MPI Special Topics:
Naturally Parallel Applications
Monte-Carlo Simulations

Martin Siegert, SFU

MPI Special Topics Series

This is about:

- Learn parallel programming using MPI
- Methods for parallelizing a program
- Use an example to illustrate MPI programming

This is not about:

- Advanced Special Topics
- Advanced Parallel Programming Techniques

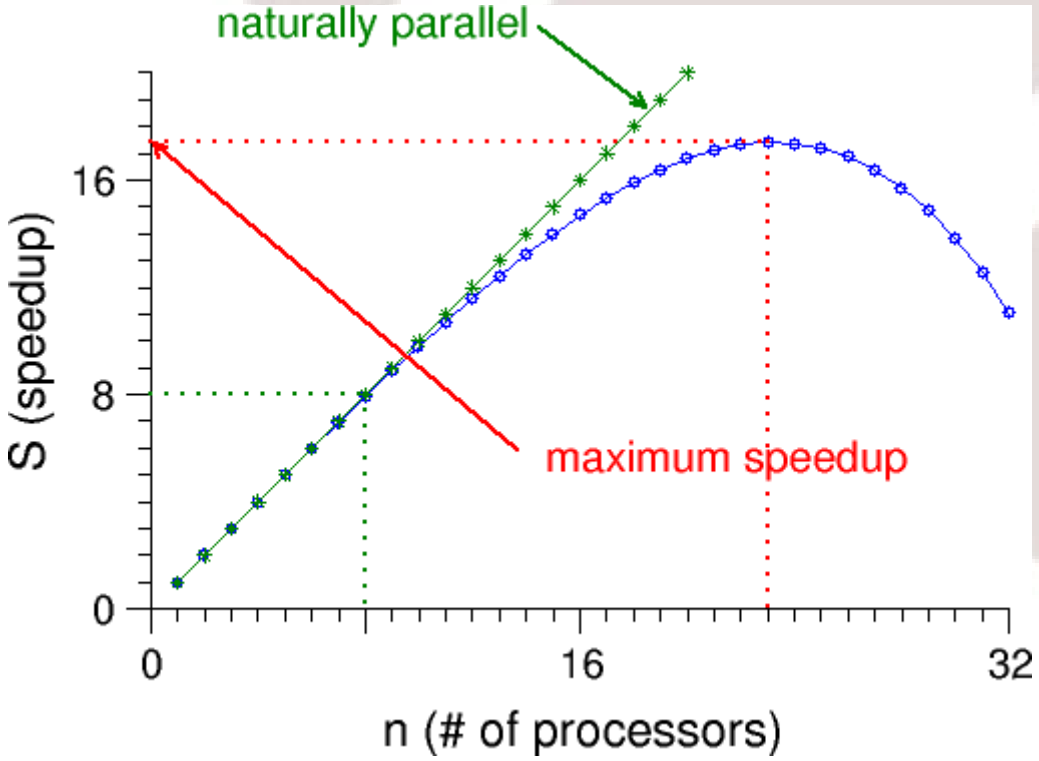
Naturally Parallel Applications Monte-Carlo Simulations

- Introduction to MPI programming
- Example: Monte-Carlo Simulations: $2d$-Ising Model
  - Similarly: Parameter Studies
  - somewhat more difficult: master-slave applications
- Checkpointing MPI applications

Parallel Performance

How can results calculated on one processor be used on another processor?

↣ transfer of data (communication) required



typical parallel application:

maximum speedup

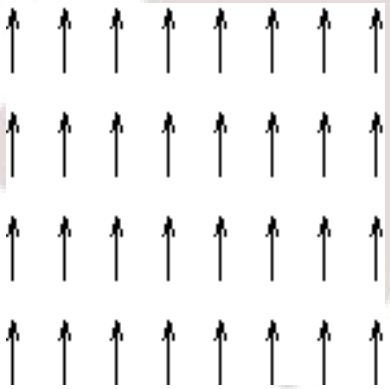
location of maximum depends on problem and interconnect.

Monte-Carlo Simulations
Ising Model

Magnetic Material

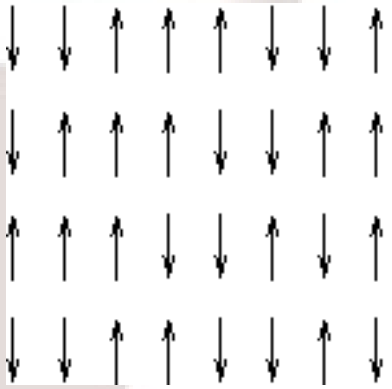
low T

M=1



high T

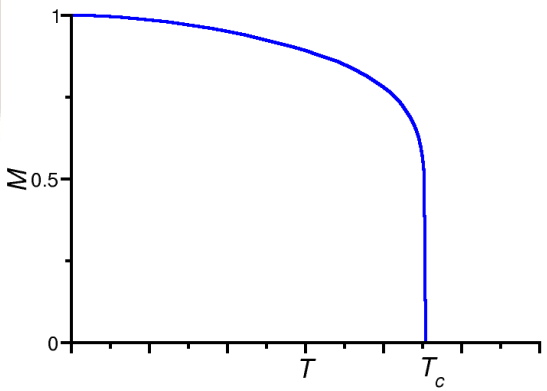
M=0



What is the configuration for arbitrary T?

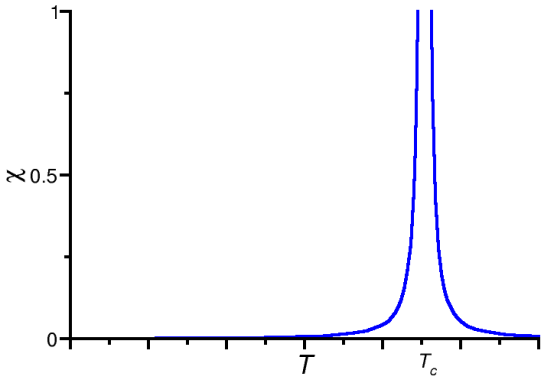
Ising Model: Critical Phenomena

Magnetization



$$\langle M \rangle \sim (T_c - T)^{1/8}$$

Susceptibility



$$\chi \sim \langle M^2 \rangle - \langle M \rangle^2 \sim |T_c - T|^{-7/4}$$

- critical exponents, universality, ...
- Finite-size scaling ...

Monte-Carlo Simulations Ising Model (cont'd)

1) start with some initial configuration
2) select particle
3) flip particle according to some probability
4) iterate 2, 3
5) measure total magnetization ***M*** and add to running average: $M_{ave} = \sum M$
6) iterate 2 – 5
7) print M_{ave}***/(no. of measurements)***

Random Number Generators

Never, ever use the system supplied random number generator.

- drand48, random, random_number, ...
- bad statistics
- small period
- slow

Source code of very good RNGs available:

- Mersenne Twister
- Marsaglia's Diehard test suite
- www.westgrid.ca/downloads/rng.pdf

Monte-Carlo Simulations
Typical Program

black: serial code run on single processor
green: code run in parallel on all processors
red: communication between processors
blue: comment

Serial Program

```
program monte_carlo
<read input data>
<generate initial configuration>
for mc=1 to mc=nrun do
   <generate new configurations acc. to probabilities>
   <measure quantities of interest>
   <update running averages>
end do
<print results>
```

Monte-Carlo Simulations
Typical Program

Parallel Program

```
program monte_carlo
<master reads input data>
<broadcast input data to all processes>
<generate initial configuration>
for mc=1 to mc=nrun/np do
   <generate new configurations acc. to probabilities>
   <measure quantities of interest>
   <update running averages>
end do
<collect averages at master process>
<master prints results>
```

Program almost 100% parallel:

➪ speedup = no. of processors

Monte-Carlo Input Data (Fortran)

```
program ising
include 'mpif.h'
<declarations>
   call MPI_Init(mpierr)
   call MPI_Comm_rank(MPI_COMM_WORLD, myid, mpierr)
   call MPI_Comm_size(MPI_COMM_WORLD, numprocs, mpierr)
   call pseeds(myid, numprocs)   ! initialize RNG
! master reads input
   if (myid == 0) then
      read*, l, t, lambda, n_ave
   end if
   call MPI_Bcast(l,1,MPI_INTEGER,0,MPI_COMM_WORLD,mpierr)
   call MPI_Bcast(t,1,MPI_DOUBLE_PRECISION,0, &
                  MPI_COMM_WORLD,mpierr)
   call MPI_Bcast(lambda,1,MPI_INTEGER,0, &
                  MPI_COMM_WORLD,mpierr)
   call MPI_Bcast(n_ave,1,MPI_INTEGER,0, &
                  MPI_COMM_WORLD,mpierr)
```

Monte-Carlo Input Data (C)

```
#include <stdio.h>
...
#include <mpi.h>
int main(int argc, char *argv[]){
...
   MPI_Init(&argc, &argv);
   MPI_Comm_rank(MPI_COMM_WORLD, &myid);
   MPI_Comm_size(MPI_COMM_WORLD, &numprocs);
   pseeds(myid, numprocs);   /* initialize RNG */
   int l=atoi(argv[1]);
   double t=atof(argv[2]);
   int lambda = atoi(argv[3]);
   int n_ave = atoi(argv[4]);
...
```

Monte-Carlo Simulation

```
    do n_upd = 1, n_ave/numprocs
       do mcs = 1, lambda
          ...                          ! Monte-Carlo Code
       end do
! take averages
       rm = dble(2*M)/rll              ! magnetization per spin
       ave_m = ave_m + abs(m)
       ...                             ! other averages
    end do
```

Code changes in red.

Monte-Carlo Simulation

```
for (n_upd=1; n_upd <= n_ave/numprocs; n_upd++){
   for (mcs=1; mcs<=lambda; mcs++){
      ...                          /* Monte-Carlo Code */
   }
/* take averages */
   rm = ((double)(2*M))/rll;     /* magnetization
                                       per spin */
   ave_m += abs(m);
   ...                          /* other averages */
}
```

Code changes in red.

Monte-Carlo Collect Results

```
! send result to master
   call MPI_Reduce(ave_m, ave_m_tot, 1, &
                   MPI_DOUBLE_PRECISION, MPI_SUM, &
                   0, MPI_COMM_WORLD, mpierr)
   ...          ! other MPI_SUM reductions
! write results
   if (myid == 0) then
      ...       ! print results
   end if
   call MPI_Finalize(mpierr)
end
```

Monte-Carlo Collect Results

```
/* send result to master */
   MPI_Reduce(&ave_m, &ave_m_tot, 1, MPI_DOUBLE,
                  MPI_SUM,0, MPI_COMM_WORLD)
   ...          /* other MPI_SUM reductions */
/* write results */
   if (myid == 0) {
      ...       /* print results */
   }
   MPI_Finalize();
}
```

MPI Summary

- Initialization/Finalization

```
MPI_Init(&argc, &argv);
MPI_Comm_rank(MPI_COMM_WORLD, &myid);
MPI_Comm_size(MPI_COMM_WORLD, &numprocs);
...
MPI_Finalize();
```

- Broadcast of data:

```
call MPI_Bcast(l, 1, MPI_INTEGER, 0, &
                    MPI_COMM_WORLD, mpierr)
```

- Collecting Results

```
MPI_Reduce(&ave_m, &ave_m_tot, 1, MPI_DOUBLE,
              MPI_SUM, 0, MPI_COMM_WORLD)
```

Checkpointing

- Save data periodically so that program can be restarted
- Important for long running jobs
- Important for jobs that run longer than the allowed maximum walltime
- particularly easy for iterative algorithms

Checkpointing MPI Programs

Parallel I/O is difficult:

- cannot simply write to a single file from several processes.
- writing one file from every process can be problematic

Recommendation:

- send all data to master.
- master writes checkpoint file
- can be combined with writing intermediate results

Checkpointing MPI Programs: Monte-Carlo Simulations

Collect results after, e.g., each 10th of a run has been completed:

```
n_ave_p = n_ave/numprocs;
n_pr = n_ave_p/10;
n_upd_pr = n_pr;
for (n_upd = 1; n_upd <= n_ave_p; n_upd++) {
    ...
    if (n_upd >= n_upd_pr || n_upd == n_ave_p){
        /* collect results */
        n_upd_pr += n_pr;
        MPI_Reduce(&ave_m, &ave_m_tot, 1, MPI_DOUBLE,
                   MPI_SUM, 0, MPI_COMM_WORLD);
        ...
        /* write checkpoint file, print results */
    }
}
```

Running a Restartable Program

Torque submission script:

```
#!/bin/bash
#PBS -l walltime=160:00:00
#PBS -r y
#PBS -l nodes=10
cd $PBS_O_WORKDIR
size=200; temp=2.268; fdat=is${size}_${temp}
if [ ! -s "`pwd`/$fdat" ]; then rm -f $fdat; fi
date
mpiexec -machinefile $PBS_NODEFILE -n 10 \
   ./ising-mpi $size $temp 100000 10000 10000 $fdat 1
echo "Job finished at `date`"
```

Preemptible and restartable job:

```
qsub -q pre ising-mpi.pbs
```

Questions?

- www.westgrid.ca/downloads/PPT/MC.pdf
- www.westgrid.ca/downloads/ising-C.tar.gz
- www.westgrid.ca/downloads/ising-F.tar.gz
- www.westgrid.ca/downloads/rng.pdf
- MPI – The Complete Reference (Vol. 1)
- www.sfu.ca/~siegert/comphys.ps.gz