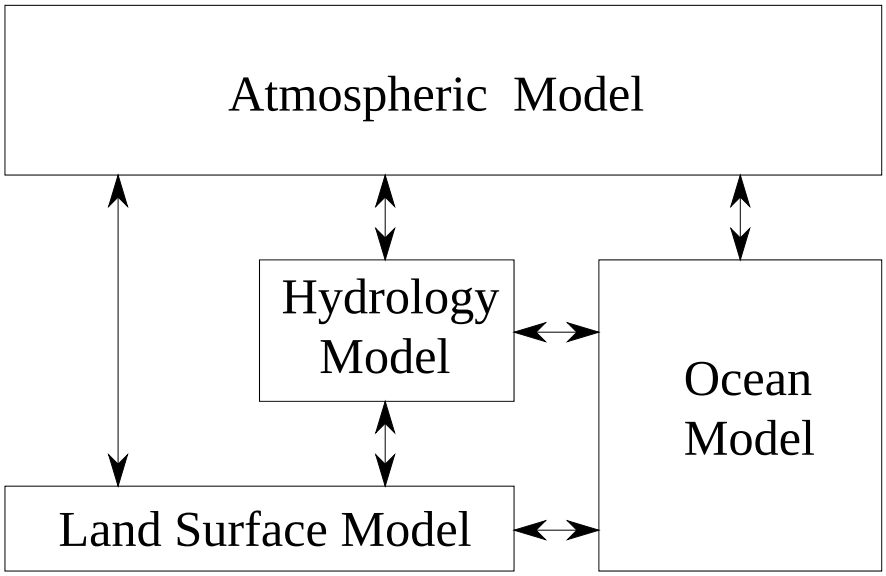

Figura 10: Decomposição Mista na simulação do clima da terra.

Análise de Partições

Para assegurar que o desenho não contém falhas óbvias, à lista de perguntas que se seguem dever-se-á, normalmente, responder afirmativmente.

1. O número de tarefas é superior, pelo menos numa ordem de grandeza, ao número de processadores físicos?
2. A partição evita a redundância nos cálculos e requisitos de armazenamento dos dados?
3. As tarefas têm todas um tamanho comparável?
4. O número de tarefas é proporcional ao tamanho do problema?
5. É possível identificar mais que uma partição alternativa?

Comunicação

As tarefas numa partição apesar de concorrentes podem, em geral, não ter execução independente.

- Estabelecimento da comunicação
  1. Definir a estrutura de canais que ligam as tarefas consumidoras às tarefas produtoras
  2. Especificar as mensagens para envio ou recepção
- Decomposição

  Domínios – díficil de determinar requisitos de comunicação

  Funções – a comunicação corresponde ao fluxo de dados

- Garantir a execução concorrente
- Evitar a criação de canais e comunicações desnecessárias

Para identificar os requisitos de comunicação e as operações associasdas consideram-se os seguintes eixos de discussão

- Local/Global
  - cada tarefa comunica apenas com as tarefas vizinhas
  - cada tarefa comunica com muitas outras tarefas
- Estruturada/Não-estruturada
  - as tarefa em conjunto com todas tarefas vizinhas constituem uma estrutura regular
  - a rede de comunicações é um grafo arbitrário
- Estática/Dinâmica
  - a identidade dos parceiros comunicantes é fixa
  - a identificação dos parceiros é determinada durante a execução
- Síncrona/Assíncrona
  - as tarefas comunicam coordenadamente
  - não há qualquer tipo de coordenação

Comunicação Local

Uma estrutura de comunicação local resulta de uma operação que precisa de dados de um pequeno número de outras tarefas.

Diferenças Finitas de Jacobi

- grelha multidimensional
- actualização do valor de cada ponto
- aplicação de uma função aos pontos de *matriz*

$$X_{i,j}^{t+1} = \frac{4X_{i,j}^{(t)} + X_{i-1,}^{(t)} + X_{i+1,j}^{(t)} + X_{i,j-1}^{(t)} + X_{i,j+1}^{(t)}}{8}$$

Decomposição em Domínios

- uma tarefa atribuída a cada ponto $X_{i,j}$
- tantos canais quanto o número de vizinhos
- cada tarefa efectua t passos sequenciais
- para cada passo são precisos os valores das tarefas vizinhas

para $t = 0$ até $T - 1$

enviar $X_{i,j}^{(t)}$ para cada vizinho

receber $X_{i-1,j}^{(t)}, X_{i+1,j}^{(t)}, X_{i,j-1}^{(t)}, X_{i,j+1}^{(t)}$ dos vizinhos

– calcular $X_{i,j}^{(t)}$ usando a fórmula

parafim

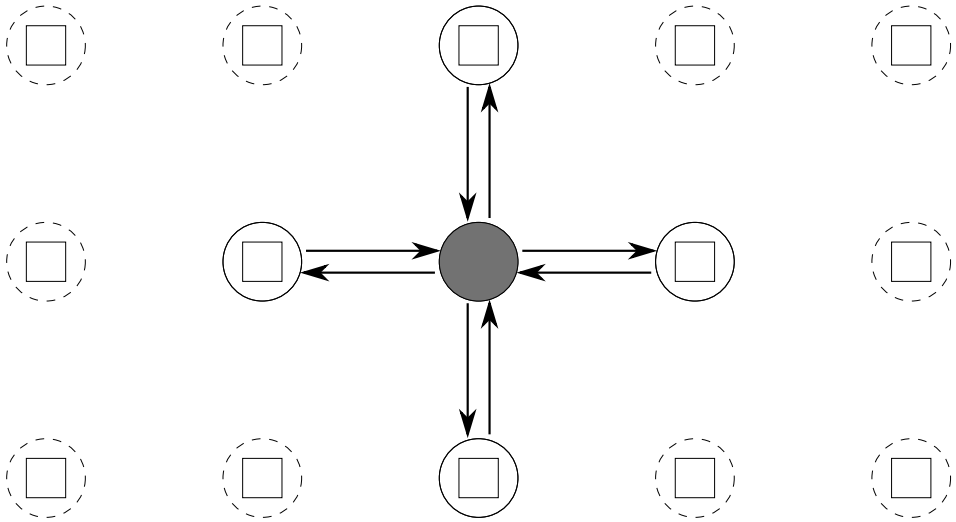

Figura 11: Estrutura de canais e tarefas para um problema de diferenças finitas usando 5 pontos de matriz

Comunicação Global

Corresponde a uma comunicação que envolve muitas tarefas.

- não chega identificar os pares consumidor/produtor
- demasiadas comunicações
- restrição à oportunidade para a execução concorrente

Cálculo do resultado S de uma operação *paralela de redução* que reduz N valores distribuídos por N tarefas

$$S = \sum_{i=0}^{N-1} X_i.$$

- tarefa mestre
- valores $X_0, X_1 ...$ distribuídos pelas tarefas $0, 1..$
- estrutura de comunicação independente
- tarefa mestre apenas pode somar um valor de cada vez
- tempo de $O(N)$ para adicionar os N números
- má solução paralela

Uma visão puramente local da comunicação pode prejudicar a execução eficiente de algoritmos paralelos

- Algoritmo *centralizado*, não distribui nem a computação nem a comunicação.
- Algoritmo *sequencial*, não favorece a execução concorrente de múltiplas comunicações e cálculos.

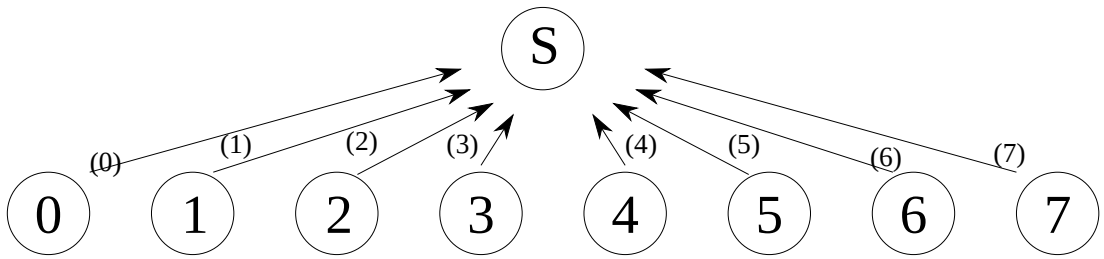

Figura 12: Algoritmo centralizado de redução por soma

Comunicação e Computação Distribuída

Algoritmo que distribui $N-1$ comunicações e somas, mas só permite a execução concorrente se o problema incluir múltiplas operações de soma.

$$S_i = X_i + S_{i-1}.$$

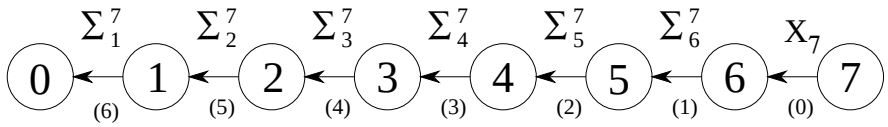

Figura 13: Algoritmo distribuído de redução por soma

- N tarefas são ligadas numa fila e $0 < i < N-1$
  1. a tarefa $(N-1)$ envia o seu valor para o seu vizinho na fila
  2. cada tarefa (1 a $N-2$) recebe a soma parcial da direita, soma-a ao seu próprio valor e envia o resultado para a esquerda.
- a tarefa 0 calcula a soma global adicionando ao seu próprio valor o valor recebido da direita.

Dividir para Paralelizar

Se um problema complexo puder ser dividido em dois ou mais sub-problemas, o processo é aplicado recursivamente, até não serem possíveis mais subdivisões

$$\sum_{i=0}^{2^n-1} = \sum_{i=0}^{2^{n-1}-1} + \sum_{i=2^{n-1}}^{2^n-1}.$$

- *Soma de N números $N = 2^n, n > 0$ para n inteiro*
  - pode haver subdivisão se $n > 1$
  - gera uma estrutura regular de comunicação em árvore
  - profundidade da árvore $n = \log n$
- *Execução concorrente ao mesmo nível na árvore*
  - comunicação limitada às tarefas vizinhas
  - tempo total de execução $O(\log n)$

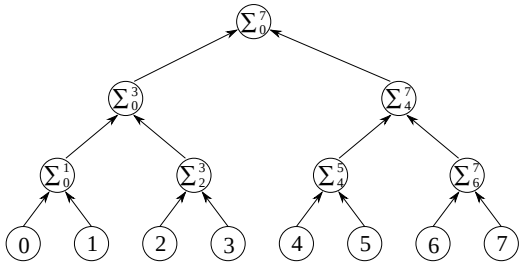

Figura 14: Estrutura em árvore ($N = 8$) para algoritmo do tipo Dividir para paralelizar

Comunicação Dinâmica Não Estruturada

A comunicação não estruturada não causa dificuldades conceptuais especiais nas fases iniciais de desenho.

Problemas nas fases mais adiantadas

- quando se agrupa tarefas
- quando se distribuem as tarefas pelos processadores

Em comunicação dinâmica

- algoritmos para determinar tamanhos óptimos de tarefas
- algoritmos para determinar estruturas de comunicação mínimas
- calcular a relação custo/benefício de desenvolvimento dos algoritmos

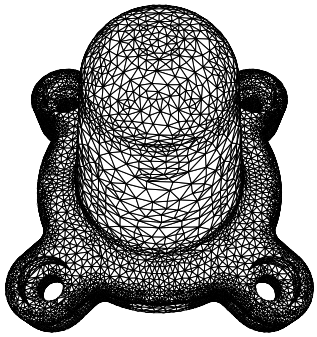

Figura 15: Comunicação não estruturada. Notar que vértices diferentes tem números diferentes de vizinhos

Comunicação Assíncrona

A comunicação assíncrona ocorre, frequentemente, quando tarefas distintas têm que aceder periodicamente aos elementos de uma estrutura de dados partilhada.

- Estrutura de dados muito grande ou alvo de acessos frequentes
  - dados distribuídos por mais que uma tarefa
  - mecanismo capaz de garantir operações assíncronas de leitura e escrita

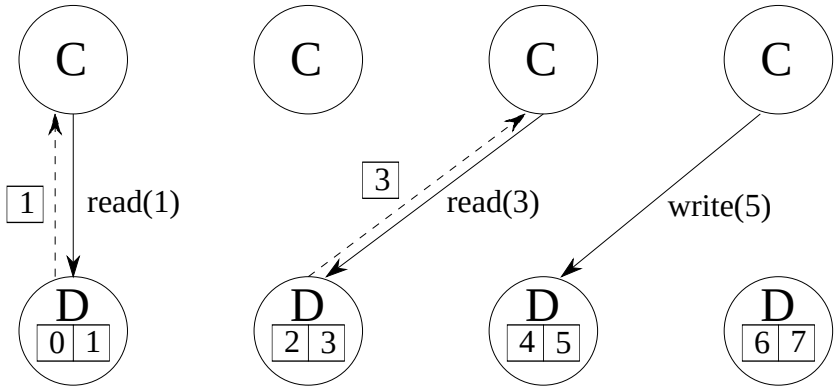

Figura 16: Uso de tarefas auxiliares (D) para leitura e escrita de dados numa estrutura distribuída. Computação realizada pelas tarefas (C)

a) *A estrutura de dados é distribuída pelas tarefas*

   - as tarefas fazem cálculos
   - geram pedidos de dados noutras tarefas
   - esperam a conclusão de pedidos pendentes

b) *Conjunto auxiliar de tarefas*

   - a distribuição da estrutura de dados num segundo grupo de tarefas
   - respondem apenas a pedidos de leitura e escrita

- Inconvenientes

  a) programas emaranhados e não modulares

  b) é difícil explorar a localidade dos dados

Análise da Comunicação

Para identificar características não escaláveis no desenho de algoritmos, à lista de perguntas que se seguem dever-se-á, normalmente, responder afirmativmente.

Perguntas

1. O número de operações realizadas por todas as tarefas é aproximadamente igual?
2. Cada tarefa comunica apenas com um número diminuto de tarefas vizinhas?
3. As comunicações podem ser realizadas concorrentemente?
4. As várias computações podem ser realizadas concorrentemente.

Em caso negativo ...

1. Distribuir ou replicar as estruturas de dados.
2. Formular a comunicação global com base numa estrutura de comunicações locais
3. Experimentar técnicas de *dividir para paralelizar* que possam exponham a concorrência.
4. Revisitar a especificação ou encontrar uma nova sequência de comunicações e computações que favoreçam a concorrência.

Agrupamento

Com o agrupamento pretende-se passar do mundo abstracto para a realidade

- *Fases Anteriores*

  Partição – a computação dividida em tarefas

  Comunicação – meio de fornecer dados às tarefas

- *Agrupamento*
  - combinar ou agrupar as tarefas
  - replicação de computação e/ou dados
  - explorar arquitecturas paralelas específicas

- *Problemas*
  - mais tarefas que processadores físicos
  - como fazer o arranjo de processadores
- *Alternativa*
  - se o computador exigir um programa **SPMD**
  - P tarefas executam em P processadores

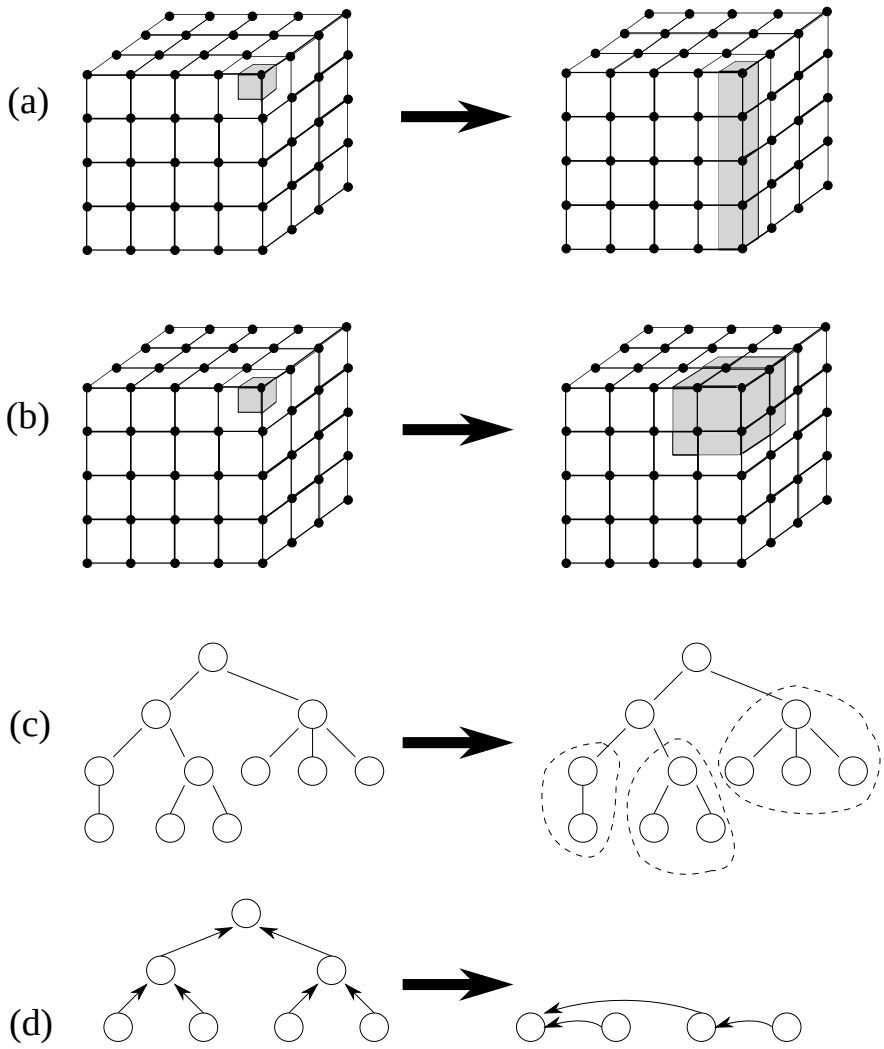

Figura 17: Exemplos de Agrupamento. a) redução do nível de decomposição. b) aumento do tamanho do grão. c) junção de ramos de árvore. d) combinação de nodos num algoritmo.

As decisões relativas ao agrupamento e replicação podem gerar conflitos face aos seguintes objectivos:

- *Reduzir custos*

  computação/comunicação – aumentando *granularidade*

  económicos – no desenvolvimento da aplicação

- *Flexibilidade*

  escalabilidade – da solução

  arranjo – dos processadores

Aumentar a Granularidade

Encontrar o maior número possível de tarefas é uma disciplina útil que não conduz necessariamente à obtenção de um algoritmo eficiente.

- Custo de criação de tarefas
- Custos de comunicação
  - suspensão da computação para enviar e receber mensagens
  - custos fixos dependentes do mecanismo de comunicação
- Para reduzir o tempo de comunicação
  - transferir menor volume de dados
  - reduzir o número de mensagens

Efeitos Superfície-Volume

A redução dos custos de comunicação pode ser obtida através do *efeito superfície-volume*

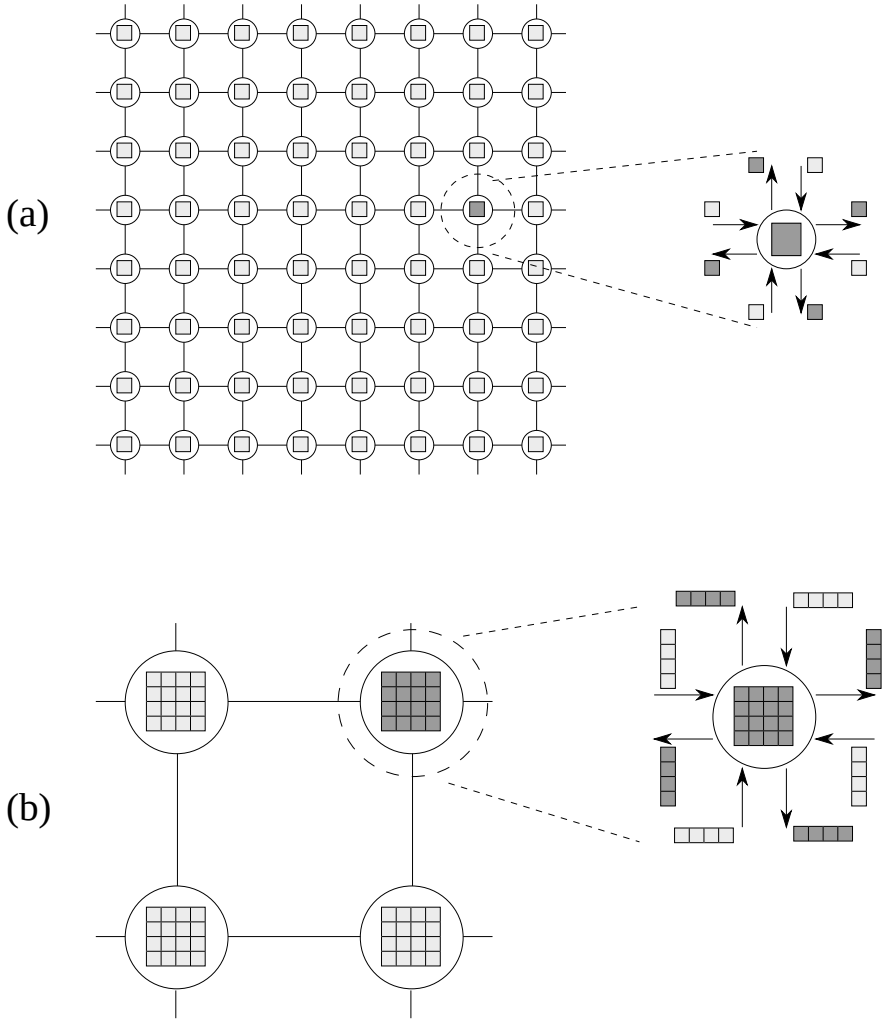

Figura 18: Efeitos do aumento de granularidade nos custos de comunicação a) `8*8 tarefas - 256 comunicações` b) `2*2 tarefas - 16 comunicações.`

- Requisitos por subdomínio de operação

  comunicação – proporcionais à superfície

  computação – proporcionais ao volume

- Problema bidimensional

  comunicação – proporcional ao tamanho

  computação – proporcional ao quadrado do tamanho

Estratégias de Agrupamento

- *Rácio comunicação/computação* – diminui com o aumento do tamanho das tarefas
- *Para melhorar a eficiência*
  - partir de uma elevada decomposição dimensional
  - em vez de reduzir a dimensão da decomposição
- *Comunicação não estruturada* – para um agrupamento eficente é necessário utilizar técnicas especializadas

Replicação da Computação

É possível encontrar um compromisso entre computação replicada e a redução do tempo de comunicação e/ou o tempo de execução.

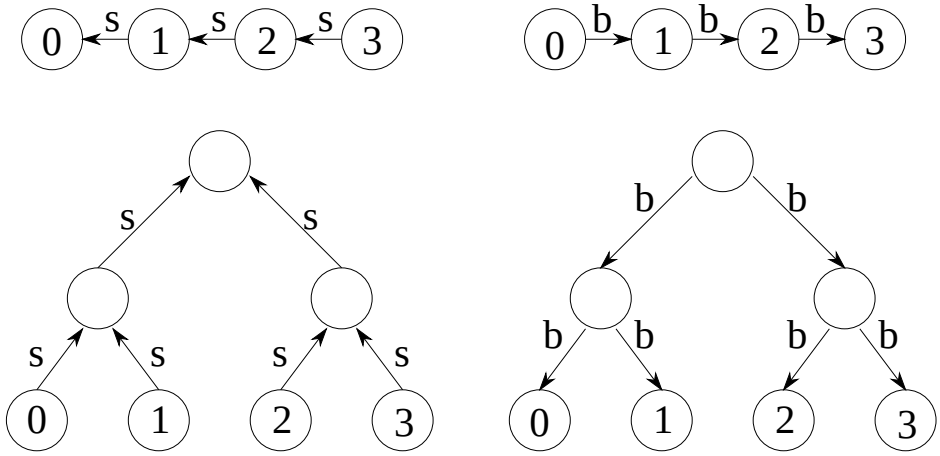

Figura 19: Soma e difusão do resultado numa topologia em lista) ou árvore.

- *Somatório replicado nas N tarefas*
  - algoritmo optimizado
  - não há comunicação ou computação redundante
- *Passos de difusão*

  lista – $2(N-1)$

  árvore – $2 \log N$

Variante que utiliza somas concorrentes

- *Topologia em Anel*
  - todas as tarefas executam o mesmo algoritmo
  - N somas parciais calculadas simultaneamente
  - $N - 1$ passos para obter a soma em todas as tarefas

- *Análise da variante*
  - evita subsequentes operações de difusão
  - adições e comunicações redundantes $(N - 1)^2$
  - passos totais para completar a adição $N - 1$

O algoritmo da soma em árvore pode ser modificado para evitar difusões em separado.

- *Estrutura de Comunicação em Borboleta*
  - número de passos $\log N$
    . cada tarefa recebe dados de duas tarefas
    . calculo da adição em cada tarefa
    . envio da soma para duas tarefas no próximo passo
  - adições e comunicações redundantes $N \log N$

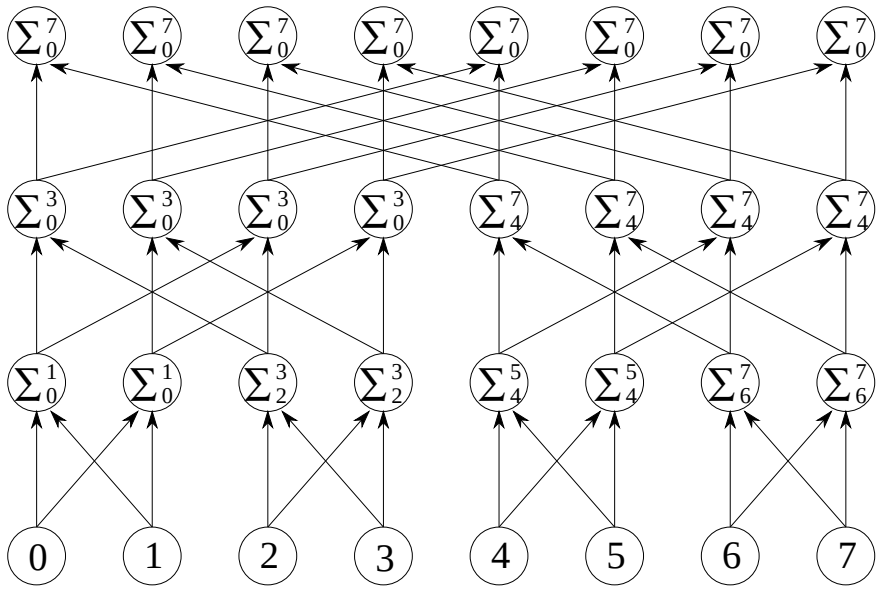

Figura 20: Estrutura de Comunicação em *borboleta*.

Evitando a Comunicação

O agrupamento é normalmente benéfico se os requisitos de comunicação revelarem que há conjuntos de tarefas que não podem ser executadas concorrentemente.

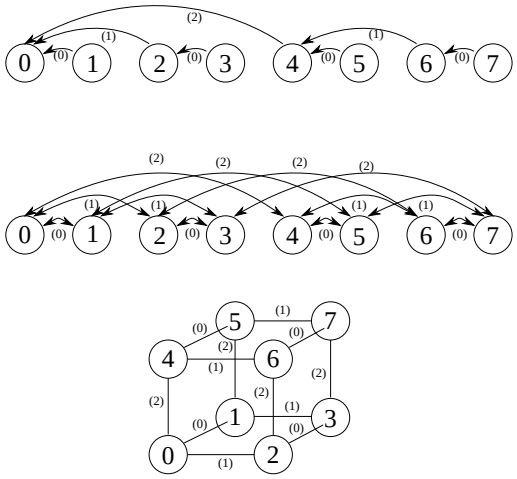

Figura 21: Estrutura de comunicação resultantes do agrupamento de tarefas ($N = 8$) em níveis diferentes numa *árvore* ou *borboleta*

- Soma simples
  - apenas as tarefas ao mesmo nível executam concorrentemente
  - para múltiplas operações de soma todas as tarefas podem estar em actividade
  - as tarefas nos diferentes níveis podem ser agrupadas

Preservação da Flexibilidade

É importante no desenho de um algoritmo não limitar, desnecessariamente, o número de tarefas que podem ser criadas porque se pode por em causa a sua escalabilidade.

- *Escalabilidade*
  - número variável de tarefas
  - adaptação ao número de processadores
  - portabilidade
- *Sobreposição computação/comunicação*
  - tarefas bloqueiam frequentemente à espera de dados
  - várias tarefas num único processador
  - aproveitar tempos de ócio do processador

- *Arranjo de processadores*
  - mais tarefas que processadores
  - aumento da gama de estratégias de arranjo
  - balanceamento da carga computacional
  - rácio *tarefas/processadores* pode atingir uma ordem de grandeza
- *Flexibilidade na criação da tarefas*
  - durante a compilação
  - parâmetro de evocação do programa

Redução dos Custos de Desenho

A estratégia de agrupamento deve ter em atenção os custos relativos de desenvolvimento, associados às diferentes estratégias de partição.

Teste de Agrupamento

O agrupamento de tarefas depende dos requisitos de comunicação, da necessidade de aumentar a granularidade da computação e da comunicação e/ou diminuir os custos desenho.

- *Perguntas sobre agrupamento*

  1. reduziu os custos de comunicação, por aumento da localidade?
  2. teve como efeito a replicação de computação?
  3. teve como efeito a replicação de dados?
  4. todas as tarefas têm custos equitativos de comunicação e de cálculo?
  5. o número de tarefas é proporcional ao tamanho do problema?
  6. no caso do agrupamento ter diminuído as oportunidades para a execução paralela, está garantida suficiente concorrência?
  7. a diminuição do número de tarefas introduz problemas: de balanceamento de carga, de custos de desenvolvimento, ou de redução da escalabilidade?
  8. se está a paralelizar um algoritmo sequencial consideram-se os custos de modificação do código?

- *Em caso negativo ...*

  1. rexaminar o algoritmo para verificar se há estratégias de agrupamento alternativa que conduzam a esse fim.
  2. verificar se os benefícios da replicação ultrapassam os custos – variando as dimensões do problema e o número de processadores.
  3. verificar se a escalabilidade do problema não fica comprometida quando se restringem as dimensões do problema, ou o número de processadores.
  4. quanto maiores forem as tarefas mais importante é igualizar os custos, especialmente quando há tantas tarefas como processadores.
  5. o algoritmo não poderá ser usado para resolver grandes problemas
  6. um algoritmo com concorrência limitada poderá continuar a ser o mais eficiente.
  7. em caso afirmativo não esquecer que os algoritmos que criam um número pequeno de tarefas gordas são muitas vezes mais simples e mais eficientes do que os que geram muitas tarefas finas.
  8. considerar algoritmos alternativos que aumentem as oportunidades de reutilização de código.

Arranjo

Na última fase do processo de desenho, procura-se minimizar o tempo de execução distribuindo, convenientemente, as tarefas pelos processadores.

- *Máquinas uni/multi processador de memória partilhada*
  - basta especificar as tarefas e as comunicações
  - sistema operativo ou hardware escalona as tarefas
- *Máquinas paralelas*
  - problema de grande dificuldade
  - não existem mecanismos automáticos

- *Estratégias para minimizar o tempo de execução*

  1. tarefas potencialmente concorrentes – em *diferentes* processadores
  2. tarefas que interagem frequentemente – no *mesmo* processador

- Problemas

  – limitação de recursos

  – restrição ao número de tarefas/processador

O arranjo é um problema `NP-completo` porque não há nenhum algoritmo polinomial para encontrar a distribuição óptima de processadores.

- *Classes de problemas*
  - número fixo de tarefas do mesmo tamanho
  - comunicações estruturadas, locais e globais
- *Técnicas e heurísticas*
  - minimizam a comunicação entre processadores
  - entregam aos processadores tarefas de grão médio

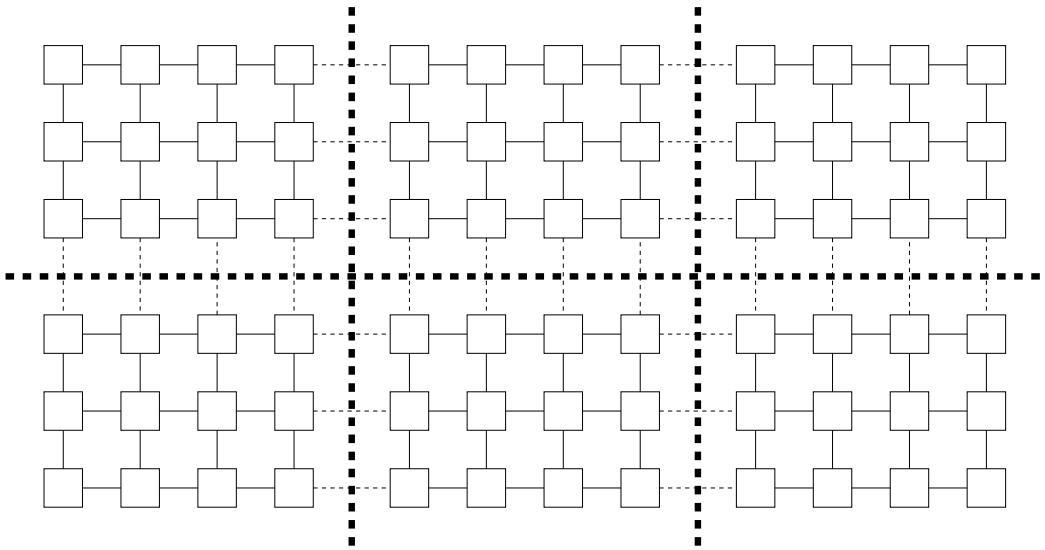

Figura 22: Arranjo de processadores. Minimização das comunicações para uma carga computacional média semelhante.

- *Classes de problemas complexos*
  - tarefas e tamanhos de tarefas variáveis
  - comunicações não-estruturadas
- *Técnicas e heurísticas*
  - **balanceamento de carga**
  - identificação de agrupamentos eficientes

O custo de execução dos algoritmos de balanceamento deverá ser contabilizado, face aos benefícios na redução do tempo de execução.

Balanceamento de Carga

- *Probabilístico*
  - não explora a estrutura da aplicação
  - baixo custo
- *Dinâmico*
  - determinar novos agrupamento e arranjos
  - execução periódica
  - grande número de execuções
  - privilegiar algoritmos **locais**
  - não precisam de conhecimento global do programa
- *Escalonamento de tarefas*
  - decomposição funcional
  - múltiplas tarefas, pequenas e de curta duração
  - alocação de tarefas a processadores ociosos
  - sincronização limitada ao ínicio e fim da execução

Algoritmos de Balanceamento de Carga

Existe uma grande variedade de técnicas de balanceamento de carga que podem ser aplicadas a algoritmos paralelos que se baseiam na decomposição em domínios.

- *Algoritmos de partição*
  - agrupar as tarefas finas, duma partição inicial, para obter tarefas de tamanho médio
  - em alternativa, o domínio de computação é dividido em subdomínios

Bissecção Recursiva

Obter subdomínios com custos de computação aproximadamente iguais, para reduzir os custos de comunicação, por diminuição do número de canais que cruzam os limites das tarefas.

- *Técnicas de bissecção*

  1. dividir o domínio ao longo de uma dimensão
  2. divisões recursivas em novos subdomínios
  3. `(número de divisões = número de tarefas)`
  4. algoritmo altamente paralelizável

Bissecção Coordenada Recursiva

Em matrizes irregulares, com uma estrutura de comunicação local, as divisões fazem-se com base nas coordenadas físicas dos pontos na matriz.

- *Técnica*
  - divisão ao longo da maior das dimensões (X)
  - em cada subdomínio os pontos têm uma ordenada maior que nos outros subdomínios
- *Vantagens*
  - abordagem simples e económica
  - bons resultados
- *Desvantagens*
  - não optimiza a eficiência das comunicações
  - pode produzir mais mensagens do que a decomposição em subdomínios quadrados

Bissecção Recursiva Não-Balanceada

Obter $P-1$ partições formando submatrizes: com $1/P$ e $(P-1)/P$ da carga, com $2/P$ e $(P-2)/P$ da carga e assim sucessivamente, escolhendo a partição que minimiza o rácio de partições.

- *Avaliação*
  - submatrizes com melhores rácios
  - reduz os custos de comunicação
  - aumenta os custos de cálculo da partição

Bissecção por grafo recursivo

Identificar as duas extremidades do grafo, entregando seguidamente cada vértice ao subdomínio correspondente à extremidade mais perto.

- *Utilização*
  - redes de elementos finitos
  - estruturas complexas não-estruturadas
- *Técnica*
  - matriz é um grafo de N vértices v_i.
  - reduz o número de lados da matriz que cruzam limites de subdomínio
  - diminui os requisitos de comunicação.

Algoritmos de Balanceamento Local

Quando a carga computacional varia constantemente, faz-se o balanceamento usando apenas informação de um número reduzido de processadores vizinhos.

- *Técnica*
  - processadores organizados numa trama lógica
  - a carga de cada processador é comparada com a dos vizinhos na trama
  - se o diferencial exceder um determinado limiar, há transferência de computação
- *Avaliação*
  - pior balanceamento de carga que os algoritmos globais
  - múltiplas operações de balanceamento local para compensar picos de carga num único processador

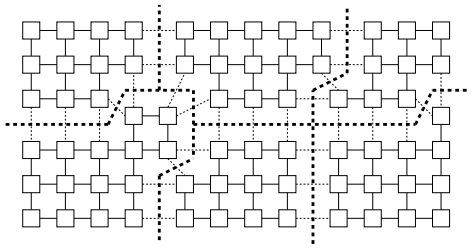

Figura 23: Distribuição de carga produzida por um algoritmo de balanceamento local

Métodos Probabilísticos

Se houver um número elevado de tarefas, distribuídas aleatoriamente pelos processadores, espera-se que a carga computacional, atribuída a cada processador, seja aproximadamente igual.

- *Utilização*
  - quando há pouca comunicação entre tarefas
  - quando há baixa localidade na comunicação
- *Vantagens*
  - baixo custo
  - escalabilidade
- *Desvantagens*
  - necessário comunicação *off-processor*, virtualmente, para todas as tarefas
  - apenas quando o número de tarefas excede largamente o número de processadores

Arranjo Cíclico

Cada um dos P processadores é alocado a cada grupo de P tarefas, de acordo com uma determinada numeração das tarefas

- *Técnica*
  - carga computacional varia, em cada ponto da matriz
  - há localidade espacial significativa na carga
  - processadores com carga computacional semelhante
- *Avaliação*
  - algoritmo probabilístico
  - aumento dos custos de computação por perda de localidade
  - uma variação do método é a **distribuição cíclica em bloco**

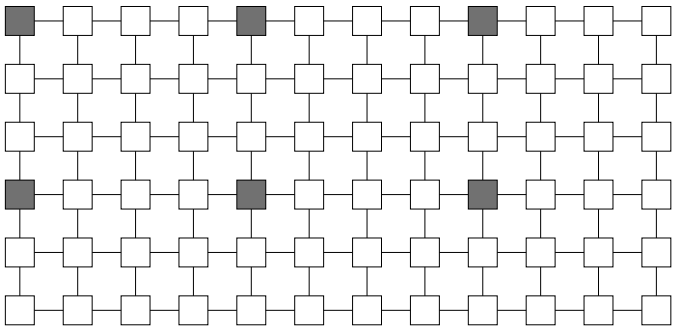

Figura 24: Distribuição de carga produzida por arranjo cíclico

Algoritmos de Escalonamento de Tarefas

As tarefas originais são transformadas em estruturas de dados que representam *problemas* para serem resolvidas por um conjunto de tarefas.

- *Técnica*
  - muitas tarefas com fracos requisitos de localidade
  - lote centralizado ou distribuído de tarefas
- *Avaliação*
  - operações independentes, para reduzir os custos decomunicação
  - conhecimento global do estado da computação, para fazer o balanceamento

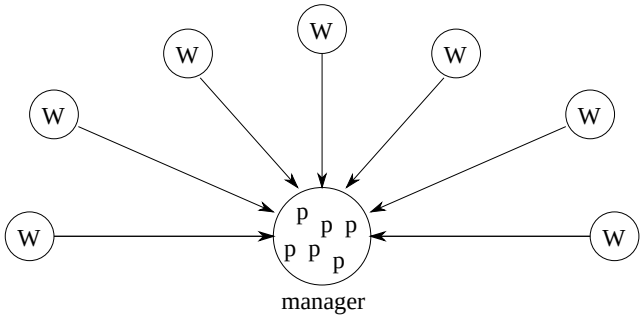

Figura 25: Algoritmo Gestor/Trabalhador

Gerente/Trabalhador

Os trabalhadores resolvem problemas, pedidas ao gerente, ou enviam novas tarefas ao gerente, para alocação a outros trabalhadores.

- *Avaliação*
  - depende do número de trabalhadores e dos custos de obtenção e execução de problemas
  - pré-busca de problemas para permitir a sobrepor computação com comunicação.
  - pré-armazenamento de problemas *cache* reduz a comunicação entre gerente e trabalhadores
- **Hierarquia Gerente/Trabalhador**
  - trabalhadores são agrupados em conjuntos separados entregues a sub-gerentes dedicados
  - os sub-gerentes comunicam com o gerente e restantes sub-gerentes

Esquemas Descentralizados

Cada processador mantém um lote separado de tarefas e os trabalhadores ociosos pedem problemas a outros processadores.

- *Técnica*
  - lote de tarefas é uma estrutura de dados distribuída
  - acesso ao lote realizado de modo assíncrono
- *Políticas de acesso*
  - trabalho provém de um conjunto, pré-definido, de processadores vizinhos
  - trabalho provém de um processador seleccionado aleatoriamente
  - na variante **híbrido centralizado/distribuídos**– pedidos enviados ao gerente principal são distribuídos circularmente pelos trabalhadores

Detecção de Terminação

- imediata nos esquemas centralizados
- mais difícil nos esquemas descentralizados
  - não existe um registo central
  - mensagens em trânsito podem conter tarefas

Teste de Arranjo

As decisões de arranjo procuraram compatibilizar a distribuição equitativa de carga com a redução dos custos de comunicação.

- *Perguntas sobre Arranjo*

  1. Face a uma solução SPMD, para um problema complexo, foi considerada a hipótese de um algoritmo baseado na criação ou destruição dinâmica de tarefas?
  2. Face a uma solução baseada na criação e destruição dinâmica de tarefas, foi considerada a hipótese de um algoritmo SPMD?
  3. Ao usar um esquema centralizado de balanceamento de carga, foi verificado se o trabalho do gerente não é fonte de congestionamento?
  4. Ao usar um esquema de balanceamento dinâmico, foi feita a avaliação relativa dos custos das diferentes estratégias?
  5. Ao usar métodos probabilísticos ou cíclicos, foi garantida a existência de um número suficientemente grande de tarefas?

- *Em caso afirmativo ...*

    1. A segunda hipótese pode conduzir a um algoritmo mais simples, mas de rendimento duvidoso.
    2. Neste caso, há um maior controlo sobre o escalonamento da computação e da comunicação, mas o algoritmo é tendencialmente mais complexo.
    3. Pode conseguir-se reduzir os custos de comunicação se se passarem ao gerente, em vez das tarefas os respectivos apontadores.
    4. A necessidade de evitar operações repetidas de balanceamento de carga sugere a utilização de esquemas probabilísticos ou de arranjo cíclicos.
    5. Tipicamente, são necessários pelo menos dez vezes mais tarefas que processadores.

Análise Quantitativa do Desenho

Os *modelos de rendimento* servem para comparar o rendimento dos diferentes algoritmos, *antes* que seja investido um esforço substancial na realização.

- avaliar a escalabilidade
- identificar ineficiências (congestionamento)
- revelar pontos a optimizar

Objectivos

- *Optimização*
  - tempo de execução e requisitos de memória
  - custos de realização e de manutenção
- *Compromissos*
  - simplicidade e rendimento
  - portabilidade

Definição de rendimento

O *rendimento* de um programa paralelo pode contemplar medidas tais como: o tempo de execução, o paralelismo, o desempenho, ou a entrada/saída de dados.

Natureza do Problema

- Previsão do Clima
  - tempo de execução
  - custo do equipamento
  - escalabilidade e fiabilidade
- Bases de Dados
  - mais rápida que a solução sequencial
  - minimização do tempo de realização
  - código adaptável aos dados e à tecnologia
- Processamento de Imagem
  - compressão de imagens/segundo
  - sistema de detecção em tempo real

Modelação de Rendimento

Lei de Amdahl's

Se a parte sequencial de um algoritmo demorar $1/s$ do tempo total de execução do programa, o aumento máximo de velocidade que pode ser obtido é s.

- Programa
  - tempo para j processadores: $T(j)$
  - fracção sequencial: β
  - fracção paralela: $(1 - \beta)$
- Tempo sequencial: $T(1)\beta$
- Tempo paralelo: $\frac{(1-\beta)T(1)}{N}$

$$
\begin{aligned}
S &= T(1)/T(N) \\
T(N) &= T(1)\beta + \frac{T(1)(1-\beta)}{N} \\
S &= \frac{N}{N\beta + (1-\beta)}
\end{aligned}
$$

Lei de Gustafson-Barsis

O paralelismo deve ser usado para aumentar o tamanho (paralelo) do problema. Se for usado apenas um processador, cabe-lhe a ele resolver tanto a parte sequencial como a parte paralela do problema.

- *Problema*

  $N = 10$ e $\beta = 0,67$

- Amdahl

  $S = \frac{1}{0,67+(0,33/10)} = 1,42$

- Gustafson-Barsis

  $S = 10 - (9)(0,6) = 3,97$

$$
\begin{aligned}
S &= T(1)/T(N) \\
T(1) &= \beta + N(1-\beta) \\
T(N) &= \beta + (1-\beta) \\
S &= N - (N-1)\beta
\end{aligned}
$$

Discussão

A maior parte dos problemas computacionais têm uma solução paralela não limitada pela existência de partes *sequenciais*.

- Limitações à escalabilidade
  - custos de comunicação
  - tempos de ócio
  - computação replicada
- Relevância da Lei Amdhal
  - paralelização de algoritmos sequenciais

Se um camião que despeja betão, num único ponto de uma estrada em construção, provoca um engarrafamento, pode-se argumentar que a estrada deveria estar a ser construída, simultaneamente, em vários pontos do traçado.

O baixo rendimento num projecto com 1000 trabalhadores em que há 999 parados enquanto o trabalhador que resta completa uma tarefa sequencial, decorre não da natureza intrinsecamente sequencial do problema mas de uma falha de gestão.

Observar para Extrapolar

A medida de rendimento resultante de um número pequeno de observações, apenas determina o rendimento numa região estreita de um vasto espaço multidimensional

- Problema
  - tamanho : $N = 10$
  - processadores: $P = 12$
  - aumento velocidade: $S = 10.8$
- E se alterarmos
  - tamanho : $N = 1000$
  - processadores: $P = 1000$
  - custos comunicação: 10 vezes superiores
  - *aumento de velocidade: ?*

Limites de observacionalidade

- Modelos de Rendimento
- Algoritmo sequencial óptimo $N + N^2$
  1. $T = N + N^2/P$.
  2. $\frac{T=(N+N^2)}{P+100.}$
  3. $\frac{T=(N+N^2)}{P+0.62P^2.}$

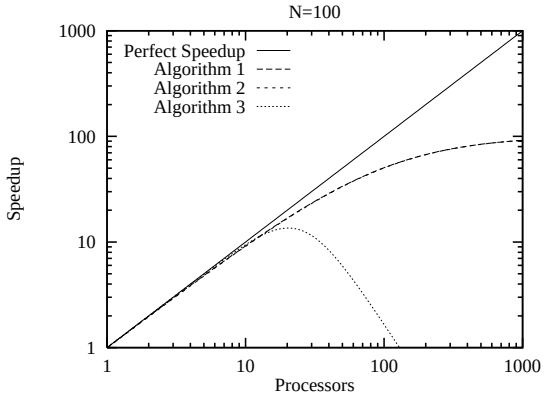

Figura 26: Eficiência como uma função de P para $N = 100$

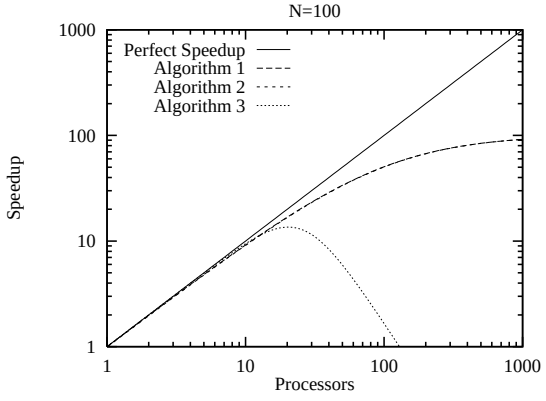

Figura 27: Eficiência como uma função de P para $N = 1000$

Complexidade de um Algoritmos

A *complexidade* de um algoritmo é o `custo` associado a resolução de um problema quando se usa esse algoritmo

- *Definições*
  - *Complexidade Temporal*: tempo necessário para executar o algoritmo
  - *Complexidade Espacial*: requisitos de memória para executar o algoritmo
  - *Complexidade Assimptótica*: o que acontece quando tamanho do problema se aproxima do infinito

Análise Assimptótica

Para avaliar a complexidade assimptótica de um algoritmo usam-se formalismos que representam os limites de rendimento de um algoritmo.

1. **Limite Superior**: Se $\exists c, N$ para $\forall n \geq N$ tal que:

$$f(n) \leq c.g(n)$$

2. **Limite Inferior**: Se $\exists c, N$ para $\forall n \geq N$ tal que:

$$f(n) \geq c.g(n)$$

3. **Limite Apertado**: Se $\exists c1, c2, N$ para $\forall n \geq N$ tal que:

$$c1.g(n) \leq f(n) \leq c2.g(n)$$

A análise assimptótica deve identificar o modelo de máquina subjacente, os coeficientes a aplicar e os valores de N e P para o regime considerado.

1. *Porque depende de valores elevados de N e P*

   - ignora os termos de baixa ordem
   - ignora situações de interesse prático
   - em $10N + N \log N$ considerar o termo $10N$ para $N < 1024$

2. *Nada é dito sobre custos absolutos*

   a) $1000N \log N$

   b) $10N^2$

   - **a** é melhor que **b**
   - **b** mais rápido para $N < 996$

3. *Modelos ideais*

   - diferença entre realidade e modelo
   - PRAM: custos de comunicação nulos

Desenvolvimento de Modelos

Um bom modelo de rendimento pode explicar as observações realizadas e fazer previsões futuras, pondo de parte detalhes pouco importantes.

- *Convencionais*
  - não satisfazem os requisitos: lei de Amdahl, análise empírica e análise assimptótica tem excesso de pormenores: simulações com pouco interesse prático
- *Especializados*
  - multi-computador
  - nível intermédio de detalhe
  - $T = f(N, P, U, ...)$.

Num programa paralelo o *tempo de execução* é o tempo que decorre, desde que o primeiro processador inicia a execução do problema, até que o último processador termine.

$$T = T^j_{comp} + T^j_{com} + T^j_{ocio}.$$

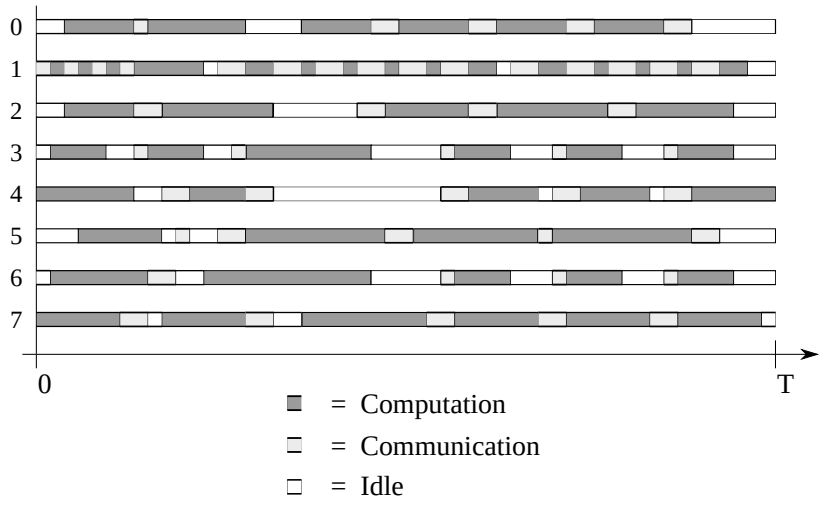

Figura 28: Actividade numa aplicação paralela

- *Tempo total de execução*

  1. $T = \frac{1}{P}(T_{comp} + T_{com} + T_{ocio})$.
  2. $T = \frac{1}{P}(\sum_{i=0}^{P-1} T_{comp} + \sum_{i=0}^{P-1} T_{com} + \sum_{i=0}^{P-1} T_{ocio})$.

As expressões do modelo de rendimento deverão ser o mais simples possíveis, mas de precisão aceitável.

- *Multi-computador*
  - hierarquias de memória e topologias de interconexão
  - ignorar detalhes físicos de baixo nível
- *Análise de importância*
  - identificar efeitos desprezáveis
  - tempo de iniciação pouco significativo
- *Estudos empíricos*
  - calibrar modelos simples
  - não desenvolver de raíz

Tempo de Computação

Pode-se determinar o tempo de computação T_{comp}, de um algoritmo paralelo, cronometrando o programa sequencial que realiza a mesma computação.

- *Dependências*
  - tamanho do problema: N ou $N_1, N_2 ... N_m$
    - rendimento da *cache*
  - replicação da computação: número de tarefas e/ou número de processadores
    - encademento dos processadores
  - ambiente heterogéneo: tipo de processador

Tempo de Comunicação

T_{com} é o tempo de comunicação que as tarefas gastam para enviar e receber mensagens.

- *Comunicação inter-processadores*
  - diferentes processadores
  - velocidade depende do tipo de ligação
- *Comunicação intra-processadores*
  - mesmo processador
  - cópias memória-memória
  - comutação de contextos
- *Custos*
  - comparáveis em termos práticos

Custos de Transmissão

$$T_{msg} = t_s + t_w L.$$

- *Arranque*
  - t_s: tempo de necessário para iniciar a comunicação
- *Transferência*
  - t_w: tempo de transferência de 4 octetos
  - limitado pela largura de banda física do canal

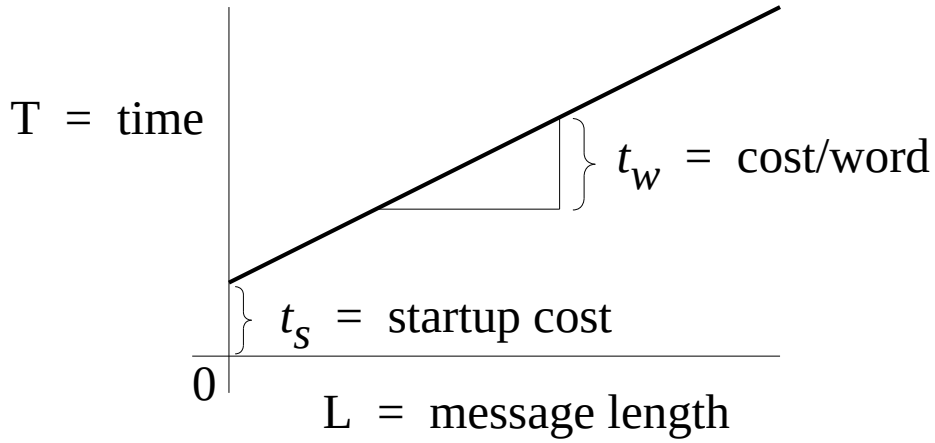

Figura 29: Modelo simplificado de comunicação

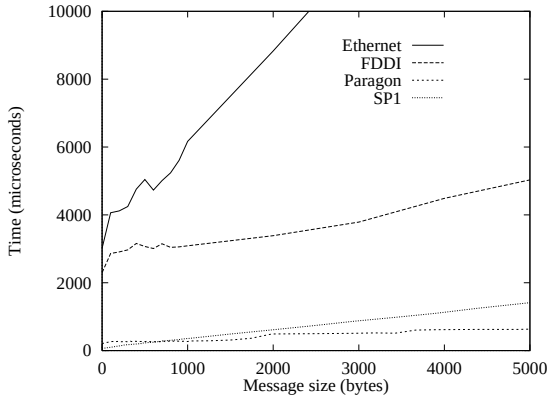

Figura 30: Tempo de ida e volta de uma mensagem

- *Tempo de ida e volta de uma mensagem*
  - valores duplos dos obtidos pela fórmula
  - pequenas mensagens: irregularidades em Ethernet e FDDI
  - Paragon: especificidade dos protocolos de comunicação
  - mensagens grandes: razoável precisão obtida pela fórmula
- *Avaliação*
  - mensagens muito grandes: apenas t_w é importante
  - mensagens pequenas: o termo t_s é normalmente dominante

Tempo de ócio

Um processador pode estar ocioso porque não tem nenhuma computação a realizar ou porque faltam dados.

- depende da ordem de execução das operações
- falta de computação
  - técnicas de balanceamento de carga
- falta de dados
  - *Sobreposição de computação e comunicação*
    1. tarefas múltiplas em cada processador
       * eficiência depende dos custos de escalonamento
    2. uma só tarefa
       * pedidos de dados remotos intermeados com computação

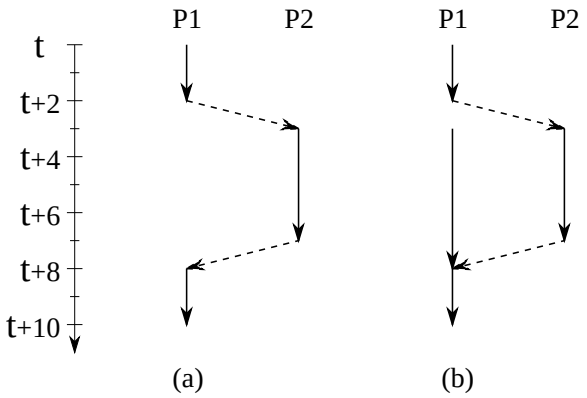

Figura 31: Sobreposição de computação e comunicação

Eficiência e Velocidade Máxima

A eficiência é uma medida que caracteriza a forma como um algoritmo usa os recursos de um computador paralelo, de uma forma independente do tamanho do problema

- *Tempo de Execução*
  - varia com o tamanho do problema
  - normalização para permitir comparar o rendimento de um algoritmo para diferentes tamanhos do problema
- *Eficiência*
  - fracção de tempo de trabalho útil
  - T_1 e T_P o tempo de execução em 1 ou P processadores

$$E_{relativa} = T_1/PT_P, \quad (1)$$

$$S_{relativa} = PE, \quad (2)$$

Quando se comparam dois algoritmos pode ser útil ter um medida independente do algoritmo diferente do tempo de execução.

- *Medidas*
- Relativas
  - definidas em relação ao algoritmo paralelo executado num único processador
- Absolutas
  - definidas em relação ao melhor algoritmo sequenciais conhecidos para um uni-processador
- *Exemplos*

  a) 1 P – 10000 segundos
  1000 P – 20 segundos

  b) 1 P – 1000 segundos
  1000 P – 5 segundos

O segundo algoritmo é o melhor para P entre 1 e 1000, mesmo se a velocidade relativa máxima é apenas 200 quando é 500 para o primeiro algoritmo.