

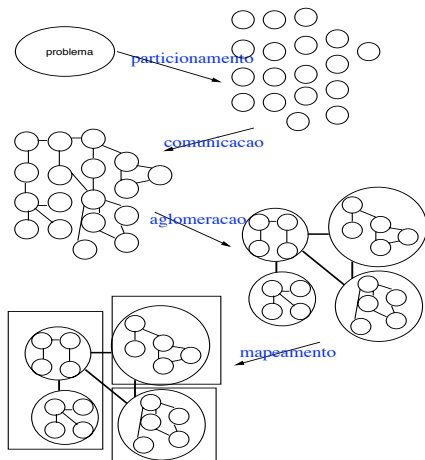
Programação Concorrente e Paralela

projeto de programas paralelos

Noemi Rodriguez

2016





- desenhada para memória distribuída mas muitas idéias em comum

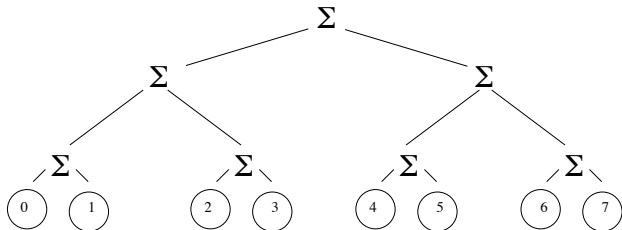


- idéia é expor oportunidades de paralelismo
 - foco em determinar uma grande quantidade de pequenas tarefas
- decomposição funcional e decomposição por domínio
- taxonomia de Flynn (transposta para programas)
 - spsd
 - **spsd**
 - mpsd
 - mpmd



Oportunidades de Paralelização

- solução paralela parte de versão sequencial?
- versão sequencial pode ou não ser a que oferece maiores oportunidades
 - exemplo soma de N números



- exemplo paralelização de computação com dependências



ordenação

- que algoritmos se prestam a paralelização?



Ordenação: bubble sort

```
for (list_length = n; list_length >= 2; list_length--)  
  for (i=0; i<list_length; i++)  
    if (a[i] > a[i+1]) {  
      tmp = a[i];  
      a[i] = a[i+1];  
      a[i+1] = tmp;  
    }
```

pouca oportunidade de paralelização!



Ordenação: *odd-even transposition*

```
for (phase = 0; phase < n; phase++)  
    if (phase%2) /* impar */  
        for (i = 1; i < n-1; i += 2)  
            if (a[i] > a[i+1]) Swap(&a[i], &a[i+1]);  
    else /* par */  
        for (i = 1; i < n; i += 2)  
            if (a[i-1] > a[i]) Swap (&a[i-1], &a[i]);
```

Parecido porém com mais oportunidades



```
for (phase = 0; phase < n; phase++)
  if (phase%2) /* impar */
    for (i = 1; i < n-1; i += 2)
      if (a[i] > a[i+1]) Swap(&a[i], &a[i+1]);
  else /* par */
    for (i = 1; i < n; i += 2)
      if (a[i-1] > a[i]) Swap (&a[i-1], &a[i]);
```

não podemos simplesmente colocar o loop externo em paralelo!




```
for (phase = 0; phase < n; phase++) {
    if (phase % 2 == 0)
#       pragma omp parallel for num_threads(thread_count) \
        default(none) shared(a, n) private(i, tmp)
        for (i = 1; i < n; i += 2) {
            if (a[i-1] > a[i]) {
                tmp = a[i-1]; a[i-1] = a[i]; a[i] = tmp;
            }
        }
    else
#       pragma omp parallel for num_threads(thread_count) \
        default(none) shared(a, n) private(i, tmp)
        for (i = 1; i < n-1; i += 2) {
            if (a[i] > a[i+1]) {
                tmp = a[i+1]; a[i+1] = a[i]; a[i] = tmp;
            }
        }
}
```

- muita criação e destruição de threads !



```
# pragma omp parallel num_threads(thread_count) \  
    default(none) shared(a, n) private(i, tmp, phase)  
for (phase = 0; phase < n; phase++) {  
    if (phase % 2 == 0)  
#        pragma omp for  
        for (i = 1; i < n; i += 2) {  
            if (a[i-1] > a[i]) {  
                tmp = a[i-1]; a[i-1] = a[i]; a[i] = tmp;  
            }  
        }  
    else  
#        pragma omp for  
        for (i = 1; i < n-1; i += 2) {  
            if (a[i] > a[i+1]) {  
                tmp = a[i+1]; a[i+1] = a[i]; a[i] = tmp;  
            }  
        }  
}
```

- exemplo retirado de: Peter Pacheco. *An Introduction to Parallel Programming*.



Verificação do Particionamento

- partição gera número de tarefas com ordem de magnitude superior à do número de processadores disponíveis?
 - flexibilidade para estágios posteriores
- tarefas são de tamanho comparável?
- número de tarefas cresce junto com tamanho da entrada do problema?
- há mais de uma alternativa de particionamento?



- como as tarefas definidas na fase 1 vão se comunicar?
 - definição de canais a serem usados
 - definição de mensagens
 - *para memória compartilhada*: custos de sincronização
- alguns critérios de classificação são úteis:
 - local x global, estruturada x não estruturada, estática x dinâmica
 - assíncrona x síncrona



- comunicações *globais* podem criar gargalos de comunicação
 - exemplos: bolsa de tarefas única, ...
- comunicações *estruturadas* podem facilitar a tarefa de aglomerar e mapear
- sincronismo: simplicidade x menos oportunidades de concorrência



- todas as tarefas têm aproximadamente a mesma carga de comunicação?
 - distribuição ou replicação de estruturas de dados
- várias operações de comunicação podem ocorrer concorrentemente?
- e a computação? pode ocorrer concorrentemente?

em memória compartilhada

- custos se referem principalmente à sincronização e a acessos a caches
 - problema do falso compartilhamento!



- possibilidade de combinar tarefas
 - j pode levar em conta arquiteturas especficas
 - deixar o nmero de tarefas igual ao de processadores?
 - diminuico dos custos de comunicao e de troca de contexto
 - replicao de computaco



- objetivos podem ser conflitantes
 - granularidade \times custos de comunicao e criao de processos
 - $n.$ tarefas $>$ processadores \Rightarrow possibilidade de sobrepor comunicao e computao
 - maior nmero de tarefas facilita balanceamento de carga



- diminumos a comunicaco?
- houve ganhos com replicaco de computaco?
- se replicamos dados, isso no compromete a escalabilidade?
 - ... o nmero de tarefas ainda cresce com o tamanho do problema?
- ainda h concorrncia suficiente?



- 1 *para sistemas de memória compartilhada*: não visível diretamente para o programador
 - 2 colocar processos que podem executar concorrentemente em máquinas diferentes
 - 3 colocar processos que se comunicam com frequência na mesma máquina
 - ... claramente pode haver conflitos
- problema geral de mapeamento é NP-completo
 - na prática muitas vezes situações mais simples...



- analisamos diferentes estratégias de criação de processos?
 - SPMD x criação dinâmica de processos



Exemplo: temperatura numa vara

- cálculo da evolução da temperatura ao longo do tempo

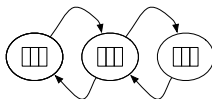
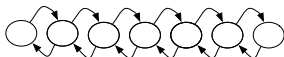
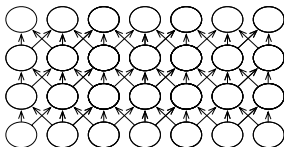


Como paralelizar

- 1 particionamento
 - cada ponto da matriz tempo \times posição é uma tarefa
- 2 comunicação
 - cálculo em cada ponto depende do resultado de outros três
- 3 aglomeração e mapeamento
 - não há como trabalhar em paralelo em diferentes pontos de uma coluna
 - aglomeração de algumas posições



Aglomeración

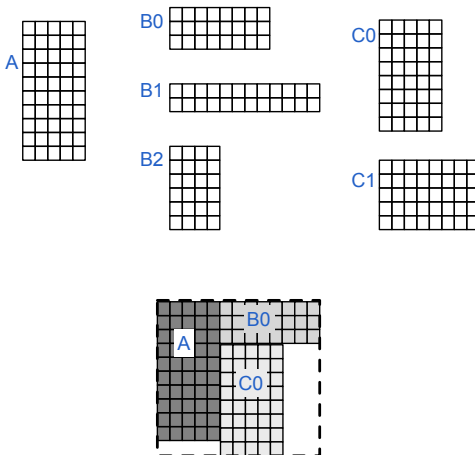


Exemplo: projeto de componentes

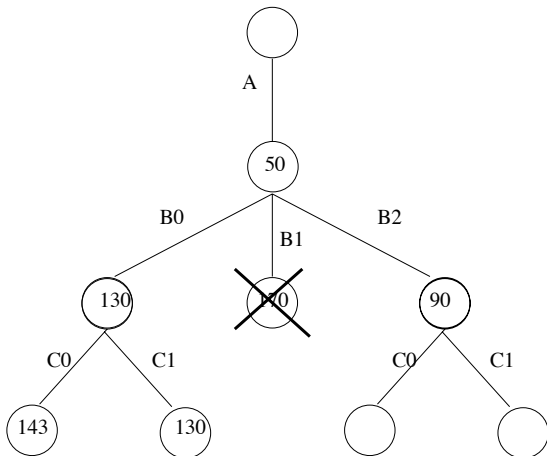
- n células têm que ser dispostas em uma placa
- existem restrições em relação às posições relativas
- exploração de todas as soluções tem custo proibitivamente alto
- uso de estratégia *branch and bound*



Componentes — alternativas



Branch and Bound



- particionamento: criação de processos para buscas paralelas
 - em primeira abordagem uma por nó...



- comunicação: divulgação de custos totais (mínimos parciais)
- balanço entre atualização do mínimo encontrado e economia de comunicação
 - gerente central pode manter soluções e mínimo
 - escalável?



- aglomeração: um único processo pode explorar sub-árvore
- mapeamento: alternativas
 - único processo raiz distribui tarefas para trabalhadores
 - diversos processos replicam trabalho inicial e assumem cada um uma sub-árvore a partir de algum ponto
 - padrão de replicação para evitar comunicação



- I. Foster. Developing and Building Parallel Programs. (disponível em www.mcs.anl.gov/~itf/dbpp/text/book.html)
- Guy Steele. The Future Is Parallel: What's a Programmer to Do? Breaking Sequential Habits of Thought

