Parallel programming

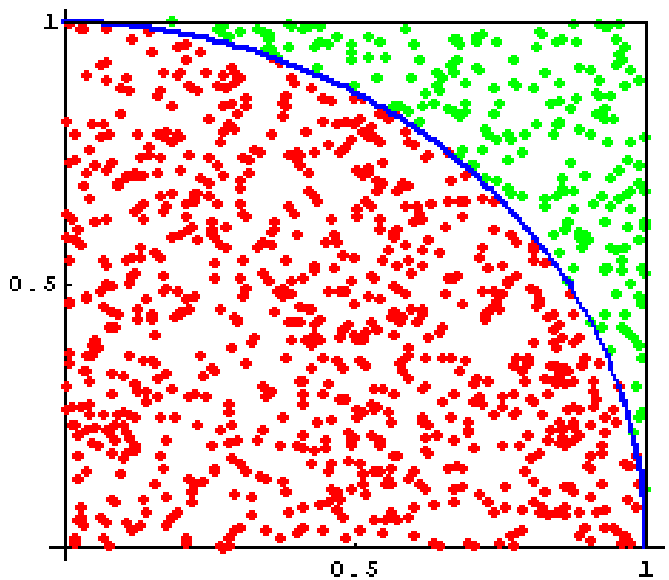
MPI Interface

03 - Monte Carlo Method for calculating π

Monte Carlo methods can be thought of as statistical simulation methods that utilize a sequences of random numbers to perform the simulation. The name "Monte Carlo" was coined by Nicholas Constantine Metropolis (1915-1999) and inspired by Stanslaw Ulam (1909-1986), because of the similarity of statistical simulation to games of chance, and because Monte Carlo is a center for gambling and games of chance. In a typical process one compute the number of points in a set **A** that lies inside box **R**. The ratio of the number of points that fall inside **A** to the total number of points tried is equal to the ratio of the two areas (or volume in 3 dimensions). The accuracy of the ratio ρ depends on the number of points used, with more points leading to a more accurate value.

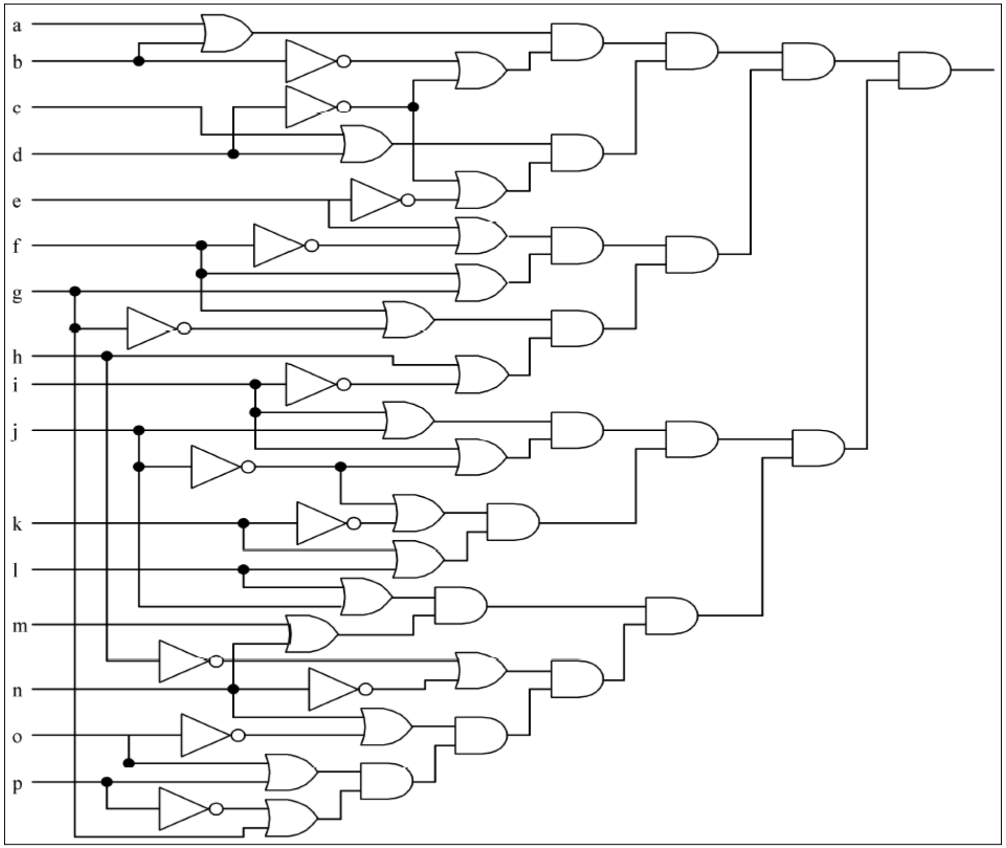
Monte Carlo Method for calculating π

A simple Monte Carlo simulation to approximate the value of $\boldsymbol{\pi}$ could involve randomly selecting points $\{(x_i, y_i)\}$ in the unit square and determining the ratio $\rho = \frac{m}{n}$, where m is number of points that satisfy $x_i^2 + y_i^2 \leq 1$. In a typical simulation of sample size $n = 1000$ there were 787 points satisfying , shown in Figure below. Using this data, we obtain $\rho = \frac{m}{n} = \frac{787}{1000}$ and $\pi = \rho * 4 = 3.148$.



04 - Circuit Satisfability

Implement the MPI program that computes whether the cicruit shown above is satisfable (in other words, for what combinations of input values (if any) will the cicruit output the value 1?), and return the value how many combinations satisfy this circuit. This problem is in class NP-complete, which means there is no known polynomial time algorithm to solve general instances of this problem.



Collective Communications

- Collective communication involves the sending and receiving of data among processes
- These "blackbox" routines hide a lot of the messy details and often implement the most efficient algorithm known for that operation
- You *must* ensure that all processors execute a given collective communication call

Examples of collective communication

- Barrier sychronization across all processes
- Broadcast from one process to all other processes
- Global reduction operations such as sum, min, max or user-defined reductions
- Gather data from all processes to one process
- Scatter data from one process to all processes
- …

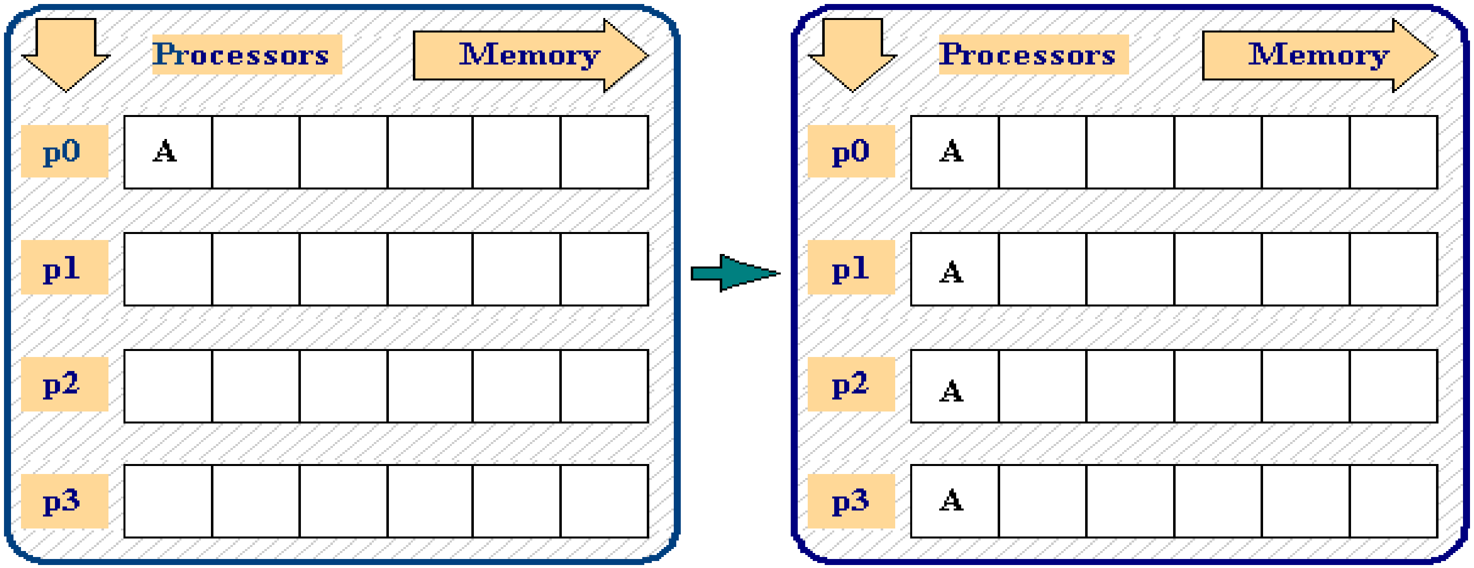
Barrier Synchronization

- There are occasions when some processors cannot proceed until other processors have completed their current instructions
- The *MPI_BARRIER* routine blocks the calling process until all group processes have called the function
- You should only insert barriers when they are truly needed

```
int MPI_Barrier ( comm  )
```

Broadcast

- The *MPI_BCAST* routine enables you to copy data from the memory of the root processor to the same memory locations for other processors in the communicator



Broadcast

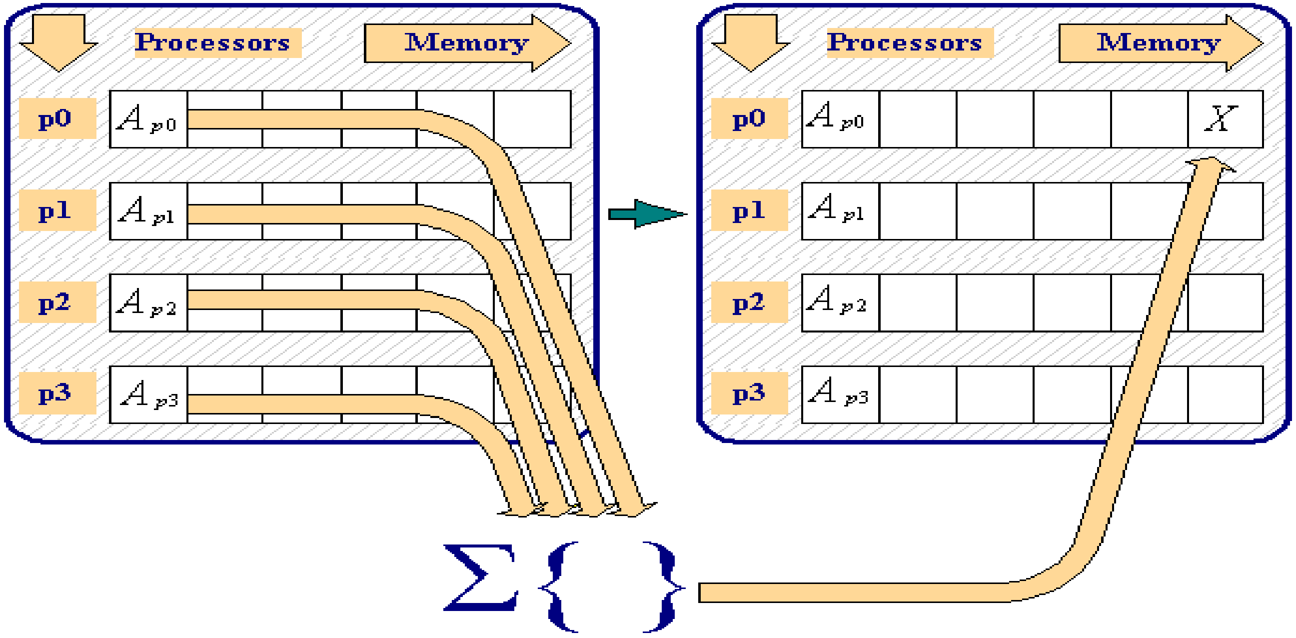
- `int MPI_Bcast ( void* buffer, int count, MPI_Datatype datatype, int rank, MPI_Comm comm )`

• buffer	in/out	starting address of send buffer
• count	in	number of elements in send buffer
• datatype	in	data type of elements in send buffer
• rank	in	rank of root process
• comm	in	mpi communicator

Example 03 - Broadcast

Reduction

- The *MPI_REDUCE* routine enables you to:
  - Collect data from each processor
  - Reduce these data to a single value (such as a sum or max)
  - Store the reduced result on the root processor



Reduction

- `MPI_Reduce( send_buffer, recv_buffer, count, data_type, reduction_operation, rank_of_receiving_process, communicator )`
- The send buffer is defined by the arguments *send_buffer*, *count*, and *datatype*
- The receive buffer is defined by the arguments *recv_buffer*, *count*, and *datatype*
- Operations: MPI_MAX, MPI_MIN, MPI_SUM, MPI_PROD, MPI_LAND, MPI_BAND, MPI_LOR, MPI_LXOR,

Example 04 - Reduction