Programação Paralela – OPRP001

Modelo de Programação Paralela

Desenvolvido por Prof. Guilherme Koslovski e Prof. Maurício Pillon

Revisando

- Memória compartilhada
  - Um único espaço de endereçamento é usado de forma implícita para comunicação entre processadores
  - Operações: load e store
- Memória não compartilhada
  - Múltiplos espaços de endereçamento privados, um para cada processador
  - Comunicação explícita através de troca de mensagens
  - Operações: send e receive

Revisando

- Memória distribuída
  - Refere-se a localização física da memória
  - Memória implementada em vários módulos e cada módulo está próximo de um processador
- Memória centralizada
  - Encontra-se a mesma distância de todos os processadores
  - Independentemente de ter sido implementada em vários módulos

Agenda

- **TOP 500**
- Análise e Otimização de Algoritmos
- Modelo de Programação Paralela
- Exemplo real

TOP 500

- http://www.top500.org
- Atualizado semestralmente desde 1993, em Novembro e Junho de cada ano
- Benchmark Linpack – álgebra linear
- Indicador de tendências arquiteturais
- **FLOPS: FLoating-point Operations Per Second**
  - **Atualmente: TFLOPS**

TOP 500

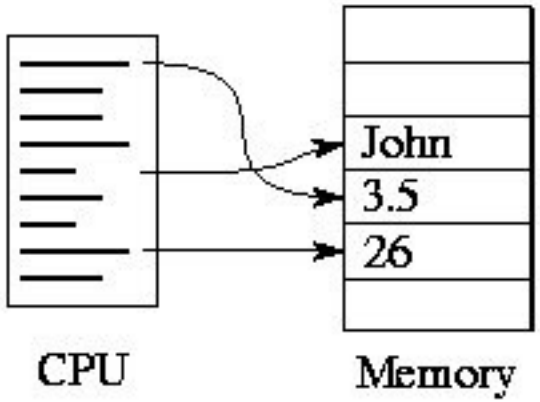
- Comparação de arquiteturas (selecione 2 + o computador pessoal)**:**
  - Descrição
  - Posição TOP500
  - Classificação Flynn
  - Classificação segundo o compartilhamento de memória
  - Número de núcleos
  - Descrição dos processadores
  - Fabricante
  - Memória total
  - Rede de comunicação
  - Consumo energético

Agenda

- TOP 500
- **Análise e Otimização de Algoritmos**
- Modelo de Programação Paralela
- Exemplo real

Modelo de Programação Sequencial vs Paralela

- A máquina de Von Neumann assume que o processador está habilitado a executar uma sequência de instruções
- Uma instrução pode especificar: uma simples adição, várias operações aritméticas, leitura/escrita na memória ou endereços de instruções
- E no paralelo?



Análise e Otimização de Algoritmos

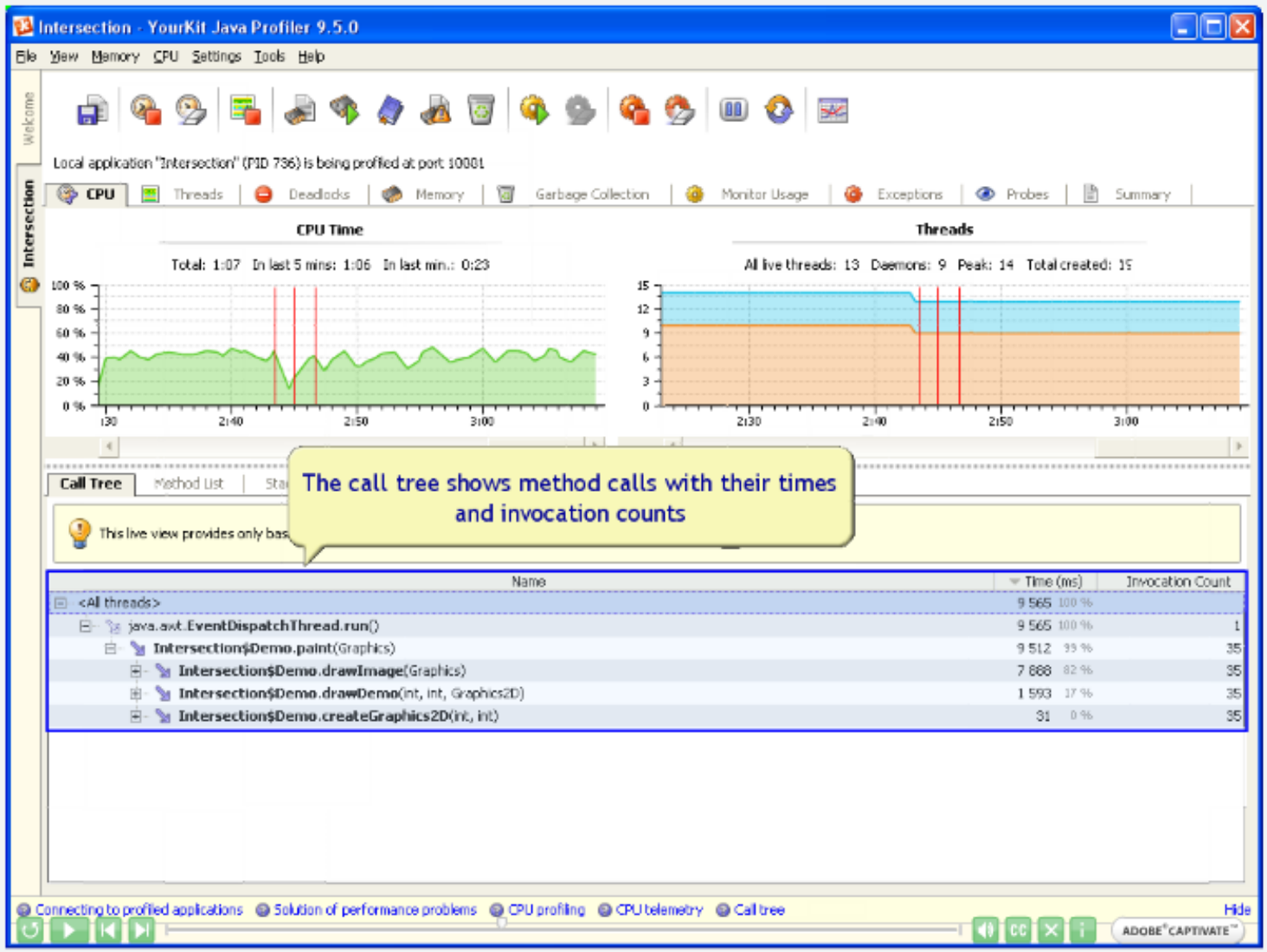
- Posso paralelizar meu programa sequencial?
  - Limitações da implementação sequencial
  - O algoritmo pode ser otimizado?
  - Por onde começar: ferramentas de profiling
    - Java: yourkit
    - C: gprof
    - Analisar o comportamento das instruções
  - Reduzir o número de instruções sem alterar a consistência do resultado final
  - Detalhes da arquitetura facilitam a exploração de recursos: uso de caches, balanceamento entre tarefas e comunicações

Profiling – GNU gprof

- man gprof
- gcc -Wall **-pg** prog.c –o prog
- ./prog
- gprof –a prog gmon.out > saida.txt

```
Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls  ms/call  ms/call  name
 33.34      0.02     0.02     7208     0.00     0.00  open
 16.67      0.03     0.01      244     0.04     0.12  offtime
 16.67      0.04     0.01        8     1.25     1.25  memccpy
 16.67      0.05     0.01        7     1.43     1.43  write
 16.67      0.06     0.01                             mcount
  0.00      0.06     0.00      236     0.00     0.00  tzset
  0.00      0.06     0.00      192     0.00     0.00  tolower
  0.00      0.06     0.00       47     0.00     0.00  strlen
  0.00      0.06     0.00       45     0.00     0.00  strchr
  0.00      0.06     0.00        1     0.00    50.00  main
  0.00      0.06     0.00        1     0.00     0.00  memcpy
  0.00      0.06     0.00        1     0.00    10.11  print
  0.00      0.06     0.00        1     0.00     0.00  profil
  0.00      0.06     0.00        1     0.00    50.00  report
...
```

Profiling – yourkit



Análise e Otimização de Algoritmos

- Como otimizar?
  - Redução de chamadas a função
    - Inlining: substituição de uma chamada de subrotina ou função pela sua definição.
    - Exemplo: pow()
    - gcc -> **https://gcc.gnu.org/onlinedocs/gcc/Optimize-Options.html**
  - Redução de condicionais:
    - A quantidade média de ocorrência de instruções de desvios é da ordem de uma para cada cinco instruções [Gonçalves et al. 2006]
  - Reestruturação de laços:
    - Evitar condicionais, eliminar chamadas desnecessárias a funções e extrair cálculos complexos [Severance and Dowd, 1998]

Análise e Otimização de Algoritmos

- Reestruturação de laços

```
1  // Laço normal
2  for( x = 0; x < 100; x = x + 1 )
3  {
4      vetor[x] = 10;
5      if( x == 55 )
6      {
7          vetor[x] = 11;
8      }
9  }
```

```
// Laço com condicional extraído
for( x = 0; x < 100; x = x + 1 )
{
    vetor[x] = 10;
}
vetor[55] = 11;
```

Listagem 3.2: Extração de Condicionais.

Otimização com gcc

-O1 → Optimize. Optimizing compilation takes somewhat more time, and a lot more memory for a large function.

Without `-O', the compiler's goal is to reduce the cost of compilation and to make debugging produce the expected results. Statements are independent: if you stop the program with a breakpoint between statements, you can then assign a new value to any variable or change the program counter to any other statement in the function and get exactly the results you would expect from the source code.

Without `-O', the compiler only allocates variables declared `register` in registers. The resulting compiled code is a little worse than produced by PCC without `-O'.

With `-O', the compiler tries to reduce code size and execution time.

Otimização com gcc

-O2 → Optimize even more. GCC performs nearly all supported optimizations that do not involve a space-speed tradeoff. The compiler does not perform loop unrolling or all function inlining when you specify `-O2'.

As compared to `-O', this option increases both compilation time and the performance of the generated code. `-O2' turns on all optional optimizations except for loop unrolling, function inlining, and strict aliasing optimizations.

-O3 → Optimize yet more. `-O3' turns on all optimizations specified by `-O2' and also turns on the `inline-functions' option (all).

-O0 → Do not optimize.

Outras opções:
https://gcc.gnu.org/onlinedocs/gcc/Optimize-Options.html

Agenda

- TOP 500
- Tipo de paralelismo
- Análise e Otimização de Algoritmos
- **Modelo de Programação Paralela**
- Exemplo real

Tipo de paralelismo

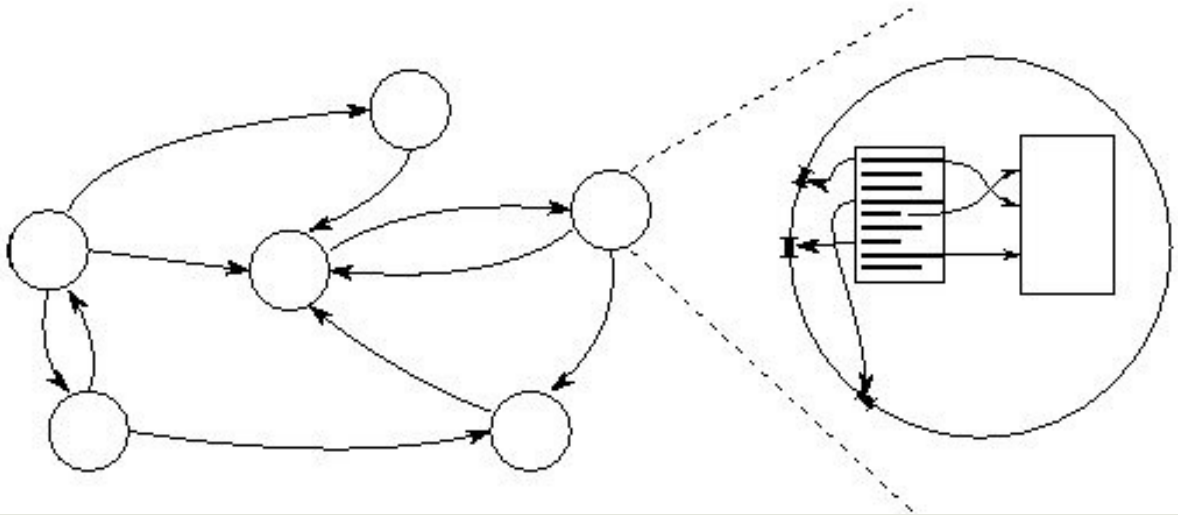
- Paralelismo de dados
  - Processo de paralelização atua sobre sequências de funções ou operações similares, não necessariamente idênticas, sendo executadas em elementos de grandes estruturas de dados
  - Dados são particionados entre tarefas
  - Técnicas de decomposição de domínio devem ser aplicadas sobre os dados que se esta operando
- Paralelismo funcional (de controle ou de tarefas)
  - Cálculos totalmente diferentes podem ser executados concorrentemente nos mesmos dados ou em conjuntos de dados distintos

Modelo de Programação Paralela

- A forma de simplificar o projeto de programas é aumentando a abstração aplicando regras de loops, estruturas de dados e procedimentos
- Esta abstração possibilita a construção de programas explorando as facilidades da modularidade
- Desta forma, objetos são manipulados sem que sua estrutura interna seja conhecida

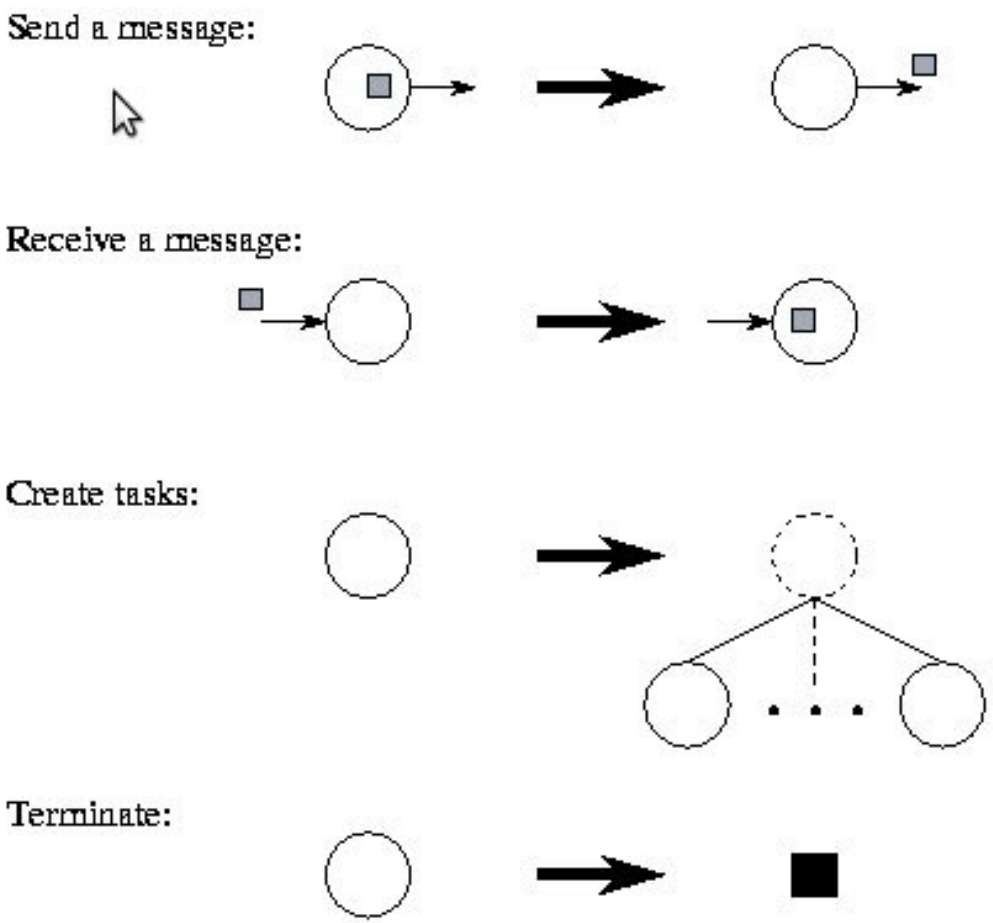
Um modelo simples de programação paralela

- A computação consiste em um conjunto de tarefas (círculos) conectados por um canal (arestas)
- A tarefa encapsula um programa em memória local e define um conjunto de portas que define suas interfaces com o ambiente
- Um canal é uma fila de mensagens no remetente que deve ser entregue a um destinatário



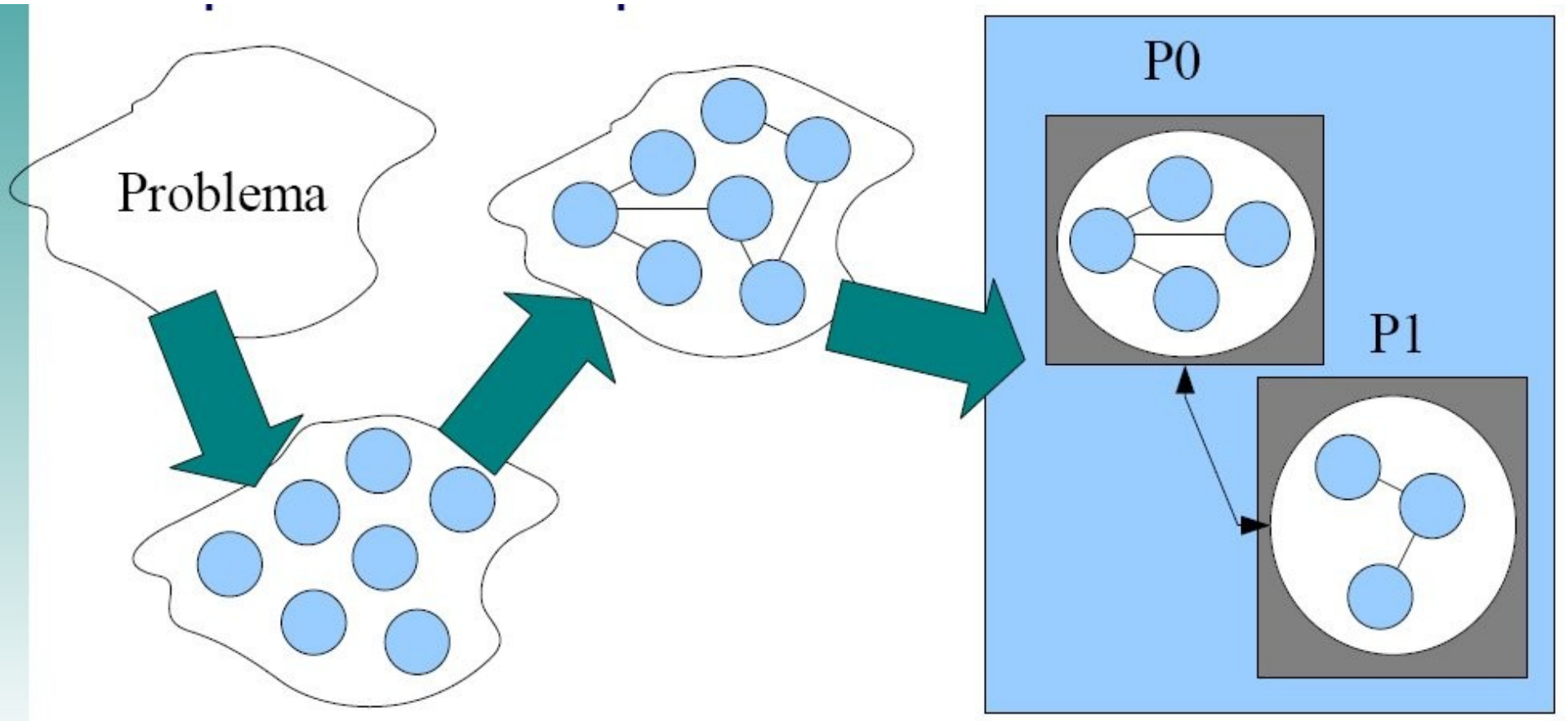
Um modelo simples de programação paralela

- Quatro ações básicas de tarefas:
  - Enviar mensagem (send)
  - Receber mensagem (receive)
  - Criar tarefa (create)
  - Terminar tarefa (terminate)



Um modelo simples de programação paralela

- Esquema de Paralelização



Impacto na programação

- **Paralelização explícita:** o programador deve dizer, explicitamente, quais as tarefas que devem ser executadas em paralelo, e a forma como essas tarefas devem cooperar entre si.
  - occam [Pountain et al., 1988], MPI [MPI Forum, 1994]
- **Paralelização implícita:** o programador desenvolve seu algoritmo em alguma linguagem que represente uma descrição do que deve ser executado sem inclusão de sequencialização.
  - Linguagem não-imperativa (SISAL) [Cann, 1993]
- **Paralelização automática:** o programador desenvolve um programa usando uma linguagem tradicional e o compilar é o responsável por extrair o paralelismo. Exemplos em [Hilhorst et al., 1987], [Wolf et al., 1991], [Chen et al., 1994], OpenMP

Impacto na programação

- Paralelização automática ainda é alvo de pesquisas
  - Atende apenas uma parcela das aplicações / arquiteturas
  - Nem sempre é eficiente
- Em geral
  - Concorrência (paralelismo e distribuição) geralmente está explicita (em maior ou menor grau)
  - **Modelos de programação adaptados à arquitetura produzem programas mais eficientes**
- Depuração?
- Traços de execução -> http://paje.sourceforge.net/index.html

Programação em multiprocessadores

- Múltiplas threads compartilhando dados (rapidez)
- **Aspecto crítico:** sincronização quando diferentes tarefas acessam os mesmos dados
- **Ferramentas de programação:**
  - Linguagens concorrentes (Ada, SR, ...)
  - Linguagens sequenciais + extensões/bibliotecas (OpenMP, threads, UPC, HPF ...)

Programação em multicomputadores

- Troca de mensagens entre tarefas (fonte de sobrecarga)

- **Aspectos críticos:**
  - desempenho da rede e distribuição dos dados
- **Ferramentas para programação:**
  - Linguagens sequencias + extensões/bibliotecas MPI: C, C++, Fortran, Java
  - PVM
  - JavaRMI, ProActive
  - Memória compartilhada distribuída: Linda, JavaSpaces

Agenda

- TOP 500
- Tipo de paralelismo
- Análise e Otimização de Algoritmos
- Modelo de Programação Paralela
- **Exemplo**

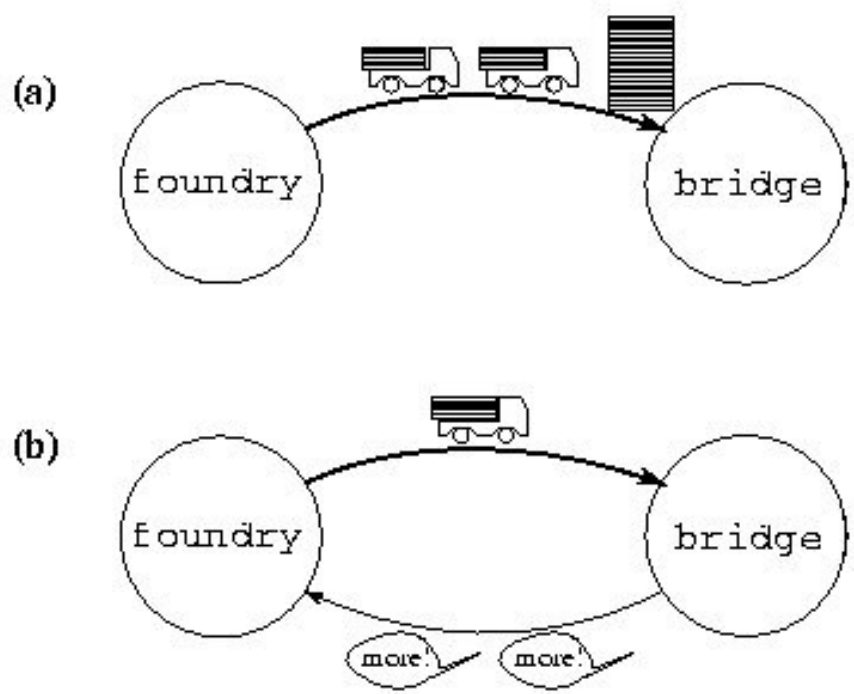
O Problema da Construção da Ponte

- Uma ponte é montada com vigas construídas por uma fundição
- Vigas são transportadas da fundição para o local de construção da ponte por caminhões
- Note que a montagem da ponte e a fabricação das vigas podem (e são) procedimentos executados em paralelo sem coordenação explícita
- **Modelo:**
  - **Fundição e a Ponte são tarefas**
  - **Fluxo de caminhões são streams de mensagens**

Proposta de soluções

a. Um único canal com um único sentido: fundição -> ponte. Assim que fabricadas as vigas são transportadas ao local da montagem da ponte

b. Dois canais: fundição -> ponte; outro para mensagens de controle

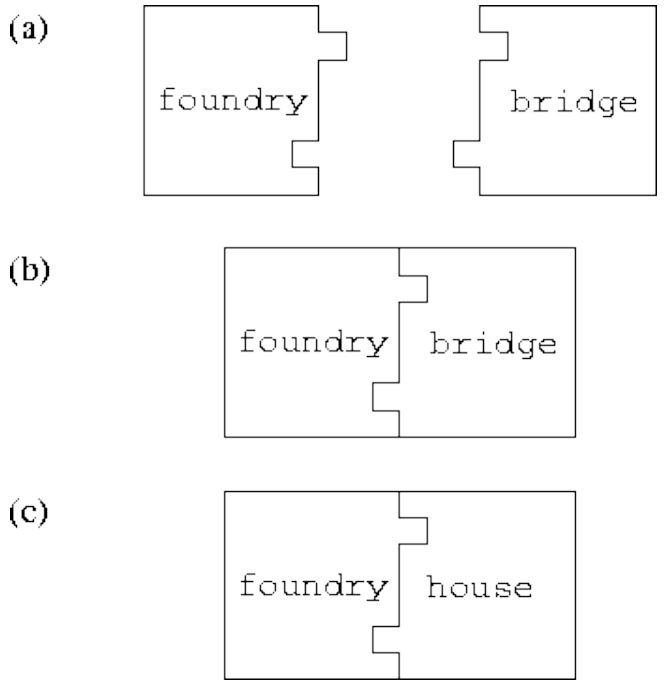


Propriedades de tarefas/canais

- **Desempenho**
  - Tarefas são mapeadas como simples tarefas sequenciais.
  - Tarefas simples que compartilham dados e que estão mapeados em processadores distintos necessitam comunicação;
  - Sempre que possível, alocá-las em um mesmo processador
- **Mapeamento da Independência**
  - Em contrapartida, criar muitas tarefas em um mesmo processador pode afetar o modelo de escalabilidade do programa

Propriedades de tarefas/canais

- **Modularidade**
  - Componentes do programa podem ser projetados e executados como módulos independentes que são combinados, com ou sem comunicação via canais, obtendo um programa.
  - O sucesso da modularidade permite reduzir a complexidade do programa e a facilidade de reuso



Propriedades de tarefas/canais

- **Determinismo**
  - Um algoritmo é determinístico se a cada execução com um conjunto de dados de entrada obtemos a mesma saída.
  - Programas não-determinísticos são mais complexos de depurar
- Outros modelos existem. De modo geral eles diferem entre si, quanto a **flexibilidade**, mecanismos de interação entre tarefas, **granularidade das tarefas**, suporte a **localidade**, **escalabilidade** e **modularidade**

Considerações finais

- Definições de programação distribuída, paralela, concorrente e sequencial
- Tipo de paralelismo: de dados e funcional
- Otimização de algoritmos
- Modelo e impacto de programação paralela