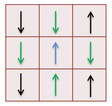
Accelerating Parallel Monte Carlo Simulations for Statistical Physics: Portability on Many-core Processors

Naw Safrin Sattar (nsattar@uno.edu)[1]; Ying Wai Li (yingwaili@lanl.gov)[2]
[1]Big Data and Scalable Computing Research Group, Dept. of Computer Science, University of New Orleans, New Orleans, LA 70148
[2] Applied Computer Science Group (CCS-7), Computer, Computational, and Statistical Sciences Division, Los Alamos National Laboratory, Los Alamos, NM 87545

Introduction

The Ising Spin Model Hamiltonian



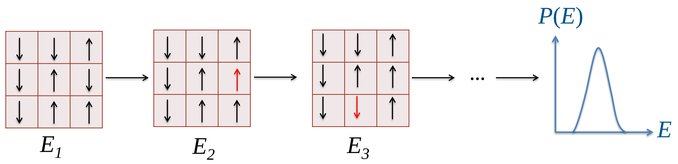
$$E = -J \sum_{<ij>} \sigma_i \sigma_j$$

σ_i: spin at site i
J: interaction strength

Interactions only between nearest neighbors

Metropolis[1] Monte Carlo (MC) Simulations for Spin Models

Given a temperature, random sampling to obtain probability distribution in energy, $P(E)$:



Energy and other physical observables are accumulated to calculate the ensemble averages $\langle E \rangle$

Importance

Why Parallel Monte Carlo?

- MC simulations are important tools for studying thermodynamics and materials properties at finite temperature
- Scaling up to larger systems allows for simulations comparable to experimental length scale

Strong and Weak Scaling

- Strong scaling is achieved by using multiple random walkers →**"Embarrassingly Parallel"**
- Weak scaling to study larger systems is harder to achieve but offers an additional layer of parallelism

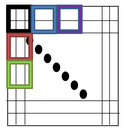


Fig 1. Domain decomposition for large system in weak scaling

Focuses of this work

- Parallelization of computation for large systems (millions of spin/atoms) within a single random walk using domain decomposition
- Study of performance of massively parallel MC simulations on many-core processors

Acknowledgement

This work was supported by the Laboratory Directed Research and Development Program of Los Alamos National Laboratory (LANL). LANL is operated by Triad National Security, LLC, for the National Nuclear Security Administration of U.S. Department of Energy (Contract No. 89233218CNA000001).

Checkerboard Algorithm

Parallel Domain Decomposition [2]

- Spin flip on a black box only affects interactions with surrounding white boxes
- Parallelization achieved by performing spin flips on all black boxes altogether
- Spin flips alternate between black boxes and white boxes

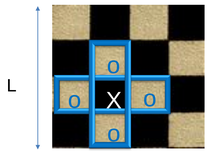


Fig 2. A 4x4 checkerboard; where Grid Size, L=4, System Size, N=4x4=16

Performance Analysis

- Serial[3] MC simulations of 20,000 MC steps as a baseline
- Implemented checkerboard algorithm with multithreading
- Repeat simulations with different number of OpenMP threads on different architectures (OpenMP 5.0 + GCC 9.4.0)

Table 1. Multicore CPU Hardware Architectures Examined

CPU Vendor	CPU Model	CPU Family	Base Clock Rate (GHz)	Total Cores	Threads per Core	Total Threads
Fujitsu	A64FX	arm	1.8	48	1	48
Cavium	ThunderX2 CN9980	arm	2.2	64	4	256
AMD	EPYC 7742	amd	2	128	2	256
Intel	Xeon Phi 7250	knl	1.4	68	4	272
Intel	Gold 6152	skylake	2.1	44	2	88
Intel	Gold 6254	cascade-lake	3.1	36	2	72

Effect of Thread Affinity

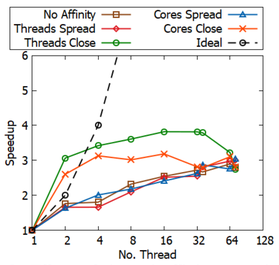


Fig 3. Effect of thread affinity examined on Intel Cascadelake. Experiments performed on a 512 × 512 system size.

Placing threads close together:

```
OMP_PLACES=threads
OMP_PROC_BIND=close
```

results in largest speedup compared to thread spreading:

```
OMP_PLACES=cores
OMP_PROC_BIND=spread
```

References

1. N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller and E. Teller, J. Chem. Phys. **21**, 1087 (1953).
2. D. Heermann and A.N. Burkitt, Parallel Comput. **13** 345–357 (1990).
3. Y. W. Li, K. C. Pitike, and USDOE. Oak Ridge Wang-Landau (OWL). Computer software. 2019. https://www.osti.gov//servlets/purl/1510020.

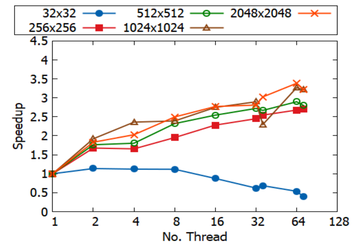


Fig 4. Performance on Intel Cascadelake without affinity.

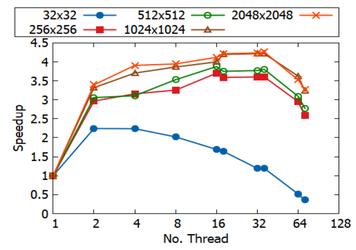


Fig 5. Performance on Intel Cascadelake with <thread,close> affinity. Skylake shows a similar pattern with slightly different speedup factors.

Speed-Up on Different Multi-Core Architectures with Optimal Affinity: <threads,close>

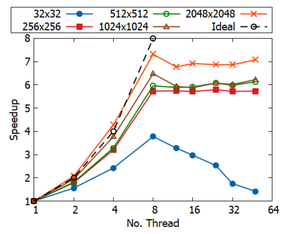


Fig 6. Performance on ARM (Fujitsu): <cores,close> yields the best performance.

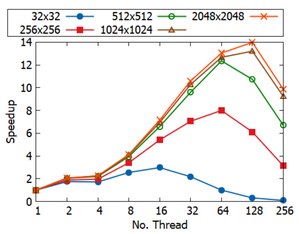


Fig 7. Performance on ARM (Cavium).

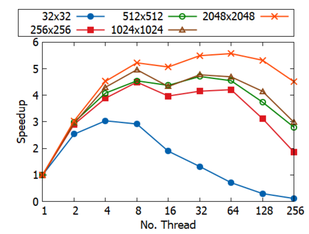


Fig 8. Performance on AMD.

Ideal Scaling Difficult to Achieve for Weak Scaling

- After each parallelization step, there are all-reduce operations to calculate the total energy
- Switching between the black and white sublattices induces overhead for context-switching

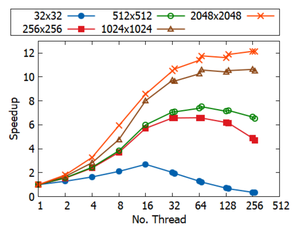


Fig 9. Performance on Intel Xeon Phi.

Table 2. Performance Summary for Million Size Largest System (2048x2048)

Multi-core Processor	Gain with Affinity More Speedup || Less Threads	Speedup (Optimal Threads)	Hyper-Threading	Multi-core Processor	Gain with Affinity More Speedup || Less Threads	Speedup (Optimal Threads)	Hyper-Threading
ARM Cavium	1.44x || 2x	13.99(128)	√	Intel Xeon Phi	1.03x || 1x	12.15(272)	√
ARM Fujitsu	1.06x || 6x	7.31(8)	X	Intel Cascade-lake	1.26x || 2.91x	4.26(36)	X
AMD	1.23x || 4x	5.57(64)	X	Intel skylake	1.29x || 1.78x	4.19(22)	X

Conclusions

- Optimal performance is obtained with placing threads close together
- ARM(Cavium) and Intel Xeon Phi chips are slower for serial code (due to lower clock rates) but achieve larger speedup when multithreaded
- Hyperthreading results in higher speedup on ARM(Cavium)-partial, Intel Xeon Phi-full, but not in others
- Larger systems provide more computational work, yielding better speedup
- Similar performance behavior is expected for higher dimensional systems and other domain decomposition applications, not only under MC context