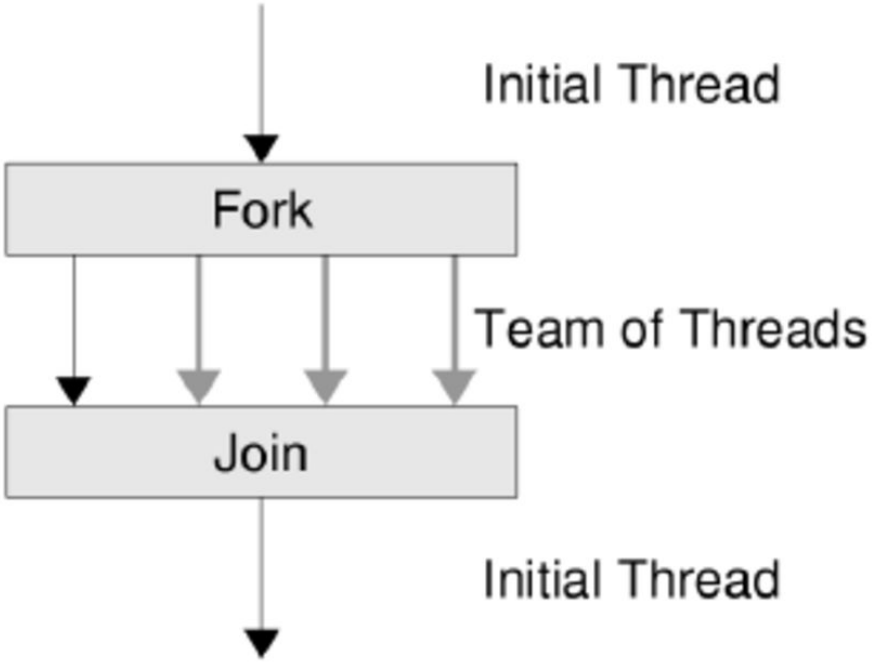
OpenMP

Enabling HPC since 1997

- Padrão
  - Diretivas
  - pequeno conjunto de funções
  - variáveis de ambiente
- suporta C/C++ e Fortran
  - C/C++: **`#pragma`**
- Várias opções de compiladores
  - GCC/GFortran

Modelo *Fork-Join*



Modo de operação do OpenMP

- Uma **região paralela** é a estrutura básica do paralelismo OpenMP
- Inicia a execução com uma única *thread*
  - Denominada ***thread master***
- **Criação automática de *threads*** em regiões paralelas
  - *Overhead* para criação e destruição
  - Importante **minimizar o número de regiões paralelas** abertas
- Existência de **variáveis privadas ou compartilhadas** nessas regiões

Diretivas e Sentinelas

- Uma diretiva é uma linha especial de código fonte com significado especial apenas para determinados compiladores
- Uma diretiva se distingue pela existência de uma sentinela no começo da linha
  - C/C++: **`#pragma omp`**
  - Seguem o padrão de diretivas de compilação para C/C++
  - Cada diretiva se aplica no máximo a próxima instrução, que deve ser um bloco estruturado

Regiões Paralelas

- Um código dentro da região paralela é executado por todas as *threads*
  - Exige incluir a biblioteca: `<omp.h>`

```
#include <omp.h>
...
#pragma omp parallel
   {
   // bloco executado em paralelo
   }
```

Cláusulas

- Especificam informação adicional na diretiva de região paralela:
- Em C/C++:

```
#pragma omp parallel [clausulas]
```

- Cláusulas são separadas por vírgula ou espaço no Fortran, e por espaço no C/C++

Principais cláusulas para atributos de dados em uma região paralela

- **`shared(list)`** - Variáveis compartilhadas - todas as *threads* acessam o mesmo endereço dos dados
- **`private(list)`** - Variáveis privadas - cada *thread* tem a sua própria cópia local
  - Valores são indefinidos na entrada e saída
- **`default(shared|none)`** - Define valores *default*:
  - `Shared` - todas os dados são compartilhados
  - `None` - nada será definido por *default*

OpenMP Team := Master + Workers

- Uma região paralela é um bloco de código executado por todas as *threads* concorrentemente
  - A *thread* Mestre tem `ID = 0`
  - Ao iniciar uma região paralela todos os *threads* estão sincronizados
  - Regiões paralelas podem ser aninhadas, mas esta característica é dependente da implementação
  - Regiões paralelas podem ser condicionadas por "`if`"

Principais funções da API OpenMP

Função	Utilidade
omp_get_num_threads()	Retorna o número de threads em execução
omp_set_num_threads(n)	Define o número de threads (n)
omp_get_thread_num()	Retorna o ID da Thread
omp_get_num_procs()	Retorna o número de processadores do sistema
omp_in_parallel()	Retorna verificação se no dado ponto do programa está-se ou não em uma região paralela
omp_get_wtime()	Retorna o *Wall Clock Time* do sistema
omp_get_wtick()	Retorna o número de *ticks* ou intervalos regularmente espaçados ditados pela freqüência do *clock* entre um segundo, no sistema

Hello World em OpenMP

```
#include <omp.h>
#include <stdio.h>
int main (int argc, char *argv[]) {
  int th_id, nthreads;
#pragma omp parallel private(th_id)
  {
    th_id = omp_get_thread_num();
    printf("Hello World from thread %d\n", th_id);
    if (th_id==0) {
      nthreads = omp_get_num_threads();
      printf("There are %d threads\n", nthreads); }
  }
  return 0; }
```

Definindo a quantidade de *threads* a executar

- Padrão: quantidade de threads = quantidade de processadores
- Dentro do código:
  - `#pragma omp parallel num_threads(N)`
  - `omp_set_num_threads(N)`
- Fora do código com variável de ambiente:
  - `export OMP_NUM_THREADS=N`

Compilação de programas em C/C++ com OpenMP

- Compiladores disponíveis: Intel C/C++ e Fortran-95, GCC e GFORTRAN, G95, PGI, PathScale, Absoft

- Usando GCC:

```
gcc -fopenmp -o <executavel> <fonte.c>
```

https://colab.research.google.com/drive/1wfFs5j_h4ARNZtcHSKiTFwklxUUu2AJX?usp=sharing

Laços Paralelos em OpenMP

- Principal fonte de paralelismo em muitas aplicações
  - Checar se as Iterações são independentes
    - podem ser executadas em qualquer ordem
  - Exemplo: considerando 2 *threads* e o laço a a seguir, pode-se fazer as iterações 0-49 em uma *thread* e as iterações 50-99 na outra

```
for (i = 0; i<100; i++) {
    a[i] = a[i] + b[i]; }
```

Laços Paralelos em OpenMP (cont.)

- `#pragma omp for [clausulas]`

- Sem cláusulas adicionais, a diretiva DO/FOR particionará as iterações o mais igualmente possível entre as *threads*
  - Contudo, isto é dependente de implementação e ainda há alguma ambiguidade:
  - Ex:. 7 iterações, 3 *threads*. Pode ser particionado como 3+3+1 ou 3+2+2

Exemplos de laços paralelos em OpenMP

For-loop with independent iterations

```
for (int i=0; i<n; i++)
  c[i] = a[i] + b[i];
```

For-loop parallelized using an OpenMP pragma

```
#pragma omp parallel for
for (int i=0; i<n; i++)
    c[i] = a[i] + b[i];
```

Thread 0	Thread 1	Thread 2	Thread 3	Thread 4
i=0-199	i=200-399	i=400-599	i=600-799	i=800-999
a[i]	a[i]	a[i]	a[i]	a[i]
+	+	+	+	+
b[i]	b[i]	b[i]	b[i]	b[i]
=	=	=	=	=
c[i]	c[i]	c[i]	c[i]	c[i]

Paralelizando laço em OpenMP

```
/* Laco perfeitamente paralelizavel */
#pragma omp parallel
#pragma omp for
for (i=0; i<n; i++) {
    b[i]= (a[i] - a[i-1)])*0.5; }
/* end parallel for */
```

A diretiva DO/FOR paralela

- Construção comum
- Existe uma forma que combina a região paralela e a diretiva DO/FOR:

```
#pragma omp parallel for [clausulas]
```

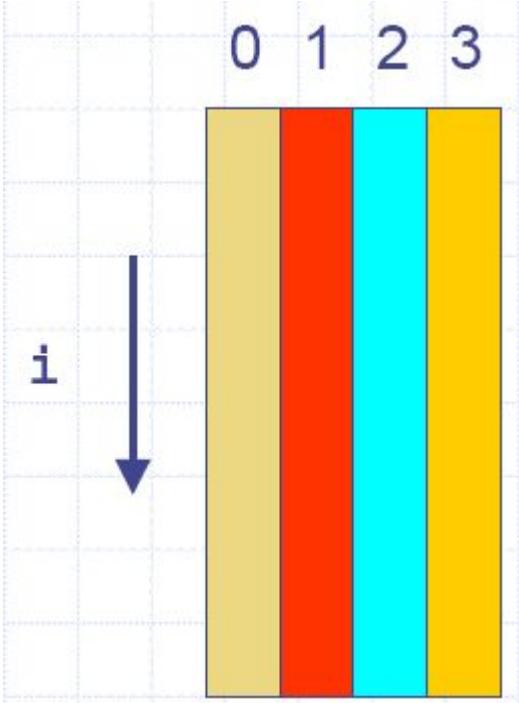
Mais exemplos de uso da diretiva DO/FOR

```
#pragma omp parallel for
for (i=0; i<n; i++) {
    b[i]= (a[i] - a[i-1])])*0.5; }
/* end parallel for */
```

```
#pragma omp parallel for
for (i=0; i < n; i++) {
    z[i] = a * x[i] + y; }
```

Variáveis Privadas e Compartilhadas (exemplo)

```
#pragma omp parallel \
    default(none) \
    private (i, myid) \
    shared(a,n)
{
myid = omp_get_thread_num();
for(i = 0; i < n; i++){
   a[i][myid] = 1.0; }
} /* end parallel */
```



Quais variáveis devem ser compartilhadas e privadas?

- A maioria das variáveis são compartilhadas
  - *Defaults*:
    - O índices dos laços são privados
    - Variáveis temporárias dos laços são compartilhadas
    - Variáveis apenas de leitura são compartilhadas
    - Arrays principais são compartilhadas
    - Escalares do tipo *write-before-read* são usualmente privados
  - A decisão pode ser baseada em fatores de desempenho

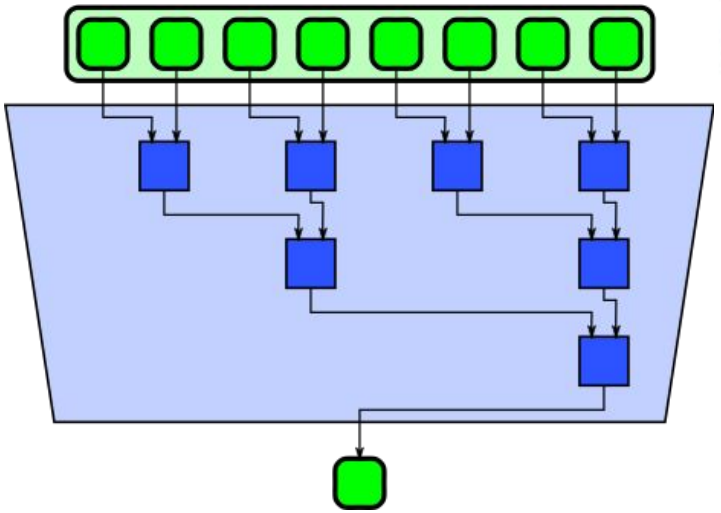
Valor inicial de variáveis privadas

- Variáveis privadas não tem valor inicial no início da região paralela.
- Para levar o valor atual para o valor inicial dentro da região paralela usa-se: **FIRSTPRIVATE**

```
b = 23.0;
. . . . .
#pragma omp parallel firstprivate(b) private(i,myid)
{
  myid = omp_get_thread_num();
  #pragma omp for
  for (i=0; i<n; i++) {
      b += c[myid][i]; }
  c[myid][n] = b;
}
```

Reduções

- combina todos os elementos de uma coleção em um único elemento a partir de uma função combinadora associativa
- Exemplos: somatório, produtório, média, máximo, mínimo, etc.



Reduções em OpenMP

- Uso da cláusula REDUCTION:
  - C/C++: **`reduction(op:list)`**
    - Onde op pode ser:

Operador (op)	Operação	Valor inicial
+	adição	0
-	subtração	0
*	Multiplicação	1
&	E lógico	Todos os bits em 1
|	OU lógico	Todos os bits em 0
^	Equivalente (lógica)	Todos os bits em 0
&&	Não equivalente (lógico)	Todos os bits em 1
||	Máximo	0

Exemplo de redução em OpenMP

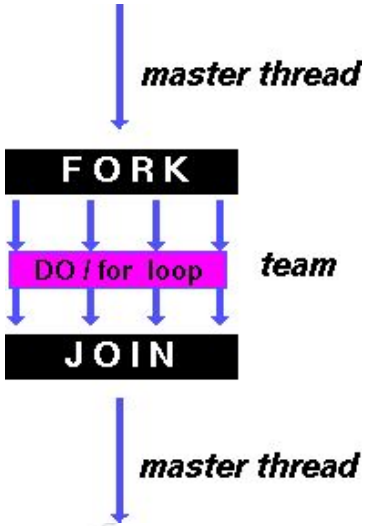
```
soma = 0;
#pragma omp parallel private (i, myid) \
        reduction (+:soma)
{
myid = omp_get_thread_num();
#pragma omp for
for (i = 0; i < n; i++) {
    soma = soma + c[i][myid]; }
}
```

Variáveis de ambiente do SO para OpenMP

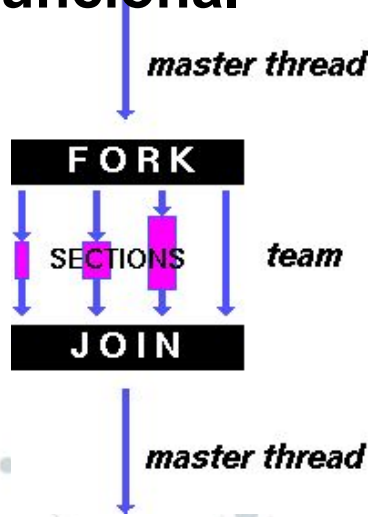
- `OMP_NUM_THREADS` – especifica o número de *threads* para serem usadas durante a execução de regiões paralelas
  - O valor padrão para esta variável é o número de processadores
  - Pode-se definir um número de *threads* independente do número de processadores físicos disponíveis

Tipos de paralelismo
Work-Share

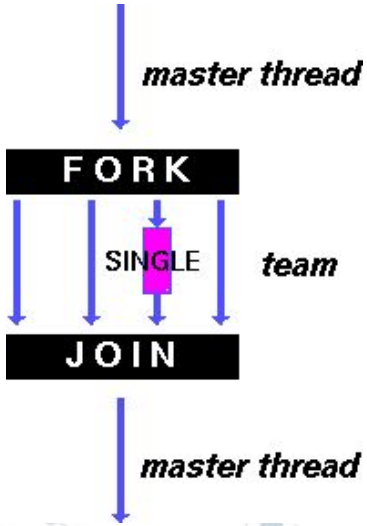
Laços paralelos.
Paralelismo de dados



Seções paralelas: cada *thread* executa uma seção
Paralelismo funcional



Execução serial dentro de uma seção paralela



Seções paralelas (paralelismo funcional)

Cada seção é executada por uma *thread*

```
#pragma omp parallel default(none)\
        shared(n,a,b,c,d) private(i)
  {
    #pragma omp sections nowait
    {
      #pragma omp section
        for (i=0; i<n-1; i++)
            b[i] = (a[i] + a[i+1])/2;

      #pragma omp section
        for (i=0; i<n; i++)
            d[i] = 1.0/c[i];

    } /*-- End of sections --*/

  } /*-- End of parallel region --*/
```