CP331: Introduction to Parallel Computing

by Scott King

Dr. Ilias Kotsireas

James Reinders

Note: Processes on SHARCNET run 0..n

Note: *procs* means processors

Oh look...our first MPI program.

```
#import <stdio.h>
#import "mpi.h"

int main(int argc, char ** argv) {
  int num_procs;
  int ID;

  if (MPI_init(&argc, &argv) != MPI_SUCCESS) {
    printf("error");
  }

  MPI_Comm_size(MPI_COMM_WORLD, &num_procs);
  MP_Comm_rank(MPI_COMM_WORLD, &ID);
  printf("hello world from process %d of %d \n", ID, num_procs);

  MPI_Finalize();
}
```

- The *Fair Share* algorithm is used in determining the queue for executing C code
- Command to run: `mpicc {name}.c -o {executable_name} -lmpi`
- Command to submit to queue: `sqsub -r 3m -q mpi -n 15 -o ofile.%J ./parallel_hello_world`
  - `sqsub`: Submit to queue
  - `-r 3m`: Time limit
  - `-q mpi`: Submit to MPI-specific queue
  - `-n 15`: How many cpu's we want
  - `-o ...`: The executable we compiled that we want to run

1 Overview of Parallel Computing

Parallel computing utilizes a parallel computer to reduce the time needed to solve a computational problem.

Applications:

- Galactic dynamics

- Climate modeling
- Molecular dynamics
- Protein folding (bioinformatics)
- Data mining
- Oil exploration
- Web search engines
- Medical imaging
- Financial modeling
- Graphics

Note: The first two applications involve numerically solving systems of PDE's.

1.1 Why do we use HPC?

1. To save time and money
2. Solve large scale problems (Computational Fluid Dynamics, CFD)
3. Provide concurrency
4. Distributed resources (SETI)
5. Limits to serial computing

1.2 Motivation for using HPC

1. Speed - large scale computation
    - Explicit numerical scheme - ie. finite differences
    - Discretize temporal and spatial variables, n points + boundary conditions
    - Typically, $O\left(10^{15}\right)$ flops on a 1 GFlop computer $\left(10^{9}\right) = 10^{6}$ secs = 280h
2. Memory
    - How much data can we get in 1GB of RAM?
    - 1GB can hold 512^3 double precision pairs
    - OR two double precision arrays of length $(n+1)^3$ each

2 Parallel Computers

Definition 1. *A computing system that allows multiple processors concurrently, in parallel, to solve a computational problem.*

2.1 Categories of Parallel Computers

2.1.1 Shared-Memory Systems

- The processors have no private memory, they access a single global memory
- Ex. SMP (symmetric multi-procs); memory access time is uniform across all procs
- Ex. NUMA (non-uniform memory access); allows us to incorporate more procs

{insert dia diagram}

2.1.2 Distributed-Memory Systems

- Each proc has a private local memory that is inaccessible by other procs
- The procs are interconnected
    1. Ethernet
    2. Dedicated high throughput, low-latency network

{insert dia diagram}

3 Parallel Computing

Parallel computing is meaningful when, a computational problem can be divided **evenly** among procs which will work in parallel, and will be coordinated in the form of information exchange and synchronization.

Information exchange is to provide private data of one proc to another proc.

Synchronization is keeping the procs at the same place when needed.

Both forms of coordination require communication among procs.

3.1 When is parallel computing beneficial?

1. All procs have sufficient workload
2. The extra overhead caused by parallelization is negligible

3.2 Work Division

Parallel computers are mainly used for task and data parallelism.

3.2.1 Task Parallelism

- Set of standalone computational tasks that exhibit clear distinction between each other
- Some of these tasks may need to follow the completion of other tasks

Example 2. Parameter analysis: $f^{(3)}(x) = a \times x\, f^{(2)}(x)$ where a is the parameter.

- Choose a range of values for a, the solve ODE numerically and study the solutions

Remark 3. All solving processes for distinct values of a are independent. Each of them can be assigned to a procs.

Math reasons. The behaviour of a solution for a^2, can be deduced and also depends on the behaviour of solutions for a. Setup a dependency graph. Parallel execution starts with the tasks that do not depend on other tasks and incrementally include more and more tasks that will become available for execution. When there are more tasks than procs, setup a dynamically allocated queue.

3.3 Data Parallelism

- When the number of tasks is small and the number of procs is large, then multiple processes have to share the work of one task → further work division
- Perform operations to different sections of the same data structure

Example 4. Evaluate a function, $f(x)$, for a set of x-values (1d array)

```
for (i = 0; i < n; i++) {
  y[i] = f(x[i]);
}
```

Assumptions

- All function evaluations are equally expensive
- All procs are equally powerful
- Each proc is assigned a subset of the same number of x-values

We have:

- P: number of procs

Then:

$$n\,p = \left\lfloor \frac{n}{p} \right\rfloor + \begin{cases} 1\text{: } p < \mathrm{mod}\,(n,p) \\ 0\text{: otherwise} \end{cases}$$

where p is the process ID=0...$P-1$.

Property. The max difference between $n\,p$, for $p = 0...P-1$ is 1.

Example 5. Numerical case study

$P = 6$, $n = 63$

$$\left\lfloor \frac{n}{p} \right\rfloor = 10, \mathrm{mod}\,(n,p) = 3$$

$$n_0 = n_1 = n_2 = 11$$

$$n_3 = n_4 = n_5 = 10$$

Note: We know the number of x-values per proc.

The start position for proc, p, is:

$$\begin{aligned} i_{\mathrm{start},p} &= p \times \left\lfloor \frac{n}{p} \right\rfloor + \min\,(p, \mathrm{mod}\,(n,p)) \\ i_{\mathrm{start},0} &= 0 + \min\,(0,) = 0 \\ i_{\mathrm{start},1} &= 1 \times 10 + \min\,(1,3) = 11 \\ i_{\mathrm{start},p} &= 2 \times 10 + \min\,(2,3) = 22 \\ &\vdots \end{aligned}$$

Work Division. n_p, $i_{\text{start},p}$, ...

```
for (i = i_start_p; i < i_start_p + n_p; i++) {
    y[i] = f(x[i]);
}
```

Distributed Memory. Each proc has two local arrays, x_p, y_p

```
for (i = 0; i < n_p; i++) {
    y_p[i] = f(x_p[i]);
}
```

We will need to know which x-values will be assigned to which proc. Assing to each proc a contiguous piece of memory (performance friendly). Index set {0,1,..,n-1} is divided in P pieces using n0 + n1 + ... np-1 = n

Example 6. Composite trapezoidal rule for numerical integration

$$\int_b^a f(x)\mathrm{dx} = h\left(\frac{1}{2}(f(a)+f(b)) + \sum_{i=1}^{n-1} f(a+i\,h)\right)$$

where $h = \frac{b-a}{n}$.

The two endpoints: $\frac{1}{2}$

$n-1$ inner points: 1

What we need to do is divide $n-1$ function evaluations at inner points among P procs, with each proc carrying out a partial summation.

$$S_p = \sum^{i_{\text{start},p}+n_p-1} f(x_i)$$

with $i = i_{\text{start},p}$ and $x_i = a + i\,h$, where $i = 1...n-1$. Also, $n_p = \lfloor \frac{n-1}{P} \rfloor + \begin{cases} 1, p < \mathrm{mod}\,(n-1,p) \\ 0, \text{otherwise} \end{cases}$

Ex. 2 $n = 200$, $P = 3$

$$\begin{aligned} n_0 &= 67 \quad & i_{\text{start},0} &= 1 \\ n_1 &= 66 \quad & i_{\text{start},1} &= 68 \\ n_2 &= 66 \quad & i_{\text{start},2} &= 134 \end{aligned}$$

And from that:

$$\begin{aligned} \text{proc } 0 &\rightsquigarrow x_1...x_{67} \\ \text{proc } 1 &\rightsquigarrow x_{68}...x_{133} \\ \text{proc } 2 &\rightsquigarrow x_{134}...x_{199} \end{aligned}$$

When all of the procs compute their S_p values in parallel, we need an additional computation:

$$h\left(\frac{1}{2}(f(a)-f(b)) + \sum_{p=0}^{P-1} S_p\right)$$

Local result, S_p, are available on each proc $\rightsquigarrow$ needed for communication.

There are **2 approaches:**

1. Designate a master proc (with slave procs)
    - Slave procs, pass on their S_p values to the master

- Master computes $\sum S_p + (...)$

2. All procs have equal role
    - Each proc passed its S_p value to every other proc
    - Each proc computes $\sum S_p + (...)$

3.3.1 MPI Terminology

1. *All-to-one* communication
2. *All-to-all*

These collective communications and their associated computations $\rightsquigarrow$ **reduction operations**. Ultimately, the time cost for these reduction operations is:

$$O(\log_2(P))$$

Example 7. ID-diffusion equation: $\frac{\delta u}{\delta t} = \frac{\delta^2 u}{d x^2} + f(x,t)$, where $u = u(x,t)$.

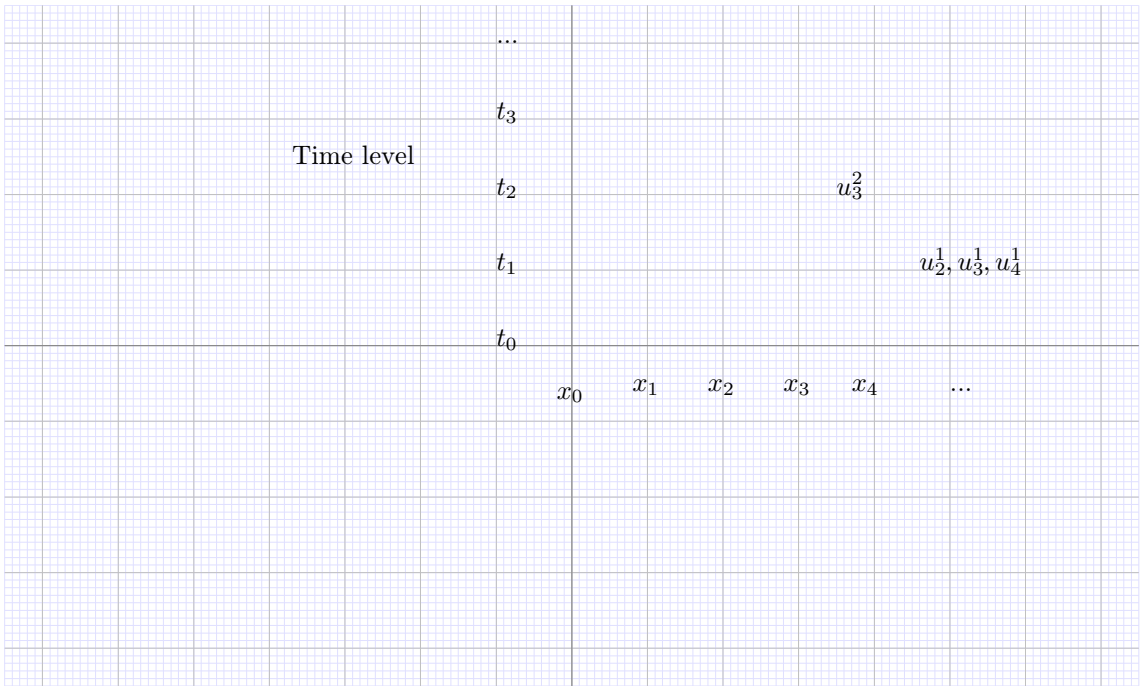
Inter-proc communication is required more often then **Ex. 2**, ie. during the parallel computations.

To solve a ID-diffused equation numerically, we employ a discretization scheme.

$$u_i^{l+1} = u_i^l + \frac{\Delta_t}{\Delta_{x^2}}(u_{i-1}^l - 2u_i^l + u_{i+1}^l) + \Delta_t f(x_i, t_e)$$

where we have $u_i^l = u(x_i, t_e)$ and x_i is the discretization of x and t_l is the discretization of t, Δ_t is the interval length and Δ_x is the interval length.

This means, that a point (x_i, t_{l+1}) we can compute the function value u_i^{l+1} from the known function values: $u_i^l, u_{i-1}^l, u_{i+1}^l$.



Comment. Boundary conditions provide initial values for the discretization scheme.

Parallelism in this example rises from the fact that the computations to find inner points u_i^{l+1} and u_j^{l+1} are independent.

The $n-1$ inner points can be computed in parallel for level $l+1$.

To compute u_i^{l+1}, we need only three values: $u_i^l, u_{i-1}^l, u_{i+1}^l$ from the previous level.

Work Division.

Partition inner points, $x_1, ..., x_{n-1}$, in P contiguous pieces. Each proc computes u_i^{l+1} for a subset of i indices in $\{1, ..., n-1\}$.

- The proc responsible for x_0, has to update u_0^{l+1}, using bondary conditions
- The proc responsible for x_n, has to update u_n^{l+1}, using the discretization scheme

Parallelism, in this example, assumes that **ALL** points at the same time level have been computed.

$\rightsquigarrow$ No proc will be allowed to proceed to level $t+1$ before all other processors have finished at level l.

$\rightsquigarrow$ Coordination between procs is achieved by a built-in synchronization mechanism, **barrier** (MPI term) which forces all pocs to wait for slower ones.

3.3.2 S-M Systems

Barrier operation is only needed for inter-proc communication.

3.3.3 D-M Systems

Proc, p, should operate on two local arrays u_p^l, u_p^{l+1} of length n_p.