Programação Paralela e Concorrente

Linguagens para Programação Paralela — PGAS

Noemi Rodriguez

2016

Linguagens de Programação Paralelas

- paralelismo de dados (data-parallel)
- GAS e PGAS
- orientação a processos
  - memória compartilhada
  - troca de mensagens

Paralelismo de Dados: exemplo HPF

Fortran 90: operações com arrays

- operadores escalares estendidos para arrays:

```
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);
```

HPF: expressão de paralelismo sobre dados

- processadores virtuais
- arrays distribuídos entre processadores *cyclic, block, ...)
- alinhamento entre arrays
- programa descreve o mundo global e não a visão de um único processo

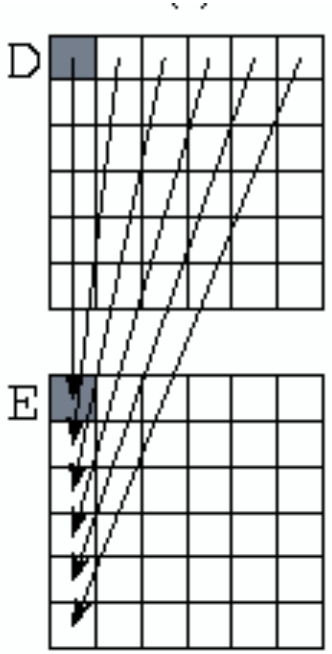
Paralelismo de Dados: exemplo HPF

```
!distribution directives not supplied - replication
  PROG1
  INTEGER BARRAY(100)
!default distribution directives supplied - replication
  PROG2
  INTEGER BARRAY(100)
!HPF$ DISTRIBUTE BARRAY(*)

  REAL X(16), Y(16)
!HPF$ DISTRIBUTE Y(BLOCK)
!HPF$ ALIGN X(I) WITH Y(I+1)
```

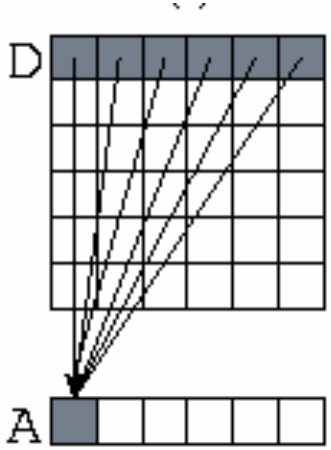
Paralelismo de Dados: exemplo HPF

```
!HPF$ PROCESSORS p(20)
      real A(100,100), B(100, 100), C(100,100)
!HPF$ ALIGN B(:,:) WITH A(:,:)
!HPF$ ALIGN C(i,j) WITH A(j,i)
!HPF$ DISTRIBUTE A(BLOCK,*) ONTO p
```

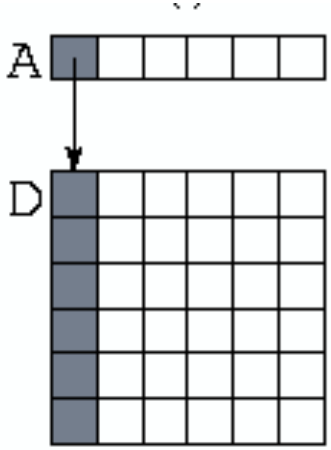
Alinhamento em HPF




```
ALIGN D(I,J)
   WITH E(J,I)
```




```
ALIGN D(:,*)
   WITH A(:)
```




```
ALIGN A(:)
   WITH D(*,:)
```

Linguagens PGAS

- Partitioned Global Address Space
- ... tipicamente final dos anos 90
- Titanium, Co-array FORTRAN, UPC

Linguagens PGAS

- HPF *retrospectivamente* PGAS
- processadores (possivelmente virtuais) chamados de *lugares* em vocabulário PGAS
- função *custo* reflete custo de acesso a diferentes dados
  - tipicamente "barato" e "caro"
- ditribuições pré-definidas para dados regulares
  - *cyclic*, *block-cyclic* e *block*
- dados remotos acessados explicita ou implicitamente

UPC

- variáveis podem ser `private` ou `shared`
- variáveis compartilhadas podem ser usadas por todos mas explicitam o custo de comunicação
  - e são de fato o único ponto onde isso fica explícito
- variáveis escalares compartilhadas: sempre na thread 0
- vetores distribuídos entre threads

UPC: compartilhamento

```
shared int hits;
main(int argc, char **argv) {
    int i, my_trials = 0;
    int trials = atoi(argv[1]);
    my_trials = (trials + THREADS - 1)/THREADS;
    srand(MYTHREAD*17);
    for (i=0; i < my_trials; i++)
      hits += hit();
    upc_barrier;
    if (MYTHREAD == 0) {
      printf("PI estimated to %f.", 4.0*hits/trials);
    }
}
```

shared variable to record hits

divide work up evenly

accumulate hits

What is the problem with this program?

UPC: compartilhamento

- Have each thread update a separate counter:
  - But do it in a shared array
  - Have one thread compute sum

```
shared int all_hits [THREADS];
main(int argc, char **argv) {
  ... declarations an initialization code omitted
  for (i=0; i < my_trials; i++)
    all_hits[MYTHREAD] += hit();
  upc_barrier;
  if (MYTHREAD == 0) {
    for (i=0; i < THREADS; i++) hits += all_hits[i];
    printf("PI estimated to %f.", 4.0*hits/trials);
  }
}
```

all_hits is shared by all processors, just as hits was

update element with local affinity

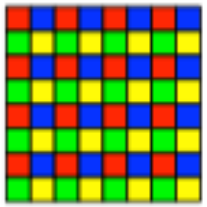
UPC: distribuição de trabalho

```
/* vadd.c */
#include <upc_relaxed.h>
#define N 100*THREADS

shared int v1[N], v2[N], sum[N];
void main() {
    int i;
    for(i=0; i<N; i++)
        if (MYTHREAD == i%THREADS)
            sum[i]=v1[i]+v2[i];
}
```

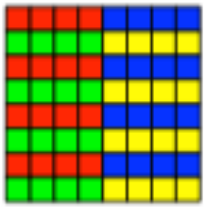
cyclic layout

owner computes

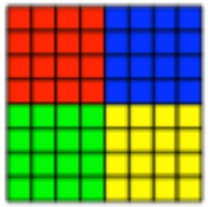
distribuição de trabalho — formas comuns



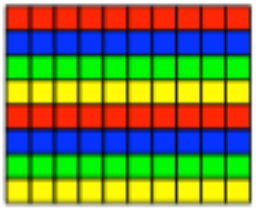
(a) (cyclic,cyclic)-distribution



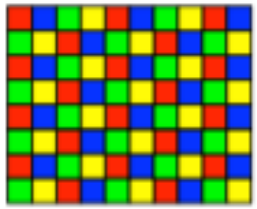
(b) (cyclic,block)-distribution



(c) (block,block)-distribution



(d) (cyclic,*)-distribution



(e) (cyclic)-distribution

UPC: distribuição de trabalho

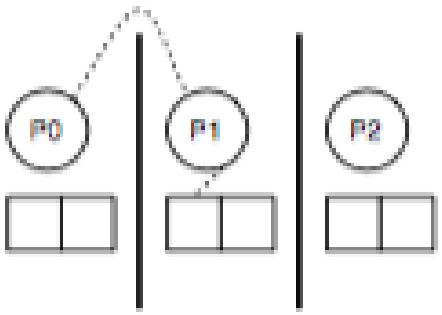
```
#define N 100*THREADS

shared int v1[N], v2[N], sum[N];

void main() {
    int i;
    upc_forall(i=0; i<N; i++; i)
            sum[i]=v1[i]+v2[i];
}
```

The cyclic data distribution may perform poorly on some machines

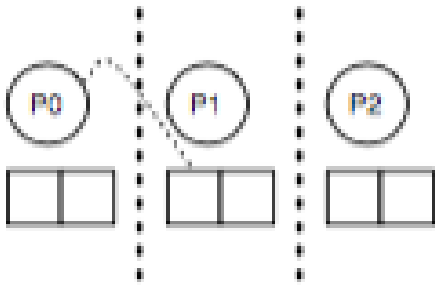
afinidade

- expressão extra no *forall*
- valor inteiro: avaliada se valor (módulo # threads) igual ao id da thread
- ponteiro: avaliada se endereço é local

PGAS: modelo de memória



(a) Message-passing



(b) Partitioned-memory

(c) Shared-memory

PGAS – anos 2000

- DARPA – projeto PERCS (Productive Easy-to-use Reliable Computer Systems)
- aumento de produtividade de programadores...
- arquiteturas *NUCC*
  - arquiteturas híbridas: non-uniform cluster computing
- X10 (IBM) e Chapel (Cray)
- ênfase em construções assíncronas

Chapel – paralelismo de dados

```
1  const BigD = {0..n+1, 0..n+1} dmapped Block(boundingBox=[0..n+1, 0..n+1]),
2            D: subdomain(BigD) = {1..n, 1..n};
3  var A, Temp: [BigD] real;
4
5  do {
6    forall (i,j) in D do
7      Temp[i,j] = (A[i-1,j] + A[i+1,j] + A[i,j-1] + A[i,j+1]) / 4;
8    const delta = max reduce abs(A[D] - Temp[D]);
9    A[D] = Temp[D];
10 } while (delta > epsilon);
```

Chapel – paralelismo de tarefas

```
1  proc quickSort(arr: [?D],
2                 thresh = log2(here.numCores()), depth = 0,
3                 low: int = D.low, high: int = D.high) {
4    if high - low < 8 {
5      bubbleSort(arr, low, high);
6    } else {
7      const pivotVal = findPivot(arr, low, high);
8      const pivotLoc = partition(arr, low, high, pivotVal);
9      serial(depth >= thresh) do cobegin {
10       quickSort(arr, thresh, depth+1, low, pivotLoc-1);
11       quickSort(arr, thresh, depth+1, pivotLoc+1, high);
12 } } }
```

Chapel e X10 – Modelo da Linguagem

- derivação de sintaxes anteriores
- sublinguagem para arrays
- APGAS – assincronismo!
  - reificação de localidade: *locales* e *places*
  - *places* fixados no início da execução do programa
- programador define distribuição de dados entre os *places*
- computação realizada por *atividades* (threads desacoplados de objetos)
  - X10: variáveis usadas por mais que uma atividade devem ser declaradas como *final*
    - imutabilidade!

Referências

PGAS M. De Wael e outros. Partitioned Global Address Space Languages. ACM Computing Surveys, 47(4), May 2015.

UPC : http://upc.lbl.gov/publications/

Chapel B. Chamberlain e outros. Parallel Programmability and the Chapel Language. International Journal of High Performance Computing Applications, August 2007, 21(3): 291-312.

X10 P. Charles e outros. X10: An Object-oriented approach to non-uniform Clustered Computing. OOPSLA 2005.