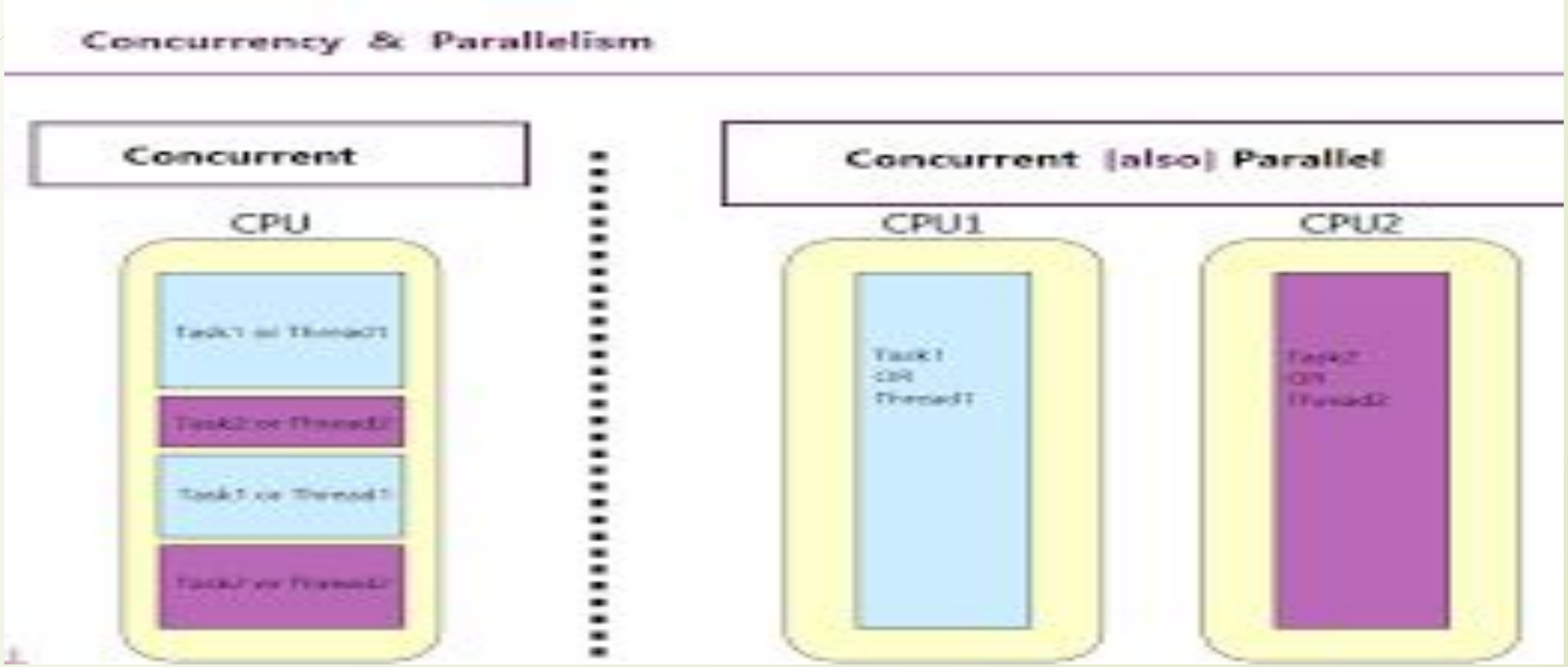
Parallel Programming

TY IT 23-24

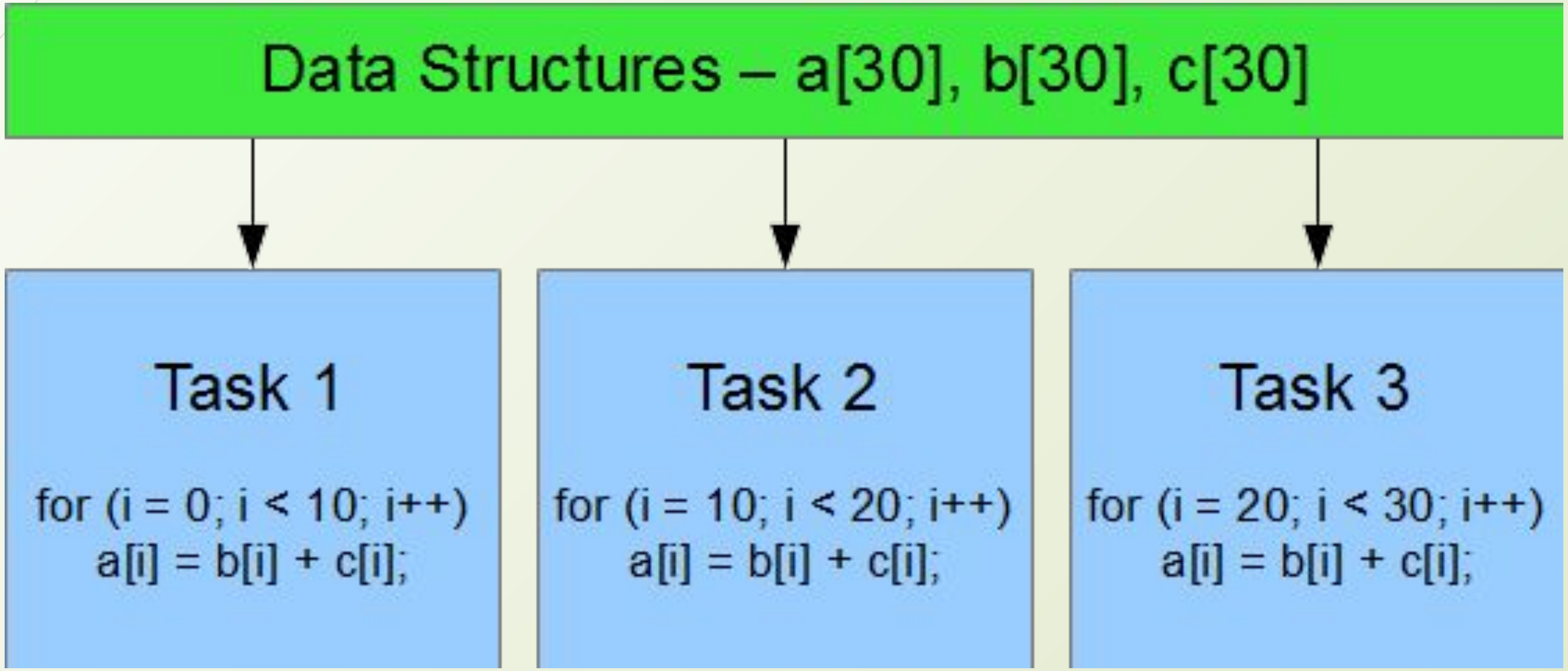
D B Kulkarni

Walchand College of Engineering, Sangli

Concurrency and Parallelism

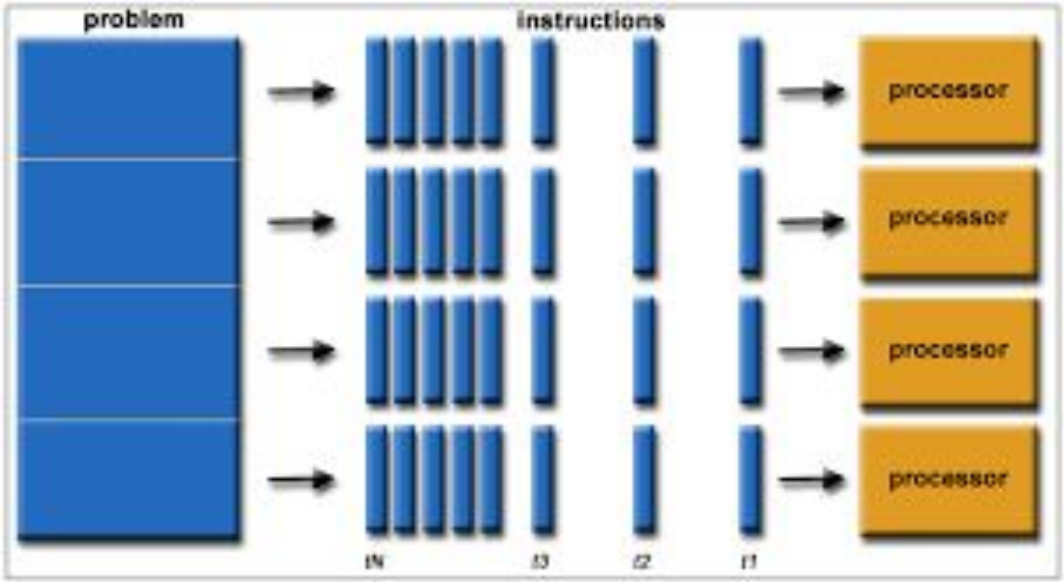


Data Parallel



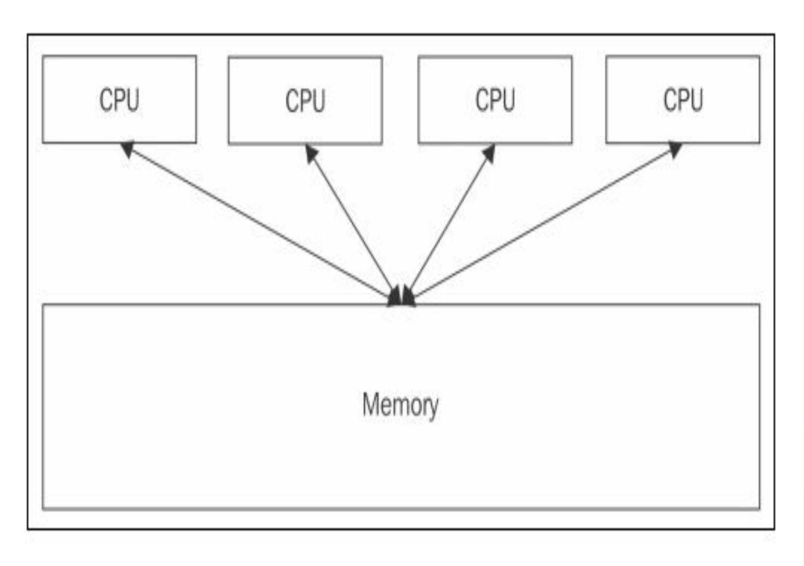
- Data parallel
  - distributing the data across different nodes, which operate on the data in parallel

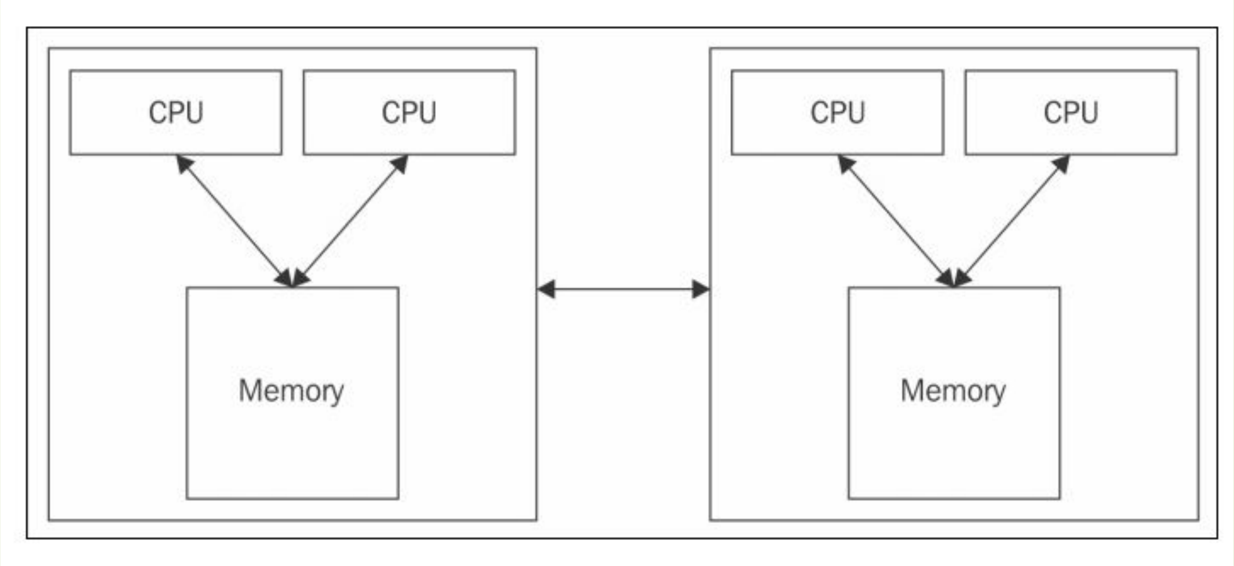
Task Parallel



- ➢ Data parallel
  - distributing the data across different nodes, which operate on the data in parallel

Parallel system





Shared Memory Model

Distributed Memory Model

Shared memory models

The shared-memory programming models are used to develop solutions for machine architectures that share one common memory space across a set of processors. The models include:

- the *pure* shared memory model
- the multi-threading models
  - the programmer-controlled model
  - the API controlled model

Pure Shared memory models

The pure shared memory programming model identifies data independently of all processors. This model does not associate data with any particular processor. It manages access to shared memory through a system of locks and semaphores.

Advantages:

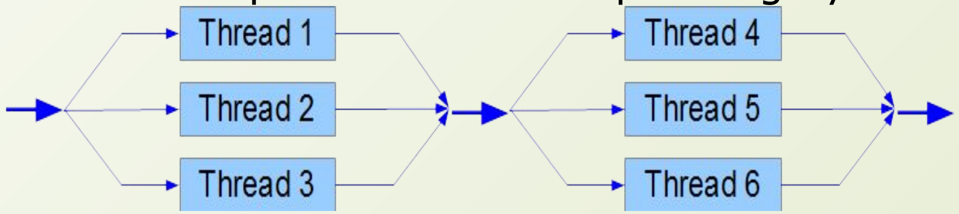
- no concept of data ownership
- program development is simple

Drawbacks:

- difficult to manage data locality
- programming instructions are low-level

Multithreaded Shared memory models

The multi-threading models divide a part of a process into threads. A thread is an independent stream of instructions that the operating system can schedule independently on the processors. A thread exists within a process and uses that process' resources. Thread-creation requires much less operating-system overhead than process-creation.



The multi-threading models associate each thread with some local data. Each thread can execute concurrently with the other threads. Each thread communicates with other threads through shared memory.

The multi-threading models require synchronization to ensure that concurrently executing threads are not updating the same memory address.

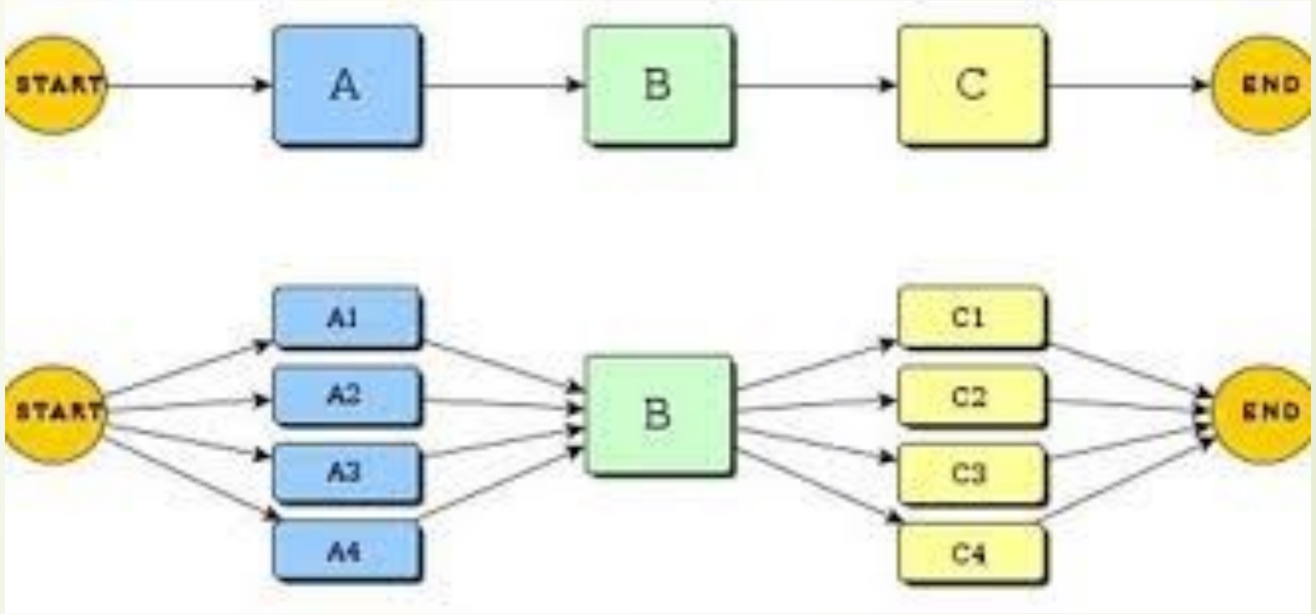
Advantages:

- programmer manages data locality
- run-time system schedules threads

Drawback:

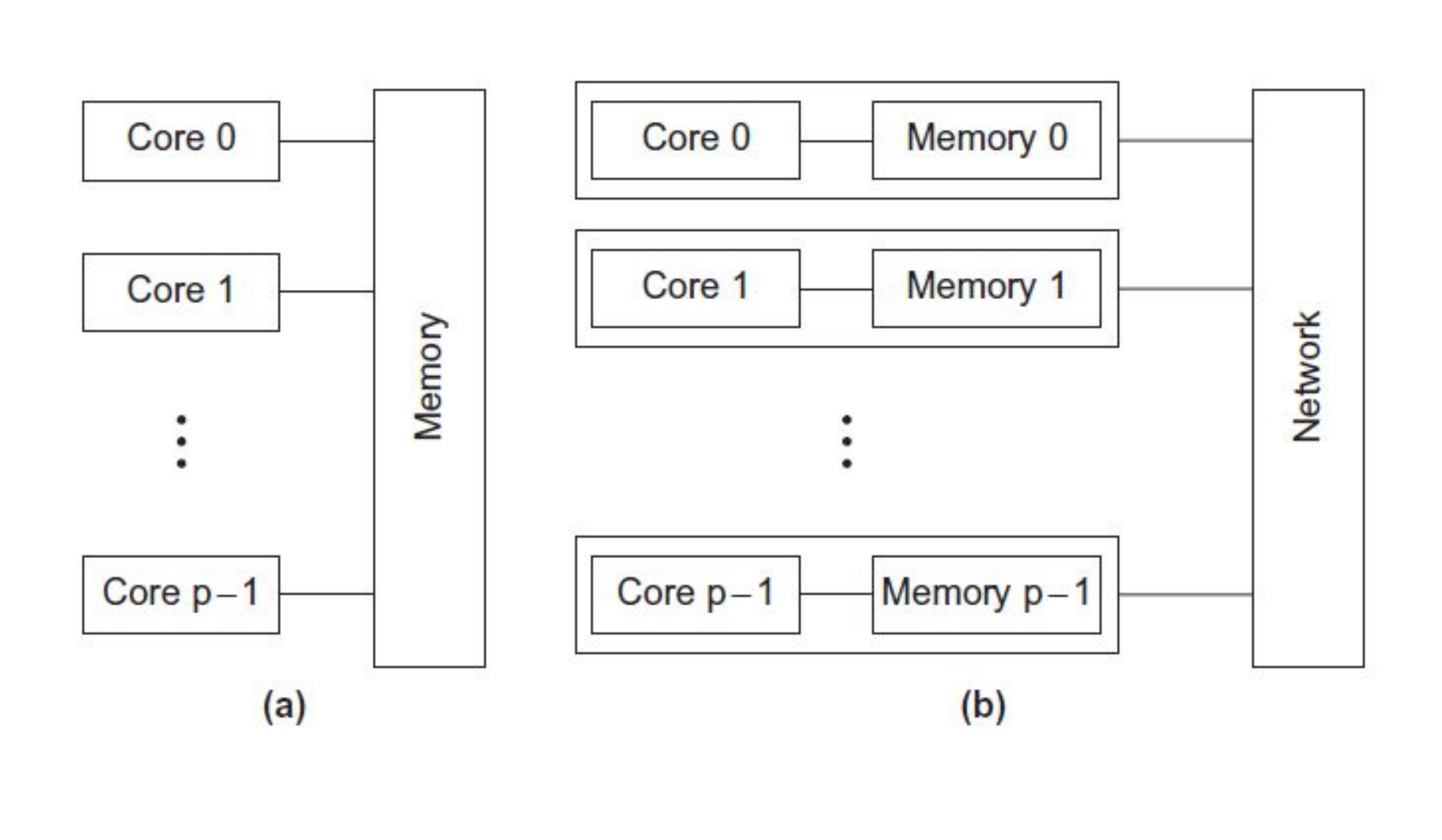
- programmer is responsible for determining the parallelism

Shared Memory Model: OpenMP



- Open Multi Processing (OpenMP)
- Data parallel
  - distributing the data across different nodes, which operate on the data in parallel

Shared vs Distributed Memory Model



Parallel Paradigm: OpenMP

OpenMP is:

- An Application Program Interface (API) that may be used to explicitly direct multi-threaded, shared parallelism
- Comprised of 3 primary components
  - Compiler directives
  - Runtime library routines
  - Environmental variables
- Use flag -fopenmp to compile using GCC:

  gcc -fopenmp hello.c –o hello

- Reference: http://www.openmp.org

OpenMP Example

```
include <stdio.h>
#include <omp.h>
int main(int argc, char *argv[]) {
#pragma omp parallel
 {
 int thread_id = omp_get_thread_num();
 int num_threads = omp_get_num_threads();
 printf("Hello World from thread %d of %d\n", thread_id,
num_threads);
 }
 return(0);
}
$ export OMP_NUM_THREADS=4
$ gcc -fopenmp hello.c
$ ./a.out
Hello World from thread 1 of 4
Hello World from thread 0 of 4
Hello World from thread 3 of 4
Hello World from thread 2 of 4
```

OpenMP Example: Modified

```
int main (int argc, char *argv[])
{  int nthreads, tid;
#pragma omp parallel private(nthreads, tid)
  {  /* Obtain thread number */
  tid = omp_get_thread_num();
  printf("Hello World from thread = %d\n", tid);
  if (tid == 0)
    {    nthreads = omp_get_num_threads();
         printf("Number of threads = %d\n",
nthreads);
 } } }
```

```
$ export OMP_NUM_THREADS=8
$ gcc -fopenmp hello.c
$ ./a.out
Hello World from thread = 0
Hello World from thread = 3
Hello World from thread = 2
Number of threads = 8
Hello World from thread = 6
Hello World from thread = 1
Hello World from thread = 4
Hello World from thread = 7
Hello World from thread = 5
```

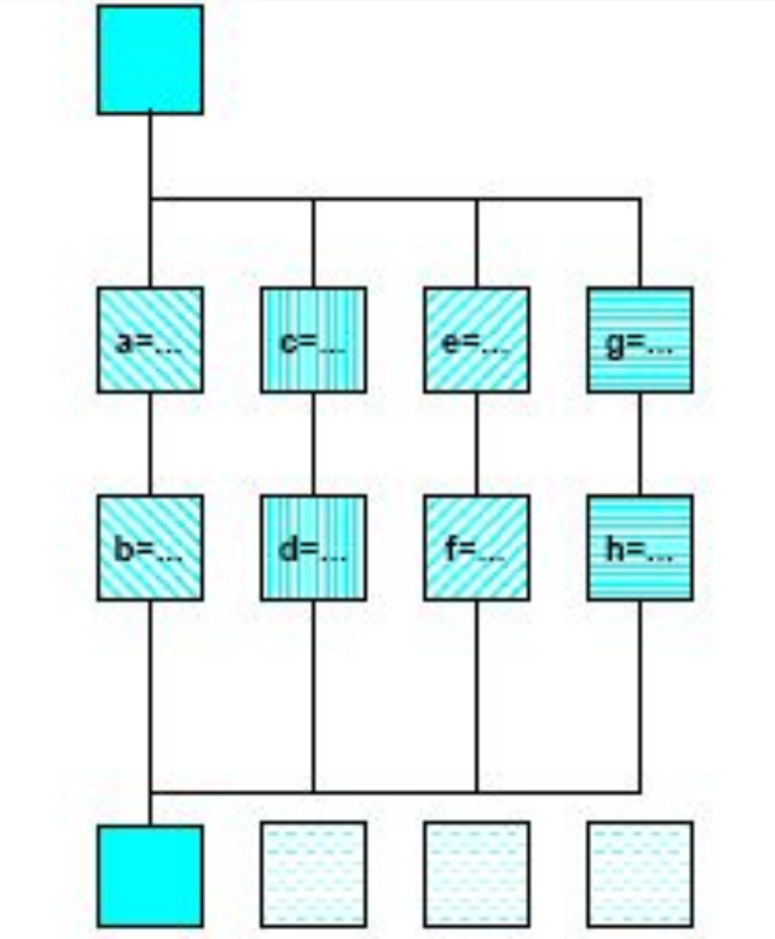
OpenMP Example: Modified

```
#define CHUNKSIZE   10
#define N       100
int main (int argc, char *argv[])
{ int nthreads, tid, i, chunk;
  float a[N], b[N], c[N];
       for (i=0; i < N; i++)
            a[i] = b[i] = i * 1.0;
  chunk = CHUNKSIZE;
#pragma omp parallel
shared(a,b,c,nthreads,chunk) private(i,tid)
  {   tid = omp_get_thread_num();
  if (tid == 0)
    {  nthreads = omp_get_num_threads();
       printf("Number of threads = %d\n", nthreads); }
  printf("Thread %d starting...\n",tid);
  #pragma omp for schedule(dynamic,chunk)
  for (i=0; i<N; i++)
    {  c[i] = a[i] + b[i];
    printf("Thread %d: c[%d]= %f\n",tid,i,c[i]);
    }}}
```

```
$ export OMP_NUM_THREADS=8
$ gcc -fopenmp hello.c
$ ??
```

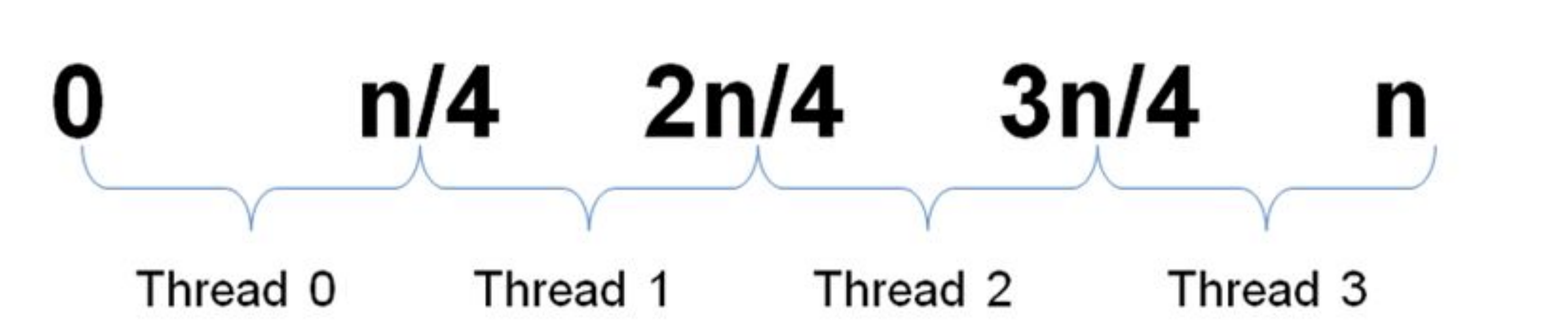
OpenMP sections directive

```
#pragma omp parallel
{
#pragma omp sections
        {{ a=...;
        b=...; }
#pragma omp section
        { c=...;
        d=...; }
#pragma omp section
        { e=...;
        f=...; }
#pragma omp section
        { g=...;
        h=...; }
} /*omp end sections*/
} /*omp end parallel*/
```



Parallel For loop

```
omp_set_num_threads(4);
#pragma omp parallel for
for(i=0;i<n;i++)
{
    ...
    ...
```



Omp parallel for loop

Clause can be one of the following:

private(list)

reduction(operator: list)

nowait

…

Implicit barrier at the end of for unless nowait is specified

If nowait is specified, threads do not synchronize at the end of the parallel loop

Storage attributes

- One can selectively change storage attribute constructs using the following clauses which apply to the lexical extent of the OpenMP construct
  - Shared
  - Private
  - Firstprivate
  - Lastprivate
- The value of a private inside a parallel loop can be transmitted to a global value outside the loop with a “lastprivate”
- The default status can be modified with:

  DEFAULT (PRIVATE | SHARED | NONE)

```
#include <omp.h>
#include <stdio.h>
#include <stdlib.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);
    #pragma omp barrier
    if ( th_id == 0 ) {
      nthreads = omp_get_num_threads();
      printf("There are %d threads\n",nthreads);
    }
  }
  return EXIT_SUCCESS;
}
```

OMP Critical and single

Critical

```
omp_set_num_threads(4);
#pragma omp parallel for
for(i=0;i<n;i++)
{
	…
	 #pragma omp critical
	{
		…
	}
	…
}
```

Single

```
#pragma omp parallel
{
	…
	….
	#pragma omp single
	{
		…
	}
	…
}
```

No wait and reduction

No-wait

- At the end of every loop is an implied barrier.
- Use “nowait” to remove the barrier at the end of the first loop:
- #pragma omp parallel

```
{
	#pragma omp for nowait
	for(i=0; i<maxi; i++)
		a[i] = b[i];
	#pragma omp for
	for(j=0; j<maxj; j++)
		c[j] = d[j];
}
```

Reductions

- for(i=0; i<n; i++)

  sum = sum + a[i];
- #pragma omp parallel for reduction(+:sum)

```
{
	for(i=0; i<n; i++)
		sum = sum + a[i];
}
```

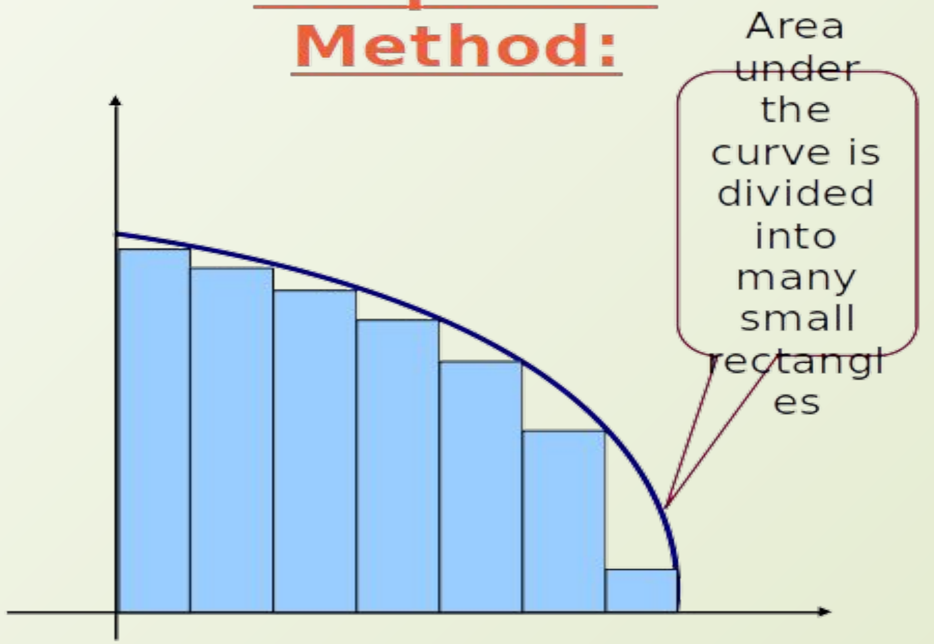
Reduction operators

SN	Operation	Operator	Initialisation
1	Addition	+	0
2	Multiplication	*	1
3	AND	&	0
4	OR	|	0
5	Bitwise AND	&&	1
6	Bitwise OR	||	0
		max, min, iand, ior, ieor. Tasking Clause	

Example: PI

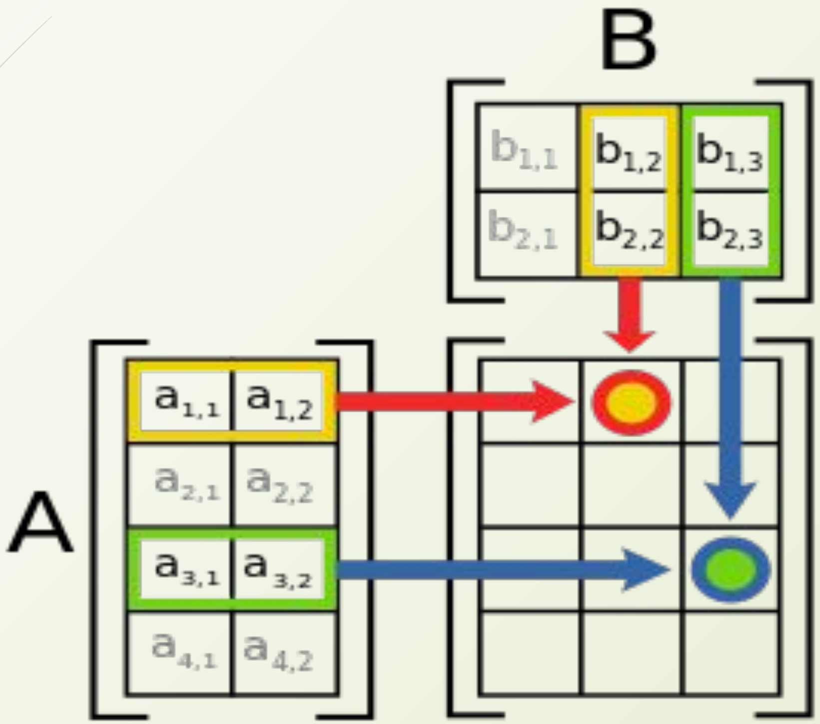
Example - PI

Simpson's Method:



Area under the curve (Approx.) = Sum of the areas of all rectangles

Matrix Multiplication



- C=A*B (Compatibility: Rows(A)=Columns(B)

Summary: Parallel Terminology

- Concurrent computing – a program is one in which multiple tasks can be in progress at any instant.
- Parallel computing – a program is one in which multiple tasks cooperate closely to solve a problem
- Distributed computing – a program may need to cooperate with other programs to solve a problem.