

Introduction to Parallel Programming with MPI

PICASso Tutorial
February 8-10, 2005

Stéphane Ethier
(ethier@pppl.gov)

Computational Plasma Physics Group
Princeton Plasma Physics Lab

Why Parallel Computing?

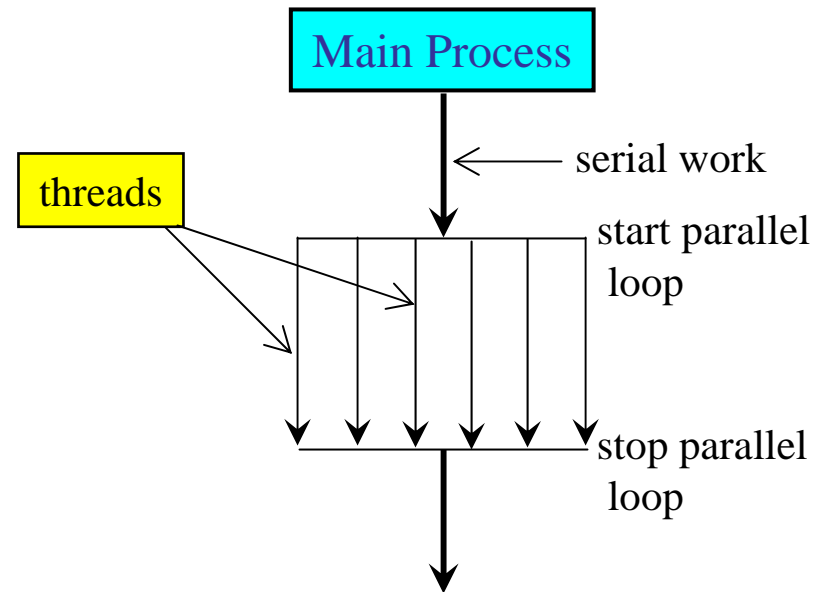
- Want to speed up a calculation.
- Solution:
 - Split the work between several processors.
- How?
 - It depends on the type of parallel computer
 - Shared memory (usually thread-based)
 - Distributed memory (process-based)
 - MPI works on all of them!

Shared memory parallelism

- Program runs inside a single process
- Several “execution threads” are created within that process and work is split between them.
- The threads run on different processors.
- All threads have access to the shared data through shared memory access.
- Must be careful not to have threads overwrite each other’s data.

Shared memory programming

- Easy to do loop-level parallelism.
- Compiler-based automatic parallelization
 - Easy but not always efficient
 - Better to do it yourself with OpenMP
- Coarse-grain parallelism can be difficult



Distributed memory parallelism

- Process-based programming.
- Each process has its own memory space that cannot be accessed by the other processes.
- The work is split between several processes.
- For efficiency, each processor runs a single process.
- Communication between the processes must be explicit, e.g. Message Passing

Most widely used method on distributed memory machines

- Run the same program on all processors.
- Each processor works on a subset of the problem.
- Exchange data when needed
 - Can be exchange through the network interconnect
 - Or through the shared memory on SMP machines
- Easy to do coarse grain parallelism = scalable

How to split the work between processors?

- Most widely used method for grid-based calculations:
 - **DOMAIN DECOMPOSITION**
- Split particles in particle-in-cell (PIC) or molecular dynamics codes.
- Split arrays in PDE solvers
- etc...
- Keep it LOCAL

What is MPI?

- MPI stands for Message Passing Interface.
- It is a message-passing specification, a standard, for the vendors to implement.
- In practice, MPI is a set of functions (C) and subroutines (Fortran) used for exchanging data between processes.
- An MPI library exists on most, if not all, parallel computing platforms so it is highly portable.

How much do I need to know?

- MPI is small (6 functions)
 - Many parallel programs can be written with just 6 basic functions.
- MPI is large (125 functions)
 - MPI's extensive functionality requires many functions
 - Number of functions not necessarily a measure of complexity
- MPI is just right
 - One can access flexibility when it is required.
 - One need not master all parts of MPI to use it.

Good MPI web sites

- <http://www.llnl.gov/computing/tutorials/mpi/>
- <http://www.nersc.gov/nusers/help/tutorials/mpi/intro/>
- <http://www-unix.mcs.anl.gov/mpi/tutorial/gropp/talk.html>
- <http://www-unix.mcs.anl.gov/mpi/tutorial/>

- MPI on Linux clusters:
 - MPICH (<http://www-unix.mcs.anl.gov/mpi/mpich/>)
 - LAM (<http://www.lam-mpi.org/>)

Batch System: PBS primer

- Submit a job script: `qsub script`
- Check status of jobs: `qstat -a` (for all jobs)
- Stop a job: `qdel job_id`

```
### --- PBS SCRIPT ---  
#PBS -l nodes=4:ppn=2,walltime=02:00:00  
#PBS -q default  
#PBS -V  
#PBS -N job_name  
#PBS -m abe  
cd $PBS_O_WORKDIR  
mpiexec a.out  
#mpirun -np 8 a.out
```

mpirun and mpiexec

- Both are used for starting an MPI job
- If you don't have a batch system, use [mpirun](#)

```
   mpirun -np #proc -machinefile mfile a.out >& out < in &
```

```
%cat mfile
```

```
machine1.princeton.edu
```

```
machine2.princeton.edu
```

```
machine3.princeton.edu
```

```
machine4.princeton.edu
```

- PBS takes care of arguments to mpiexec

Compilation

- mpich provides scripts that take care of the include directories and linking libraries
 - mpicc
 - mpiCC
 - mpif77
 - mpif90
- Otherwise, must link with the right MPI library

Makefile

- Always a good idea to have a Makefile

```
%cat Makefile
```

```
CC=mpicc
```

```
CFLAGS=-O
```

```
% : %.c
```

```
$(CC) $(CFLAGS) $< -o $@
```