FARABI UNIVERSITY
DEPARTMENT OF COMPUTER SCIENCE



Lecture 10.
Hybrid Parallel Computing

Dr. Bazargul Matkerim, Professor-Researcher

Outline

1. **OpenMP 3.0 tasks**

   task directive, taskwait, Fibonacci

2. **Hybrid MPI/OpenMP**

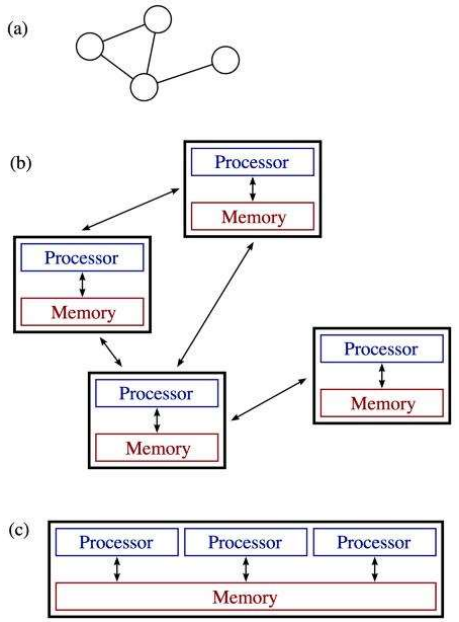
   vector mode, task mode, three communication concepts

3. **Examples**

   conjugate gradient, steady-state heat distribution

4. **Thread support in MPI**

   MPI_Init_thread, four levels



Source: Distributed vs parallel, commons.wikimedia.org (CC BY-SA 3.0)

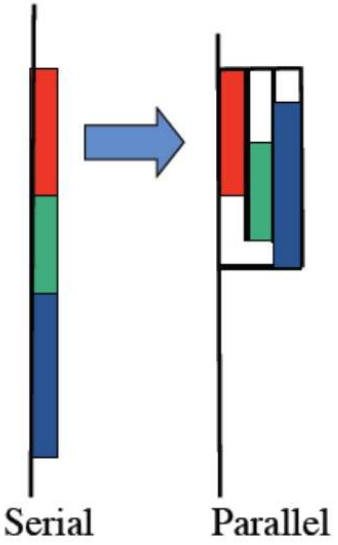
OpenMP 3.0 and Tasks

Tasks: independent units of work
- may be deferred or executed immediately
- the runtime system decides

Idea: a task queue

Irregular problems:
- unbounded loops
- recursive algorithms
- manager/worker schemes

A task has **code**, **data** and an assigned thread



Source: Lecture 15 material “Hybrid parallel computing”, slide 119

Task Directive and Synchronization

```
#pragma omp task [clauses]
   if (logical expression)
   untied
   shared (list)   private (list)
   firstprivate (list)
   default (shared | none)
 structured block
```

Each encountering thread **creates a task**

Tasks can be nested

Barrier: all tasks are completed

#pragma omp taskwait: wait for child tasks

Example: traverse a list

```
void traverse_list(List *L) {
  Element *e;
  for (e = L->first; e != NULL; e = e->next)
    #pragma omp task
      do_work(e);
  #pragma omp taskwait
  /* all tasks are completed here */
}
```

Source: Lecture 15 material, slides 122 and 125

Example: Fibonacci with Tasks

Fibonacci series

$$f(1) = f(2) = 1,$$
$$f(n) = f(n-1) + f(n-2)$$

One thread starts in omp single

Each call creates **two tasks**

taskwait before i + j

```
static int fib(int n) {
  int i, j;
  if (n < 2) return n;
  #pragma omp task shared(i)
    i = fib(n-1);
  #pragma omp task shared(j)
    j = fib(n-2);
  #pragma omp taskwait
  return i + j;
}
```

Based on Lecture 15 material, slide 123

Hybrid MPI/OpenMP

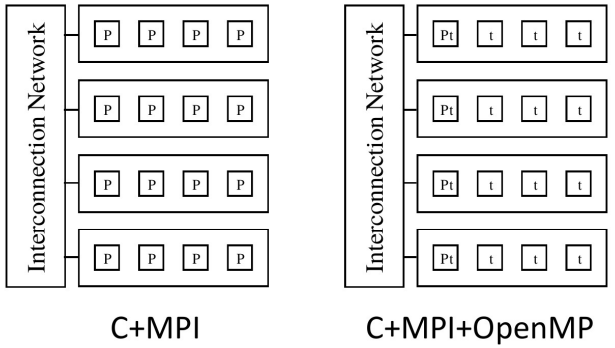
Vector mode: MPI only outside parallel regions

Task mode: some threads do special tasks
e.g. communication in the background

C+MPI: one **process** per core

C+MPI+OpenMP: one process per node, **threads** inside

Compile: mpicc -fopenmp test.c



Source: Lecture 15 material “Hybrid parallel computing”, slide 134

Communication Concepts

Concept 1: root MPI process controls communication
one MPI process per SMP node, fixed number of threads

Concept 2: master OpenMP thread of each process communicates

Concept 3: all OpenMP threads may use MPI calls
most flexible, needs <my_rank, omp_thread_id>
critical sections give some control

```
#pragma omp master
{
  if (0 == my_rank)
    /* MPI call as root process */
  else
    /* MPI call as non-root process */
}
```

Concept 1, Lecture 15 material, slide 136

```
#pragma omp critical
{
  /* MPI call as an MPI process */
}
```

Concept 3, Lecture 15 material, slide 140

Example: Conjugate Gradient

Start with an MPI program

OpenMP for in the **matrix-vector product**

Vector pieces: MPI_Allgatherv

Steps:

1. gradient g = A x − b
2. direction vector d
3. step size s
4. new approximation x = x + s d

Direction vector

$$d^{(t)} = -g^{(t)} + \frac{g^{(t)T} g^{(t)}}{g^{(t-1)T} g^{(t-1)}} d^{(t-1)}$$

```
#pragma omp parallel for private(i,j,tmp)
for (i = 0; i < BLOCK_SIZE(id,p,n); i++) {
  tmp = 0.0;
  for (j = 0; j < n; j++)
    tmp += a[i][j] * b[j];
  piece[i] = tmp;
}
new_replicate_block_vector(id, p, piece,
                           n, c, MPI_DOUBLE);
```

Source: Lecture 15 material, slide 145

Example: Steady-State Heat Distribution

$$u_{xx} + u_{yy} = f(x,y), \qquad 0 \le x \le a, \quad 0 \le y \le b$$

Row decomposition

Boundary rows: MPI_Send/Recv

Update: #pragma omp for

Global difference: MPI_Allreduce with MPI_MAX

Jacobi update

$$w_{i,j} = \frac{u_{i-1,j} + u_{i+1,j} + u_{i,j-1} + u_{i,j+1}}{4}$$

```
#pragma omp parallel private(i,j,tdiff)
{
  tdiff = 0.0;
  #pragma omp for
  for (i = 1; i < my_rows-1; i++)
    for (j = 1; j < N-1; j++) {
      w[i][j] = (u[i-1][j] + u[i+1][j]
                + u[i][j-1] + u[i][j+1]) / 4.0;
      ...
    }
  #pragma omp critical
  if (tdiff > diff) diff = tdiff;
}
MPI_Allreduce(&diff, &global_diff, 1,
   MPI_DOUBLE, MPI_MAX, MPI_COMM_WORLD);
```

Source: Lecture 15 material, slide 147

Thread Support Levels in MPI

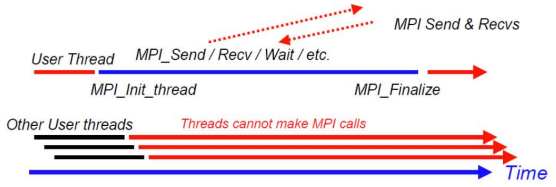
MPI_THREAD_SINGLE: no multithreading

MPI_THREAD_FUNNELED: only the **master** thread calls MPI

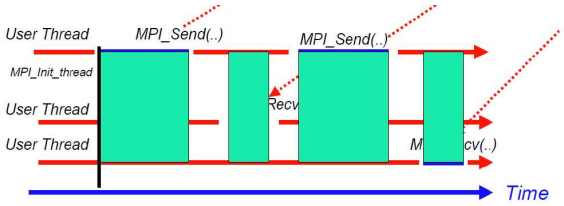
MPI_THREAD_SERIALIZED: several threads, **one at a time**

MPI_THREAD_MULTIPLE: **any thread**, any time

Check: MPI_Is_thread_main



Source: Lecture 15 material “Hybrid parallel computing”, slide 148



Source: Lecture 15 material “Hybrid parallel computing”, slide 149

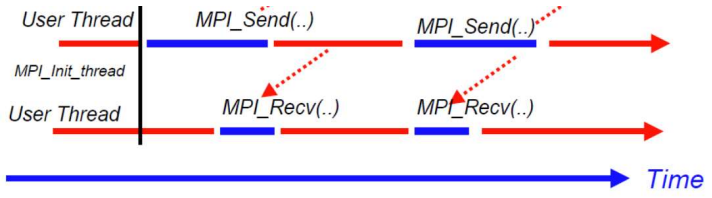
Threaded MPI Initialization

```
int MPI_Init_thread(int *argc, char ***argv,
                    int required, int *provided)
```

required: desired level

provided: level given by the system

FUNNELED or higher: **overlap** of communication and computation



Source: Lecture 15 material “Hybrid parallel computing”, slide 149

```
MPI_Init_thread(&argc, &argv,
    MPI_THREAD_MULTIPLE, &mpisupport);
MPI_Comm_rank(MPI_COMM_WORLD, &rank);
#pragma omp parallel private(omp_rank)
{
  omp_rank = omp_get_thread_num();
  printf("Process %d, thread %d\n",
         rank, omp_rank);
}
MPI_Finalize();
```

Source: Lecture 15 material, slide 151