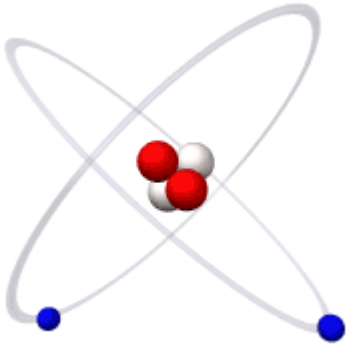


HPSC 5576 Final presentation

Florian Rappl & Christoph Preis

Contents

A. Physics
   - The Ising model
   - Helium mixing with Monte Carlo

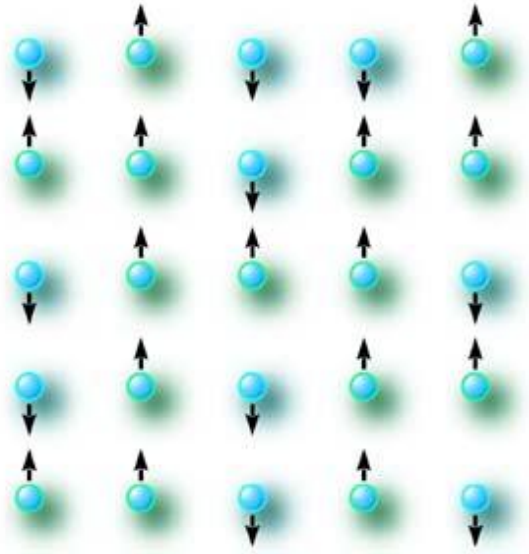
B. Computation
   - Splitting up the lattice & ghost-sites
   - Shared memory on blades
   - Reducing communication with a checkerboard
   - MPI used

C. Results

Part A

PHYSICS

The Ising model[1]

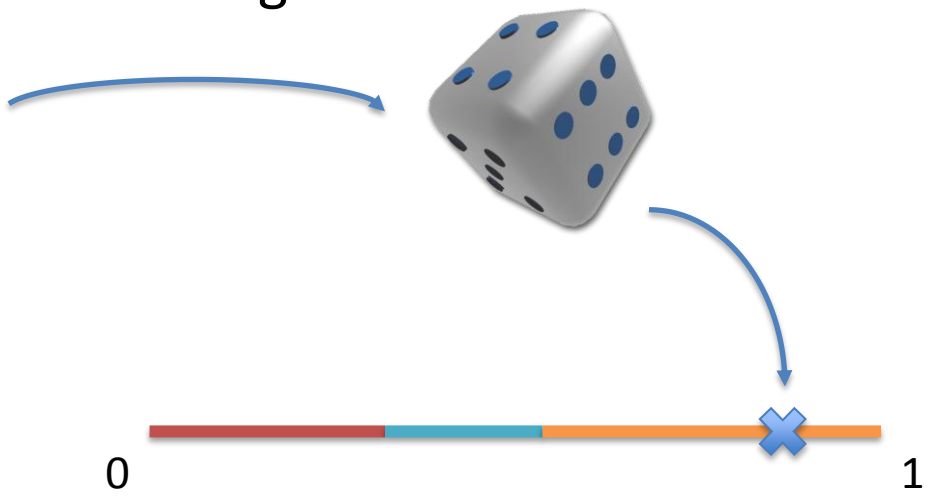


- Introduced to describe ferromagnestism
- Contains only nearest neighbor interaction
- Lattice with values of ± 1 on each site
- Parameter: inverse temperature β
- 2D Ising is simplest to show transition

[1] J. P. Sethna: Entropy, Order Parameters, and Complexity (Oxford, 2009)

Reminder on Monte Carlo[2]

- Pick site on lattice
- Choice can be random or determined
- Calculate energy H after possible change
- Generate random number
- Accept or reject the change



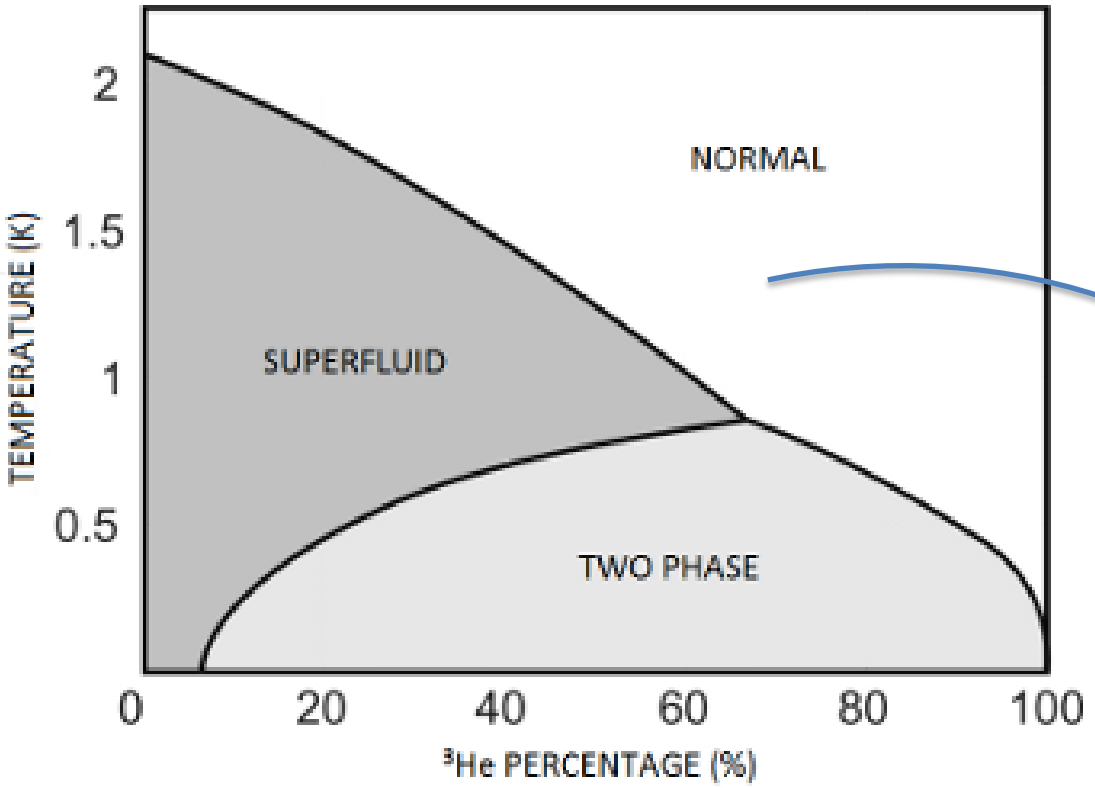
[2]T. DeGrand : Lattice Methods for Quantum Chromodynamics (World Scientific, 2006)

He-Mixing with Monte Carlo[3]

- In principle the same as the Ising model
- Now values of $0, \pm 1$ for the sites
- Introduced a new parameter μ
- So now 2 parameters (β, μ) overall
- Energy reduces to Ising model when $\mu = 0$

[3]M. Blume, V. J. Emery and R. B. Griffiths: Ising model for the μ Transition and Phase Separation in He^3-He^4 Mixtures, Phys. Rev. A **4(3)**, 1071 (1971)

What can we simulate?



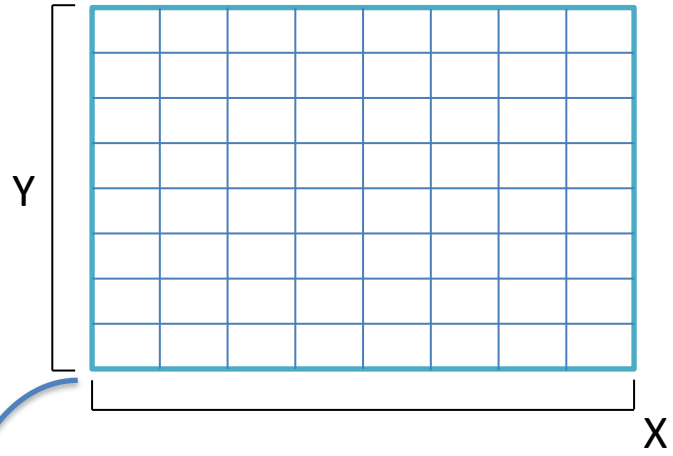
*Can we **simulate** the **experimental** findings representing this helium mixing phase diagram?*

Part B

COMPUTATION

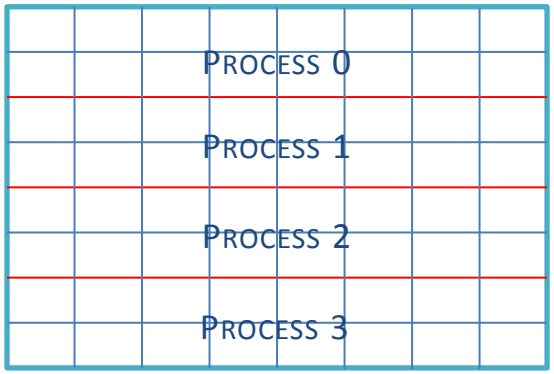
Splitting up the lattice

- We built our program with three dimensions – idea in 2D



- The lattice should be split up equally
- We reduce dimensions starting from z, y, x
- Should make ghostsite calculation very easy

Divide 8x8 lattice into 4 (8x2)

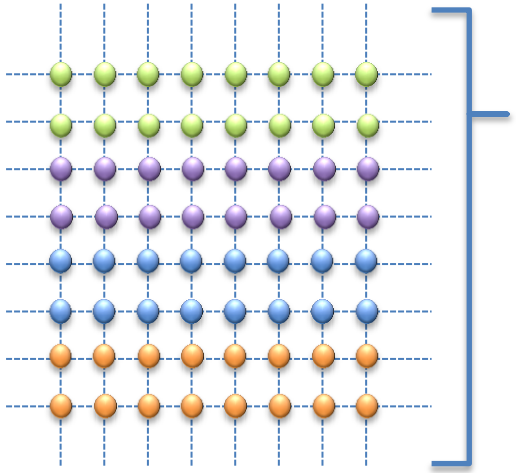


Divide 8x8 lattice into 16 (4x1)

Process 0	Process 1
Process 2	Process 3
Process 4	Process 5
Process 6	Process 7
Process 8	Process 9
Process 10	Process 11
Process 12	Process 13
Process 14	Process 15

Ghostsites from the site‘s view

- Every node prepares a vector for each side to be sent



The green processor prepares four arrays:

Top – send > orange

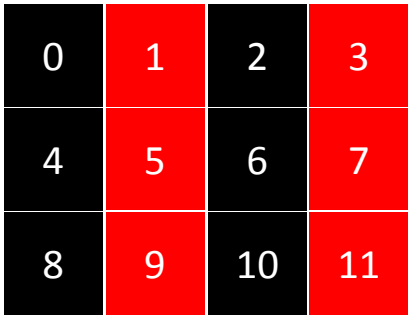
Bottom – send > purple

Left – no send

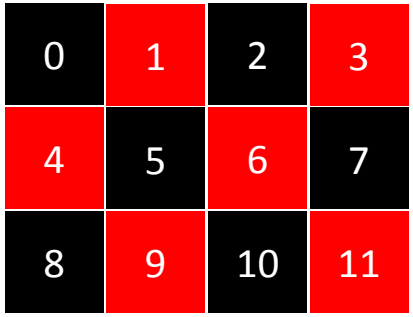
Right – no send

Ghostsites from the process‘s view

- For sending and receiving we need to make a processor grid



- In x-direction we have ± 1 to the next neighbor
- We first send with even to the odd x-processes
- Then we switch and send with the odd to the even



- In y-direction we have $\pm P_X$ to the next neighbor
- We first send with even to the odd y-processes
- Then we switch and send with the odd to the even

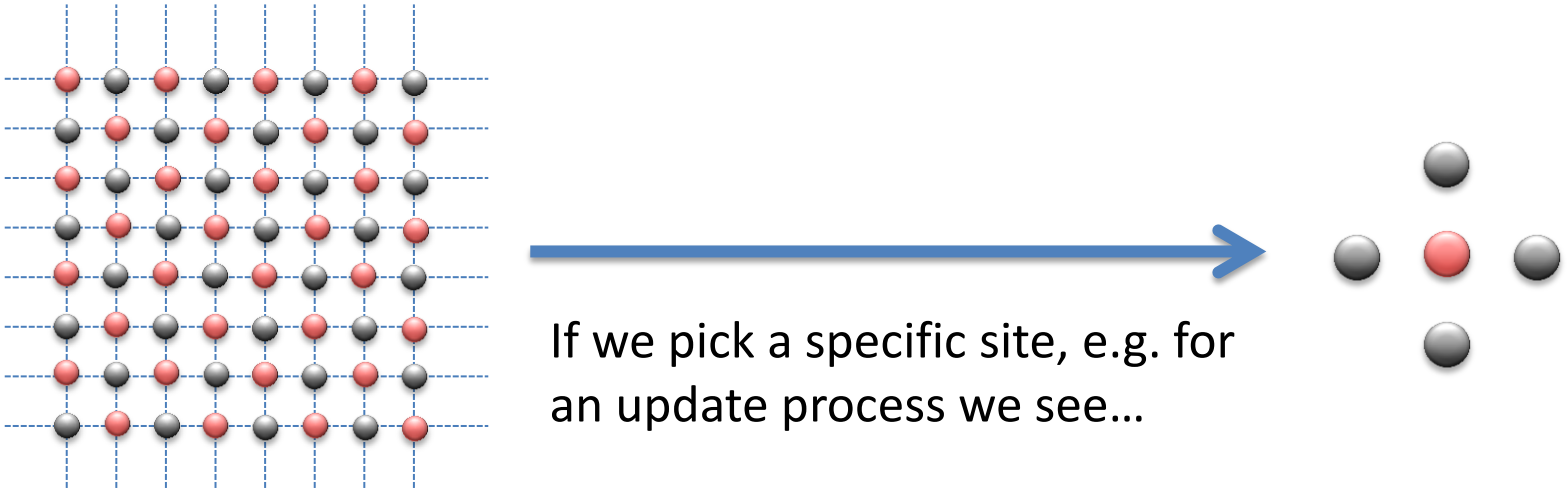
Shared memory on PSC SGI **Blacklight**

- 1 Blade = 1 Configuration
- n times 16 CPUs ➔ n configurations
- Shared memory enough for huge lattice
- Fast communication in Blade
- Inter-Blade communication small

➔ Both benefits – large lattices and more accurate statistics

Reducing communication

- Random choice problem: communication per iteration
- Determined choice: red/black checkerboard



If we pick a specific site, e.g. for an update process we see…

- Therefore we reduce communication with factor $V/2p$

MPI used

a. Wrapped Send / Recv
b. Derived datatypes
c. Custom communicators
d. Broadcast & Reduce

a. Wrapped Send/Recv

```
int COMM_Send(void* message, int count, int dest, int tag)
{

#ifdef MPICOMM
    // execute MPI code
    return MPI_Send(message, count, MPI_DOUBLE, dest, tag,
MPI_COMM_WORLD);

#else
    //no communication needed.
    return 0;

#endif /*MPICOMM*/
}
```

b. Derived datatypes

```
struct inputData
{
    /* lattice dimensions */
    int nx, ny, nz;
...
};

struct inputData data;

void distributeInputData()
{
#ifdef MPICOMM
    // Broadcast input data. Process 0 sends, all other processes
    // receive the data.
    MPI_Bcast(&data, sizeof(struct inputData), MPI_CHAR, 0,
    MPI_COMM_WORLD);
#else
    // nothing.
#endif /*MPICOMM*/
}
```

c. Custom communicators

```
#ifdef MPICOMM
    MPI_Comm MPI_COMM_BLADE;
    MPI_Comm MPI_COMM_MASTERS;
#endif /*MPICOMM*/


// Blade communicator: All processes on one blade in a communicator
MPI_Comm_split(MPI_COMM_WORLD, latticeIndex, globalIndex,
&MPI_COMM_BLADE);

// Master communicator: All blade masters in a communicator.
MPI_Comm_split(MPI_COMM_WORLD, nodeIndex, globalIndex,
&MPI_COMM_MASTERS);
```

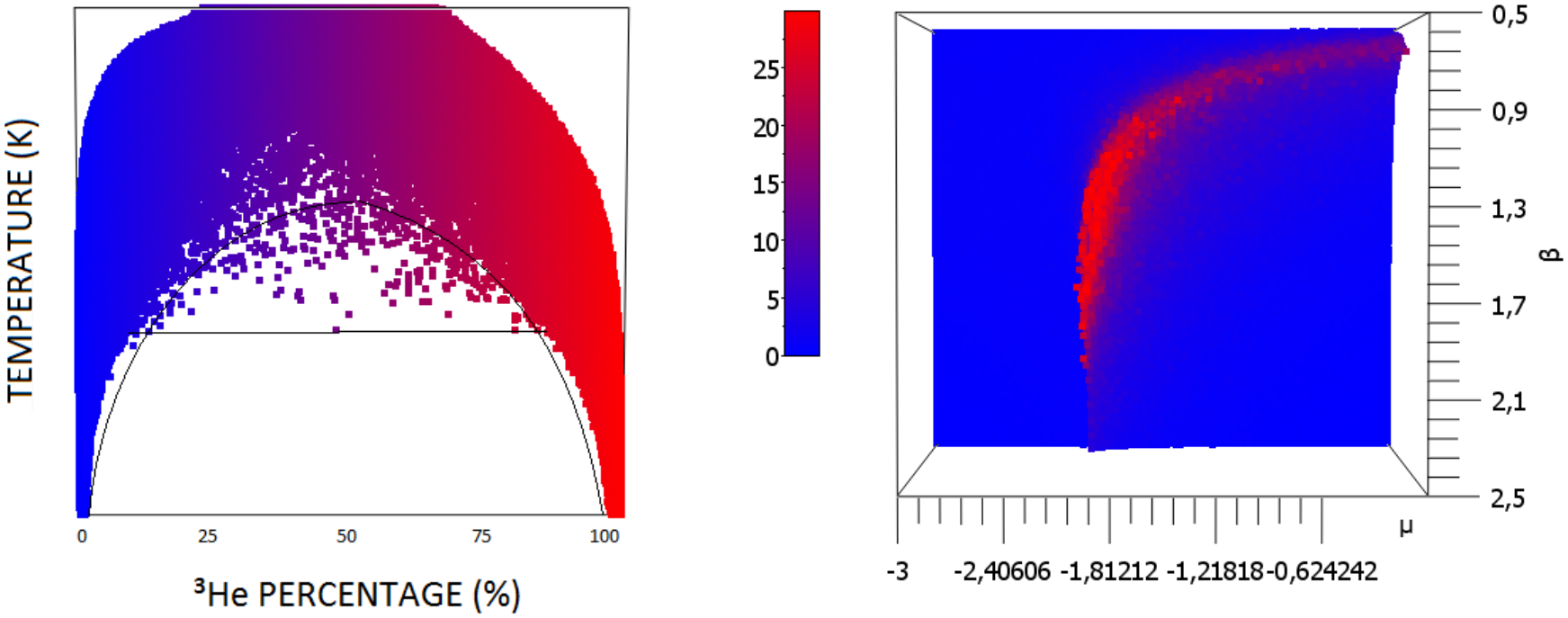
d. Broadcast & Reduce

```
void vectorsReduceSum(double* vec, double* sum, double* sumsq)
{
#ifdef MPICOMM
    double *out;
    /* reduce vectors from all sites */
    MPI_Reduce(vec, out, 1, MPI_DOUBLE, MPI_SUM, 0,
MPI_COMM_BLADE);
    /* sum on root node of each blade */
    if(nodeIndex == 0)
    {
        sum[0] = out[0];
        sumsq[0] = out[0] * out[0];
    }

#else
    /* execute single processor code */
    sum[0] = vec[0];
    sumsq[0] = vec[0] * vec[0];

#endif /*MPICOMM*/
}
```

Part C

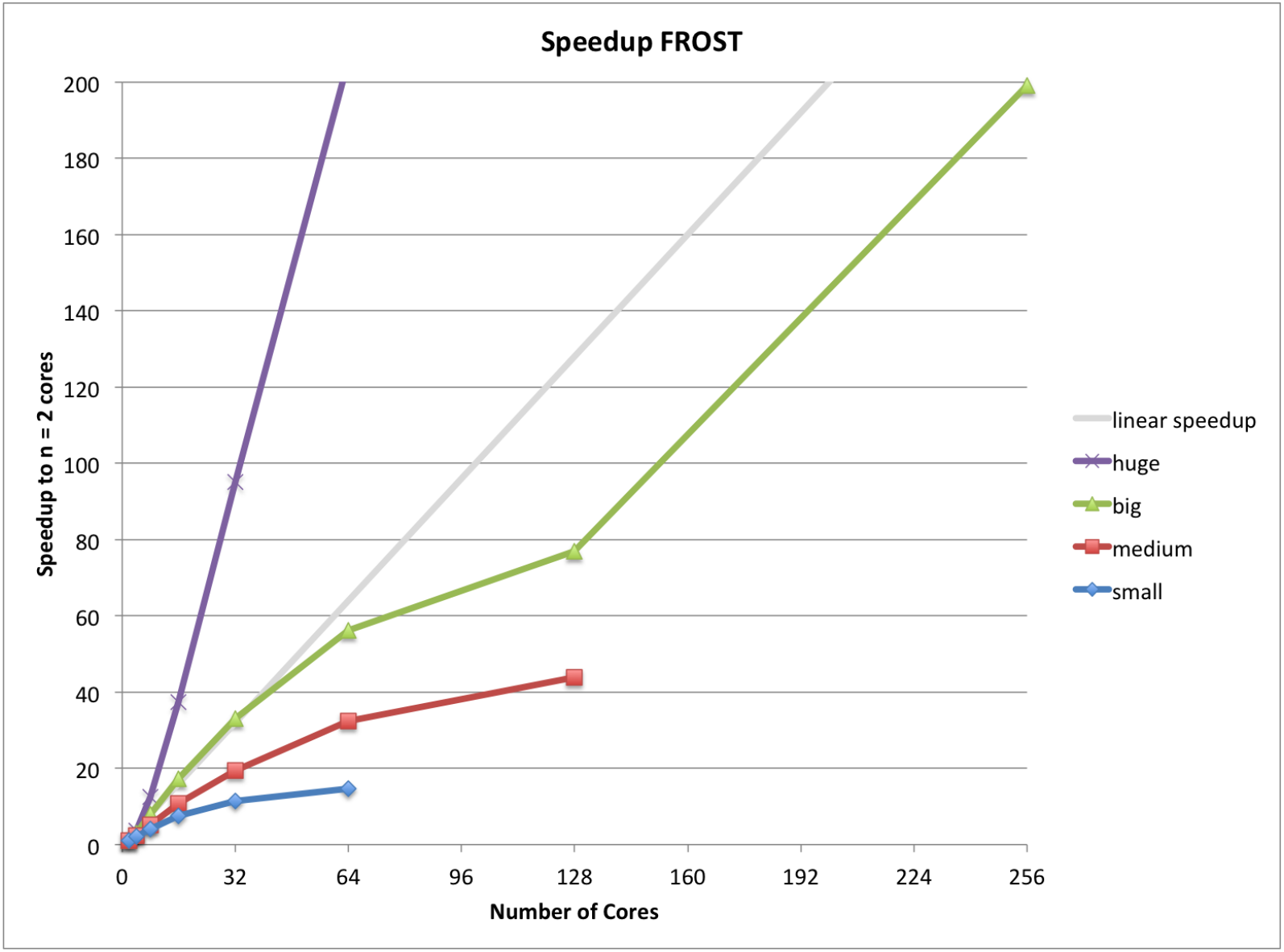
RESULTS

Physical results



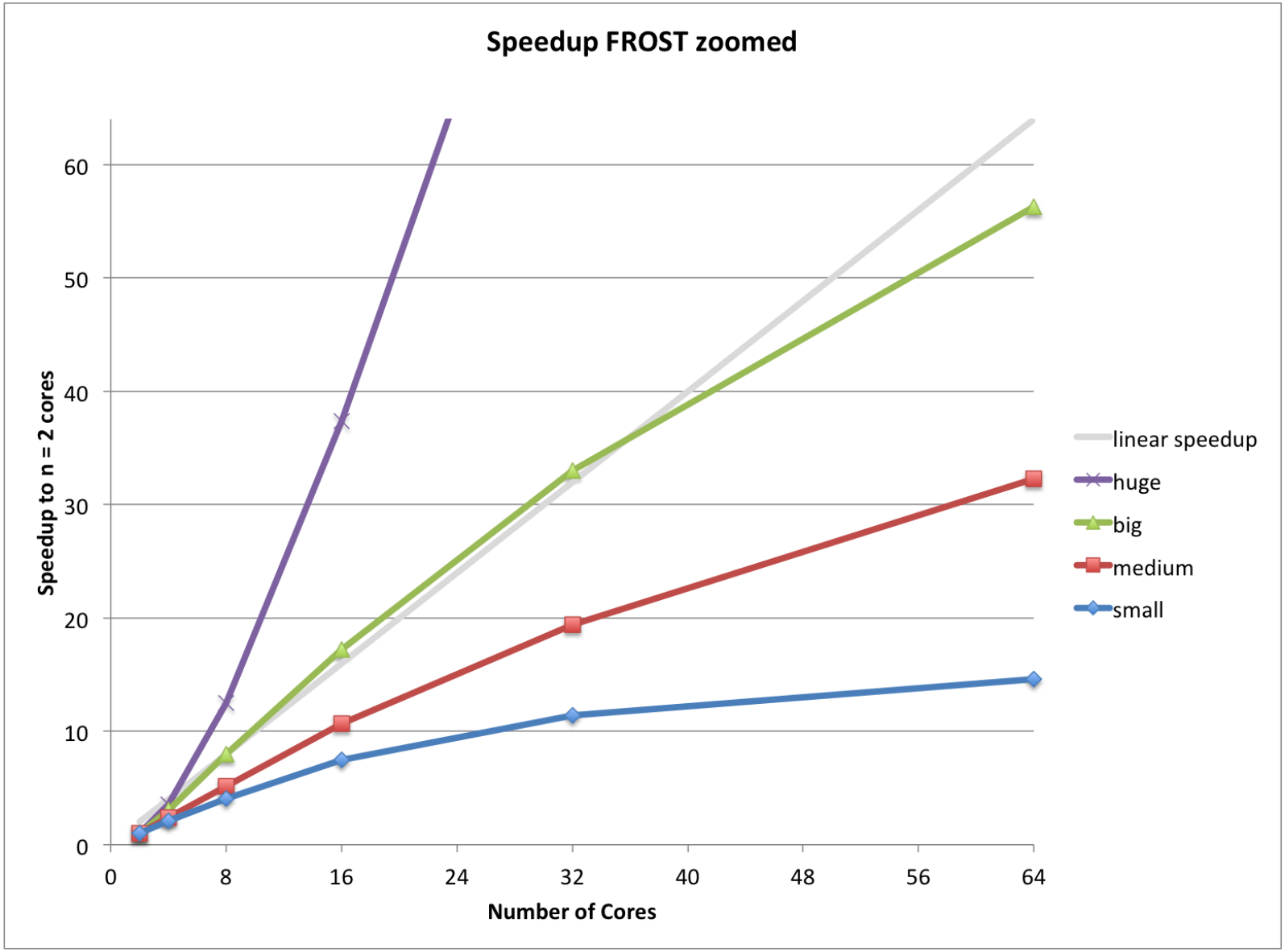
- To answer the question: YES
- We can simulate the phase diagram
- Simulation also shows predicted (critical) line

MPI speedup #1



small = 64 x 64 x 1, medium = 128 x 128 x 1, big = 256x 256 x 1, huge = 512 x 512 x 1

MPI speedup #2



small = 64 x 64 x 1, medium = 128 x 128 x 1, big = 256x 256 x 1, huge = 512 x 512 x 1

MPI speedup #3

2D huge = 512 x 512 x 1, 3D huge = 64 x 64 x 64

Thanks for your attention!

Any questions?