ALGORÍTMOS PARALELOS
(Aula 6)

Neyval C. Reis Jr.

OUTUBRO/2004

Laboratório de Computação
de Alto Desempenho

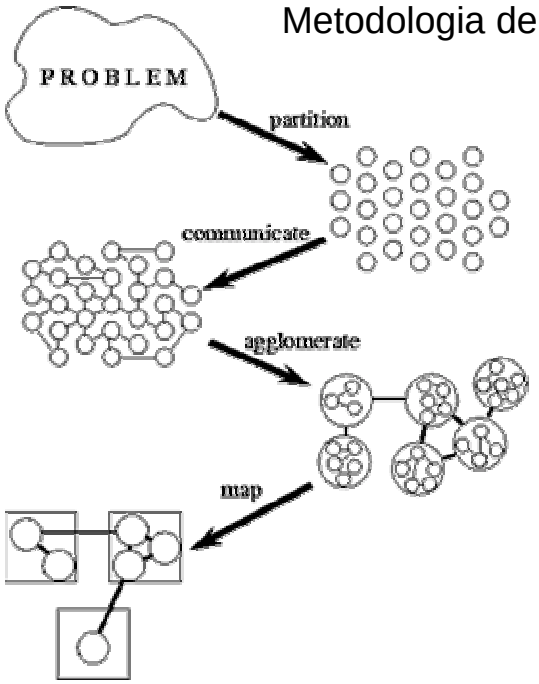
DI/UFES

Tópico	20 janeiro	27 janeiro	3 fev	10 fev	17 fev	24 fev	3 março
Paradigma de Paralelismo de Dados	■			Carnaval			
Memória Compartilhada e Distribuída		■					
Memória Compartilhada (OPEN MP)		■					
Trabalho - Análise de complexidade de algoritmos			■				
Modelos de Programação Paralela			■				
Trabalhos - Super LU					■		
Trabalhos - Otimização					■		
Trabalhos – Medição de desempenho						■	
Trabalhos - POM						■	
Trabalhos - GMRES							■
Trabalhos – Algoritmos Assíncronos							■

Programa do Curso

1. Introdução
2. Arquitetura de Computadores
3. Arquiteturas de Sistemas Paralelos
4. Computação de Alto Desempenho
5. Programação Paralela (modelos e paradigmas)
6. Análise de Desempenho e Instrumentação
7. Aplicações

Metodologia de design



Tipos de Ambientes e Paradigmas de Programação

Troca de Mensagens (Message Passing): é o método de comunicação baseada no envio e recebimento de mensagens através da rede seguindo as regras do protocolo de comunicação entre vários processadores que possuam memória própria. O programador é responsável pela sincronização das tarefas. (Aula anterior)

Paralelismo de Dados (Data Parallel): é a técnica de paralelismo de dados, normalmente automática ou semi-automática, ou seja, é o método que se encarrega de efetuar toda a comunicação necessária entre os processos de forma que o programador não necessita entender os métodos de comunicação.

Memória Distribuída e Compartilhada (Distributed-Shared Memory): Emula uma máquina de memória compartilhada em uma máquina de memória distribuída. O método que se encarrega de efetuar toda a comunicação necessária entre os processos de forma que o programador não necessita entender os métodos de comunicação.

Tipos de Ambientes e Paradigmas de Programação

Troca de Mensagens (Message Passing): é o método de comunicação baseada no envio e recebimento de mensagens através da rede seguindo as regras do protocolo de comunicação entre vários processadores que possuam memória própria. O programador é responsável pela sincronização das tarefas.

Paralelismo de Dados (Data Parallel): é a técnica de paralelismo de dados, normalmente automática ou semi-automática, ou seja, é o método que se encarrega de efetuar toda a comunicação necessária entre os processos de forma que o programador não necessita entender os métodos de comunicação.

Memória Distribuída e Compartilhada (Distributed-Shared Memory): Emula uma máquina de memória compartilhada em uma máquina de memória distribuída. O método que se encarrega de efetuar toda a comunicação necessária entre os processos de forma que o programador não necessita entender os métodos de comunicação.

Paralelismo de Dados (Data Parallel):

- Motivação
- Paradigma e Linguagens
- Visão geral do HPF
- Programando
  - Distribuição de dados
  - Loops em paralelo
- Comparação de Performance

Troca de Mensagens

- **Programando para sistemas de memória distribuída:**
  - programas escritos em linguagens concencionais (FORTRAN, C ou C++)
  - Todas as variáveis são locais
  - Comunicação através de chamadas de sub-rotinas
- **Controle completo: distribuição de dados e tarefas**
- **Performance é resultado de um balanço entre comunicação e paralelismo**
- **As trocas de mensagens são base do processamento paralelo.**

Troca de Mensagens

Extremamente flexível, porém ...

- Codificação e debugging são tarefas difíceis
- Programador é completamente responsável pela produção de um código eficiente e em manter os processadores ocupados, balanceados e em constante comunicação.

Paradigma do Paralelismo de Dados

Motivação

- **Programação de alto nível.**
- **Paralelismo implícito na distribuição dos dados.**
- **Detalhes de comunicação não são percebidos pelo programador.**
- **Permite ao programador um maior foco na aplicação.**

Paradigma do Paralelismo de Dados

O objetivo principal é afastar o foco da arquitetura e dos detalhes de linguagem necessários para a utilização eficiente de processamento paralelo. A maior motivação para isso é permitir que não apenas cientistas da computação tenham acesso a HPC (High Performance Computing), mas também engenheiros, físicos, meteorologistas, etc.

A meta é atrair programador que deseje explorar a razão custo/benefício oferecida por sistemas paralelos de memória distribuída, mas não tenham desejo de investir o tempo necessário para adquirir os conhecimentos relacionados a um ambiente de troca de mensagens.

Motivação para a Distribuição de Dados

- On distributed memory machines, each processor has a local memory
  → arrays must be distributed across many different processors
- Important: data distribution implies work distribution (owner computes rule)
- Good data distribution reduces communication overhead and produces more effective code
- Automatic data distribution by compiler is not sophisticated enough
- Compiler directives needed to specify type of distribution to use
- Mapping directives can also be used to take better advantage of the cache on shared memory machines

Paradigma do Paralelismo de Dados

- ***A idéia central do paradigma do Paralelismo de Dados é permitir que todas as operações sobre vetores e matrizes sejam executadas em paralelo.***
- ***Tipicamente, um único programa controla a distribuição de dados e operações entre os processadores (SPMD – Single Program Multiple Data).***
- ***As linguagens utilizadas variam de FORTRAN padrão ao C e C++, contando com extensões para lidar com o paralelismo.***
- ***Na maioria dos casos a distribuição de dados e operações é efetuada pelo compilador com a "orientação" do programador.***

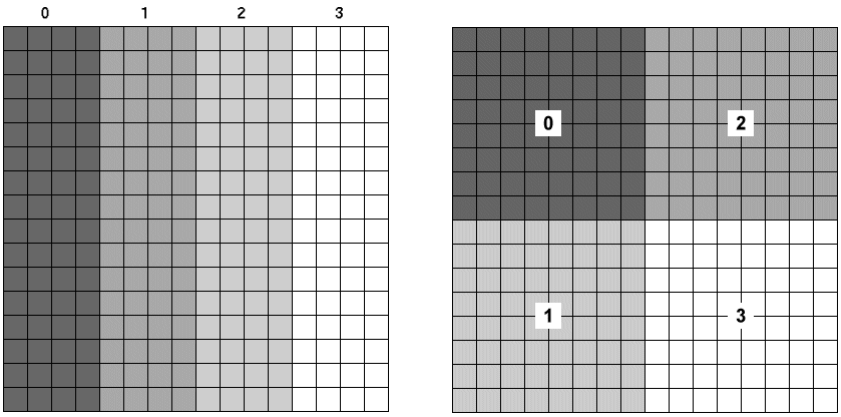
Paradigma do Paralelismo de Dados

Características principais:

- Um único programa controla todas as operações
- Espaço de endereçamento global: o programador apenas vê uma memória global (compartilhada), com todos os detalhes de baixo nível para distribuição dos dados, acessos de memória e comunicação deixados para o compilador.
- Execução "relativamente" assíncrona: A execução de cada instrução não é efetuada de maneira estritamente síncrona entre todos os processadores, todavia, alguns comandos forçam a sincronização (como o fim de um loop, por exemplo).
- Operações em paralelo: As operações sobre vetores são efetuadas simultaneamente por todos os processadores.

Paradigma do Paralelismo de Dados

Distribuição de dados



Paradigma do Paralelismo de Dados

Operações sobre vetores

2	12
8	25
18	42

c

=

1	4
2	5
3	6

a

*

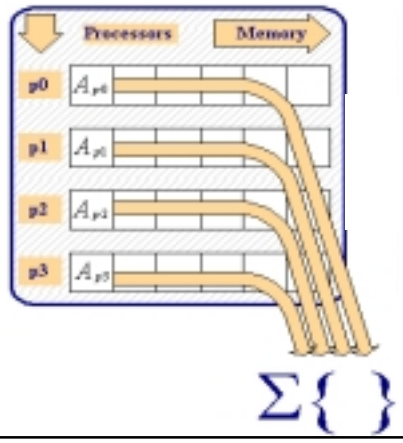
2	3
4	5
6	7

b

Paradigma do Paralelismo de Dados

A distribuição de dados guia a distribuição de tarefas e as trocas de mensagens ocorrem de maneira transparente para o programador

Exemplo : Reduções ou somatórias



Paradigma do Paralelismo de Dados

Vantagens: Fácil de Usar

- Fácil escrever programas (não existe troca de mensagens explícita)
- Tarefas de debug são mais simples
- Maior portabilidade

Desvantagens: Flexibilidade e Controle

- Controle restrito sobre a distribuição de dados e tarefas
- Mais difícil de obter performance ótima
- *Fortemente dependente de bons compiladores*

Paradigma do Paralelismo de Dados

- **Muitas linguagens implementadas com este paradigma:**
  - Fortran-Plus, MP Fortran, CM Fortran, C*, CRAFT, Fortran D, Vienna Fortran, POOMA (C++)
- **Cada linguagem expressa as operações de Paralelismo de Dados de maneira diferente**
- **Problemas:**
  - Muitas linguagens e dialetos
  - Linguagens específicas para cada arquitetura
- **Resultado:**
  - Falta de portabilidade (programadores ficam presos aos fabricates e novos usuários muitas vezes não têm vontade de aprender)
- **Necessidade de padronização** → HPF ou HPC++

High Performance Fortran Forum

Represented Organizations

Alliant Computer Szstems Cor.	ICASE	Rice University
Amoco Production Company	Intel Supercomputer S. Div.	Schlumberger
Applied Parallel Research	Lahey Computer	Shell
Archipel	Lawrence Livermore Nat. Lab.	State Univ. of N.Y. at Buffalo
CONVEX Computer Corporation	Los Alamos National Lab.	SunPro, Sun Microsystems
Cornell Theory Center	Louisiana State University	Syracuse University
Craz Research, Inc.	MasPar Computer Corporation	TNO-TU Delft
DEC MPS Group	Meiko, Inc.	Thinking Machines Corporation
Fujitsu America	nCUBE, Inc.	United Technologies Corporation
Fujitsu Laboratories	Ohio State University	University of Stuttgart
GMD, I1.T	Oregon Grad. Inst. of Science/T.	University of Vienna
Hewlett Packard	Portland Group, Inc.	Yale University
IBM	Research Inst. for Advanced C.S.	

Implementações HPFF

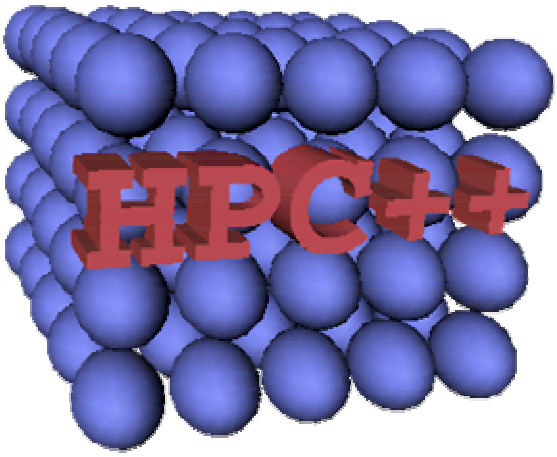
Número de implementações crescendo rapidamente:

- **Produtos Anunciados:**
  - HP, Hitachi, IBM, Intel, NEC, Portland Group, Kuck & Associates, Meiko, Motorola, NA Software, Pacific-Sierra Research, Applied Parallel Research
- **Compiladores de domínio Público:**
  - University of Southampton, GMD
- **Esforço de desenvolvimento já anunciado:**
  - ACE, Convex, Cray Computer, Fujitsu, Lahey, MasPar, Nag, nCUBE, TMC
- **Interessados:**
  - ACSET, Cray Research (with PGI), Silicon Graphics, Sun

High Performance C++ Consortium

- Consórcio criado por Indiana University, University of Oregon, Los Alamos Advanced Computing Lab e a Real World Computing Partnership, que está se dedicando ao desenvolvimento do HPC++,

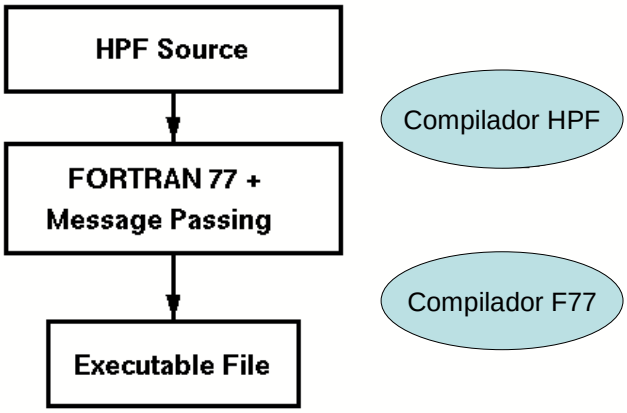
- A linguagem dá suporte ao desenvolvimento de aplicações com paralelismo de dados e tarefas com base no paradigma de Paralelismo de Dados.



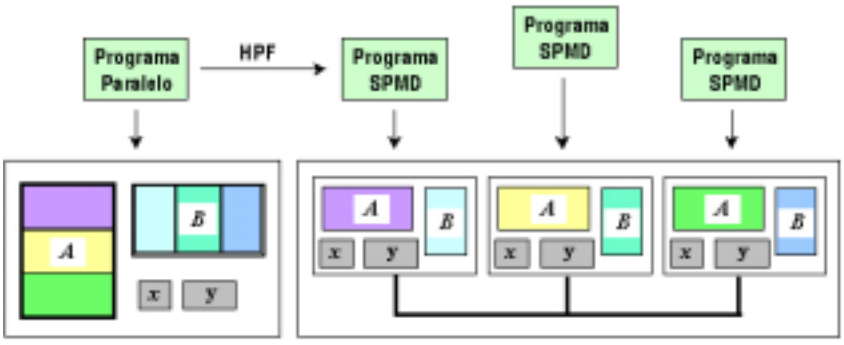
http://www.extreme.indiana.edu/hpc++/index.html

HPF

Compilação HPF



Modelo HPF de Execução



Implementações HPFF

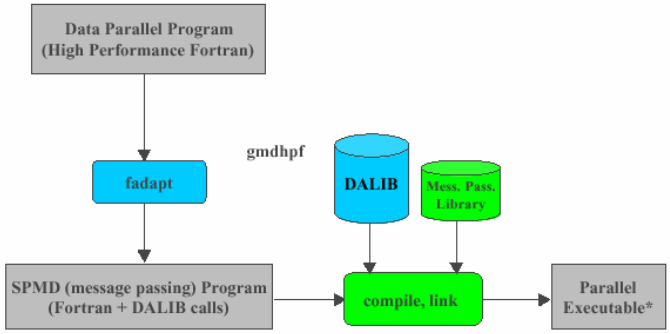
Número de implementações crescendo rapidamente:

- **Produtos Anunciados:**
  Applied Parallel Research, HP, Hitachi, IBM, Intel, Kuck & Associates, Meiko, Motorola, NA Software, NEC, Pacific-Sierra Research, Portland Group
- **Compiladores de domínio Público:**
  University of Southampton, GMD

HPF - ADAPTOR

- **Esforço de desenvolvimento já anunciado:**
  ACE, Convex, Cray Computer, Fujitsu, Lahey, MasPar, Nag, nCUBE, TMC
- **Interessados:**
  ACSET, Cray Research (with PGI), Silicon Graphics, Sun

HPF ADAPTOR (utilizado no LCAD)

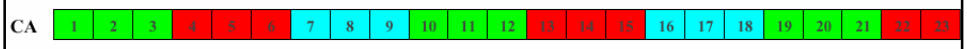


- **Interactive source-to-source translation fadapt**
- **runtime system DALIB**
- **compiler driver gmdhpf**
  features for performance analysis
  coupling with HPF related tools

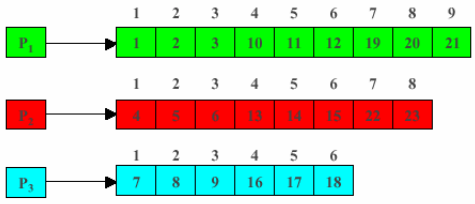
Distribuição de Dados

- **Compiler directive to specify type of distribution to use**
- **Specifies the distribution for each dimension of array**
- **Formats:**
  - !HPF$ DISTRIBUTE a(*distribution*)
  - !HPF$ DISTRIBUTE (*distribution*) :: a, b
- **!HPF$ is used for all HPF compiler directives -- this is a comment to non-HPF compilers**
- ***distribution* is a comma-separated list of the distributions for each array dimension**

Distribuição de Dados



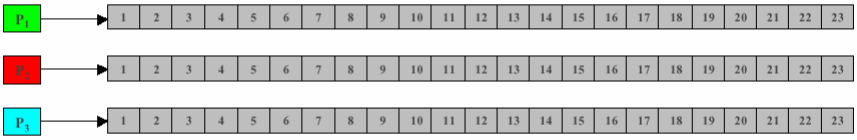

```
      real, dimension (23) :: CA
!hpf$ processors P(number_of_processors())
!hpf$ distribute CA (cyclic(3)) onto P
```



- **every processor allocates only memory for its local part / section**
- **dynamic arrays (number of processors not known at compile time)**
- **global addresses must be translated to local addresses**

Layout of Arrays without Mapping

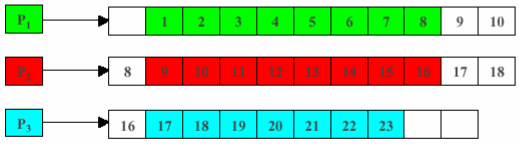

```
real, dimension (23) :: A
```



- treated like scalar variables
- every processor gets a full copy of the array
- every processor is owner of the array
- every processor makes all updates on the array

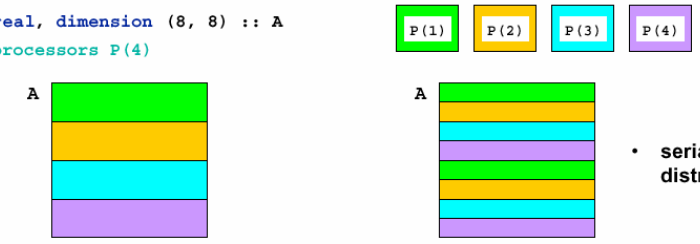
Shadow


```
real, dimension (23) :: A
!hpf$ distribute A (block)
!hpf$ shadow A (1:2)
```



- shadow provides additional memory to store non-local values of neighbored processors

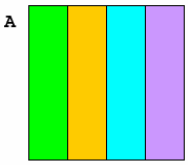
```
real, dimension (8, 8) :: A
!hpf$ processors P(4)
```



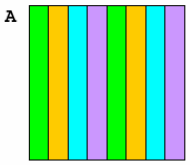
distribute (block, *) onto P

distribute (cyclic,*) onto P

- serial (collapsed) + distributed dimension



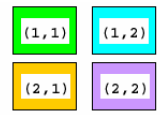
stribute (*, block) onto P

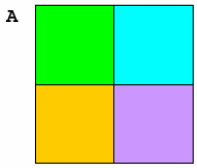


distribute (*,cyclic) onto P

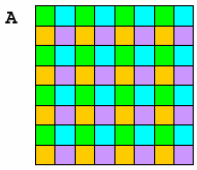
```
real, dimension (8, 8) :: A
!hpf$ processors P(2,2)
```

- two distributed dimensions

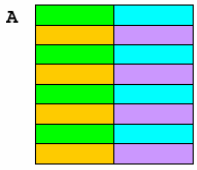




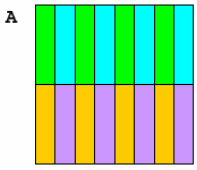
distribute (block, block) onto P



distribute (cyclic, cyclic) onto P



distribute (cyclic, block) onto P



distribute (block, cyclic) onto P

ALIGN / TEMPLATE Directive

```
      real, dimension     (N, N) :: U, V, W, U_OLD, V_OLD, W_OLD
      real, dimension (3, N, N) :: A

!hpf$ distribute (block, block) :: U, V, W, U_OLD, V_OLD, W_OLD
!hpf$ distribute (*, block, block) :: A
```

Better Solution: align arrays with a template

```
!hpf$ template SPACE (N,N)
!hpf$ distribute (block, block) :: SPACE
!hpf$ align (I,J) with SPACE (I,J) :: U, V, W, U_OLD, V_OLD, W_OLD
!hpf$ align (*,I,J) with SPACE (I,J) :: A
```

- **compiler might generate more efficient code**
- **easier to change the distribution of all arrays**

HPF: Data Parallelism

a) Array Operations

```
A(2:N-1) = (A(1:N-2)+A(3:N)) * 0.5
A = (cshift(A, dim=1, shift=1) + A) * 0.5
```

b) FORALL statement / construct

```
forall (I=2:N-1)
   A(I) = (A(I-1)+A(I+1)) * 0.5
end forall
```

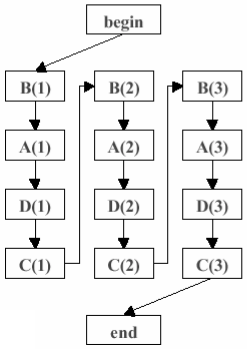
c) INDEPENDENT directive and related clauses

```
!hpf$ independent
      do I=1,N
        Y(K(I)) = X(I) + C(K(I)) * Y(K(I))
      end do
```

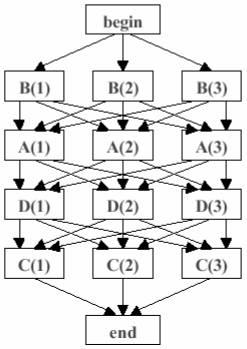
d) automatic parallelization of DO loops (do not rely on it !!!)

Precedence Graphs

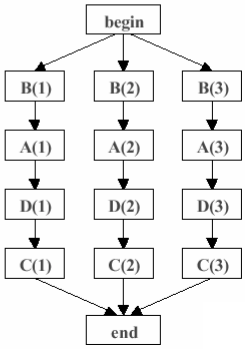
```
do I = 1, 3
  A(I) = B(I)
  C(I) = D(I)
end do
```




```
forall (I=1:3)
  A(I) = B(I)
  C(I) = D(I)
end forall
```




```
!hpf$ independent
      do I = 1, 3
        A(I) = B(I)
        C(I) = D(I)
      end do
```



$$\begin{bmatrix} 11 & 0 & 0 & 0 & 0 \\ 0 & 22 & 0 & 0 & 0 \\ 0 & 0 & 33 & 0 & 0 \\ 0 & 0 & 0 & 44 & 0 \\ 0 & 0 & 0 & 0 & 55 \end{bmatrix}$$

Applying the FORALL statement

FORALL (i = 2:5) C(i, i) = C(i-1, i-1)

to this array produces the following result:

$$\begin{bmatrix} 11 & 0 & 0 & 0 & 0 \\ 0 & 11 & 0 & 0 & 0 \\ 0 & 0 & 22 & 0 & 0 \\ 0 & 0 & 0 & 33 & 0 \\ 0 & 0 & 0 & 0 & 44 \end{bmatrix}$$

$$\begin{bmatrix} 11 & 0 & 0 & 0 & 0 \\ 0 & 22 & 0 & 0 & 0 \\ 0 & 0 & 33 & 0 & 0 \\ 0 & 0 & 0 & 44 & 0 \\ 0 & 0 & 0 & 0 & 55 \end{bmatrix}$$

However, the following apparently similar DO loop

```
DO i = 2, 5
  C(i, i) = C(i-1, i-1)
END DO
```

produces a completely different result:

$$\begin{bmatrix} 11 & 0 & 0 & 0 & 0 \\ 0 & 11 & 0 & 0 & 0 \\ 0 & 0 & 11 & 0 & 0 \\ 0 & 0 & 0 & 11 & 0 \\ 0 & 0 & 0 & 0 & 11 \end{bmatrix}$$

$$\begin{bmatrix} 11 & 0 & 0 & 0 & 0 \\ 0 & 22 & 0 & 0 & 0 \\ 0 & 0 & 33 & 0 & 0 \\ 0 & 0 & 0 & 44 & 0 \\ 0 & 0 & 0 & 0 & 55 \end{bmatrix}$$

Applying the FORALL statement

```
FORALL (i = 2:5) C(i, i) = C(i-1, i-1)
```

to this array produces the following result:

$$\begin{bmatrix} 11 & 0 & 0 & 0 & 0 \\ 0 & 11 & 0 & 0 & 0 \\ 0 & 0 & 22 & 0 & 0 \\ 0 & 0 & 0 & 33 & 0 \\ 0 & 0 & 0 & 0 & 44 \end{bmatrix}$$

However, the following apparently similar DO loop

```
DO i = 2, 5
  C(i, i) = C(i-1, i-1)
END DO
```

produces a completely different result:

$$\begin{bmatrix} 11 & 0 & 0 & 0 & 0 \\ 0 & 11 & 0 & 0 & 0 \\ 0 & 0 & 11 & 0 & 0 \\ 0 & 0 & 0 & 11 & 0 \\ 0 & 0 & 0 & 0 & 11 \end{bmatrix}$$

NEW Clause

- allows private (temporary) data in an INDEPENDENT DO loop
- modifies meaning of the INDEPENDENT directive
- overcomes some of the restrictions
- asserts that the value of the NEW variable is not needed at the start and end of each iteration
- value is undefined outside the loop
- can only be used with DO loops

```
!hpf$ independent, new (T)
      do I = 1, N
         T = A(I) * B(I)
         A(I) = sqrt (T)
      end do
```

REDUCTION Clause

- restricted use reduction variable (no read of intermediate results)
- reduction operation is associative and commutative (beside rounding errors)
- every processor gets own copy of the reduction variable, global reduction at the end of the loop
- can only be used with DO loops

```
!hpf$ independent, reduction (S)
      do I = 1, N
         S = S + A(I) * B(I)
      end do
```

Numerical Integration: Parallelism

- **Array Syntax**

```
V = ( (/(I,I=1,N)/) - 0.5 ) * W
V = 4.0 / (1.0d0 + V * V)
GSUM = sum (V)
```

- **FORALL construct**

```
forall (I=1:N)
    V(I) = (I - 0.5) * W
    V(I) = 4.0 / (1.0d0 + V(I)*V(I)
end forall
GSUM = sum (V)
```

- **INDEPENDENT loop**

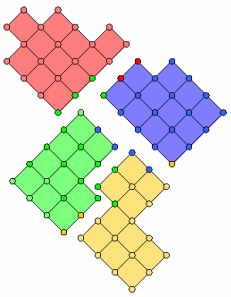
```
!hpf$ independent
      do I = 1, N
        V(I) = (I - 0.5) * W
        V(I) = 4.0/(1.0d0+V(I)*V(I))
      end do
      GSUM = sum (V)
```

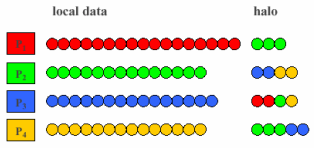
- **INDEPENDENT loop + clauses**

```
!hpf$ independent, new(V1), &
!hpf$ reduction (GSUM)
      do I = 1, N
         V1 = (I - 0.5) * W
         V1 = 4.0 / (1.0d0 + V1*V1)
         GSUM = GSUM + V1
      end do
```

Outros comandos

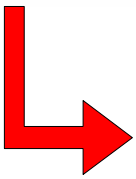
Halos



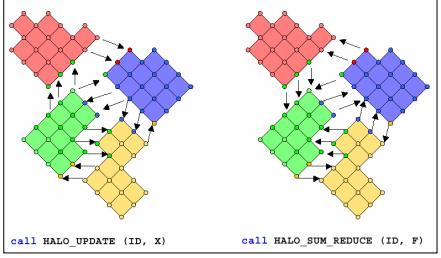



```
call HALO_DEFINE (ID, NODES, IX)
call HALO_CREATE_SHADOW (ID, X)
call HALO_CREATE_SHADOW (ID, F)
```

- arrays aligned to 'nodes' need halo (shadow area)



Update / Reduce for Halos



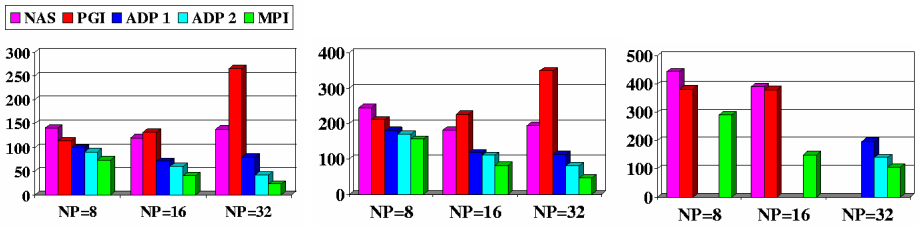
```
call HALO_UPDATE (ID, X)
```

```
call HALO_SUM_REDUCE (ID, F)
```

Performance

Comparação entre 3 compiladores HPF (NAS, PGI e Adaptor) e MPI para a paralelização de um programa de CFD

Test Case Name	Sizes and Number of Subdomains	Total Mesh Points	Processing Nodes
Small	(65 x 33 x 9) x 8	154440	1-8
Medium	(65 x 65 x 9) x 8	304200	2-16
Large	(129 x 65 x 9) x 8	603720	4-64



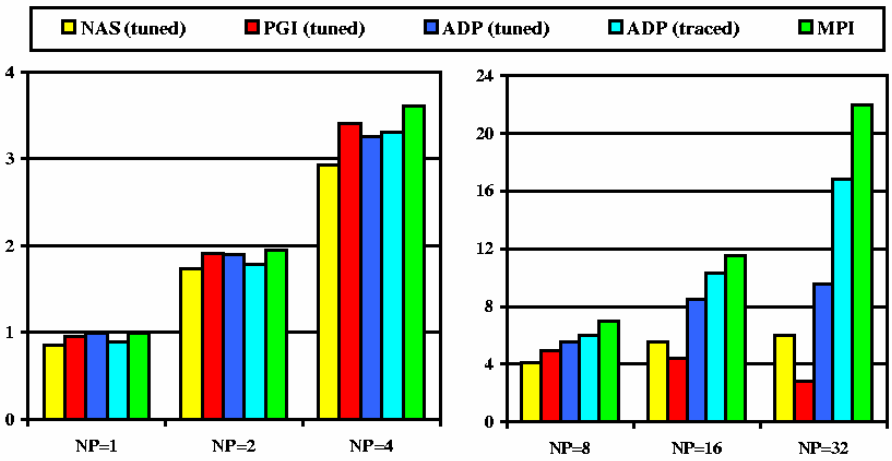
Pequeno

Médio

Grande

Performance

Speed-up Caso Grande



Fontes de informação adicionais

HPF:

http://www.epcc.ed.ac.uk/computing/training/document_archive/hpf-slides/hpf-course-slides-1.html
http://www.hp.com/techservers/tutorials3/hpf.html

HPC++:

http://www.extreme.indiana.edu/hpc++/index.html

HPF – ADAPTOR:

http://www.gmd.de/SCAI/lab/adaptor/www/adaptor_home.html

Exemplo

Fatoração LU:

$$\mathbf{A = L \,.\, U}$$

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ L_{21} & 1 & 0 \\ L_{31} & L_{32} & 1 \end{bmatrix} \begin{bmatrix} U_{11} & U_{12} & U_{13} \\ 0 & U_{22} & U_{23} \\ 0 & 0 & U_{33} \end{bmatrix}$$

Fontes de informação adicionais

HPF:

http://www.epcc.ed.ac.uk/computing/training/document_archive/hpf-slides/hpf-course-slides-1.html
http://www.hp.com/techservers/tutorials3/hpf.html

HPC++:

http://www.extreme.indiana.edu/hpc++/index.html

HPF – ADAPTOR:

http://www.gmd.de/SCAI/lab/adaptor/www/adaptor_home.html

Fontes de informação adicionais

HPF:

http://www.epcc.ed.ac.uk/computing/training/document_archive/hpf-slides/hpf-course-slides-1.html
http://www.hp.com/techservers/tutorials3/hpf.html

HPC++:

http://www.extreme.indiana.edu/hpc++/index.html

HPF – ADAPTOR:

http://www.gmd.de/SCAI/lab/adaptor/www/adaptor_home.html

Fontes de informação adicionais

HPF:

http://www.epcc.ed.ac.uk/computing/training/document_archive/hpf-slides/hpf-course-slides-1.html
http://www.hp.com/techservers/tutorials3/hpf.html

HPC++:

http://www.extreme.indiana.edu/hpc++/index.html

HPF – ADAPTOR:

http://www.gmd.de/SCAI/lab/adaptor/www/adaptor_home.html

Fontes de informação adicionais

HPF:

http://www.epcc.ed.ac.uk/computing/training/document_archive/hpf-slides/hpf-course-slides-1.html
http://www.hp.com/techservers/tutorials3/hpf.html

HPC++:

http://www.extreme.indiana.edu/hpc++/index.html

HPF – ADAPTOR:

http://www.gmd.de/SCAI/lab/adaptor/www/adaptor_home.html

Fontes de informação adicionais

HPF:

http://www.epcc.ed.ac.uk/computing/training/document_archive/hpf-slides/hpf-course-slides-1.html
http://www.hp.com/techservers/tutorials3/hpf.html

HPC++:

http://www.extreme.indiana.edu/hpc++/index.html

HPF – ADAPTOR:

http://www.gmd.de/SCAI/lab/adaptor/www/adaptor_home.html

Fontes de informação adicionais

HPF:

http://www.epcc.ed.ac.uk/computing/training/document_archive/hpf-slides/hpf-course-slides-1.html
http://www.hp.com/techservers/tutorials3/hpf.html

HPC++:

http://www.extreme.indiana.edu/hpc++/index.html

HPF – ADAPTOR:

http://www.gmd.de/SCAI/lab/adaptor/www/adaptor_home.html