

Linguagens para Programação Paralela

November 26, 2009



- expressividade
- facilidade de uso
- integração com programas e programadores

Modelos de Linguagens Paralelas (lista não completa)

- paralelismo de dados (data-parallel)
- GAS e PGAS
- orientação a processos



Paralelismo de Dados

- suporte para operações sobre estruturas de dados
- linguagens *planas* e *aninhadas*
 - planas* cada elemento da estrutura de dado deve ser escalar
 - aninhadas* elementos podem ser novas estruturas de dados



- operações escalares podem ser aplicadas a arrays inteiros

```
real A(10,20), B(10,20)
```

```
logical L(10,20)
```

```
A = A + 1.0 ! Adds 1.0 to each element of A
```

```
A = SQRT(A) ! Computes square root of each element of A
```

```
L = A .EQ. B ! Sets L(i,j) to .true. if A(i,j)= B(i,j);
```

- uso de triplas (*lowerbound, upperbound, stride*) para manipular seções de arrays

```
A(1:7) = B(1:7) + B(2:8)
```



- HPF: diretivas para distribuição de dados
- mapeamento de arrays para conjunto de processadores (estrutura abstrata)

ALIGN define que itens devem ficar no mesmo processador

DISTRIBUTE especifica forma de distribuição entre processadores

- instrução forall e diretiva INDEPENDENT

Exemplo F77

```
C      diferenças finitas
      program f77_finite_diff
      real X(100,100), New(100,100)
      do i = 2,99
         do j = 2,99
            New(i,j) = (X(i-1, j) + X(i+1, j) +
$           X(i, j-1) + X(i, j+1))/4
         enddo
      enddo
      diffmax = 0.0
      do i = 1,100
         do j = 1,100
            diff = abs(New(i,j)-X(i,j))
            if (diff .gt. diffmax) diffmax = diff
         enddo
      enddo
      end
```



Exemplo HPF

```
program hpf
!HPF$ PROCESSORS pr(4)
    real X(100,100), New(100,100)
!HPF$ ALIGN New(:, :) WITH X(:, :)
!HPF$ DISTRIBUTE X(BLOCK,*) onto pr
    New(2:99) =(X(1:98, 2:99) + X(3:100, 2:99) +
$              X(2:99, 1:98) + X(2:99, 3:100))/4
    diffmax = MAXVAL(ABS(New-X))
end
```



- linguagem com características funcionais baseada em ML
- estrutura de dados básica: sequência

```
[2, 1, 9, -3]
```

- estrutura de controle paralelo principal: *apply-to-each*

```
{a * a : a in [3, -4, -9, 5]};
```

- paralelismo aninhado

```
{sum(a) : a in [[2,3], [8,3,9], [7]]};
```



NESL – loops com paralelismo aninhado

```
function my_sort(a) =  
  let  
    rank = {count({x < y : x in a})  
           : y in a};  
    result = permute(a,rank);  
  in result;  
sort([2, 3, -7, 6, 5, 22, -8]);
```



- visão intermediária entre paralelismo de dados e de processos: compartilhamento de dados com conhecimento de localização
 - Titanium, UPC, co-array Fortran
 - biblioteca GASNet: biblioteca para construção de sistemas PGAS
- modelos de programação fortemente síncronos

Titanium - baseada em Java

```
Point<3> startCell = myBlockPos * numCellsPerBlockSide;
Point<3> endCell = startCell + (numCellsPerBlockSide - [1,1,1]);
double [3d] myBlock = new double[startCell:endCell];
/**/ "blocks" is a temporary 1D array that is used
    to construct the "blocks3D"
array double [1d] single [3d] blocks =
    new double [0:(Ti.numProcs()-1)] single [3d];
blocks.exchange(myBlock);
/**/ create local "blocks3D" array (indexed by 3D block position)
double [3d] single [3d] blocks3D =
    new double [[0,0,0]:numBlocksInGridSide - [1,1,1]] single [3d]
/**/ map from "blocks" to "blocks3D" array
foreach (p in blocks3D.domain())
    blocks3D[p] = blocks[procForBlock- Position(p)];
```



- ver slides em separado

- DARPA – projeto PERCS (Productive Easy-to-use Reliable Computer Systems)
- aumento de produtividade de programadores...
- X10 (IBM) e Chapel (Cray)
- ênfase em construções assíncronas

- visão global (... PGAS)
- suporte a paralelismo de forma geral (dados e tarefas)
 - arquiteturas *NUCC*
- separação de algoritmo e implementação
- facilidades “de mercado”: genéricos, inferência de tipos, módulos, ...
- abstrações de dados
- desempenho
- transparência do modelo de execução
- portabilidade
- ...



X10 – Modelo da Linguagem

- derivação de Java (?)
- sublinguagem para arrays
- PGAS – reificação de localidade: *places*
- *places* (espaços de endereçamento?) são fixados no início da execução do programa
- computação realizada por *atividades* (threads desacoplados de objetos)
 - variáveis usadas por mais que uma atividade devem ser declaradas como *final*

```
final place Other = here.next();
final T t = new T();
finish async (Other){
    final T t1 = new T();
    async (t) t.val = t1;
}
```



- *pontos* são usados para definir *regiões*
- *distribuições* mapeiam os pontos de uma região a um lugar
- regiões e distribuições não se alteram durante a vida de um array
- distribuições podem ser definidas na declaração e consultadas durante execução

```
int value [.] A = new int[ [1:10,1:10] ]  
    (point[i,j]) { return i+j; } ;
```

- *for*: iteração sequencial
- *foreach*: iteração paralela sobre pontos em uma região

```
foreach (point[i]: A.region.rank(0))
  for (point[j]: [(A.region.rank(1).low()+1):
                  A.region.rank(1).high()])
    A[i,j] = A[i,j-1] + 1;
```
- *ateach*: iteração paralela sobre pontos em uma distribuição

```
ateach ( point [i,j] : A ) A[i,j] = f(B[i,j]) ;
```

- `async` A dispara de atividade A
- `finish` A suspende criador de A até que A termine
- blocos atômicos
- blocos atômicos condicionais: `when (c) S`
- `future(P)` E dispara atividade para acaviar E e retorna futuro

- ênfase em processos e mensagens
 - communicating sequential processes
- escalabilidade de processos: tempo e custo
- Occam (super antiga mas com derivação Occam-Pi), Erlang

Exemplo Erlang

```
start(Sys, Proc) ->
    counter(Sys, Proc, 0).

counter(Sys, Proc, Tot) ->
    receive {
        Proc ? bump =>
            Tot1 = Tot + 1,
            counter(Sys, Proc, Tot1);
        Sys ? report =>
            Sys ! Tot,
            counter(Sys, Proc, Tot);
        Sys ? reset =>
            counter(Sys, Proc, 0).
    }
```

