

Modelos de Execução

Bolsas de Tarefas

November 5, 2009

- todos os processos executam o mesmo código
- quantidade de trabalho determinada estaticamente:
 - tarefas pre-definidas
 - quantidade de trabalho em cada tarefa é fixa ou bem conhecida
 - poder computacional homogêneo

- ambientes heterogêneos
- ambientes com carga externa à aplicação
- tarefas em quantidades diferentes de trabalho
 - caso da quadratura adaptativa
 - ou outra visão: tarefas criadas dinamicamente
- uma solução é manter número de tarefas muito maior que número de processadores e distribuí-las ao longo da execução
 - **bolsa de tarefas**

- ou **mestre/trabalhador**
- ou ainda distribuição **sob demanda**
- trabalhadores repetidamente consomem tarefas de uma *bolsa de tarefas* e enviam resultados para mestre
- apropriado para programas embaraçosamente paralelos
 - dependências não permitem alocação livre
- forma rudimentar mas bastante eficiente de balanceamento de carga
 - atende flutuações em carga interna e externa

- MPI “feito” para SPMD tradicional
- quando começamos a pensar em outras estruturas...
- outras notações podem ser mais convenientes

- processo mestre lê vetor, distribui para todos, lê matriz e entra em loop:

```
do
    recebe pedido
    envia linha
while (tem linha);
```

- demais processos ficam em loop

```
do
    envia pedido
    recebe linha
    if (linha) calcula elemento
while (linha);
```

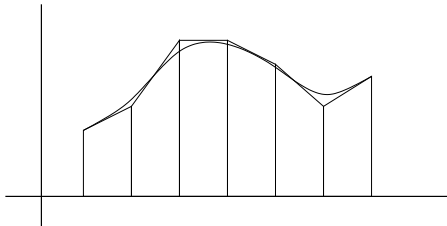
Utilidade?

- nesse problema a quantidade de trabalho é bem conhecida
 - divisão inicial, estática, evita custos com comunicação
- bolsa de tarefas pode ser útil apenas se:
 - processadores com capacidades diferentes
 - processadores sobrecarregados com outras tarefas



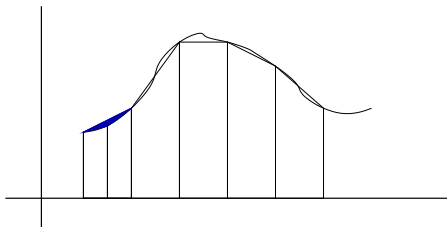
Exemplo: Quadratura

- cálculo da área abaixo de uma curva: aproximação por trapézios



Quadratura Adaptativa

- adaptação dinâmica à curva
 - diferença entre área de 1 trapézio e 2 subtrapézios indica se trabalho em certo intervalo deve continuar ou não



- trabalhador pode fazer chamadas recursivas enquanto intervalos não aceitáveis...
 - nesse caso a quantidade de trabalho associada a cada intervalo inicial é diferente
 - pode valer a pena quebrar em muitos intervalos e usar a bolsa de tarefas para distribuí-los dinamicamente
- ou trabalhador pode gerar novas tarefas na bolsa de tarefas

Implementações possíveis

- mais comum: mestre mantém a bolsa de tarefas
 - mestre pode ou não agir como trabalhador também
 - perda de um processador trabalhador X demoras em respostas
- estruturas de comunicação podem facilitar outras implementações
 - canais ou mailboxes
 - espaço de tuplas

```

module Manager
  op getTask(result double left, right);
  op putResult(double area);
body Manager
  process manager {
    double a, b;          # interval to integrate
    int numIntervals;    # number of intervals to use
    double width = (b-a)/numIntervals;
    double x = a, totalArea = 0.0;
    int tasksDone = 0;
    while (tasksDone < numIntervals) {
      in getTask(left, right) st x < b ->
        left = x; x += width; right = x;
      [] putResult(area) ->
        totalArea += area;
        tasksDone++;
      ni
    }
    print the result totalArea;
  }
end Manager

double f() { ... }      # function to integrate
double quad(...) { ... } # adaptive quad function

process worker[w = 1 to numWorkers] {
  double left, right, area = 0.0;
  double fleft, fright, lrarea;
  while (true) {
    call getTask(left, right);
    fleft = f(left); fright = f(right);
    lrarea = (fleft + fright) * (right - left) / 2;
    # calculate area recursively as shown in Section 1.5
    area = quad(left, right, fleft, fright, lrarea);
    send putResult(area);
  }
}

```

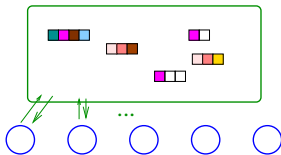
Figure 9.2 Adaptive quadrature using manager/workers paradigm.

Criação Dinâmica de Tarefas

- exemplo artigo Andrews: outra implementação de quadratura adaptativa
- trabalhadores depositam novas tarefas no canal
- ... como ficaria esse programa no MPI?

outra notação: Espaços de Tuplas

- proposta original: Linda
 - para FORTRAN e C
 - primitivas in, out, read, eval
- implementação mais conhecida atualmente: JavaSpace
 - acompanha ambiente Jini
 - primitivas take, write, read



```
int faztarefa (t_descricao descricao) {  
    ...  
}  
void worker ()  
    int res; t_descricao desc;  
    while (1) {  
        in ("tarefa", ?desc);  
        res = faztarefa(desc);  
        out ("resultado", res);  
    }  
}
```

- quem está fazendo o trabalho de distribuição?!?

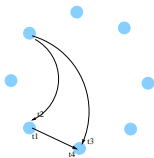
- exemplo livro MPDP (Greg Andrews), 7.16
- utilização do espaço de tuplas para:
 - tarefas
 - números primos
 - indicação de parada

Detecção de Terminação

- nos dois exemplos vistos trabalhadores ficam “trancados”
- para terminar mais elegantemente:
 - implementação que permita verificar que bolsa está vazia
 - funciona quando tarefas são todas criadas no início do programa
 - mestre coloca tarefa especial indicando final da aplicação
 - detecção de terminação
 - exemplo da quadratura adaptativa

Algoritmo de Detecção de Terminação

- problema: recolher estado global da aplicação
 - relacionado ao problema geral de *snapshot*



- exemplo de algoritmo “lateral”
- ref: Michel Raynal. *Distributed Algorithms and Protocols*. Wiley, 1992.

- balanceamento convencional requer algum tipo de escolha
 - possivelmente com monitoramento do sistema
- BdT realiza balanceamento interno à aplicação sem esse custo
 - por outro lado, não há sobreposição de comunicação e computação

- muitas vezes usados em conjunto
- necessidade de técnica de “retirada”

```
int id;
t_descricao desc;
void worker () {
    int res;
    while (1) {
        in ("tarefa", ?id, ?desc);
        res = faztarefa(desc);
        out ("resultado", id, res);
    }
}
void on_retreat () {
    out ("tarefa", id, desc);
} /* cuidado com condições de corrida! */
```

- modelo simples e muito usado
- tamanho das tarefas: flexibilidade X granularidade
 - discussão faz sentido quando cada tarefa tem tamanho conhecido
 - tamanho de tarefa recebida pode variar dinamicamente
 - processos que respondam muito rapido podem ganhar tarefas maiores
 - ... com bolsa de tarefas ativa!
- detecção de terminação

- Andrews, G. *Foundations of Multithreaded, Parallel, and Distributed Programming*. Addison-Wesley, 2000.
- Andrews, G. Paradigms for process interaction in distributed programs. *ACM Comput. Surv.* 23(1), Mar. 1991, 49-90.
- Carriero, N. and Gelernter, D. How to write parallel programs: a guide to the perplexed. *ACM Comput. Surv.* 21(3), Sep. 1989, 323-357.