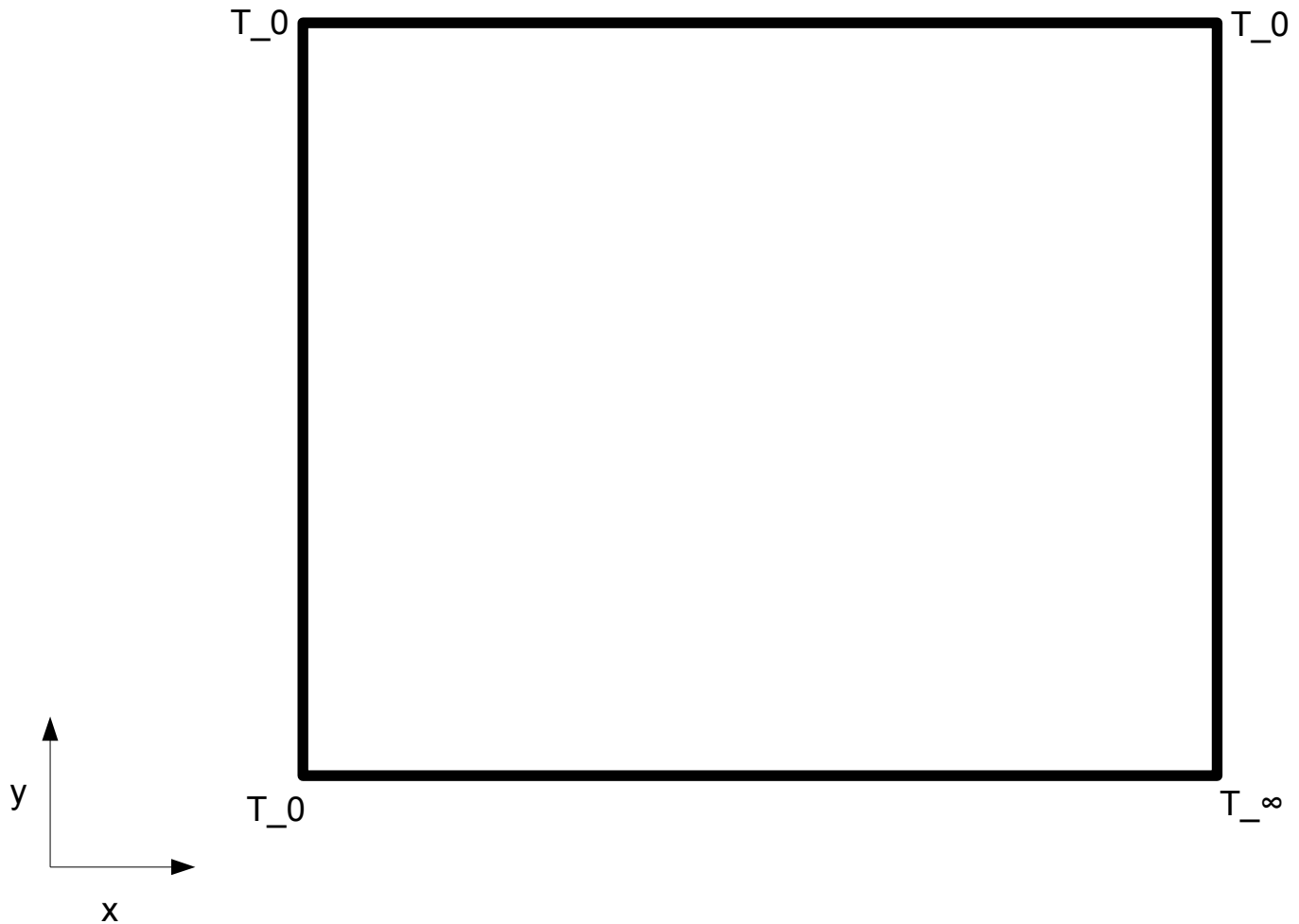
High Performance Scientific computing
Lecture 5

S. Gopalakrishnan

Steady State Heat Conduction

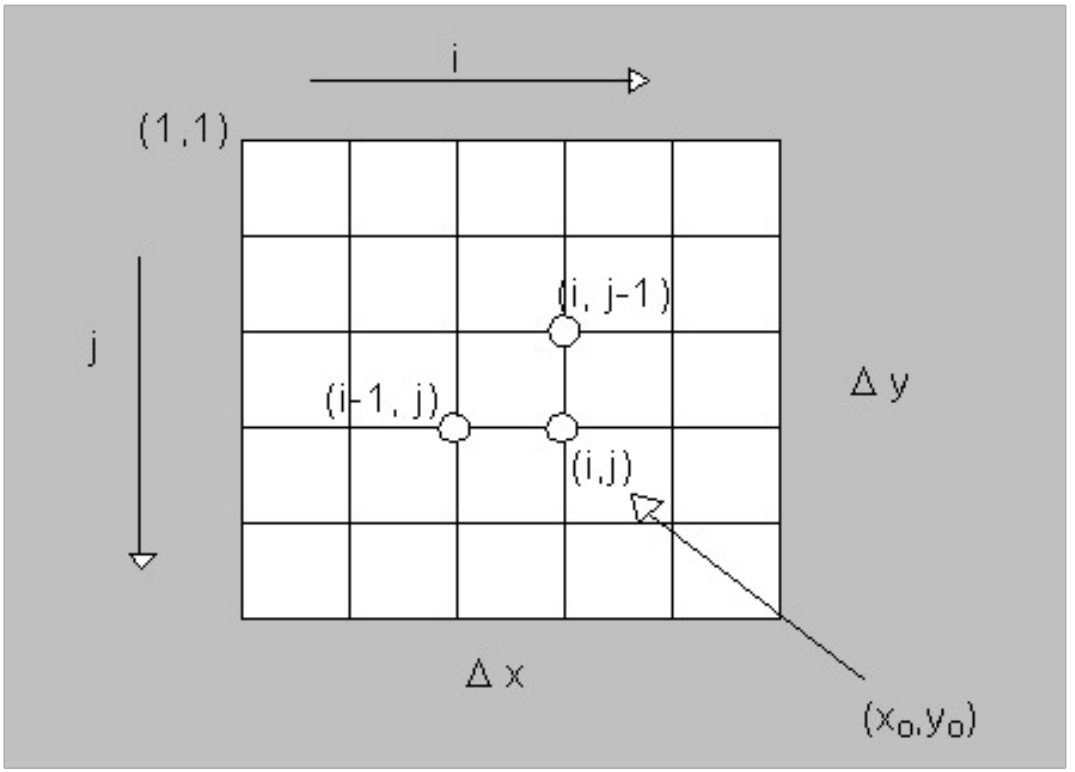


Steady State Heat Conduction

Phenomenon is modelled using Laplace's equation

$$\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2} = \nabla^2 T = 0$$

which can be discretised on a grid as,



Steady State Heat Conduction

The discretised equation at a single point (i,j) is

$$\frac{T_{i-1,j} - T_{i,j} + T_{i+1,j}}{(\Delta x)^2} + \frac{T_{i,j-1} - T_{i,j} + T_{i,j+1}}{(\Delta y)^2} = 0$$

Assemble all the equations for all unknown points in the Matrix form and then solve

$$Ax = B$$

You can choose any Linear Algebra Solver (iterative or Direct). Iterative is more efficient.

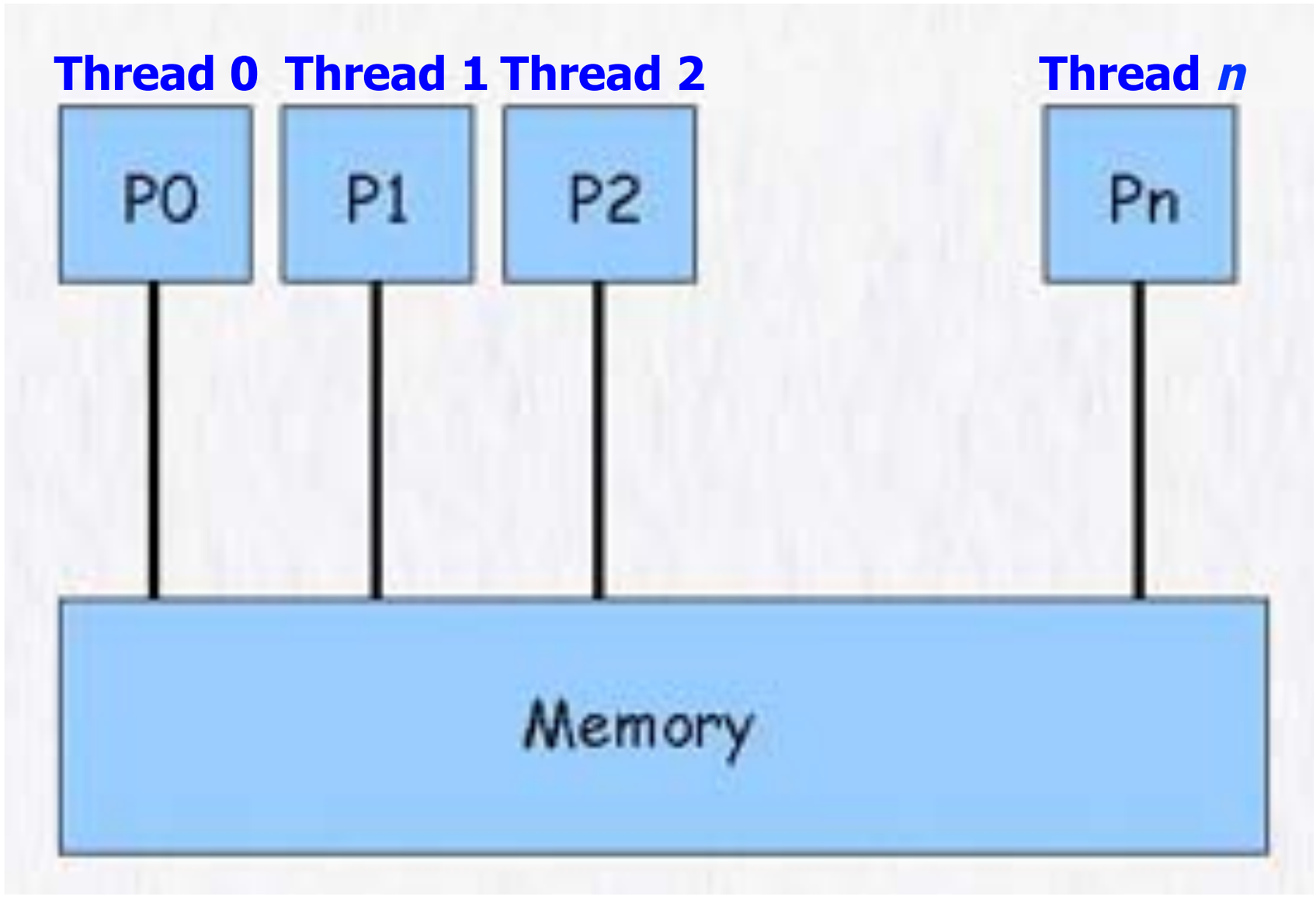
Introduction to OpenMP

Introduction

- OpenMP is designed for shared memory systems.
- OpenMP is easy to use
  - achieve parallelism through compiler directives
  - or the occasional function call
- OpenMP is a “quick and dirty” way of parallelizing a program.
- OpenMP is usually used on existing serial programs to achieve moderate parallelism with relatively little effort

Computational Threads

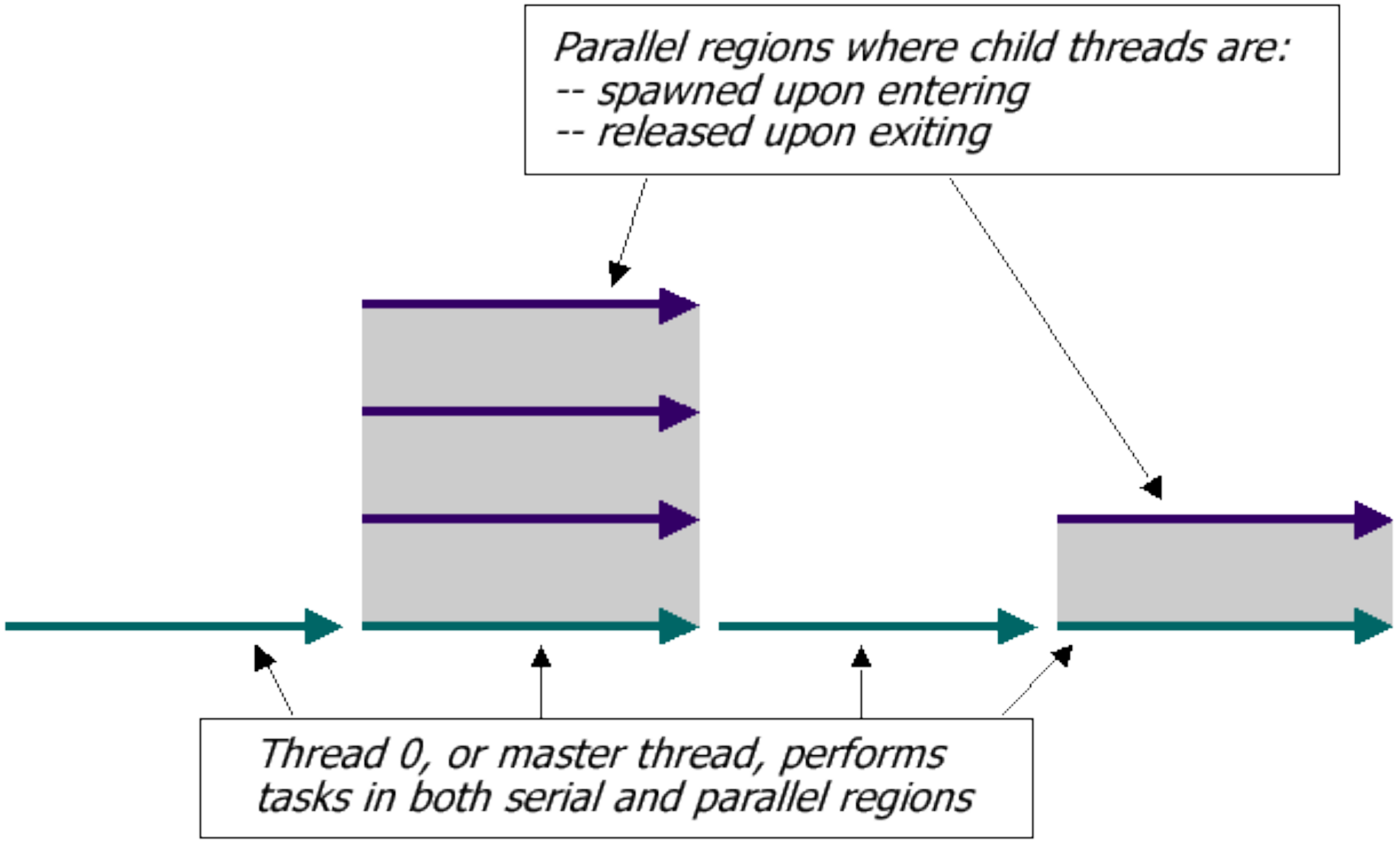
- Each processor has one thread assigned to it
- Each thread runs one copy of your program



OpenMP Execution Model

- In MPI, all threads are active all the time
- In OpenMP, execution begins only on the master thread. Child threads are spawned and released as needed.
  - Threads are spawned when program enters a parallel region.
  - Threads are released when program exits a parallel region

OpenMP Execution Model



Parallel Region Example:
For loop

Fortran:

```
!$omp parallel do
do i = 1, n
 a(i) = b(i) + c(i)
enddo
```

C/C++:

```
#pragma omp parallel for
for(i=1; i<=n; i++)
   a[i] = b[i] + c[i];
```

This comment or pragma tells openmp compiler to spawn threads *and* distribute work among those threads

These actions are combined here but they can be specified separately between the threads

Pros of OpenMP

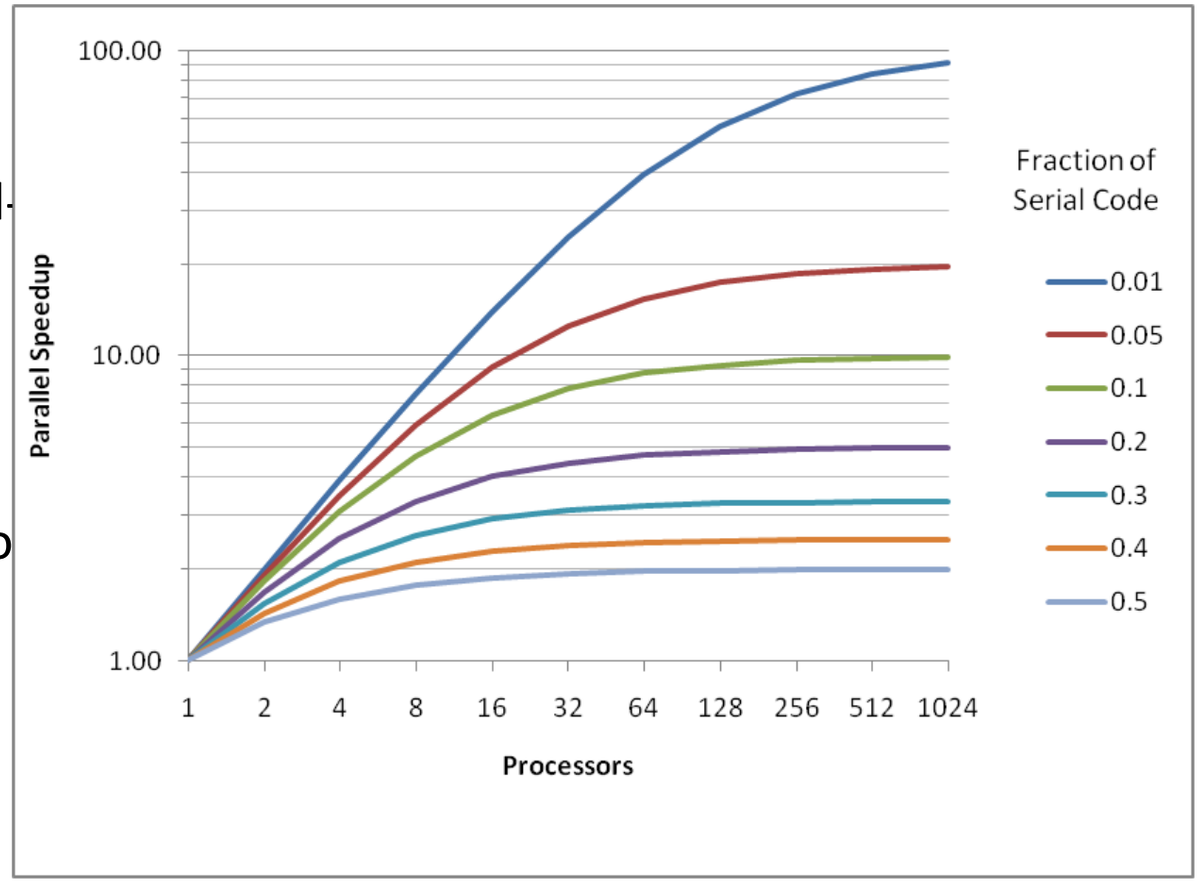
- Because it takes advantage of shared memory, the programmer does not need to worry (that much) about data placement
- Programming model is “serial-like” and thus conceptually simpler than message passing
- Compiler directives are generally simple and easy to use
- Legacy serial code does not need to be rewritten

Cons of OpenMP

- Codes can only be run in shared memory environments!
  - In general, shared memory machines beyond ~8 CPUs are much more expensive than distributed memory ones, so finding a shared memory system to run on may be difficult
- Compiler must support OpenMP
  - whereas MPI can be installed anywhere
  - However, gcc 4.2 now supports OpenMP

Cons of OpenMP

- In general, only moderate speedups can be achieved.
  - Because OpenMP codes tend to have serial-only portions, Amdahl's Law prohibits substantial speedups



- Amdahl's Law:

  F = Fraction of serial execution time that cannot be parallelized

  N = Number of processors

$$\text{Execution time} = \frac{1}{F + (1 - F)/N}$$

If you have big loops that dominate execution time, these are ideal targets for OpenMP

Compiling and Running OpenMP

- True64: **-mp**
- SGI IRIX: **-mp**
- IBM AIX: **-qsmp=omp**
- Portland Group: **-mp**
- Intel: **-openmp**
- gcc (4.2) **-fopenmp**

Compiling and Running OpenMP

- OMP_NUM_THREADS environment variable sets the number of processors the OpenMP program will have at its disposal.
- Example script

```
#!/bin/tcsh
setenv OMP_NUM_THREADS 4
mycode < my.in > my.out
```

Sections: Functional parallelism

```
#pragma omp parallel
{
    #pragma omp sections
    {
     #pragma omp section
          block1
     #pragma omp section
          block2
    }
}
```

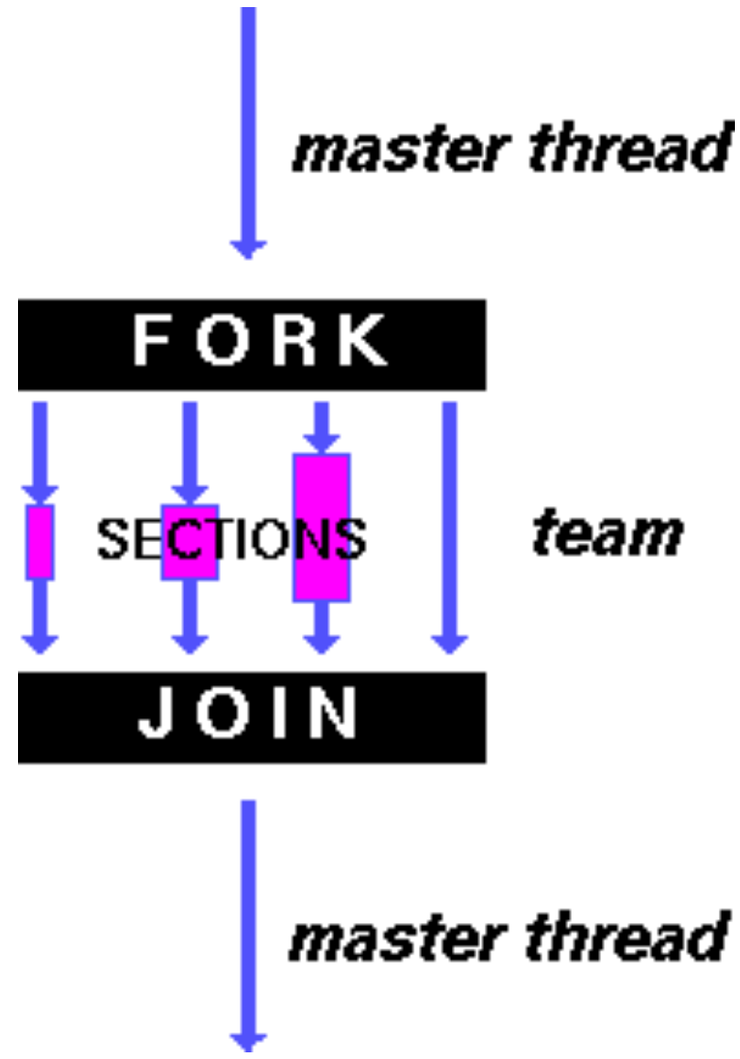


Image from: https://computing.llnl.gov/tutorials/openMP

Parallel DO/for:
Loop level parallelism

Fortran:

```
!$omp parallel do
do i = 1, n
 a(i) = b(i) + c(i)
enddo
```

C/C++:

```
#pragma omp parallel for
for(i=1; i<=n; i++)
   a[i] = b[i] + c[i];
```

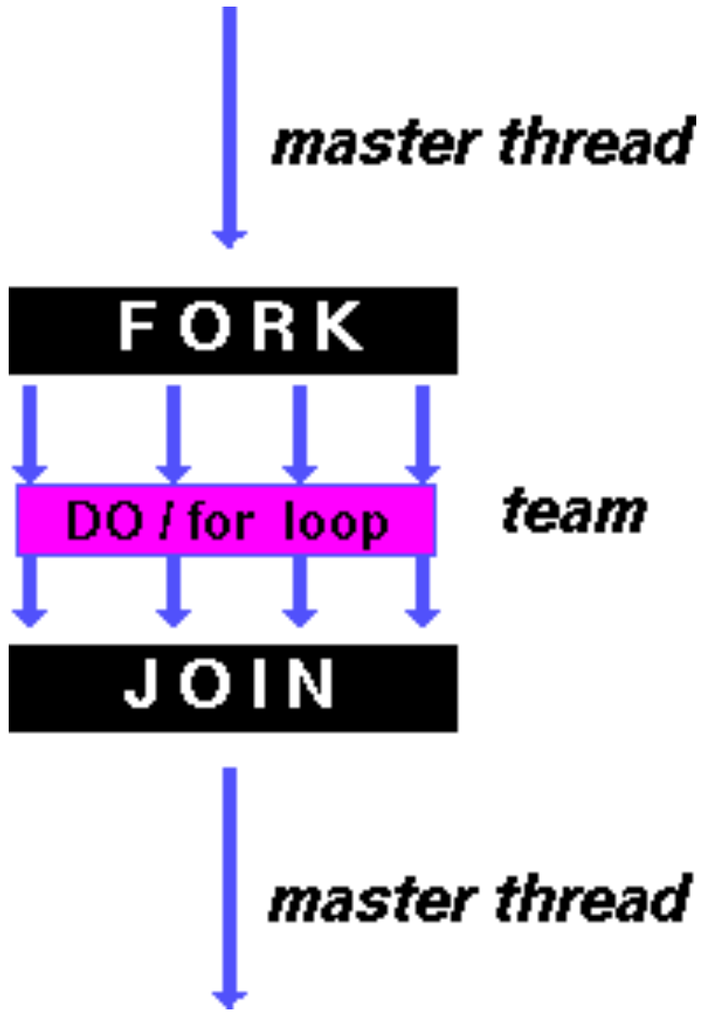


Image from: https://computing.llnl.gov/tutorials/openMP