Bibliografia em processamento paralelo

- Andrews, "Concurrent Programming: Principles and Practice"

- Foster, "Designing and Building Parallel Programs"

- Wolfe, "High Performance Compilers for Parallel Computing"

- Hwang, "Advanced Computer Architecture"

- Culler, "Computer Architecture"

Por que Paralelismo?

- Limites físicos
- Paralelismo natural
- Software de sistema
- Interesse intrínseco

Aplicações: científicas, sistemas de transação, sistemas de tempo real, tratamento de interrupções

Dimensões do Paralelismo

- Processos e threads
- Modelos de programação
- Concorrente x // x distribuído
- Sistemas //s e distribuídos
- Arquiteturas //s
- Linguagens e runtime
- Métricas de desempenho

Processos e Threads

Processo $\rightarrow$ espaço de endereçamento

Thread $\rightarrow$ mesmo espaço que pai

Processo (lógico) != processador (físico)

Processo abstração de processador

Modelo Von Neumann $\rightarrow$ 1 fluxo de controle

Progs concorrentes $\rightarrow$ +1 fluxo

Modelo de Programação

Define interface usada pelo programador

Paralelismo, comunicação, sincronização, etc

Exemplos: sequencial, memória compartilhada, passagem de mensagens

Concorrente x Paralelo x Distribuído

Concorrente: +1 fluxo de controle

Paralelo: concorrente onde comunicação via memória compartilhada

Distribuído: concorrente onde comunicação via passagem de mensagens

Sistemas Paralelos e Distribuídos

Paralelo: espaço de endereçamento único em hardware

Distribuído: múltiplos espaços de endereçamento

Possível rodar progs distribuídos em sistemas //s e vice-versa

Arquiteturas Paralelas

Arquitetura determina conj de construções e técnicas de programação que podem ser implementadas eficiente/

Mais comuns: SIMD e **MIMD**

Exs: Power Challenge, KSR-1, Sequent, Origin, T3E, Paragon, CM-5. Dash, Flash, Alewife, Typhoon, Enterprise, Multiplus, NCP I, NCP2.

Linguagens, Compiladores e Bibliotecas

Vants lings: sintaxe especial, efeitos colaterais e contexto implícito, integração (verificação de tipos, threads, tratamento de exceções, etc)

Vant compiladores: simplicidade do modelo

Vants biblio: facilidade de modificação, uso com lings existentes, uso com múltiplas lings (principalmente passagem de msgs)

Métricas de Desempenho

Amdahl's Law:

Speedup s = T(1)/T(p)

Trabalho total c = T_s + T_p = T(1)

T(p) = T_s + T_p/p

s = (T_s + T_p) / (T_s + T_p/p) =
= c / (T_s + T_p/p) $\rightarrow$ c/T_s qdo p $\rightarrow$ inf

Um Pouco de História

Computadores inicial/ eram single user – anos 50

1a motivação: devices de E/S; espera ocupada desperdício

Timeslicing (batch) → multiprogramação (concorrência)

E/S programável → condições de corrida (//ismo restrito)

Interrupções de E/S assíncronas – meados dos anos 60

Comunicação entre processos de usuário – início dos 70

Redes (processa/ distribuído) – início dos 70

Multiprocs baseados em mem multi-ported – fim dos 70

Multicomps e multiprocs baseados em barramento – início dos 80

Multiprocs escaláveis – meados dos anos 80

Multiprocs com coerência em HW – início dos 90

Projeto e Verificação de Progs //s

Importante: garantir liveness e safety

Liveness: coisas boas eventualmente acontecem

Safety: coisas más nunca acontecem

Exs liveness: nenhum processo espera para sempre, o prog termina

Exs safety: exclusão mútua, não acontece overflow de buffers

Modelos de Programação + Comuns

- Sequencial
- Memória compartilhada
- Passagem de mensagens
- SPMD vs. MPMD ou //ismo de dados vs. tarefas

Outros Modelos de Programação

- Linda
- Actors
- Dataflow
- Logic
- Functional
- Constraints

Modelo de Programação Sequencial

Modelo mais simples

//ismo implementado pelo compilador ou software + básico

```
for i = 1 to N
    a[i] = 1
```

Exs: HPF (quase) e outros Fortrans (compilador); Prolog paralelo (ex: Andorra-I, YapOr etc) e outras linguagens declarativas (runtime)

Modelo de Memória Compartilhada

Modelo mais complexo, mas ainda próximo do sequencial

//ismo implementado pelo programador através da linguagem ou chamadas ao software + básico

Sincronização necessária

Comunicação transparente (implementada por sw ou hw)

Exs: SR (linguagem), TreadMarks (runtime), SoftFLASH (SO)

Modelo de Memória Compartilhada

```
doall i = 1 to N
    a[i] = 1

for j = 1 to NPROCS-1
    fork(compute,j)
compute(0)

lock(mutex)
    x = x + 1
unlock(mutex)
```

Modelo de Passagem de Mensagens

Modelo extremamente complexo

//ismo implementado pelo programador através da linguagem ou chamadas ao software + básico

Comunicação explícita através de passagem de msgs

Sincronização geralmente associada às msgs

Exs: SR e Occam (linguagem), PVM e MPI (runtime)

Modelo de Passagem de Mensagens

```
Proc pid:

chunk = N/NPROCS
for j = pid*chunk to (pid+1)*chunk-1
    a[i] = 1
send(dest,&a[pid*chunk],chunk*sizeof(int))
```

Outros Modelos

Linda

- + complexo que compart, mas + simples que pass de msgs

- Híbrido: espaço de tuplas + comunic explícita + sincr implícita

- Comandos de acesso a tuplas: in (receive), out (send), rd (consulta), eval (criação de processos), inp e rdp (acessos não bloqueantes)

- Primitivas de Linda podem ser adicionadas a lings tradicionais

Exemplo: SOR

- Computação sobre uma matriz
- Grupo de linhas consecutivas para cada proc
- Cada elemento calculado usando vizinhos
- Comunicação nas bordas

Exemplo: SOR

Modelo sequencial

```
for num_iters
    for num_linhas
        compute
```

Modelo de memória compartilhada

```
for num_iters
    for num_linhas in //
        compute
```

ou

```
for num_iters
    for num_minhas_linhas
        compute
    barreira
```

Exemplo: SOR

Modelo de passagem de msgs (send nao bloqueante)

```
define submatriz local
for num_iters
    if pid != 0
        envia primeira linha a pid-1
        recebe limite superior de pid-1
    if pid != P-1
        envia ultima linha a pid+1
        recebe limite inferior de pid+1
    for num_linhas
        compute
```

Comparação de Modelos

- Sequencial ideal, mas depende de software sofisticado

- Mem compart leva a programas mais simples, mas sincr explícita

- Pass de msgs leva a comunicação eficiente e sincr implícita, mas dificulta programação

SPMD vs. MPMD

- Classificação a nível de programas
- SPMD = //ismo de dados = prog p/ SIMD rodando em MIMD
- MPMD = //ismo de tarefas; Mestre/escravo exemplo