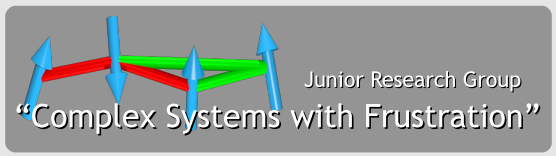
Monte Carlo methods for massively parallel architectures

Martin Weigel

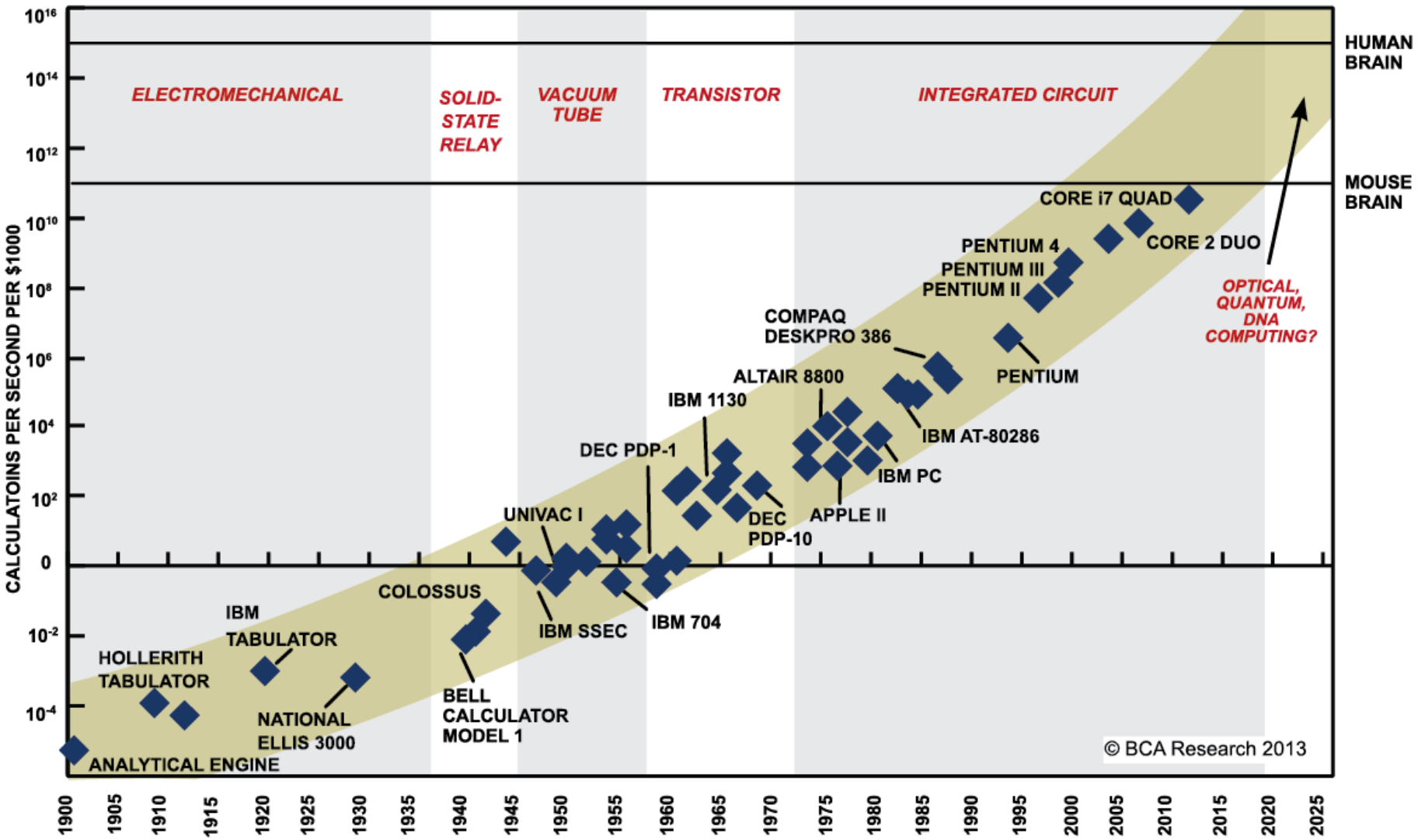
Applied Mathematics Research Centre, Coventry University, Coventry, United Kingdom

CSC at Lunch Seminar
Centre for Scientific Computing
University of Warwick, November 5, 2018.



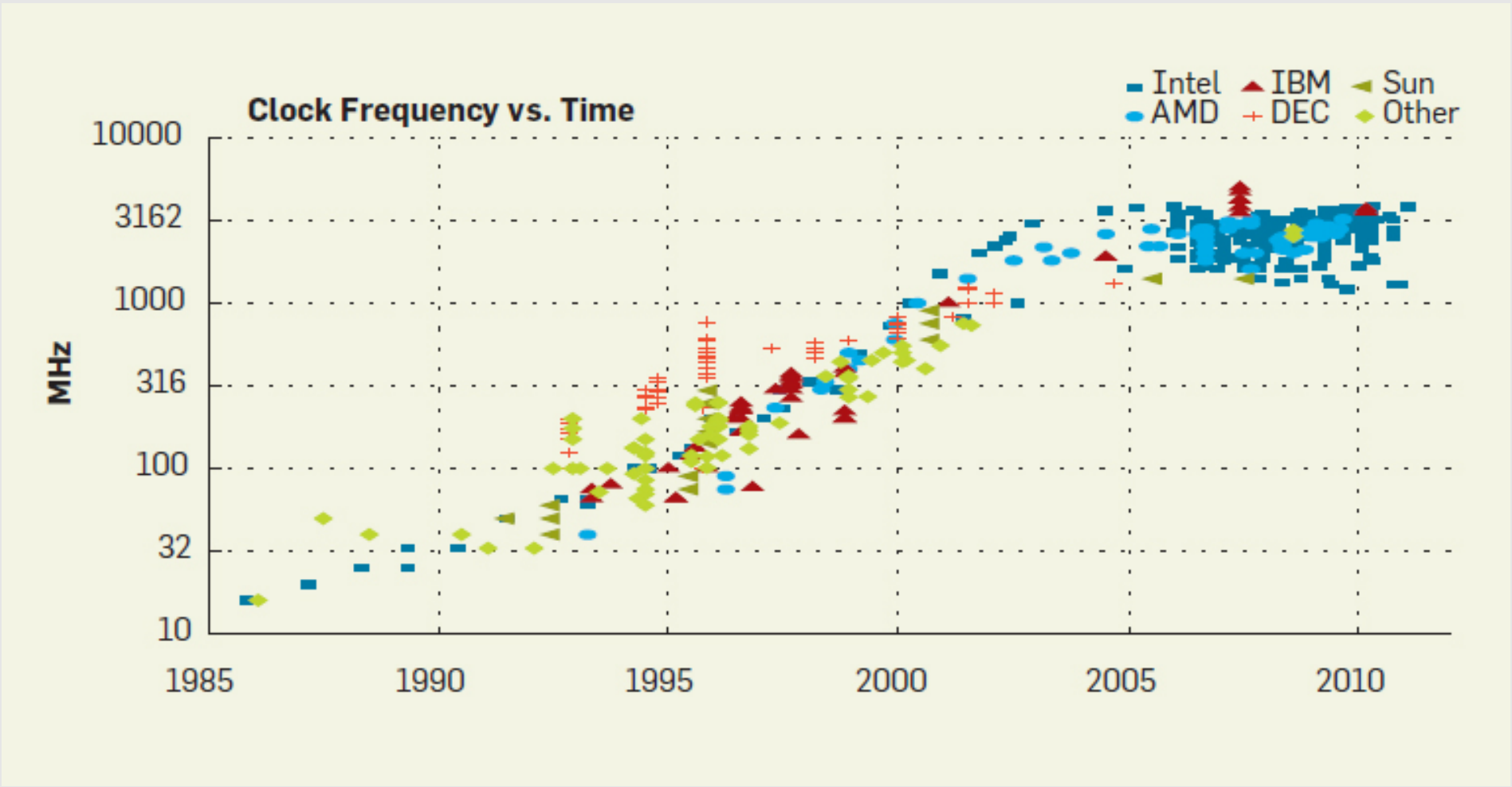
Parallel Computing and Monte Carlo

Moore's law

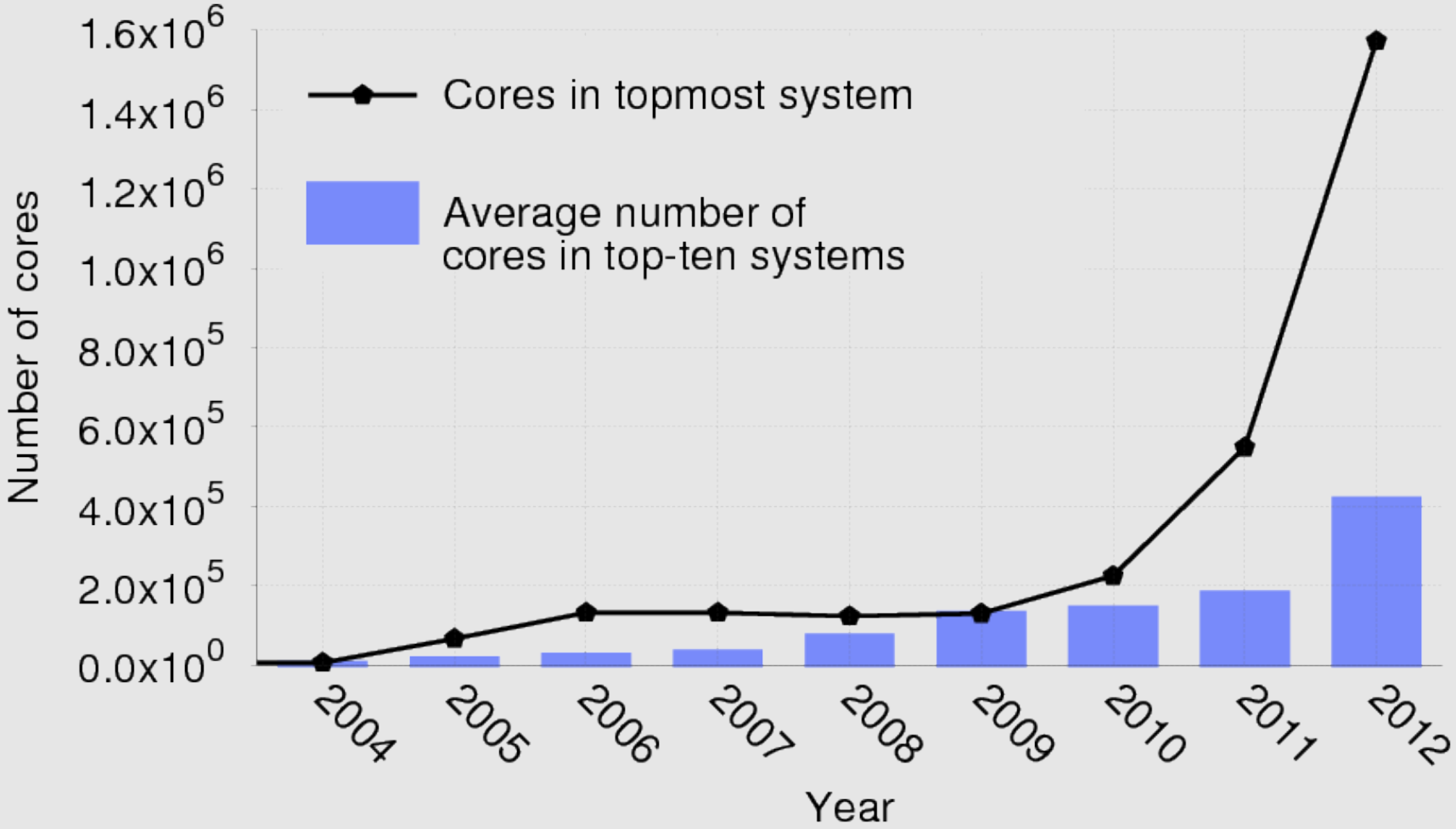


SOURCE: RAY KURZWEIL, "THE SINGULARITY IS NEAR: WHEN HUMANS TRANSCEND BIOLOGY", P.67, *THE VIKING PRESS*, 2006. DATAPOINTS BETWEEN 2000 AND 2012 REPRESENT BCA ESTIMATES.

Moore's law


Clock Frequency vs. Time
Intel
IBM
Sun
AMD
DEC
Other
MHz
10000
3162
1000
316
100
32
10
1985
1990
1995
2000
2005
2010

Moore's law


Number of cores
1.6x10^6
1.4x10^6
1.2x10^6
1.0x10^6
8.0x10^5
6.0x10^5
4.0x10^5
2.0x10^5
0.0x10^0
Cores in topmost system
Average number of cores in top-ten systems
2004
2005
2006
2007
2008
2009
2010
2011
2012
Year

Monte Carlo simulations

Most successful approach is importance sampling through Markov chains, an inherently sequential process.

Monte Carlo simulations

Most successful approach is importance sampling through Markov chains, an inherently sequential process.

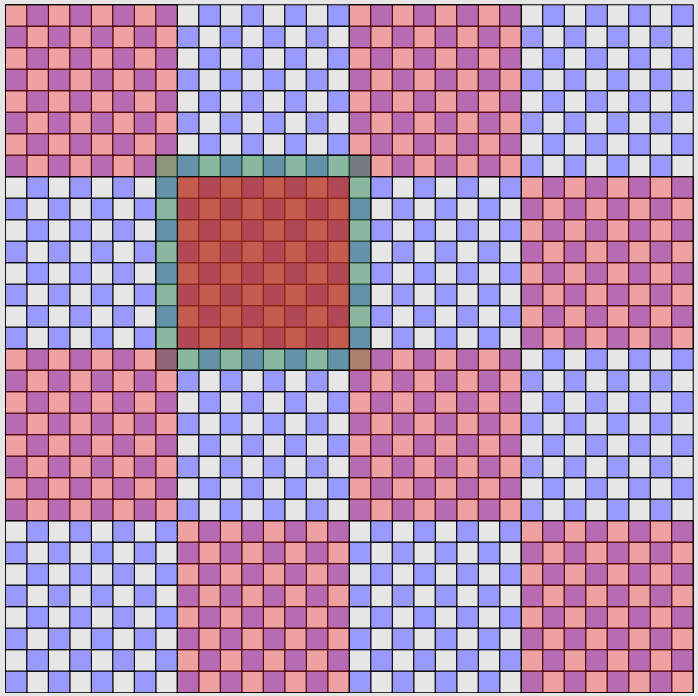
What to do it the era of parallel computing?

Monte Carlo simulations

Most successful approach is importance sampling through Markov chains, an inherently sequential process.

What to do it the era of parallel computing?

- use domain decompositions

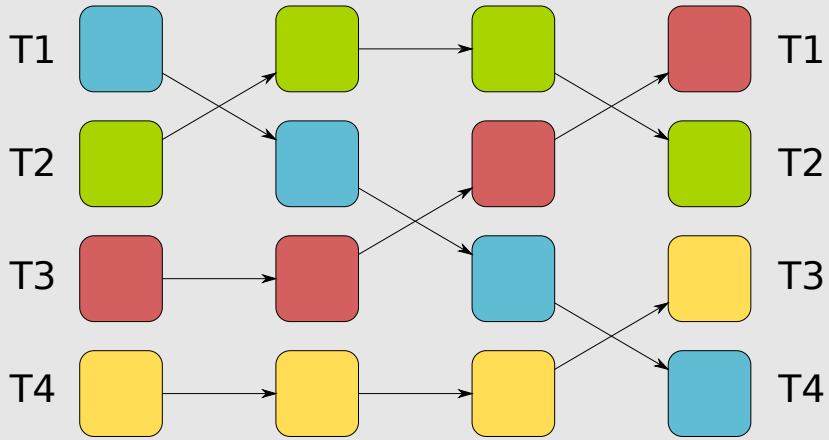


Monte Carlo simulations

Most successful approach is importance sampling through Markov chains, an inherently sequential process.

What to do it the era of parallel computing?

- use domain decompositions
- moderately parallel variants such as parallel tempering (Swendsen and Wang, 1986; Geyer, 1991; Hukushima and Nemoto, 1996)

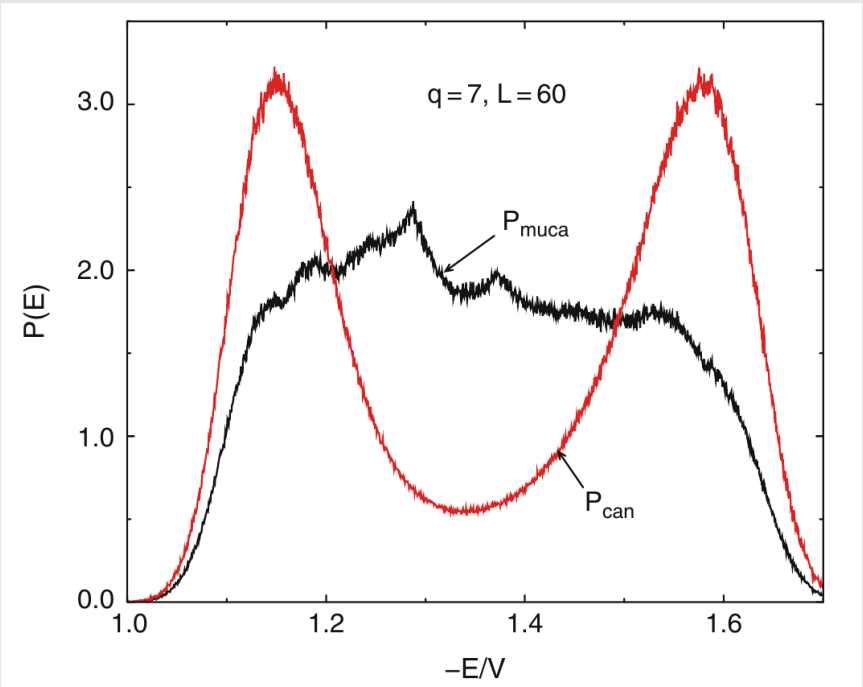


Monte Carlo simulations

Most successful approach is importance sampling through Markov chains, an inherently sequential process.

What to do it the era of parallel computing?

- use domain decompositions
- moderately parallel variants such as parallel tempering (Swendsen and Wang, 1986; Geyer, 1991; Hukushima and Nemoto, 1996)
- parallel multicanonical (Zierenberg et al., 2013) and Wang-Landau simulations (Vogel et al., 2013)

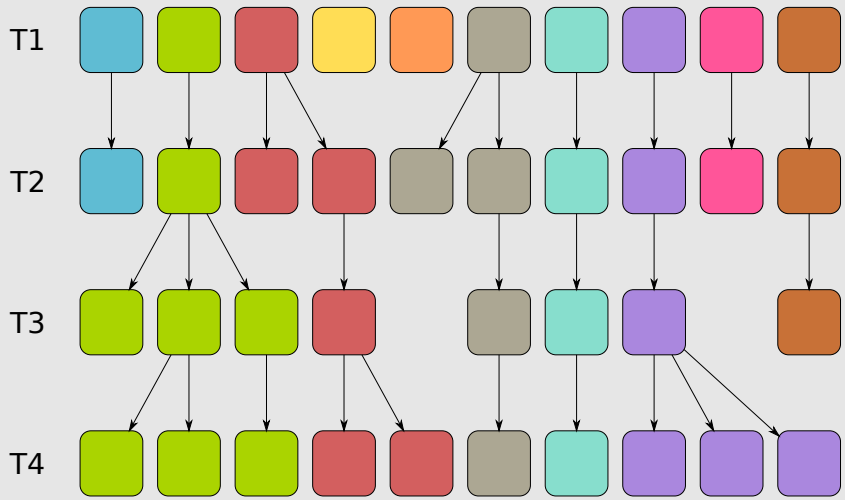


Monte Carlo simulations

Most successful approach is importance sampling through Markov chains, an inherently sequential process.

What to do it the era of parallel computing?

- use domain decompositions
- moderately parallel variants such as parallel tempering (Swendsen and Wang, 1986; Geyer, 1991; Hukushima and Nemoto, 1996)
- parallel multicanonical (Zierenberg et al., 2013) and Wang-Landau simulations (Vogel et al., 2013)
- population annealing method

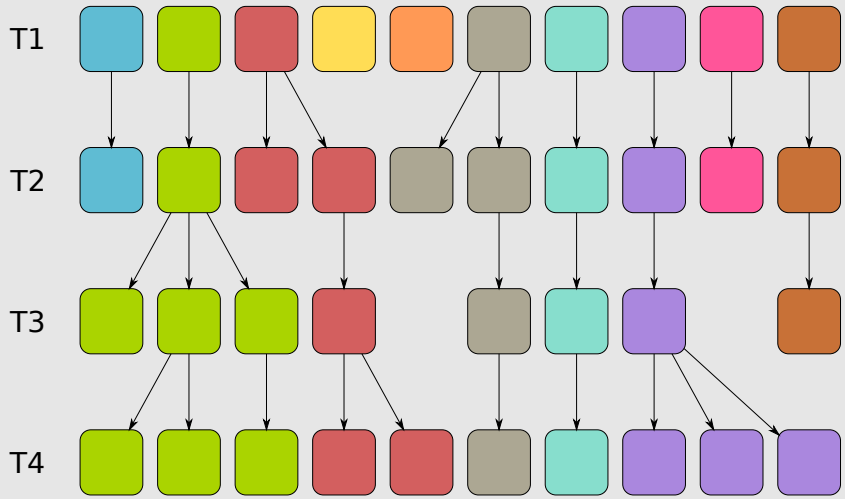


Monte Carlo simulations

Most successful approach is importance sampling through Markov chains, an inherently sequential process.

What to do it the era of parallel computing?

- use domain decompositions
- moderately parallel variants such as parallel tempering (Swendsen and Wang, 1986; Geyer, 1991; Hukushima and Nemoto, 1996)
- parallel multicanonical (Zierenberg et al., 2013) and Wang-Landau simulations (Vogel et al., 2013)
- population annealing method



Which methods work for 10^5 or 10^6 cores?

Canonical Monte Carlo

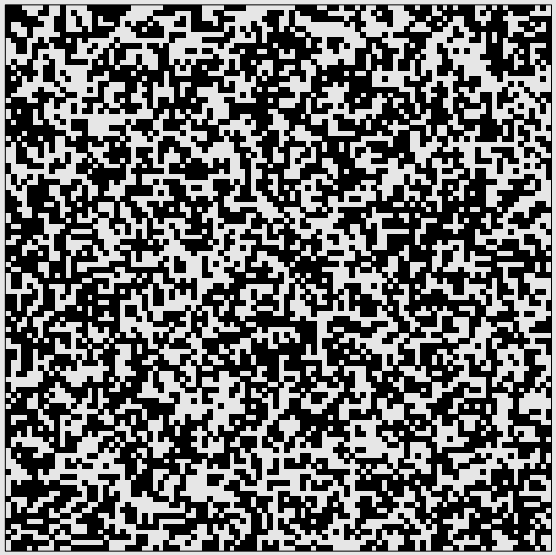
Benchmark: the 2D Ising model

Check results for the fruit fly of statistical mechanics, the 2D Ising model.

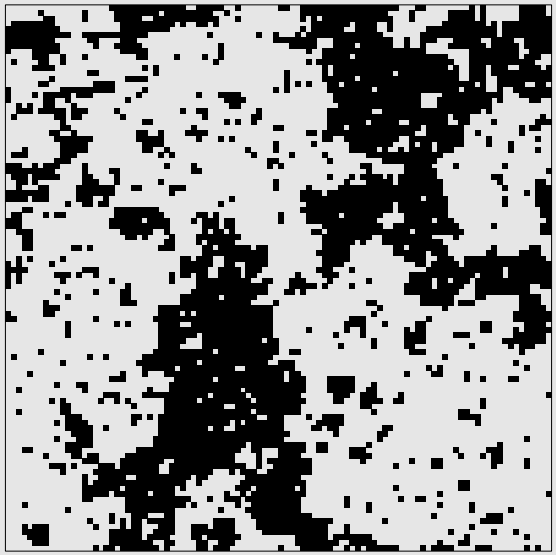
Hamiltonian

$$\mathcal{H} = -J \sum_{\langle i,j \rangle} s_i s_j, \quad s_i = \pm 1$$

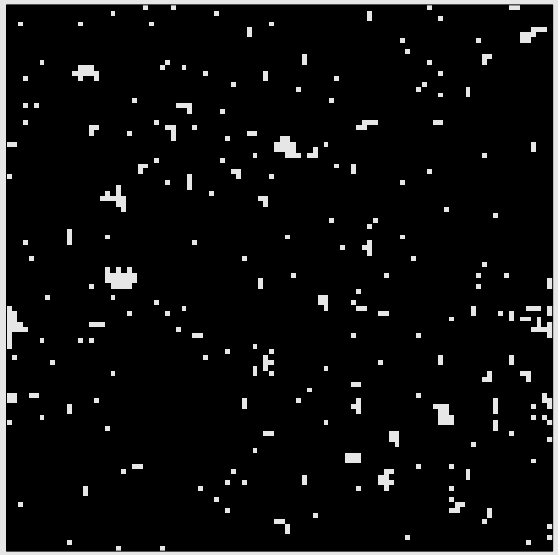
$T \gg T_c$



$T \approx T_c$



$T \ll T_c$



Parallel Metropolis

Consider spin models on regular lattices, for instance

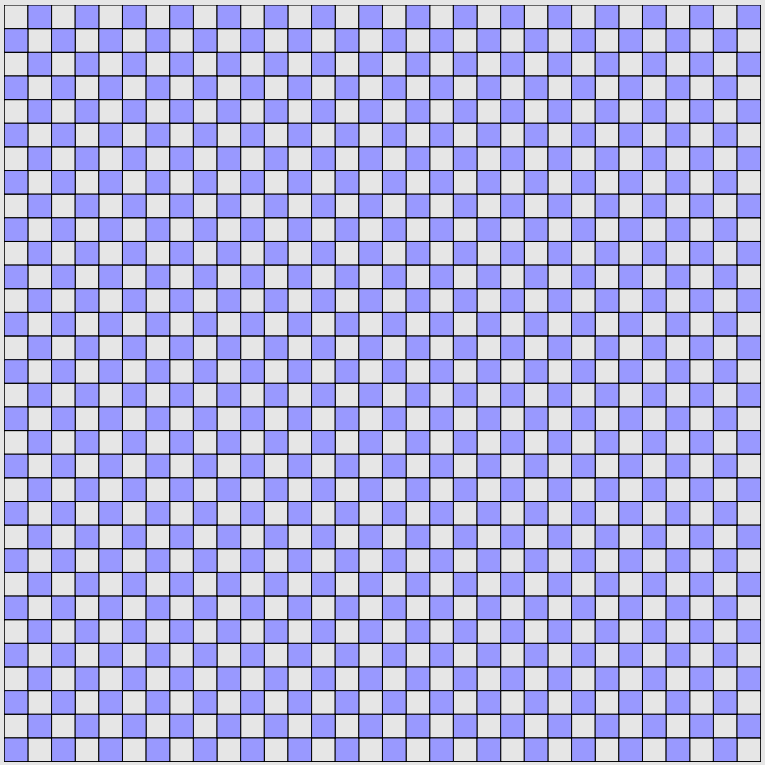
$$\mathcal{H} = -\sum_{\langle i,j \rangle} J_{ij} s_i s_j - \sum_i h_i s_i .$$

Parallel Metropolis

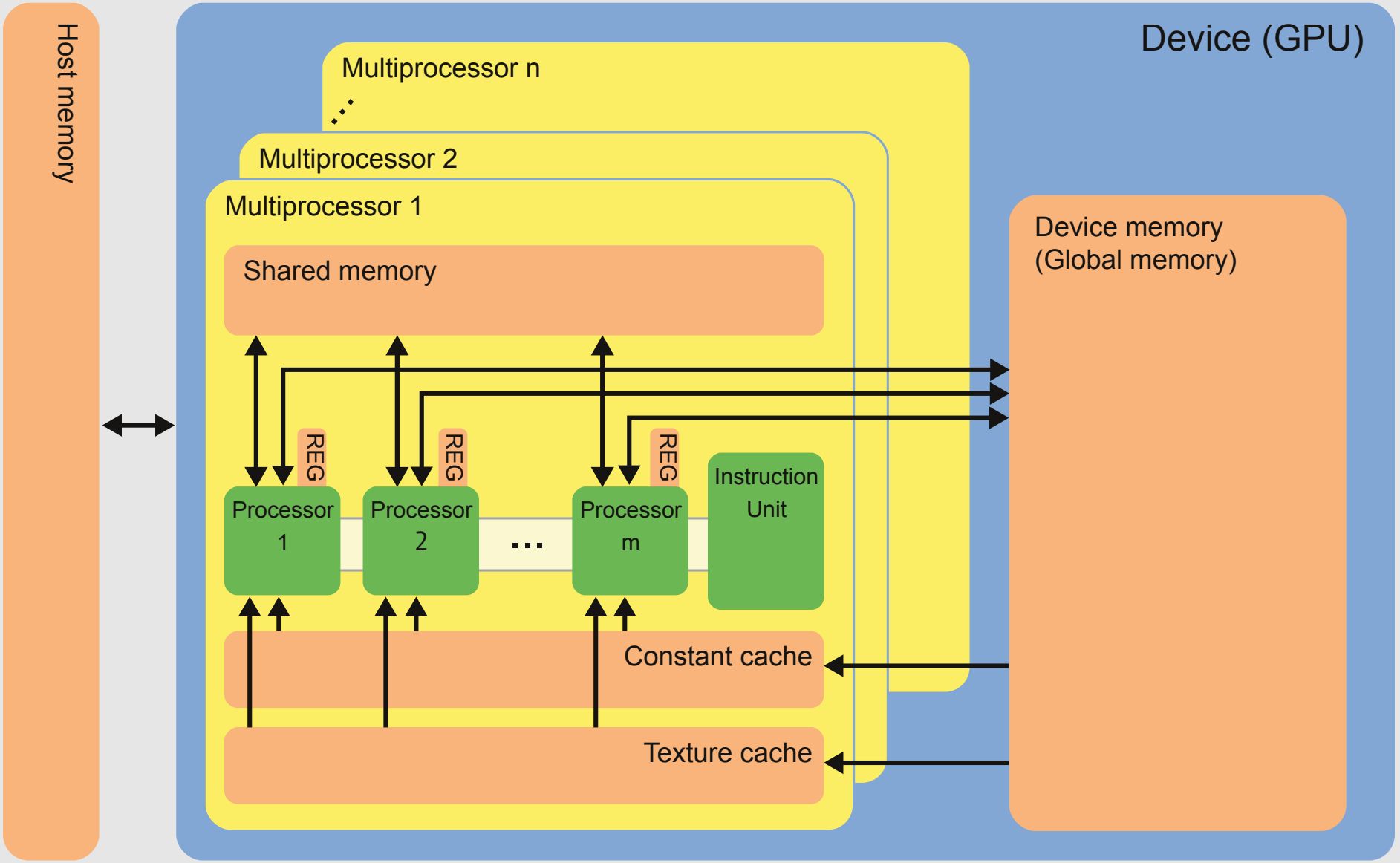
Consider spin models on regular lattices, for instance

$$\mathcal{H} = -\sum_{\langle i,j \rangle} J_{ij} s_i s_j - \sum_i h_i s_i .$$

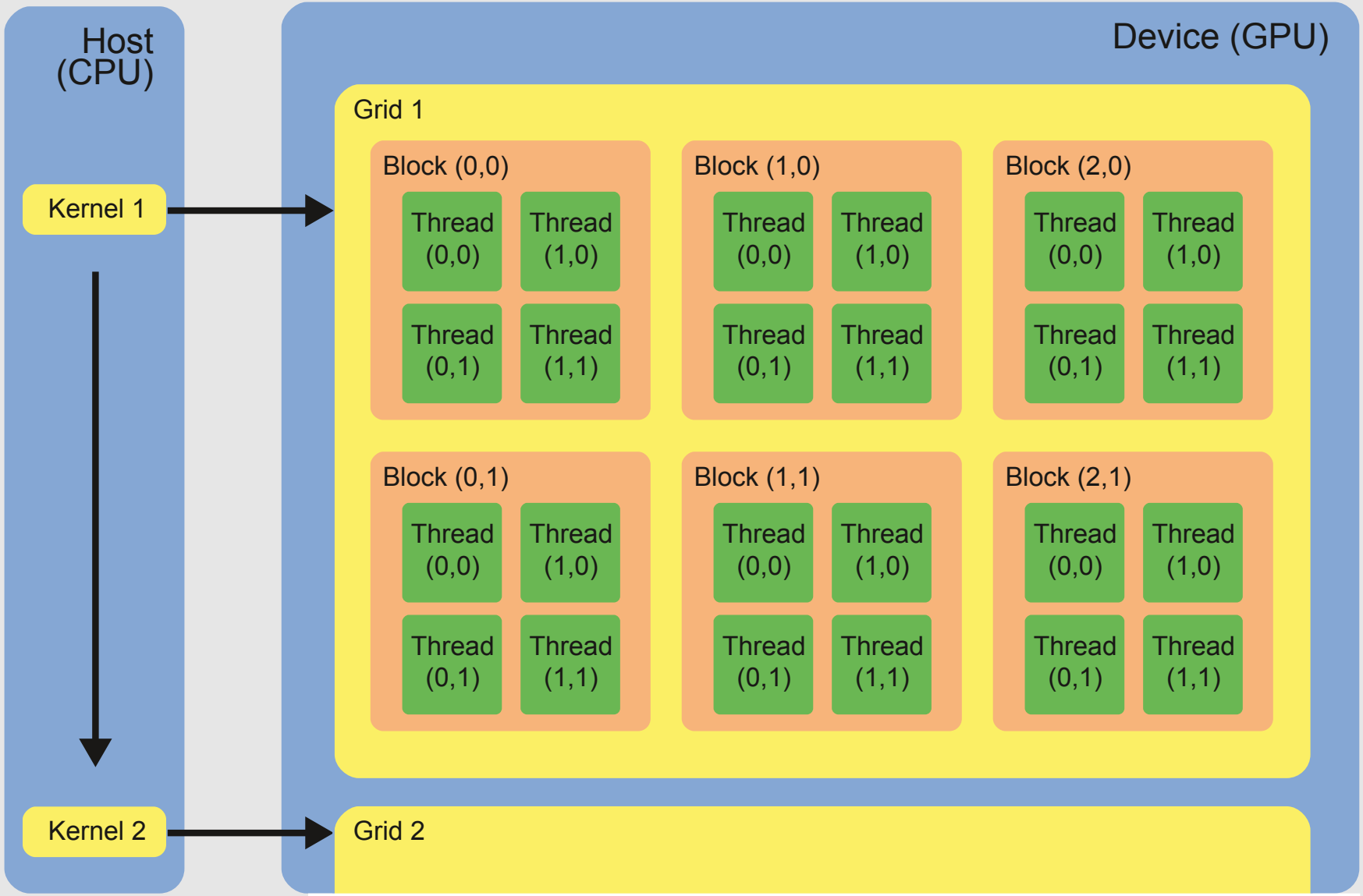
For short-range interactions, we can use a checkerboard decomposition.



NVIDIA architecture

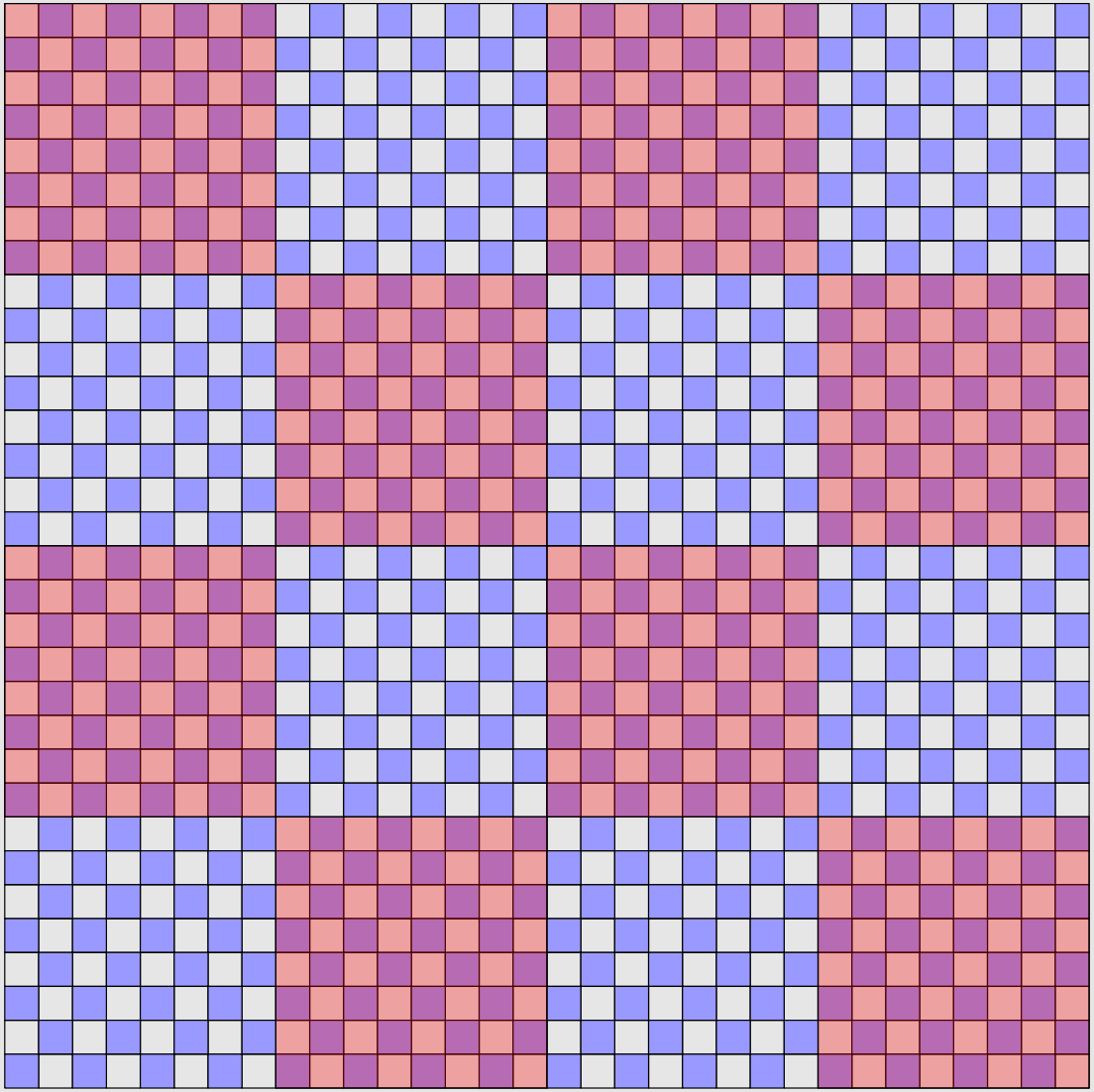


NVIDIA architecture



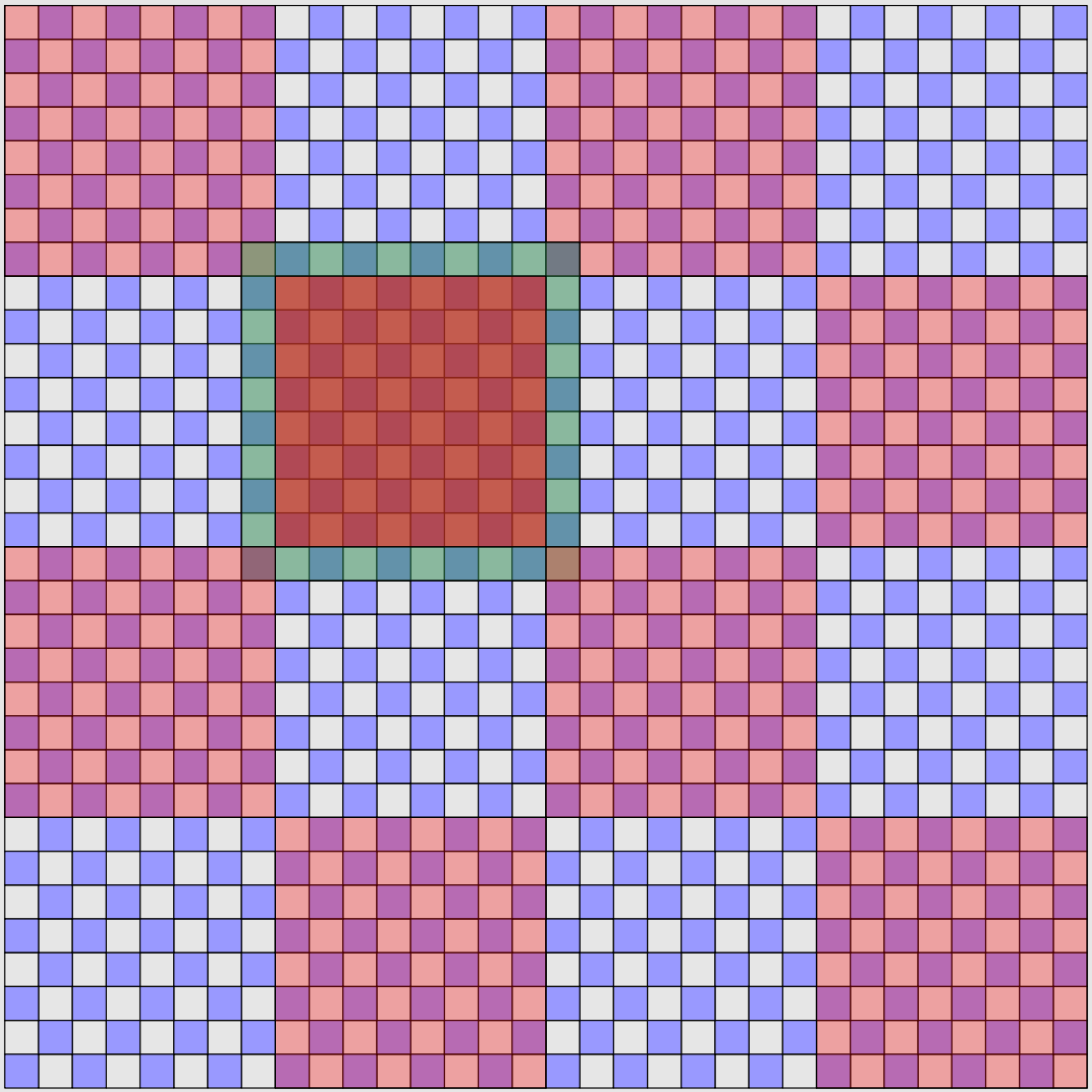
Checkerboard decomposition

- (red) large tiles: thread blocks
- (red) small tiles: individual threads

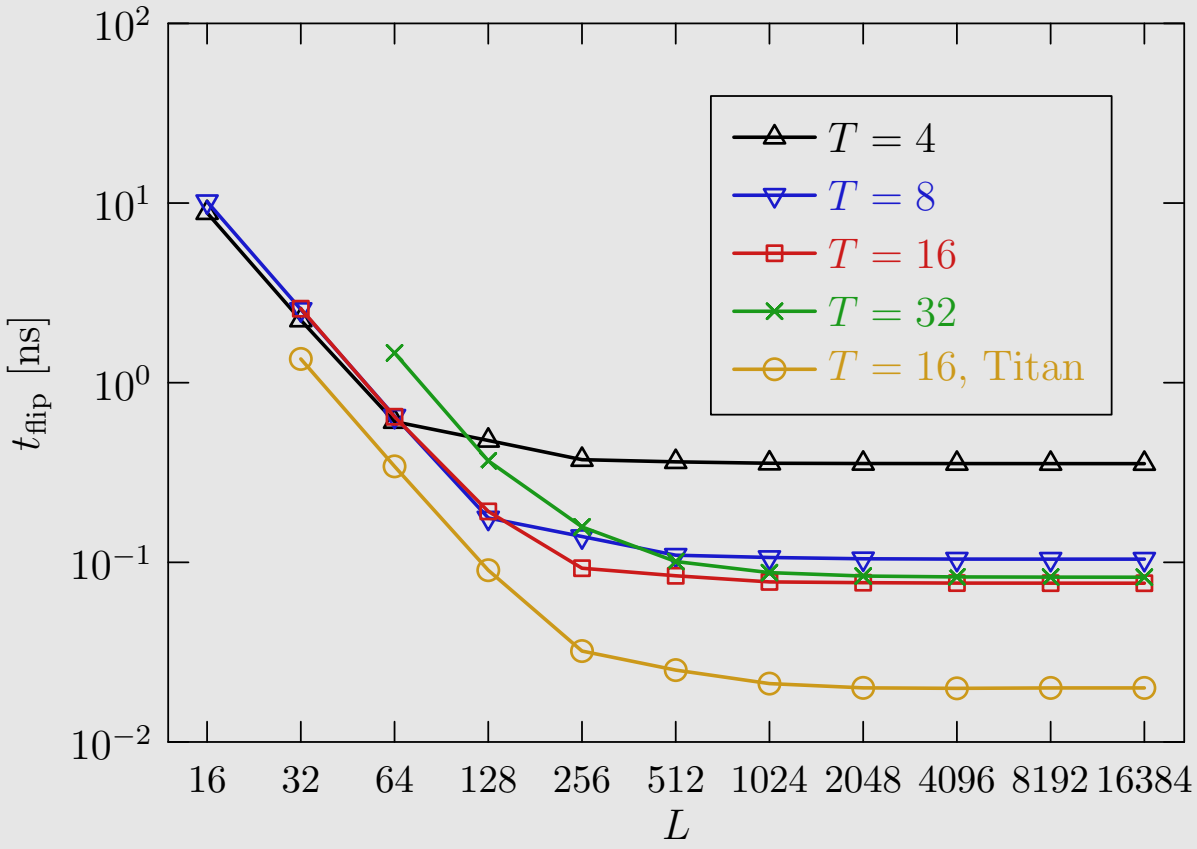


Checkerboard decomposition

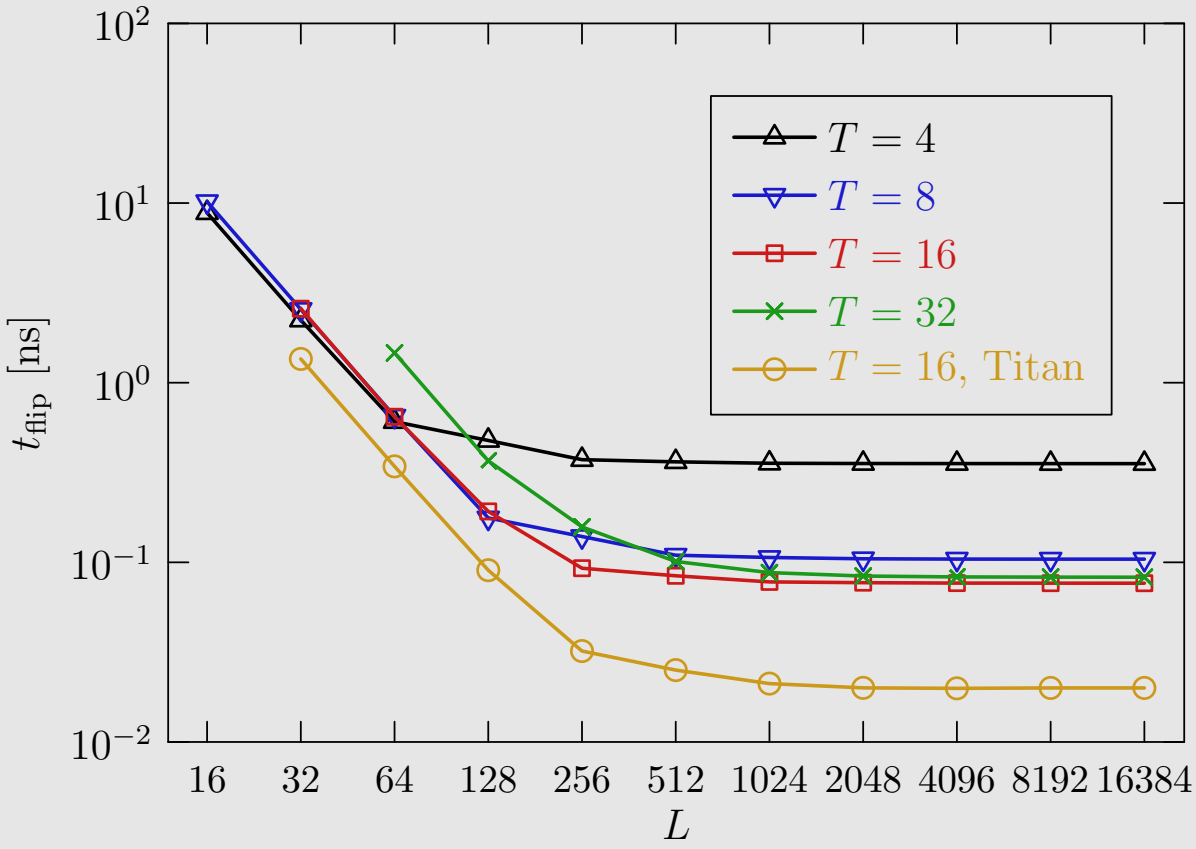
- (red) large tiles: thread blocks
- (red) small tiles: individual threads
- load one large tile (plus boundary) into shared memory
- perform several spin updates per tile



Performance

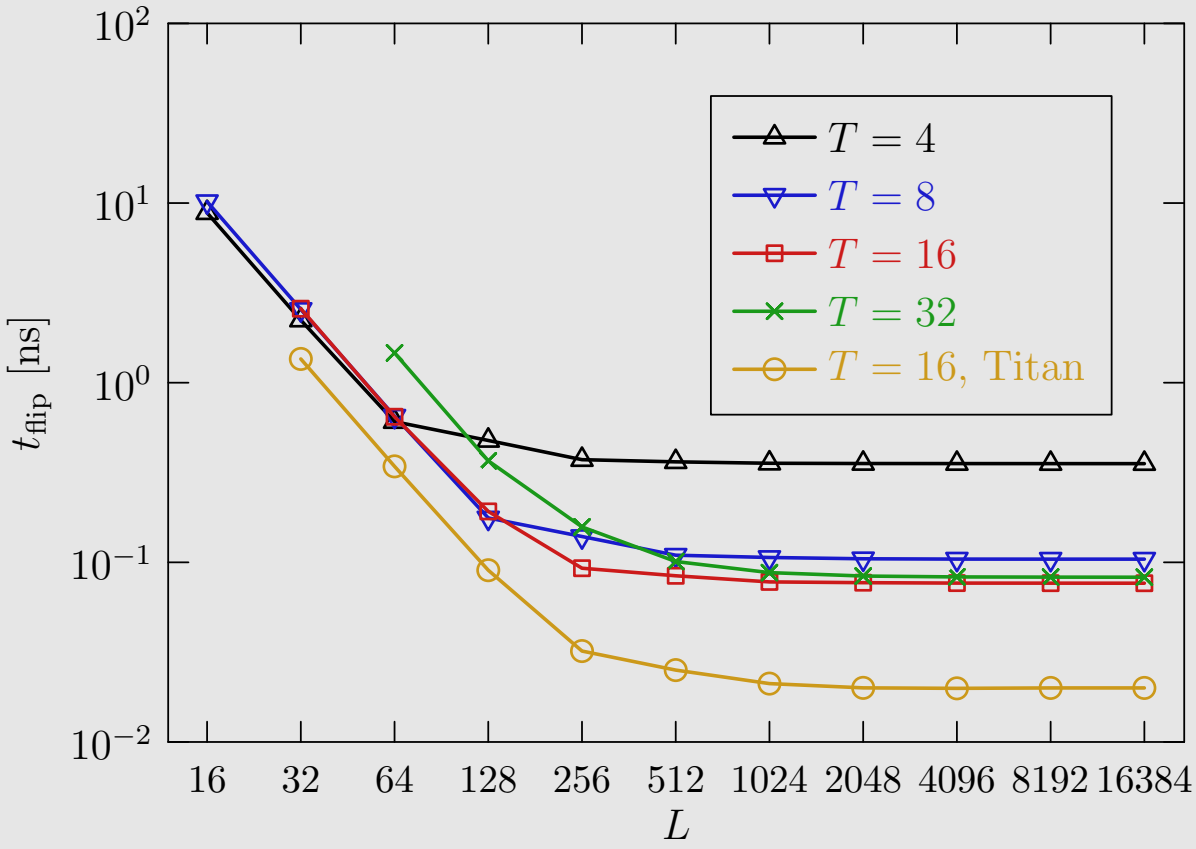


Performance



For sufficiently large lattices, one achieves spin-flip times as low as 20 ps, about 250 times faster than a single CPU core.

Performance



For sufficiently large lattices, one achieves spin-flip times as low as 20 ps, about 250 times faster than a single CPU core.

The number of threads is limited by the number of spins.

Multicanonical simulations

Multicanonical simulations

Generalized ensembles

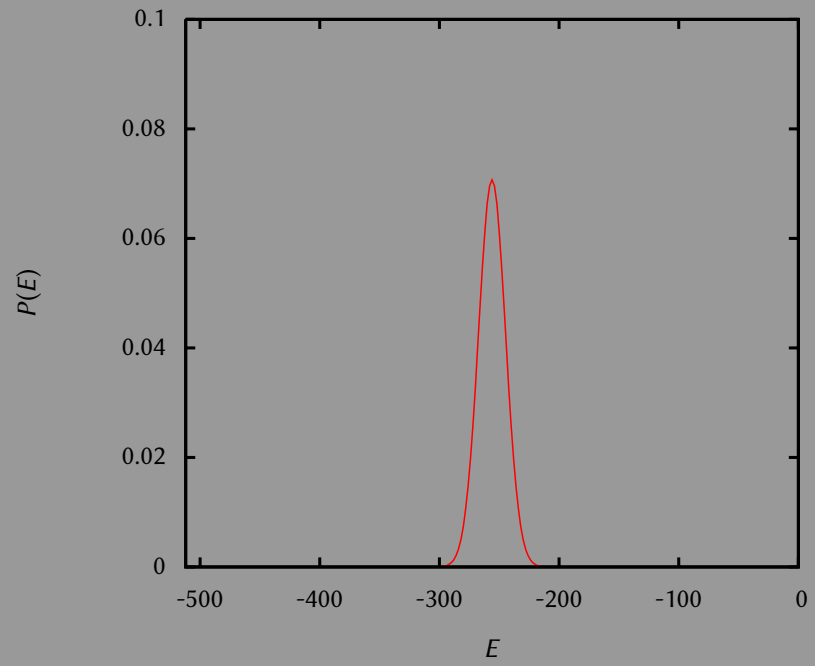
Instead of simulating the canonical distribution,

$$P_K(E) = \frac{1}{Z_K}\Omega(E)e^{-KE},$$

consider using a more general distribution

$$P_{\mathrm{muca}}(E) = \frac{\Omega(E)/W(E)}{Z_{\mathrm{muca}}} = \frac{\Omega(E)e^{-\omega(E)}}{Z_{\mathrm{muca}}},$$

engineered to overcome barriers, improve sampling speed and extend the reweighting range.



Multicanonical simulations

Choice of weights

To overcome barriers, we need to *broaden* $P(E)$, in the extremal case to a *constant* distribution,

$$P_{\text{muca}}(E) = Z_{\text{muca}}^{-1}\,\Omega(E)/W(E) = Z_{\text{muca}}^{-1}\,e^{S(E)-\omega(E)} \stackrel{!}{=} \text{const},$$

where $S(E) = \ln \Omega(E)$ is the microcanonical entropy.

Under these assumptions, $W(E) = \Omega(E)$ is optimal, i.e., we again desire to estimate the density of states. This is not known a priori, so (again) use histogram estimator

$$\hat{\Omega}(E) = Z_{\text{muca}}\,\hat{H}_{\text{muca}}(E)/N \times e^{\omega(E)}.$$

Canonical averages can be recovered at any time by reweighting:

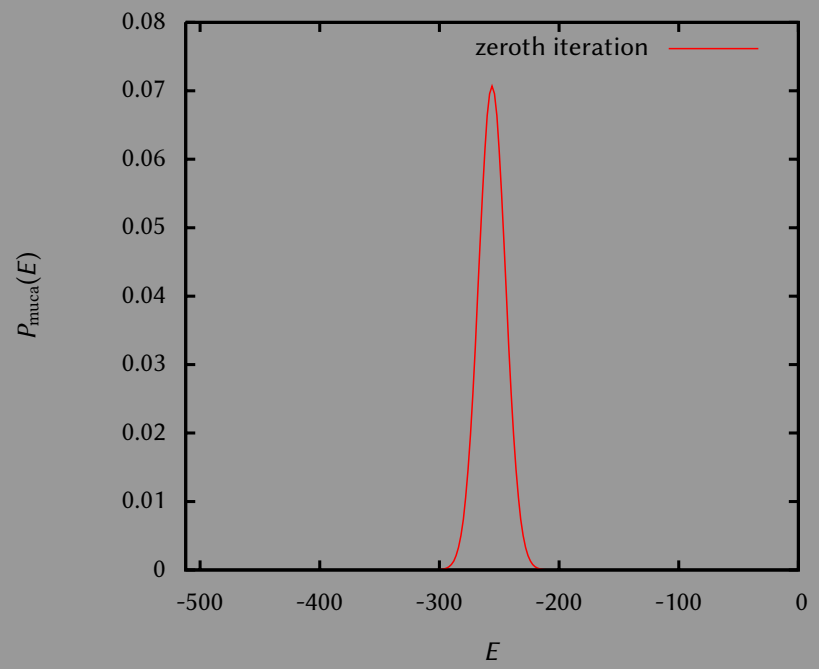
$$\langle A \rangle_K = \frac{\sum_E A(E) P_K(E)/P_{\text{muca}}(E)}{\sum_E P_K(E)/P_{\text{muca}}(E)}$$

Multicanonical simulations

Muca iteration

Determine muca weights/density of states iteratively:

1. Use, e.g., a $K = 0$ canonical simulation to get initial estimate $\hat{S}_0(E) = \ln \hat{\Omega}_0(E)$.

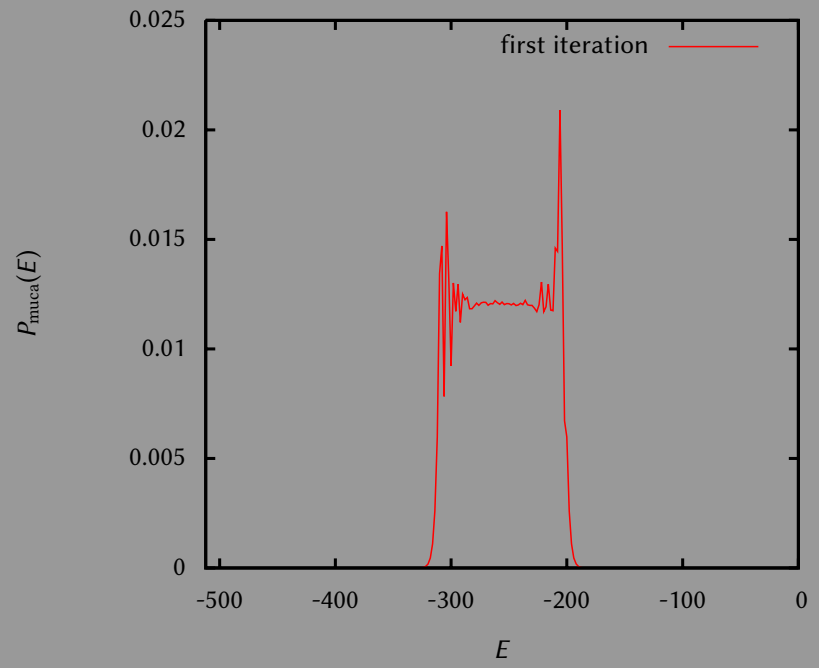


Multicanonical simulations

Muca iteration

Determine muca weights/density of states iteratively:

1. Use, e.g., a $K = 0$ canonical simulation to get initial estimate $\hat{S}_0(E) = \ln \hat{\Omega}_0(E)$.
2. Choose multicanonical weights $\omega_1(E) = \hat{S}_0(E)$ for next simulation.

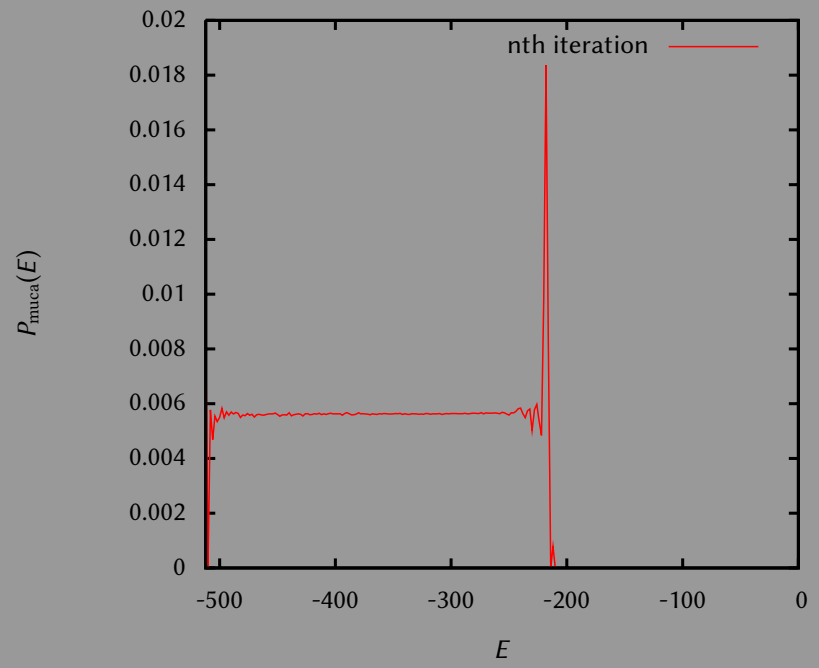


Multicanonical simulations

Muca iteration

Determine muca weights/density of states iteratively:

1. Use, e.g., a $K = 0$ canonical simulation to get initial estimate $\hat{S}_0(E) = \ln \hat{\Omega}_0(E)$.
2. Choose multicanonical weights $\omega_1(E) = \hat{S}_0(E)$ for next simulation.
3. Iterate.

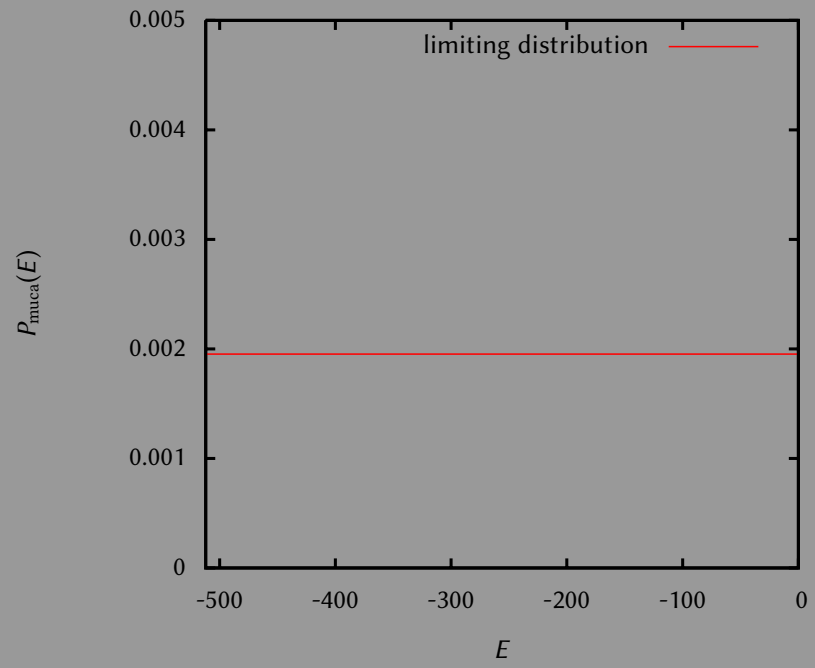


Multicanonical simulations

Muca iteration

Advantages:

- always in equilibrium
- arbitrary distributions possible
- system ideally performs an unbiased random walk in energy space $\rightarrow$ fast(er) dynamics

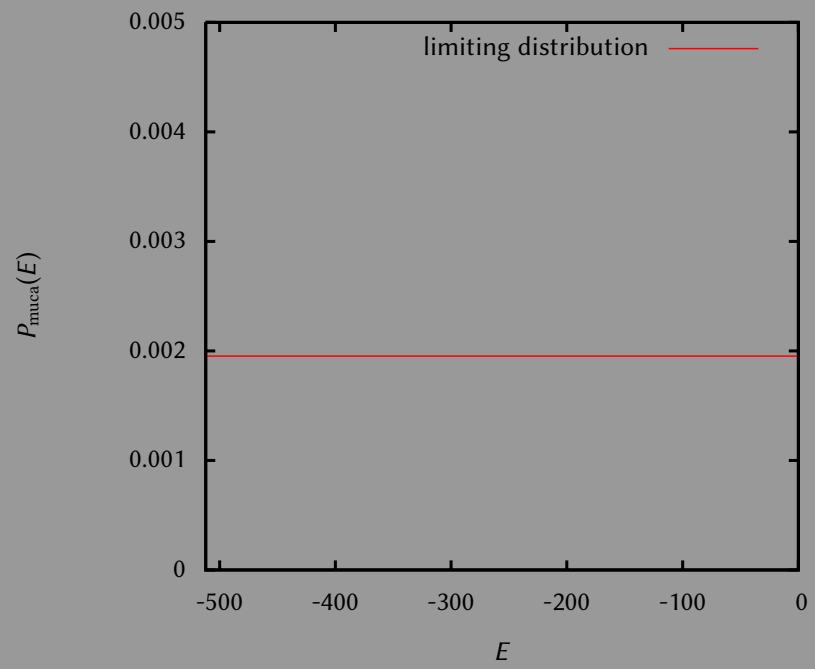


Multicanonical simulations

Multicanonical simulations

Variants:

- umbrella sampling, entropic sampling (identical)
- multiple Gaussian modified ensemble
- (broad histogram method)
- transition-matrix Monte Carlo
- metadynamics
- Wang-Landau sampling
- ...



Multicanonical simulations

Wang-Landau sampling

Muca weights are updated as

$$\omega_{i+1}(E) - \omega_i(E) = \text{const} + \ln \hat{H}_i(E),$$

i.e., if an energy E is visited more often than others, it receives *less* weight in future iterations.

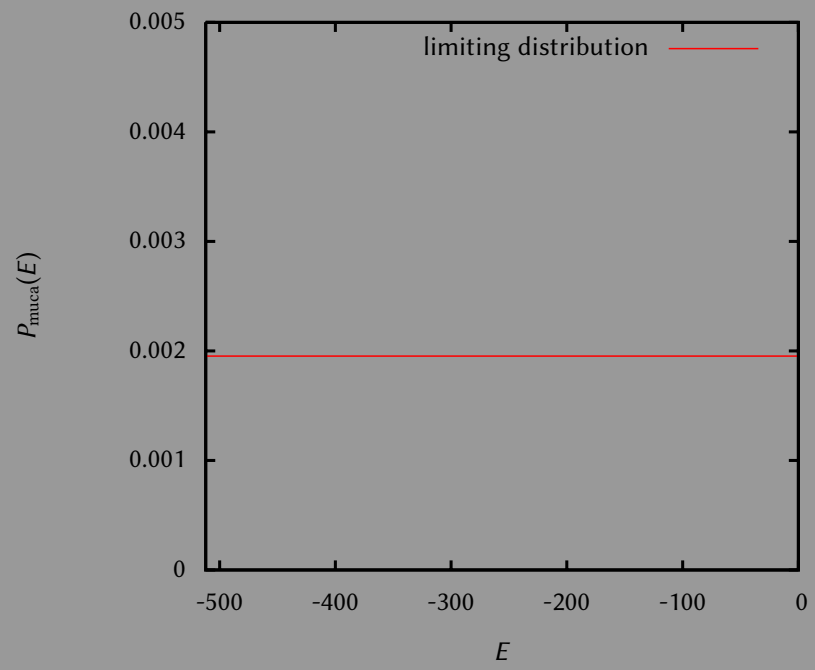
This behavior can be imitated in a one-step iteration: simulate

$$P_{\text{WL}}(E) \propto \Omega(E) e^{-\omega(E)},$$

but update

$$\omega_{i+1}(E) - \omega_i(E) = \phi,$$

each time an energy E is seen.



Multicanonical simulations

Wang-Landau sampling

Muca weights are updated as

$$\omega_{i+1}(E) - \omega_i(E) = \text{const} + \ln \hat{H}_i(E),$$

i.e., if an energy E is visited more often than others, it receives *less* weight in future iterations.

This behavior can be imitated in a one-step iteration: simulate

$$P_{\text{WL}}(E) \propto \Omega(E) e^{-\omega(E)},$$

but update

$$\omega_{i+1}(E) - \omega_i(E) = \phi,$$

each time an energy E is seen.

WL iteration

1. Start with $\omega(E) = 0 \; \forall E$.
2. Simulate "sufficiently long" while continuously updating $\omega(E)$.
3. Reduce modification factor, e.g.,
$$\phi \to \phi/2$$
4. Iterate till $\phi < \phi_{\text{thres}}$.

Multicanonical simulations

Use and justification

Different possible interpretations of this scheme:

- Rather efficient way of calculating muca weights.
- Standalone algorithm for estimating the density of states (convergence?).
- Violates detailed balance for any $\phi > 0$, but convergence can be proved as a stochastic approximation (instead of MCMC) algorithm for

$$\phi = \frac{t_0}{\max(t, t_0)} \sim \frac{1}{t}, \quad t > t_0$$

instead of

$$\phi = \phi_0 \, 2^{-t}$$

(cf. simulated annealing)

WL iteration

1. Start with $\omega(E) = 0 \; \forall E$.
2. Simulate "sufficiently long" while continuously updating $\omega(E)$.
3. Reduce modification factor, e.g.,

$$\phi \to \phi/2$$

4. Iterate till $\phi < \phi_{\text{thres}}$.

Parallel muca

Each update requires the value of the current energy to evaluate $W(E)/W(E')$, effectively serializing all spin flips!

Intrinsically serial algorithm?

Suggested ways out:

Parallel muca

Each update requires the value of the current energy to evaluate $W(E)/W(E')$, effectively serializing all spin flips!

Intrinsically serial algorithm?

Suggested ways out:

- divide the energy range in windows, or

Parallel muca

Each update requires the value of the current energy to evaluate $W(E)/W(E')$, effectively serializing all spin flips!

Intrinsically serial algorithm?

Suggested ways out:

- divide the energy range in windows, or
- use multiple walkers (`Zierenberg et al., 2013`)

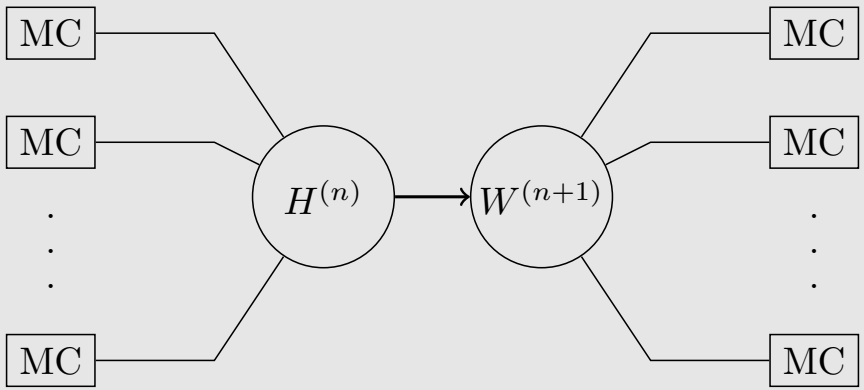
Parallel muca

Each update requires the value of the current energy to evaluate $W(E)/W(E')$, effectively serializing all spin flips!

Intrinsically serial algorithm?

Suggested ways out:

- divide the energy range in windows, or
- use multiple walkers (`Zierenberg et al., 2013`)



Parallel muca (cont'd)

Each walker samples its own histogram, all of them are combined for the next weight update,

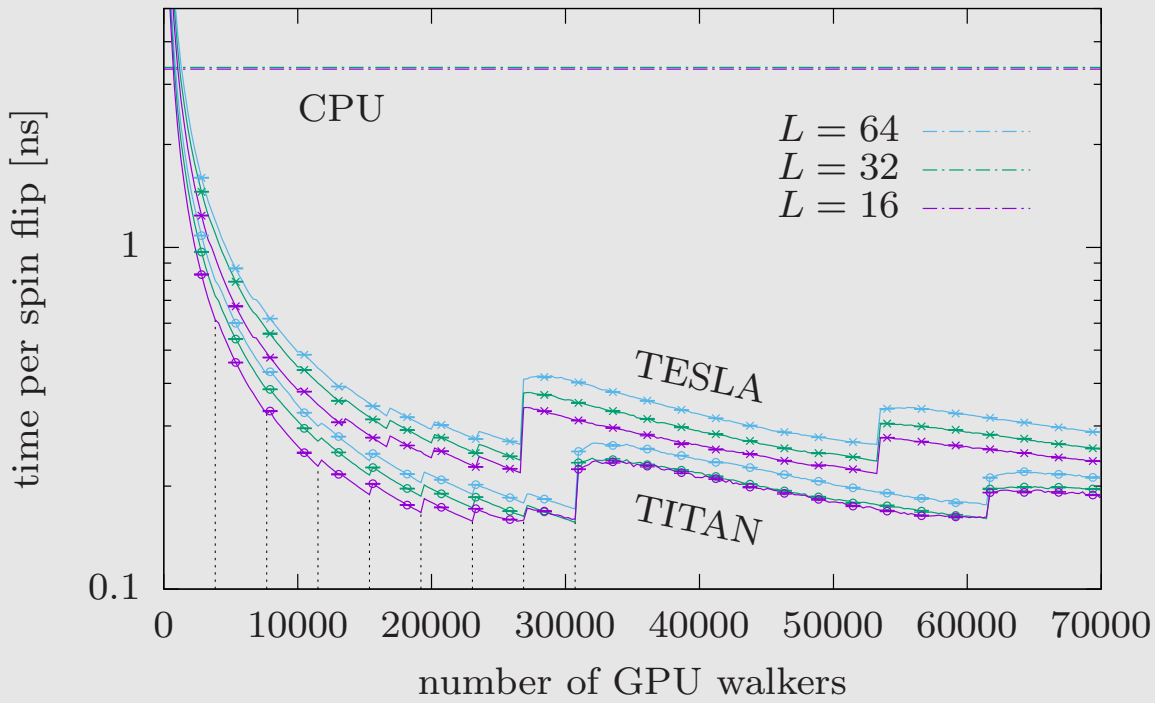
$$H^{(n)}(E) = \sum_i H_i^{(n)}(E).$$

Parallel muca (cont'd)

Each walker samples its own histogram, all of them are combined for the next weight update,

$$H^{(n)}(E) = \sum_i H_i^{(n)}(E).$$

This scheme can be efficiently implemented on MPI clusters (Zierenberg et al., 2013) and on GPUs.

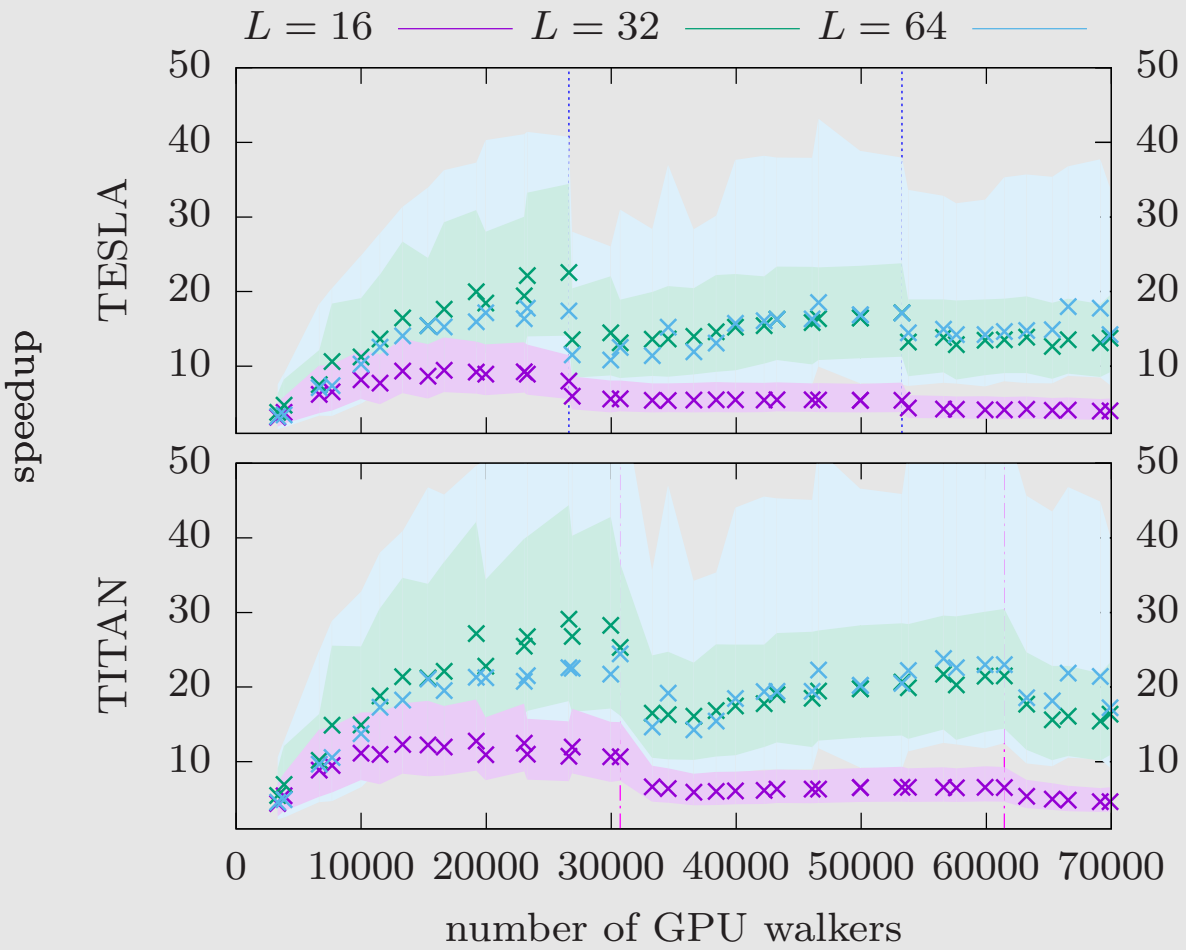


Parallel muca (cont'd)

Additional walkers lead to a faster convergence of the weight iteration.

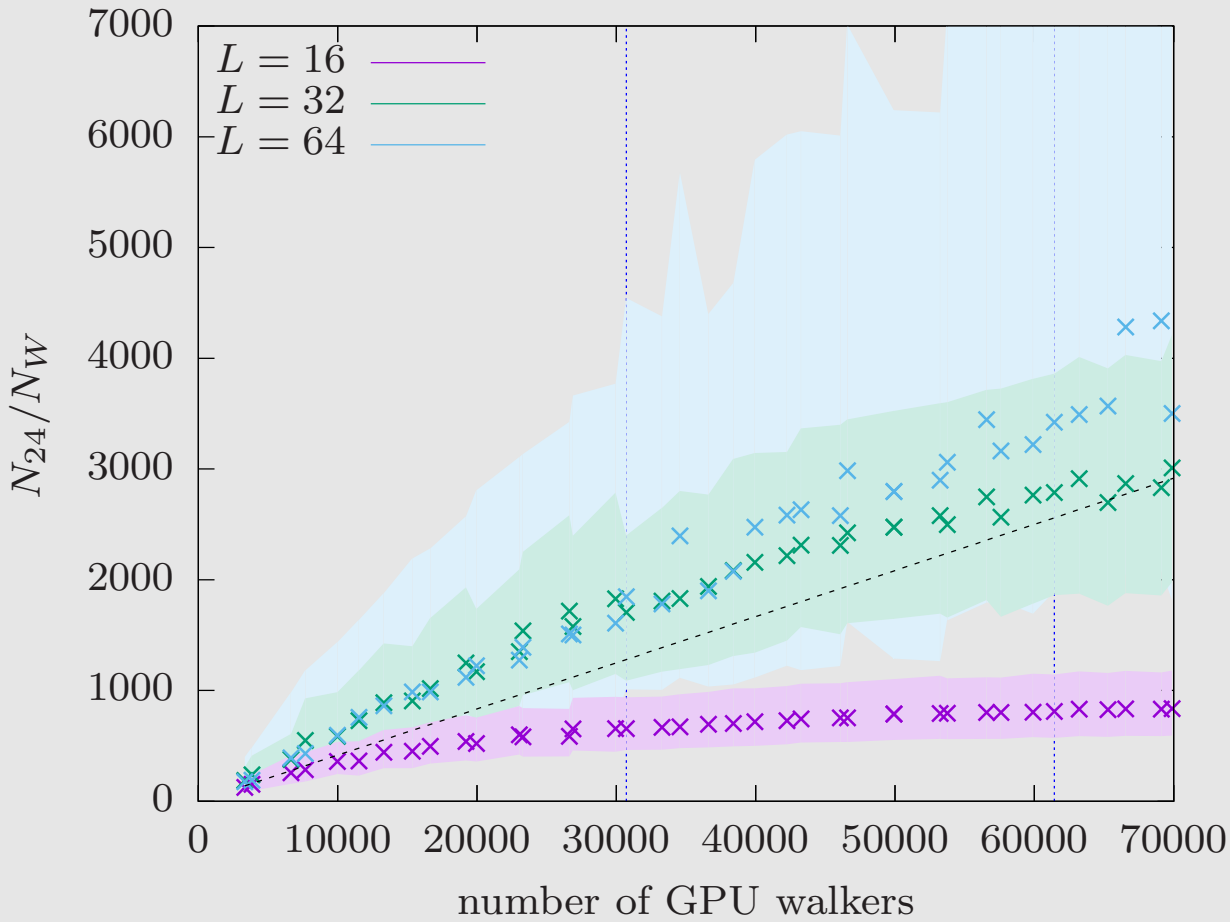
Parallel muca (cont'd)

Additional walkers lead to a faster convergence of the weight iteration.



Parallel muca (cont'd)

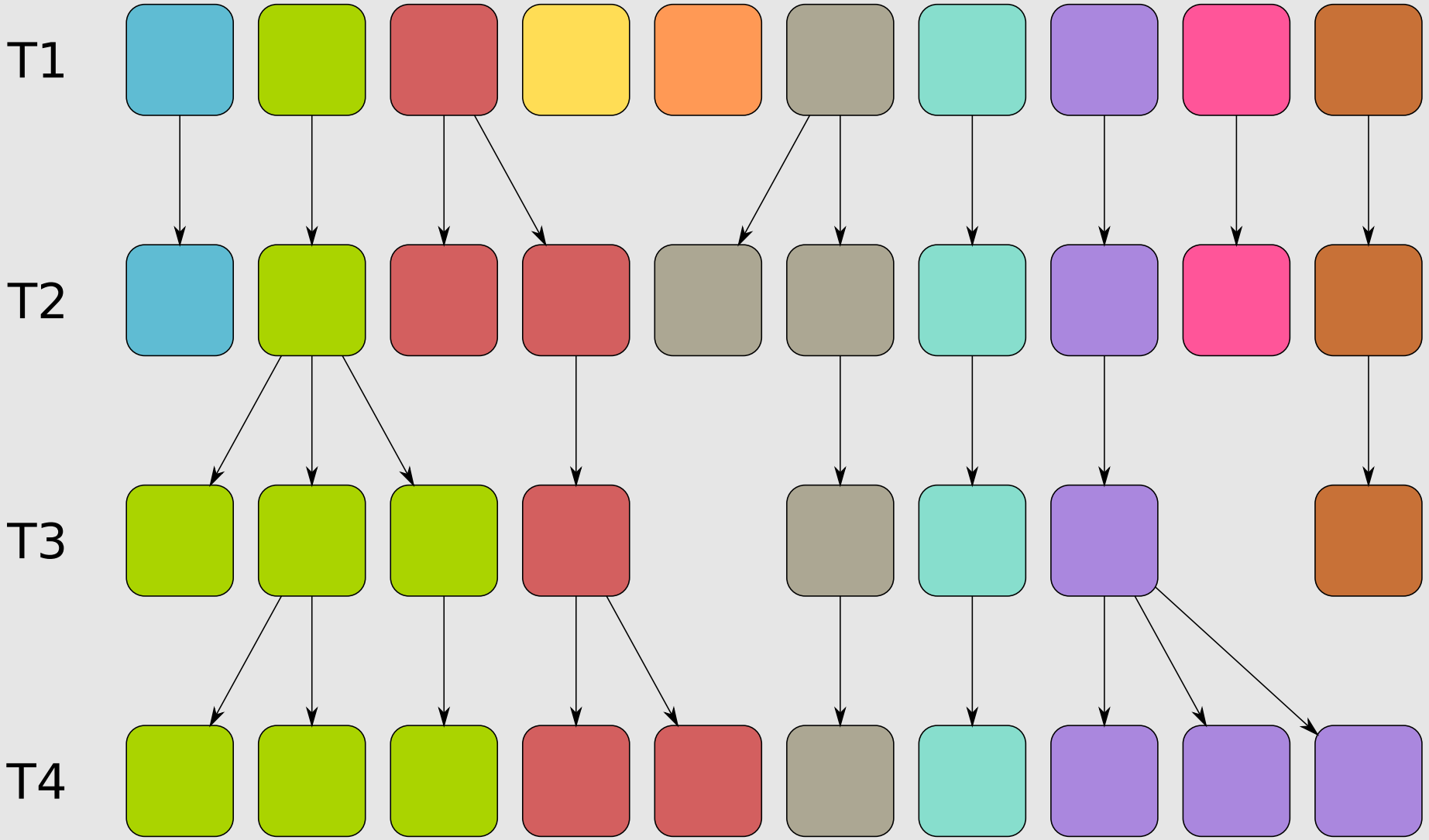
Additional walkers lead to a faster convergence of the weight iteration.



There is super-linear scaling at least up to 70, 000 threads.

Population Annealing

Population annealing



Population annealing

Population annealing algorithm (Hukushima + Iba, 2003; Machta, 2010):

1. Set up an equilibrium ensemble of R independent copies of the system at inverse temperature β_0. Typically $\beta_0 = 0$, where this can be easily achieved.

Population annealing

Population annealing algorithm (Hukushima + Iba, 2003; Machta, 2010):

1. Set up an equilibrium ensemble of R independent copies of the system at inverse temperature β_0. Typically $\beta_0 = 0$, where this can be easily achieved.
2. To create an approximately equilibrated sample at $\beta_i > \beta_{i-1}$, resample configurations with their relative Boltzmann weight $\exp[-(\beta_i - \beta_{i-1})E_j]/Q$, where $Q = \sum \exp(-(\beta_i - \beta_{i-1})E_j)$.

Population annealing

Population annealing algorithm (Hukushima + Iba, 2003; Machta, 2010):

1. Set up an equilibrium ensemble of R independent copies of the system at inverse temperature β_0. Typically $\beta_0 = 0$, where this can be easily achieved.
2. To create an approximately equilibrated sample at $\beta_i > \beta_{i-1}$, resample configurations with their relative Boltzmann weight $\exp[-(\beta_i - \beta_{i-1})E_j]/Q$, where $Q = \sum \exp(-(\beta_i - \beta_{i-1})E_j)$.
3. Update each copy (replica) by θ rounds of an MCMC algorithm at inverse temperature β_i.

Population annealing

Population annealing algorithm (Hukushima + Iba, 2003; Machta, 2010):

1. Set up an equilibrium ensemble of R independent copies of the system at inverse temperature β_0. Typically $\beta_0 = 0$, where this can be easily achieved.
2. To create an approximately equilibrated sample at $\beta_i > \beta_{i-1}$, resample configurations with their relative Boltzmann weight $\exp[-(\beta_i - \beta_{i-1})E_j]/Q$, where $Q = \sum \exp(-(\beta_i - \beta_{i-1})E_j)$.
3. Update each copy (replica) by θ rounds of an MCMC algorithm at inverse temperature β_i.
4. Calculate estimates for observable quantities $\mathcal{O}$ as population averages $\sum_j \mathcal{O}_j/R$.

Population annealing

Population annealing algorithm (Hukushima + Iba, 2003; Machta, 2010):

1. Set up an equilibrium ensemble of R independent copies of the system at inverse temperature β_0. Typically $\beta_0 = 0$, where this can be easily achieved.
2. To create an approximately equilibrated sample at $\beta_i > \beta_{i-1}$, resample configurations with their relative Boltzmann weight $\exp[-(\beta_i - \beta_{i-1})E_j]/Q$, where $Q = \sum \exp(-(\beta_i - \beta_{i-1})E_j)$.
3. Update each copy (replica) by θ rounds of an MCMC algorithm at inverse temperature β_i.
4. Calculate estimates for observable quantities $\mathcal{O}$ as population averages $\sum_j \mathcal{O}_j/R$.
5. Goto step 2 until target temperature is reached.

Population annealing

Population annealing algorithm (Hukushima + Iba, 2003; Machta, 2010):

1. Set up an equilibrium ensemble of R independent copies of the system at inverse temperature β_0. Typically $\beta_0 = 0$, where this can be easily achieved.
2. To create an approximately equilibrated sample at $\beta_i > \beta_{i-1}$, resample configurations with their relative Boltzmann weight $\exp[-(\beta_i - \beta_{i-1})E_j]/Q$, where $Q = \sum \exp(-(\beta_i - \beta_{i-1})E_j)$.
3. Update each copy (replica) by θ rounds of an MCMC algorithm at inverse temperature β_i.
4. Calculate estimates for observable quantities $\mathcal{O}$ as population averages $\sum_j \mathcal{O}_j/R$.
5. Goto step 2 until target temperature is reached.

The MCMC is not strictly necessary but significant for the overall performance.

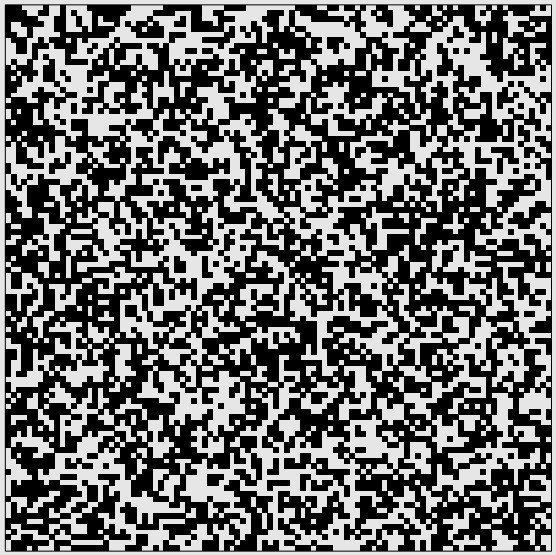
Benchmark: the 2D Ising model

Check results for the fruit fly of statistical mechanics, the 2D Ising model.

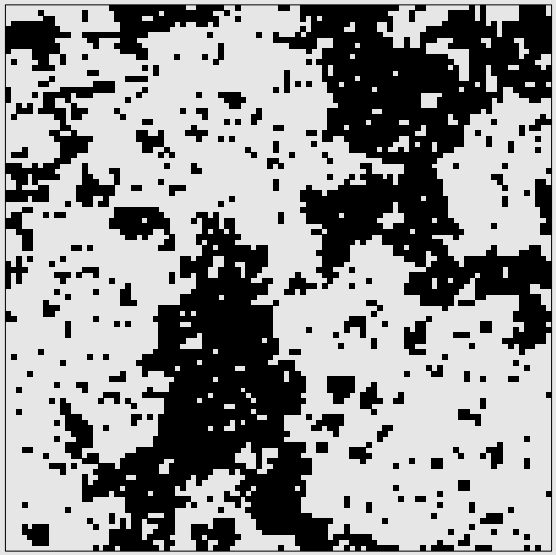
Hamiltonian

$$\mathcal{H} = -J \sum_{\langle i,j \rangle} s_i s_j, \quad s_i = \pm 1$$

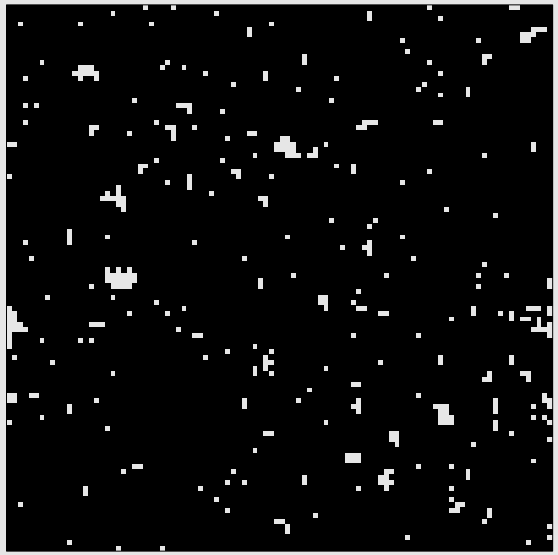
$T \gg T_c$



$T \approx T_c$

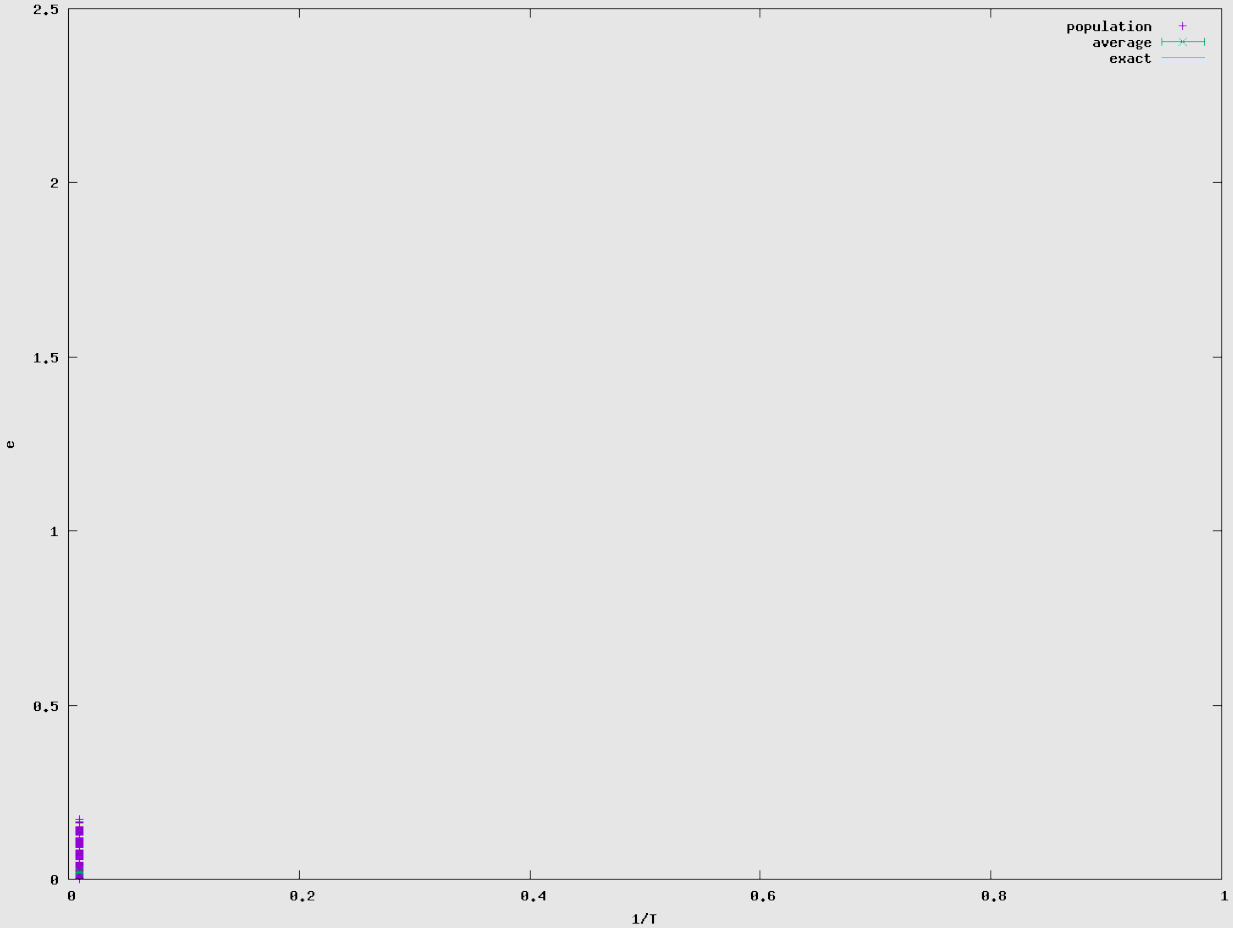


$T \ll T_c$



Population annealing

A sequential annealing of the population from infinite temperature, $\beta = 0$, down to $\beta = 1$.

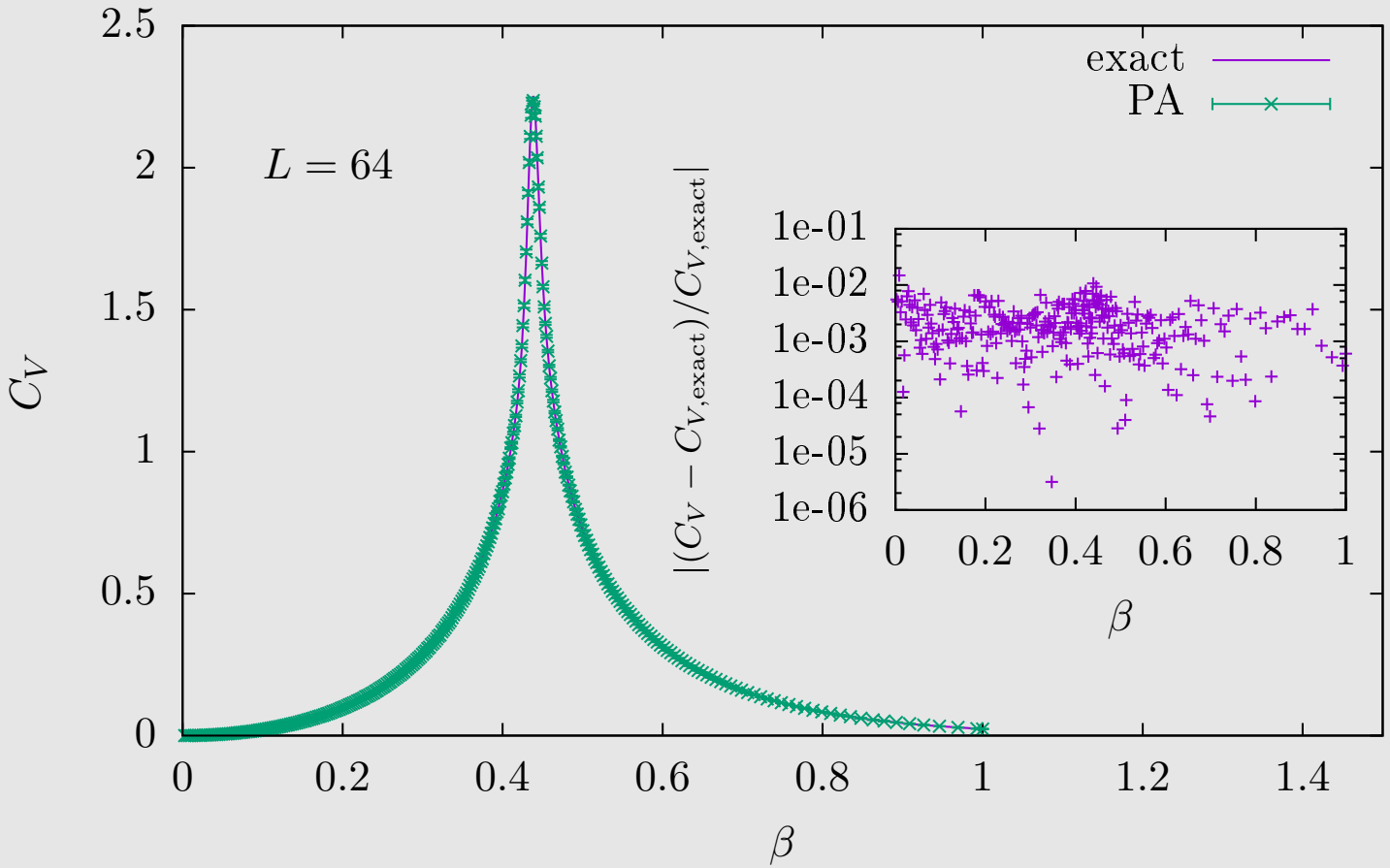


Correct results?

Exact results are available for finite lattices for the internal energy, specific heat and free energy (Ferdinand + Fisher, 1969).

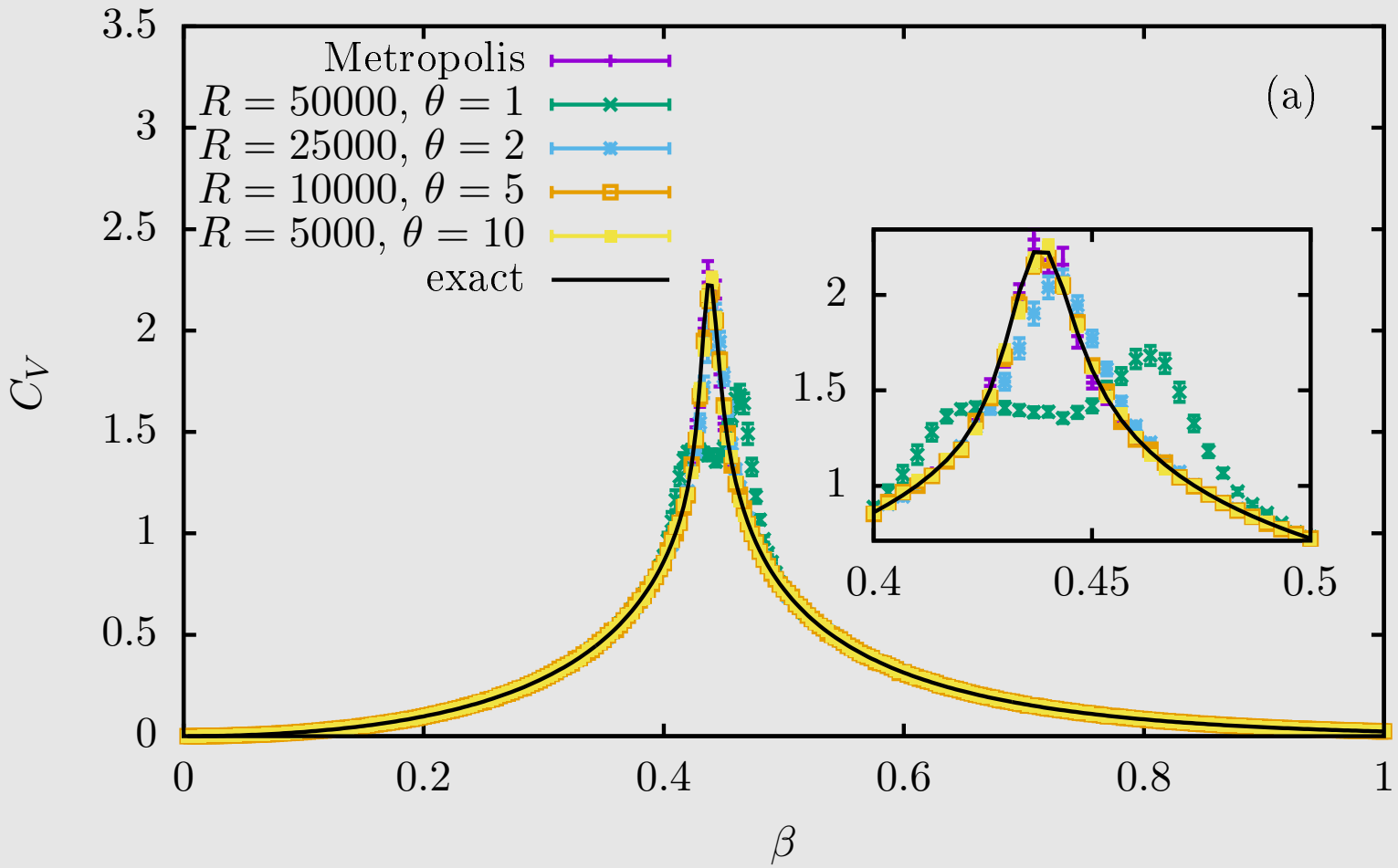
Correct results?

Exact results are available for finite lattices for the internal energy, specific heat and free energy (Ferdinand + Fisher, 1969).

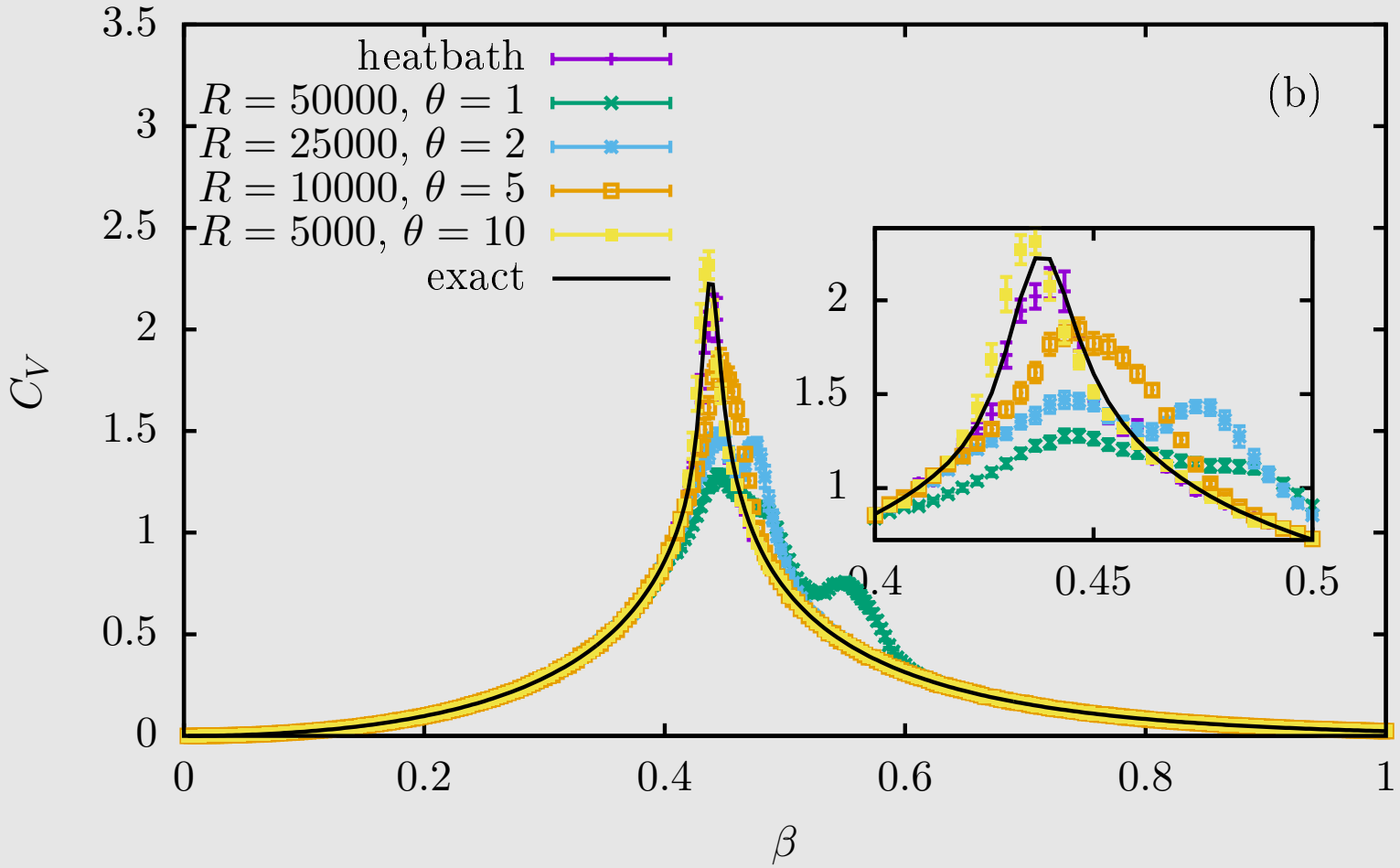


$R = 50\,000$, $\theta = 10$

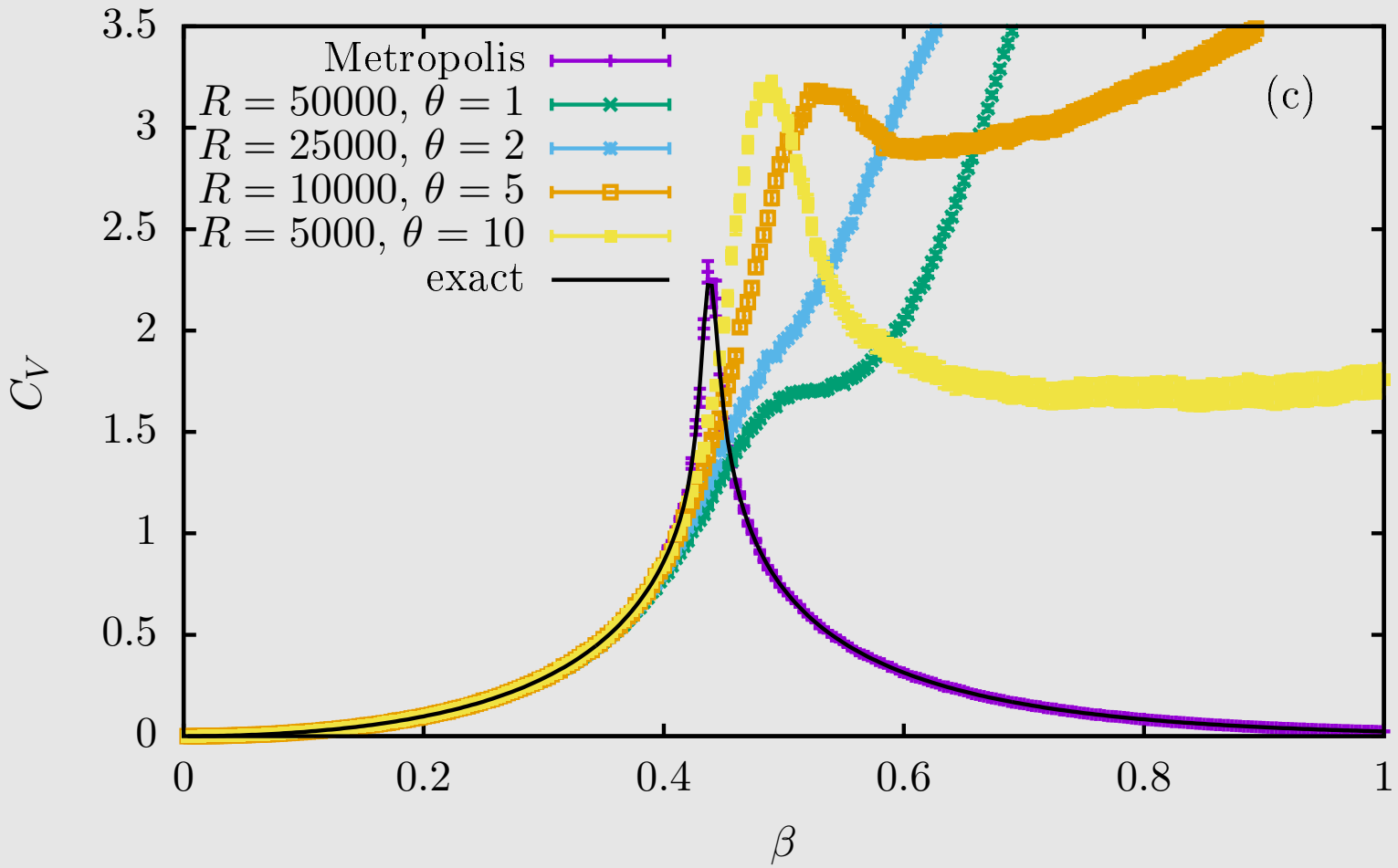
Not always


Metropolis
$R = 50000,\ \theta = 1$
$R = 25000,\ \theta = 2$
$R = 10000,\ \theta = 5$
$R = 5000,\ \theta = 10$
exact
(a)
C_V
β
3.5
3
2.5
2
1.5
1
0.5
0
0
0.2
0.4
0.6
0.8
1
2
1.5
1
0.4
0.45
0.5

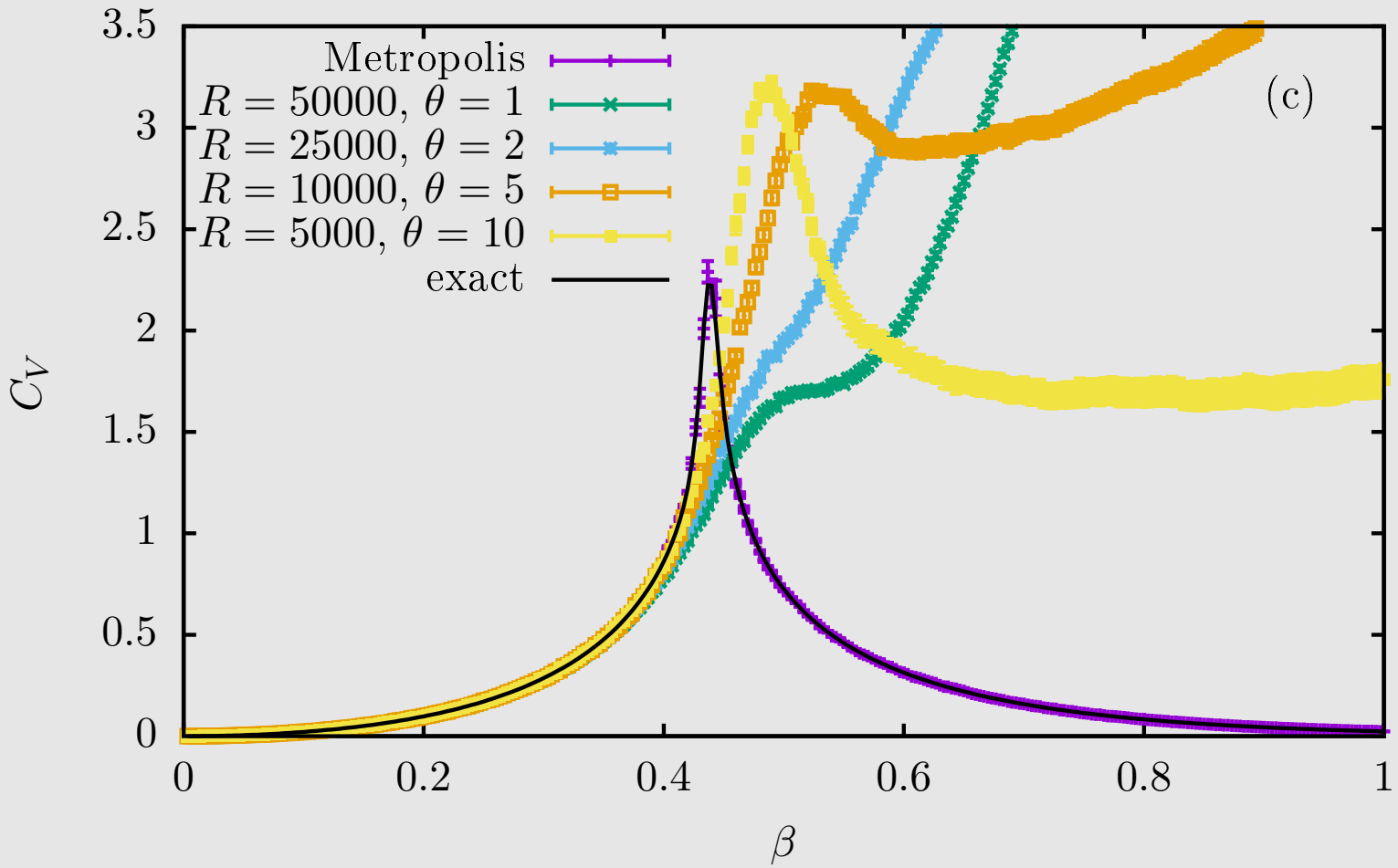
Not always


heatbath
$R = 50000,\ \theta = 1$
$R = 25000,\ \theta = 2$
$R = 10000,\ \theta = 5$
$R = 5000,\ \theta = 10$
exact
(b)
C_V
β
0
0.5
1
1.5
2
2.5
3
3.5
0.2
0.4
0.6
0.8
0.45

Not always


Metropolis
$R = 50000,\ \theta = 1$
$R = 25000,\ \theta = 2$
$R = 10000,\ \theta = 5$
$R = 5000,\ \theta = 10$
exact
(c)
C_V
β
0
0.5
1
1.5
2
2.5
3
3.5
0
0.2
0.4
0.6
0.8
1

Not always



Need to understand dependence on parameters, R, θ, $\Delta\beta$.

Correlations

The replicas in the population are not independent:

- resampling creates copies, so increases correlations
- MCMC moves decorrelate configurations

Correlations

The replicas in the population are not independent:

- resampling creates copies, so increases correlations
- MCMC moves decorrelate configurations

How can the effect be measured?

Correlations

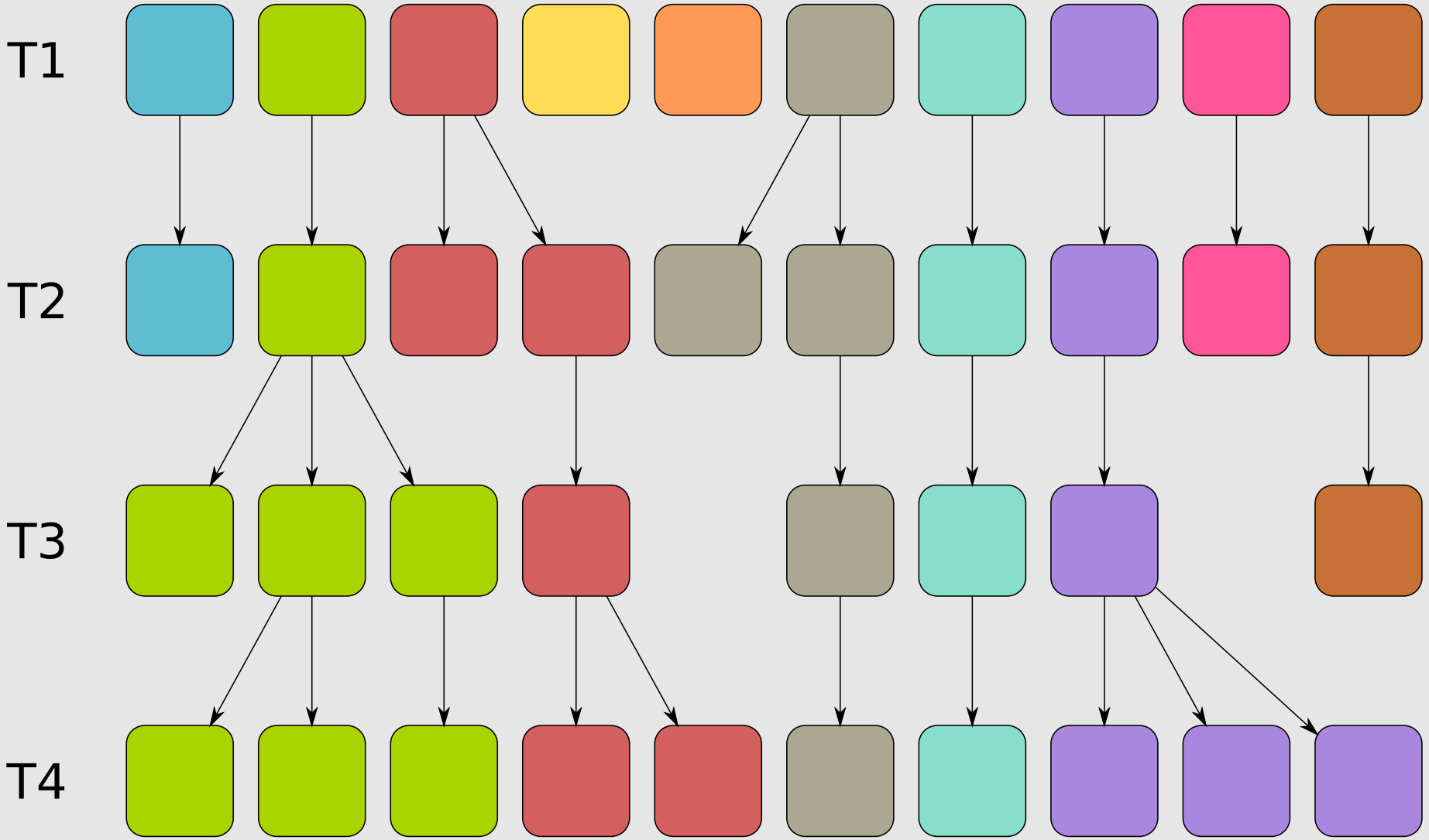
The replicas in the population are not independent:

- resampling creates copies, so increases correlations
- MCMC moves decorrelate configurations

How can the effect be measured?

An upper bound is provided by considering the statistics of *families* (W. Wang et al., 2015).

Population annealing



Correlations

The replicas in the population are not independent:

- resampling creates copies, so increases correlations
- MCMC moves decorrelate configurations

How can the effect be measured?

An upper bound is provided by considering the statistics of *families* (W. Wang et al., 2015).

For a better estimate, learn from time-series analysis.

$$\sigma^2(\overline{A}) = \frac{\sigma^2(A)}{N_{\text{eff}}}, \quad N_{\text{eff}} = N/2\tau_{\text{int}}$$

Correlations

The replicas in the population are not independent:

- resampling creates copies, so increases correlations
- MCMC moves decorrelate configurations

How can the effect be measured?

An upper bound is provided by considering the statistics of *families* (`W. Wang et al., 2015`).

For a better estimate, learn from time-series analysis.

$$\sigma^2(\overline{A}) = \frac{\sigma^2(A)}{N_{\text{eff}}}, \quad N_{\text{eff}} = N/2\tau_{\text{int}}$$

The effective population size R_{eff} can be determined from blocking,

$$R_{\text{eff}} = \frac{\sigma^2(\bar{O}^1)}{\sigma^2(\bar{O}^{N_b})}.$$

Correlations (cont'd)

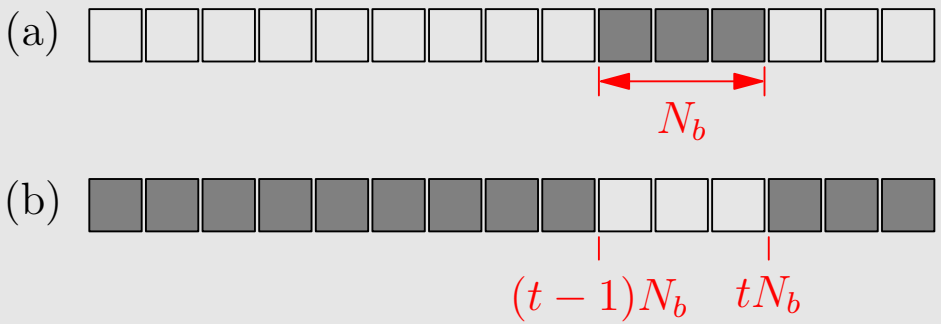
The effective population size R_{eff} can be determined from blocking,

$$R_{\text{eff}} = \frac{\sigma^2(\bar{O}^1)}{\sigma^2(\bar{O}^{N_b})}.$$

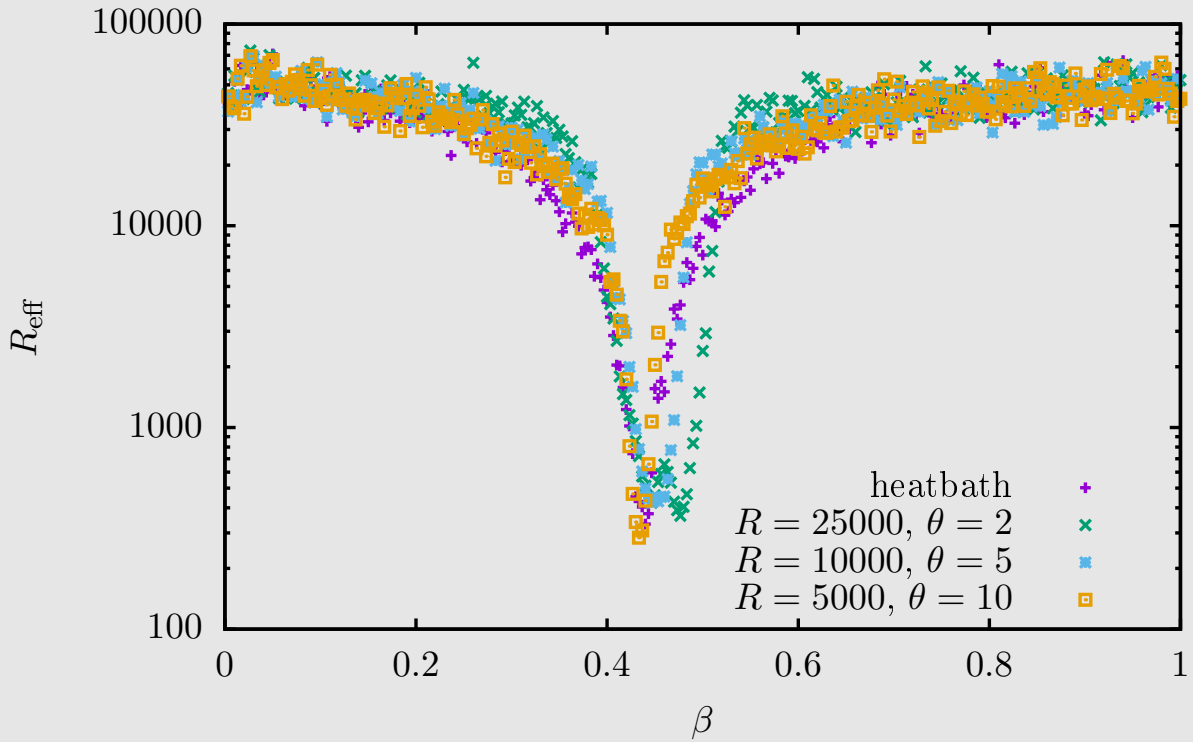
Correlations (cont'd)

The effective population size R_{eff} can be determined from blocking,

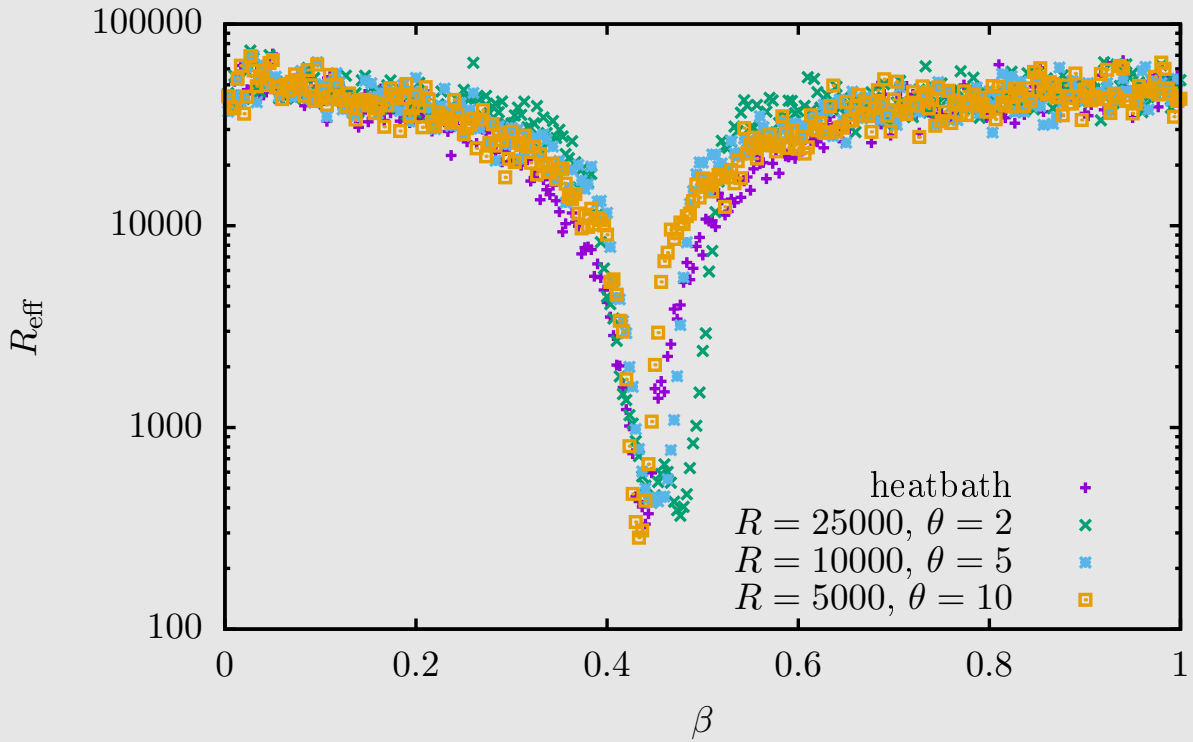
$$R_{\text{eff}} = \frac{\sigma^2(\bar{O}^1)}{\sigma^2(\bar{O}^{N_b})}.$$



Correlations (cont'd)

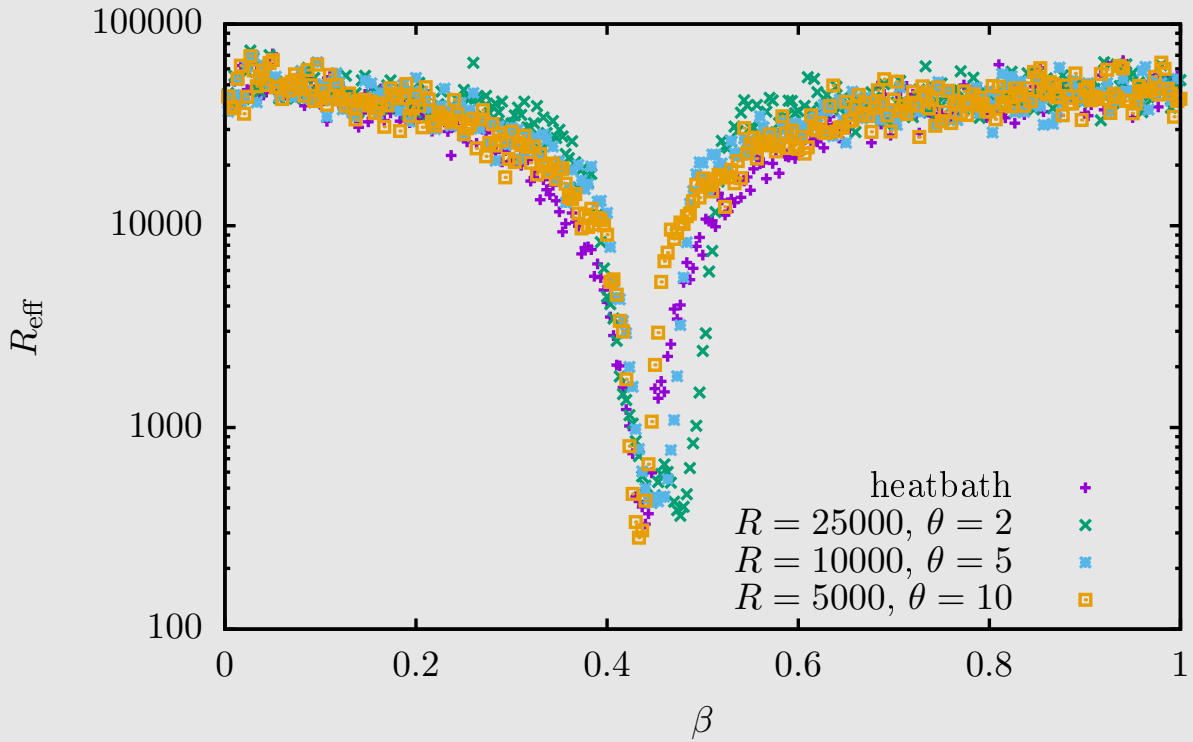


Correlations (cont'd)



- Can use regular jackknife with B blocks.
- Alternatively use *delete-m jackknife* on families.

Correlations (cont'd)



- Can use regular jackknife with B blocks.
- Alternatively use *delete-m jackknife* on families.

R_{eff} is central for the further analysis. Self-consistency demands

$$2\tau_{\text{int}} \ll N_b \quad \Rightarrow R_{\text{eff}} \gg B.$$

Need of the order of $B = 100$ blocks for reliable estimates, hence R_{eff} must be $O(10^3 - 10^4)$, independent of R.

Bias and statistical error

We can show that the bias

$$\Delta A \sim \frac{\Delta\beta}{R_{\text{eff}}} e^{-\theta/\tau},$$

and

$$R_{\text{eff}} \approx R(1 - e^{-\theta/\tau_{\text{eff}}}).$$

Hence, increasing θ is more efficient in reducing bias than increasing R, whenever the MCMC is efficient.

Statistical errors also depend on R_{eff}:

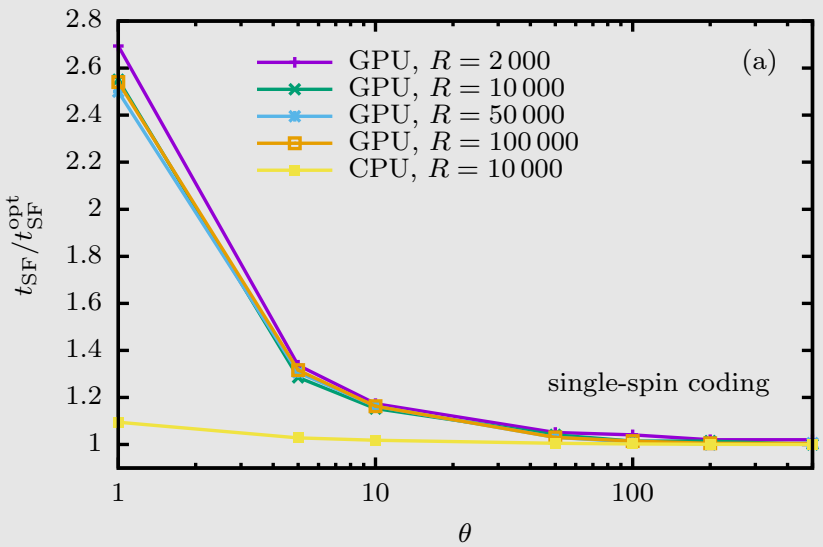
$$\sigma(\overline{A}) \sim \frac{1}{\sqrt{R_{\text{eff}}}}.$$

Asymptotically, statistical errors dominate. Overall recipe:

- use adaptive temperature steps (see below)
- increase θ only up to a point where bias is negligible over fluctuations
- then maximize R to reduce statistical errors

Massively parallel approach

The approach is naturally suitable for an implementation on massively parallel hardware such as GPUs.



L. Barash, MW, M. Borovský, W, Janke, and L. Shchur,
Comput. Phys. Commun. 220, 341 (2017).
Code at `github.com/LevBarash/PAising`.

Massively parallel approach

The approach is naturally suitable for an implementation on massively parallel hardware such as GPUs.

	CPU	GPU			
		SSC		MSC	
L	t_{SF} [ns]	t_{SF} [ns]	speedup	t_{SF} [ns]	speedup
16	23.1	0.092	251	0.0096	2406
32	22.9	0.094	243	0.0095	2410
64	22.6	0.095	238	0.0098	2306
128	22.6	0.098	230	0.0098	2306
256	22.5	0.099	227	0.0098	2295

L. Barash, MW, M. Borovský, W, Janke, and L. Shchur,
Comput. Phys. Commun. 220, 341 (2017).
Code at github.com/LevBarash/PAising.

Parallel scaling

Compare MCMC and PA regarding parallel scaling.

Parallel scaling

Compare MCMC and PA regarding parallel scaling.

Consider total work of parallel implementation. For MCMC we have

$$W \propto pE + T.$$

and statistical errors are $\propto 1/\sqrt{T}$. On the other hand, for PA one needs

$$W \propto R.$$

Parallel scaling

Compare MCMC and PA regarding parallel scaling.

Consider total work of parallel implementation. For MCMC we have

$$W \propto pE + T.$$

and statistical errors are $\propto 1/\sqrt{T}$. On the other hand, for PA one needs

$$W \propto R.$$

The parallel speedup is hence

$$S = \frac{T_1}{T_p} = \begin{cases} \frac{E+T}{E+T/p} & \stackrel{p\to\infty}{\longrightarrow} \quad 1 + \frac{T}{E} \quad \text{MCMC}, \\ p & \stackrel{p\to\infty}{\longrightarrow} \quad \infty \quad \text{PA} \end{cases}$$

Parallel scaling

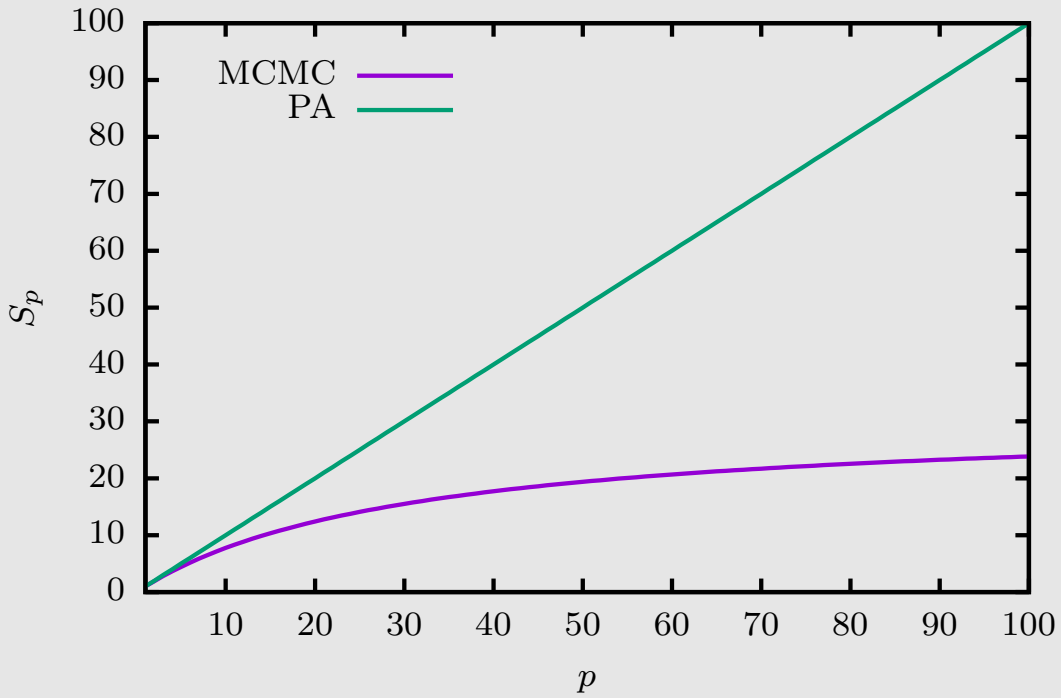
Compare MCMC and PA regarding parallel scaling.

Consider total work of parallel implementation. For MCMC we have

$$W \propto pE + T.$$

and statistical errors are $\propto 1/\sqrt{T}$. On the other hand, for PA one needs

$$W \propto R.$$



Improvements

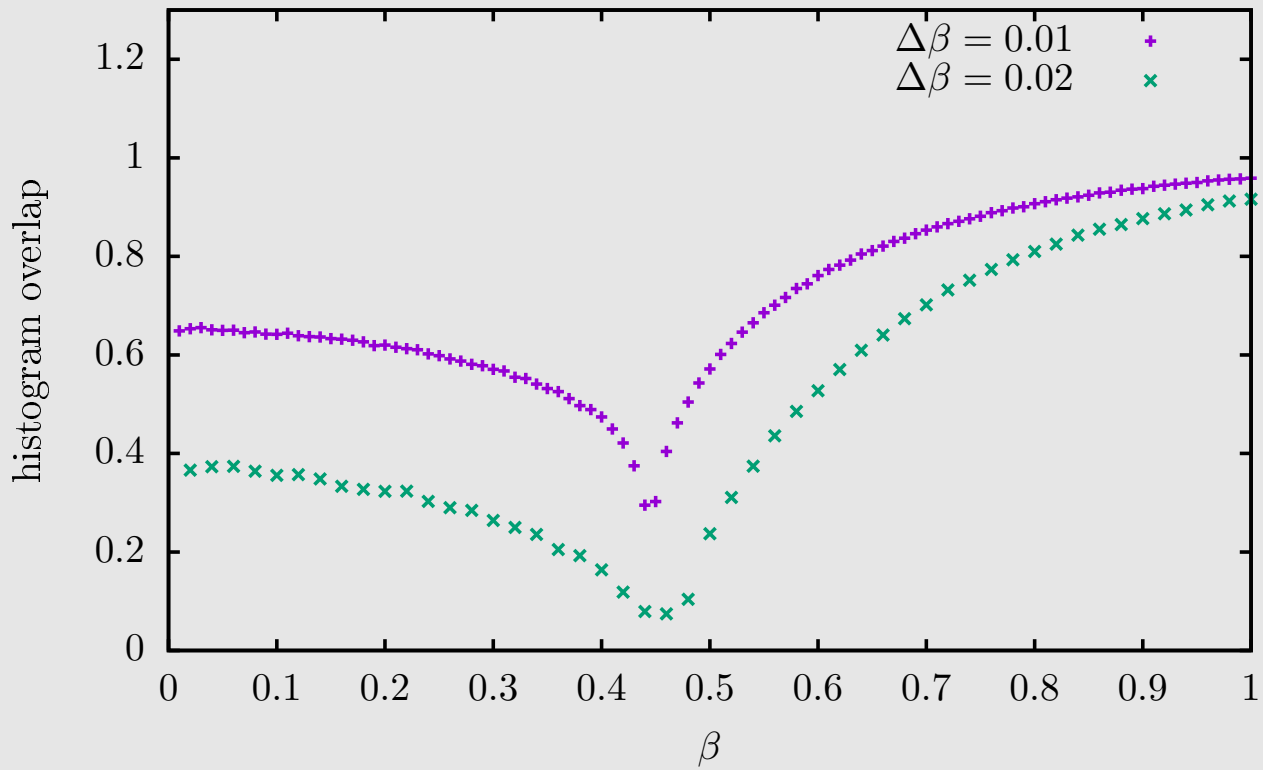
Three natural extensions that improve the algorithm significantly:

1. **Adaptive temperature steps:** Efficiency and bias of the resampling depends on histogram overlap.

Improvements

Three natural extensions that improve the algorithm significantly:

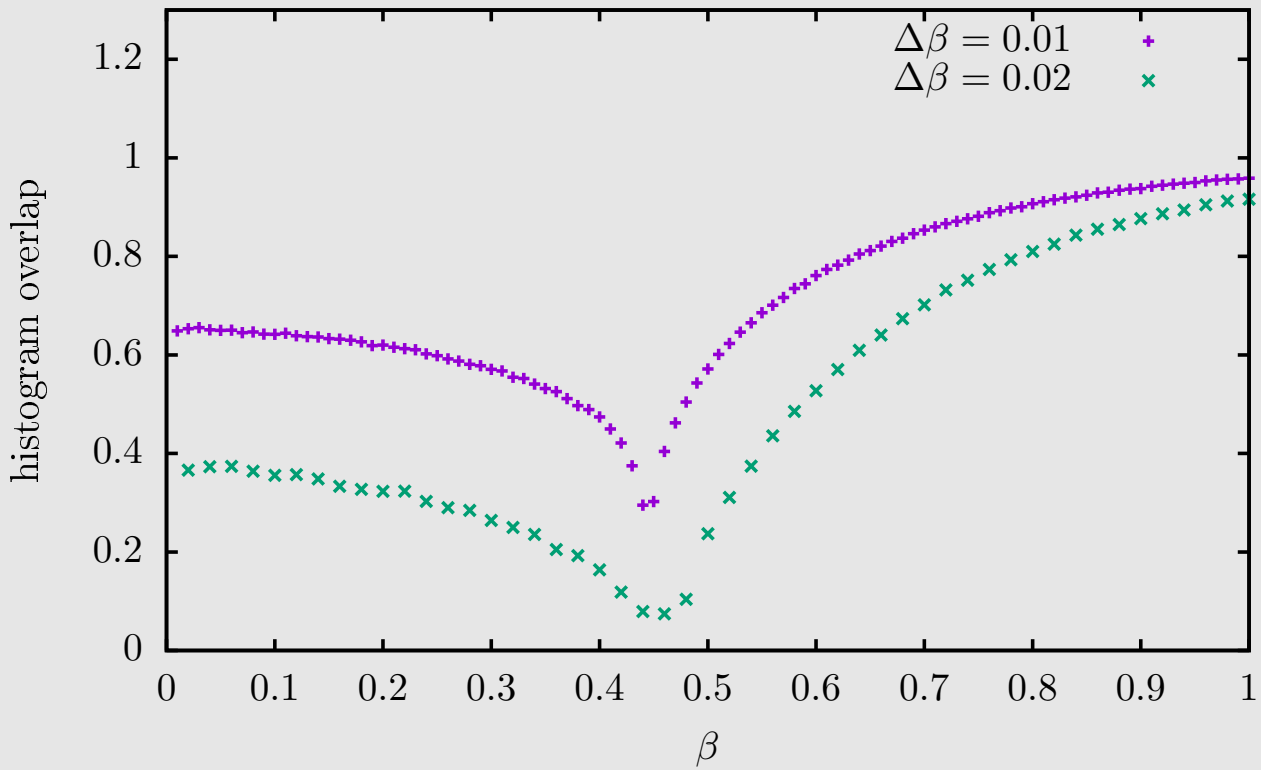
1. **Adaptive temperature steps:** Efficiency and bias of the resampling depends on histogram overlap.



Improvements

Three natural extensions that improve the algorithm significantly:

1. **Adaptive temperature steps:** Efficiency and bias of the resampling depends on histogram overlap.

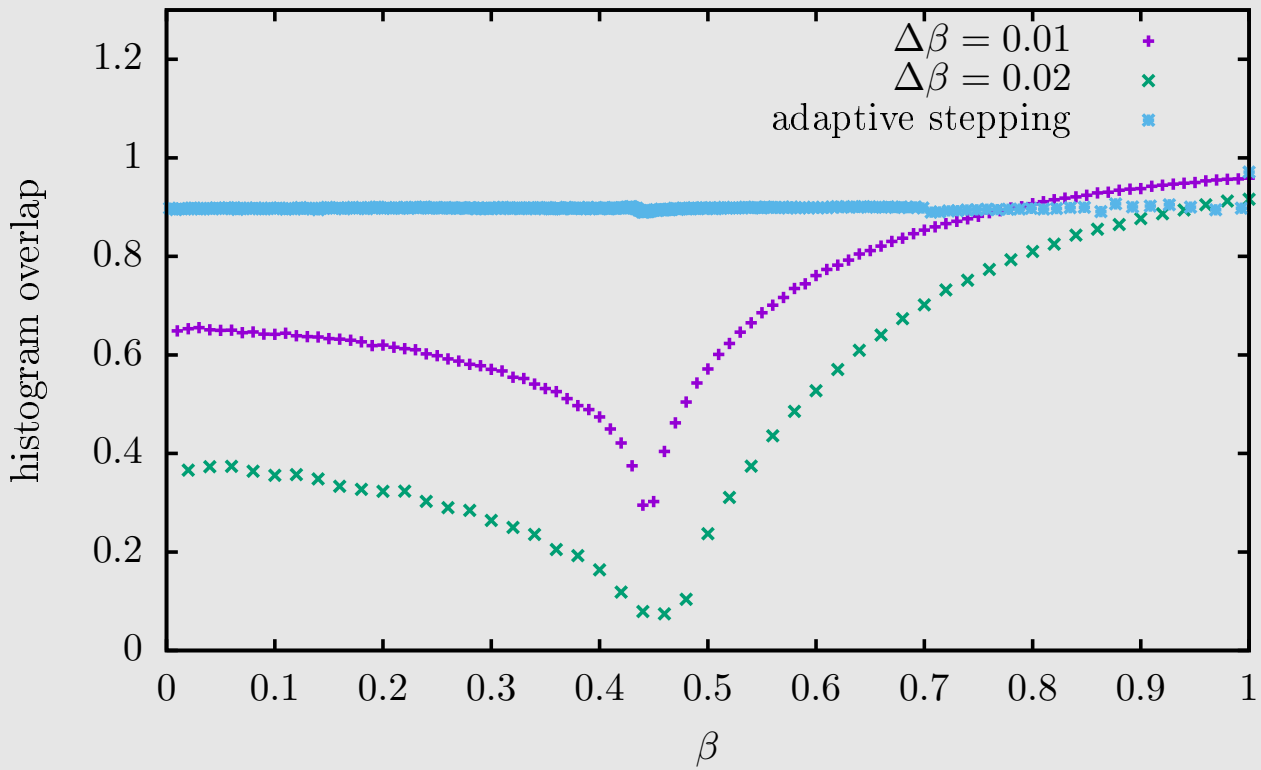


$\Rightarrow$ choose temperature step adaptively on the fly to ensure fixed overlap of neighboring energy histograms (as estimated from populations).

Improvements

Three natural extensions that improve the algorithm significantly:

1. **Adaptive temperature steps:** Efficiency and bias of the resampling depends on histogram overlap.



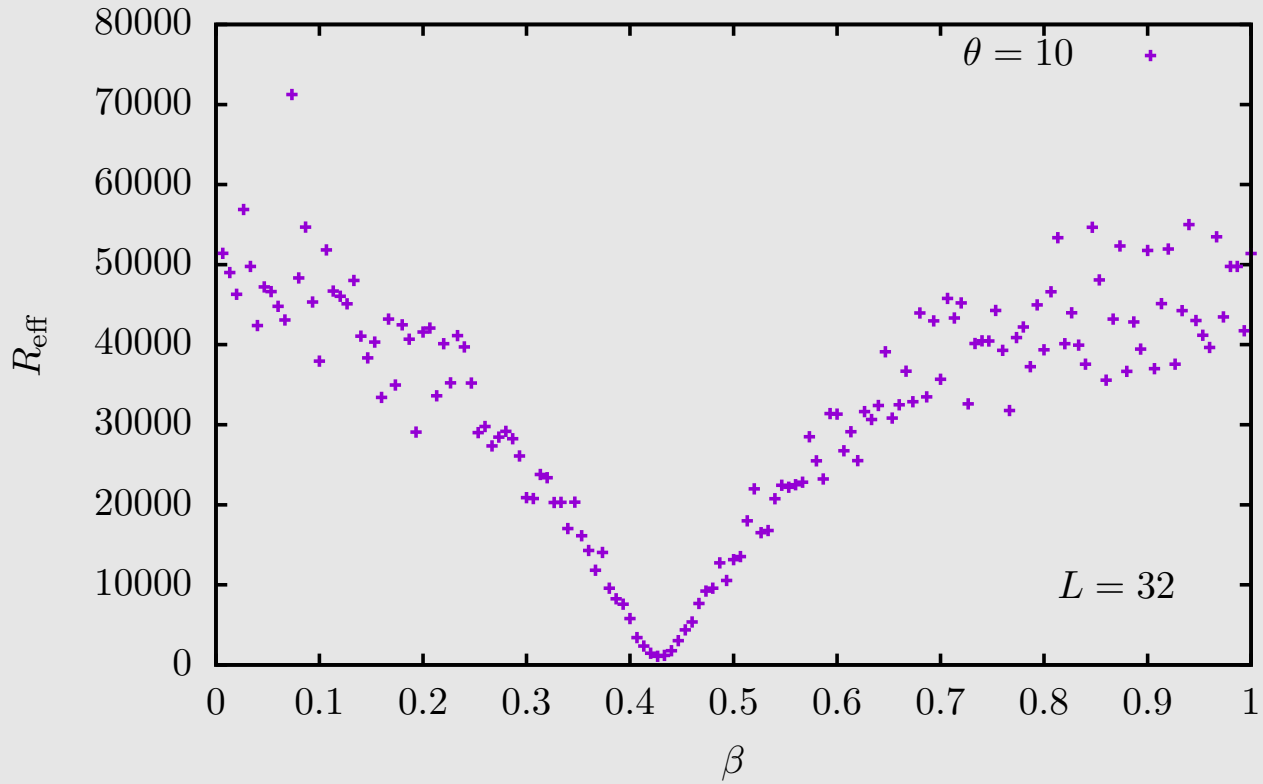
$\Rightarrow$ choose temperature step adaptively on the fly to ensure fixed overlap of neighboring energy histograms (as estimated from populations).

Improvements

2. **Adaptive time steps:** Number of independent replicas R_{eff} crucially determines bias as well as statistical errors.

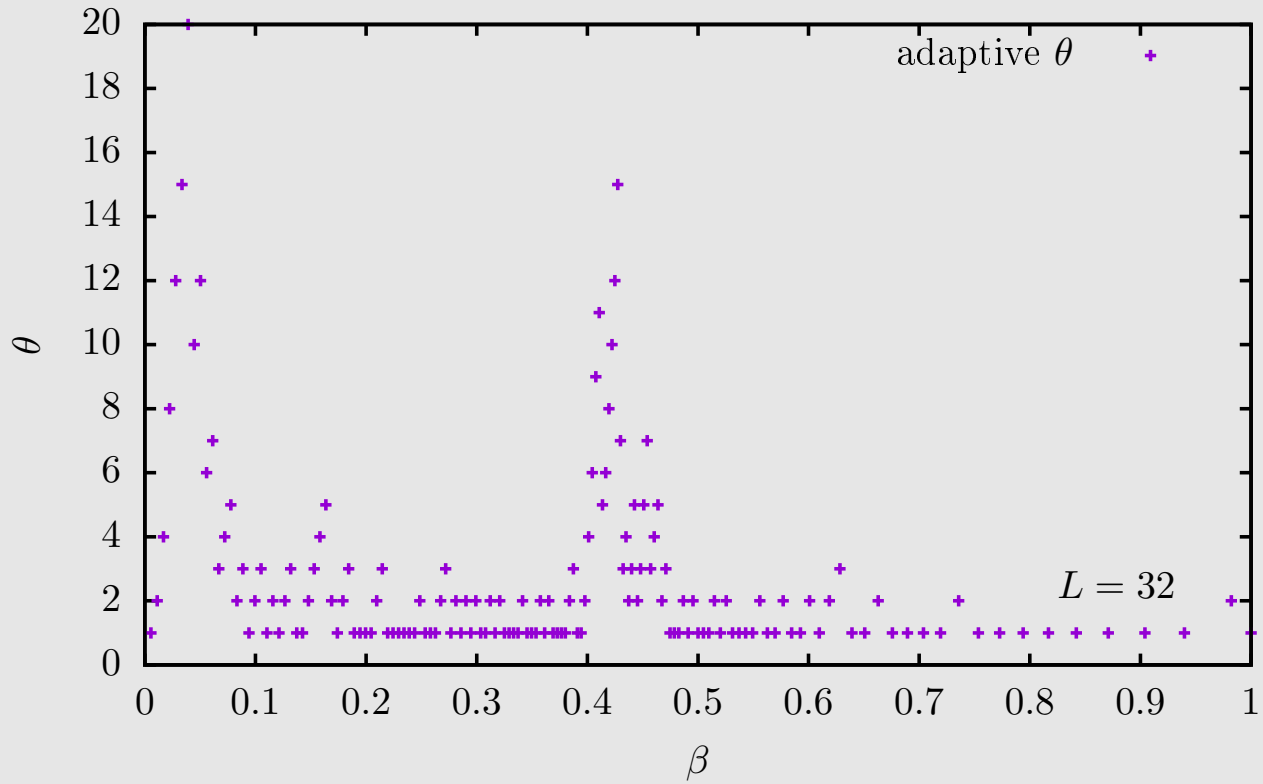
Improvements

2. **Adaptive time steps:** Number of independent replicas R_{eff} crucially determines bias as well as statistical errors.



Improvements

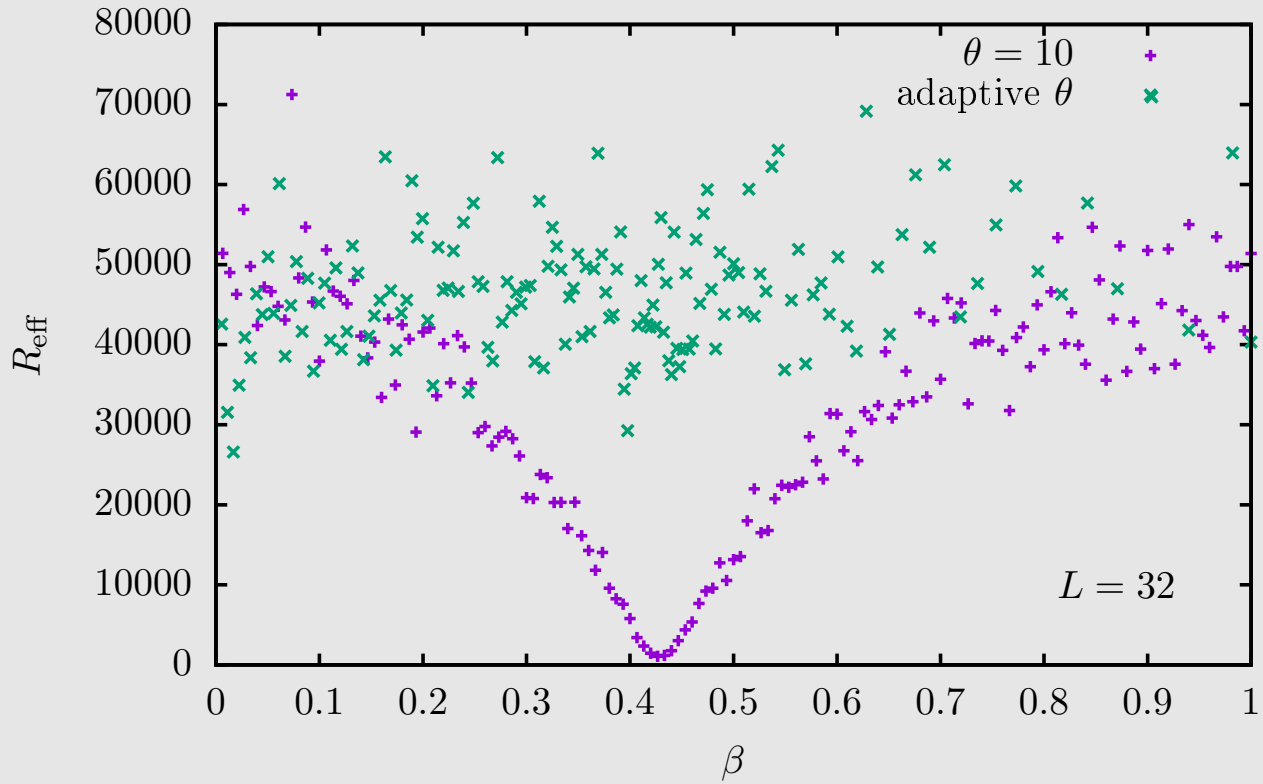
2. **Adaptive time steps:** Number of independent replicas R_{eff} crucially determines bias as well as statistical errors.



$\Rightarrow$ choose $\theta \propto R/R_{\text{eff}}$ to effectively decorrelate configurations.

Improvements

2. **Adaptive time steps:** Number of independent replicas R_{eff} crucially determines bias as well as statistical errors.



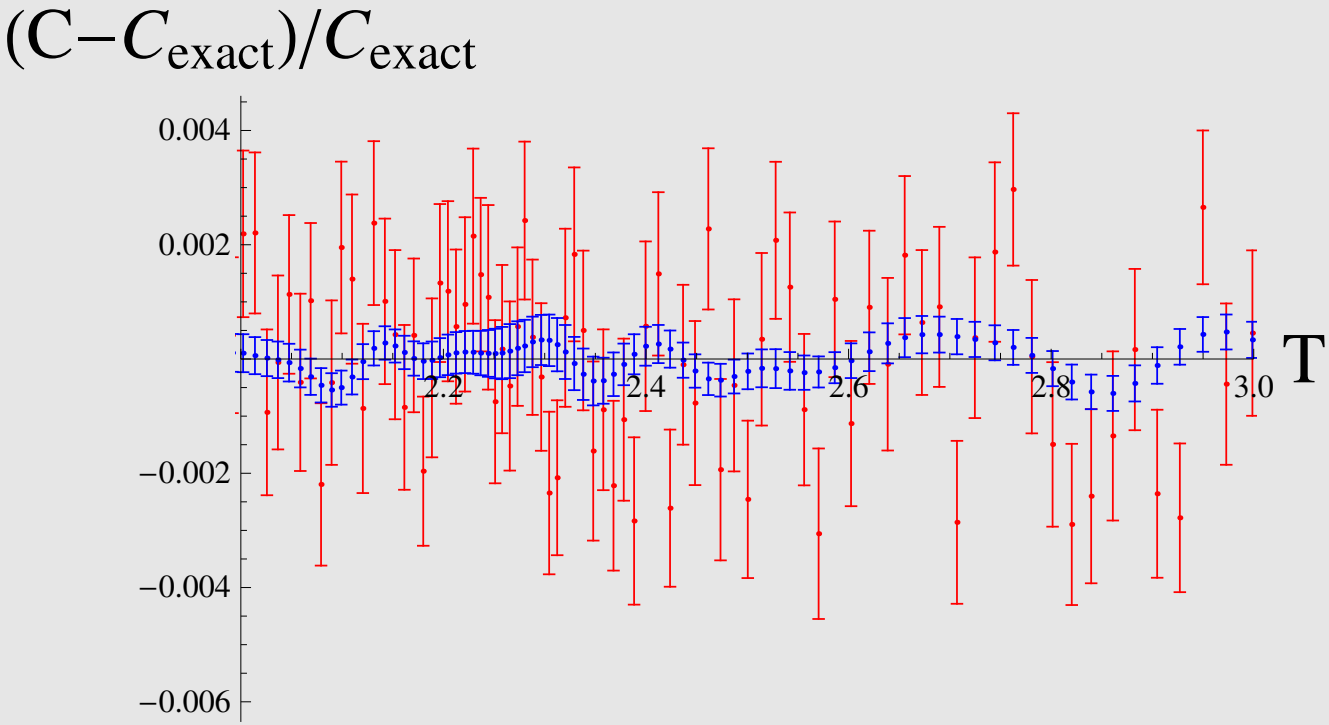
$\Rightarrow$ choose $\theta \propto R/R_{\text{eff}}$ to effectively decorrelate configurations.

Improvements

3. **Multi-histogram analysis:** Information from neighboring temperatures is also relevant.

Improvements

3. **Multi-histogram analysis:** Information from neighboring temperatures is also relevant.



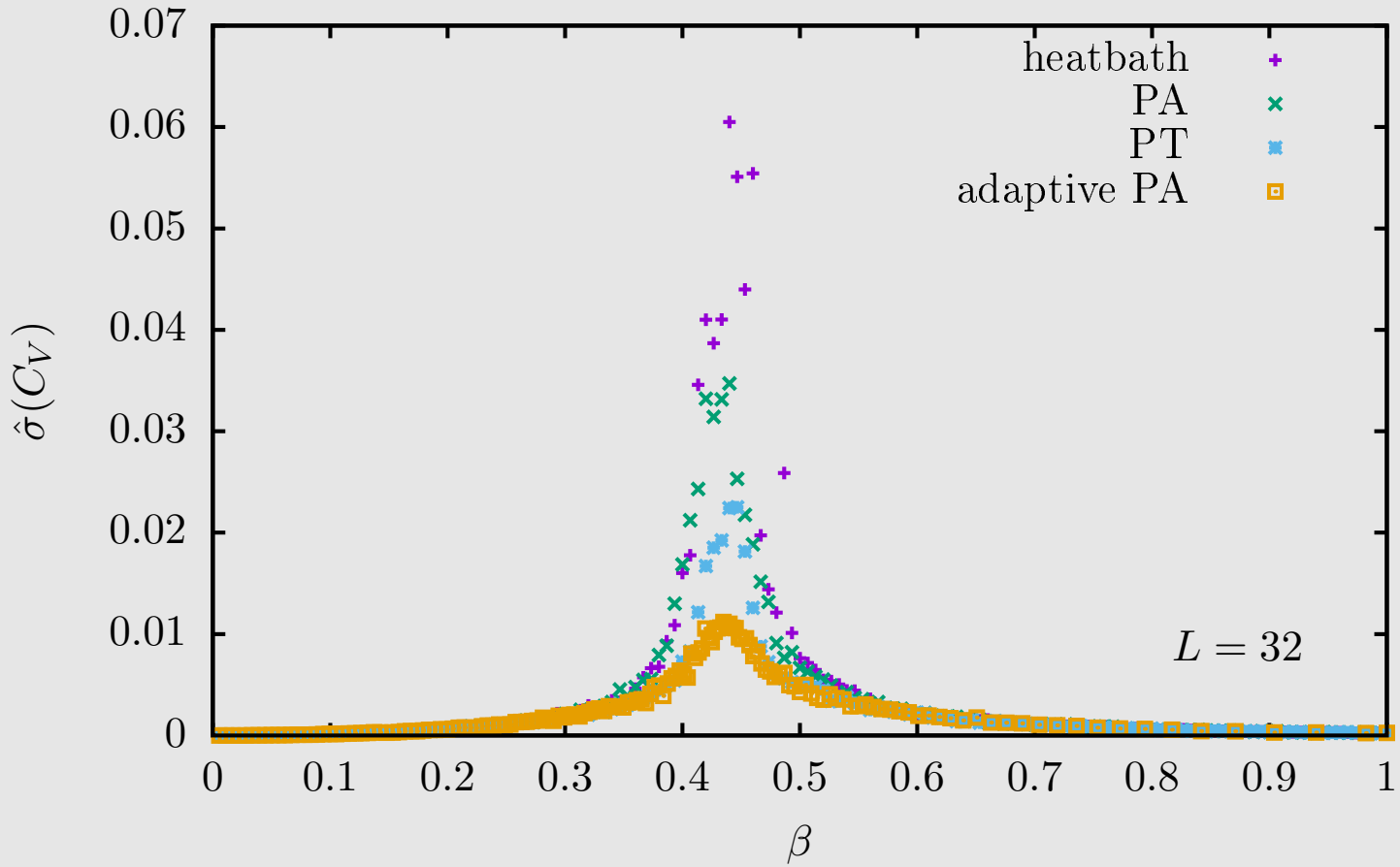
This also allows to estimate the density of states. Iterations as in the Ferrenberg/Swendsen scheme are not required.

Comparison

Adaptive scheme performs significantly better than original one.

Comparison

Adaptive scheme performs significantly better than original one.

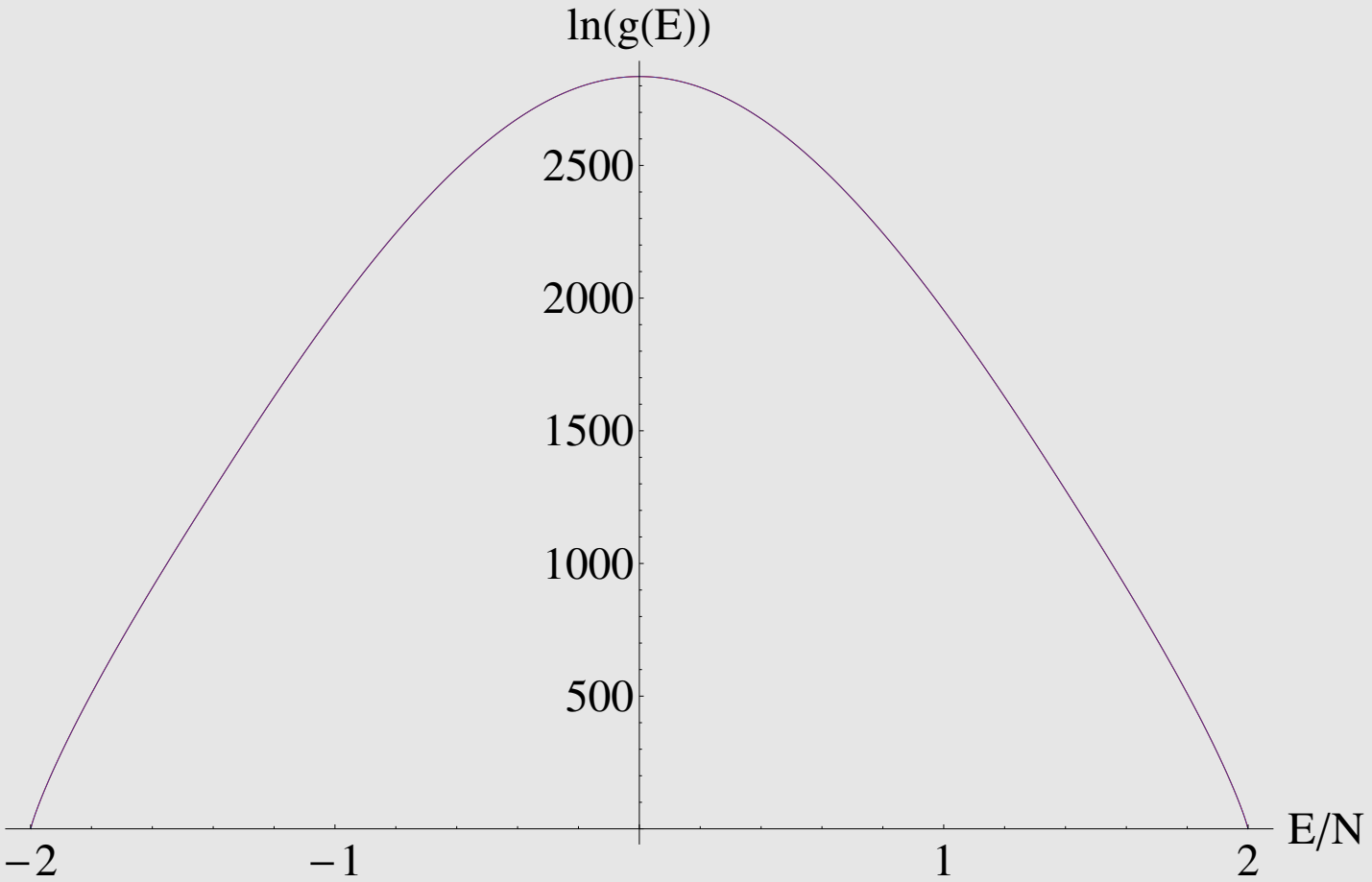


Sampling the density of states

Something that we normally think can only be done with multicanonical or Wang-Landau simulations.

Sampling the density of states

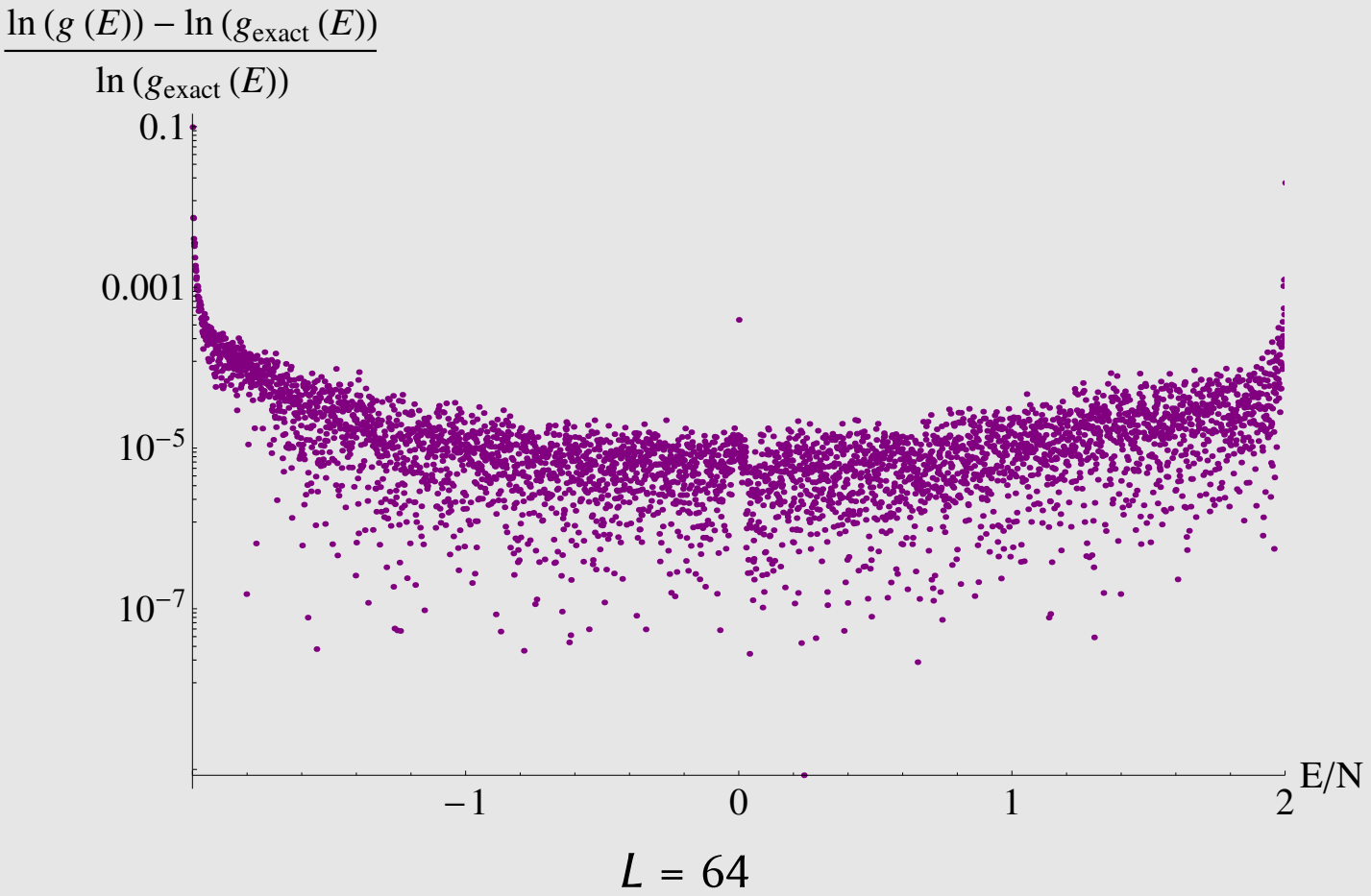
Something that we normally think can only be done with multicanonical or Wang-Landau simulations.



$L = 64$

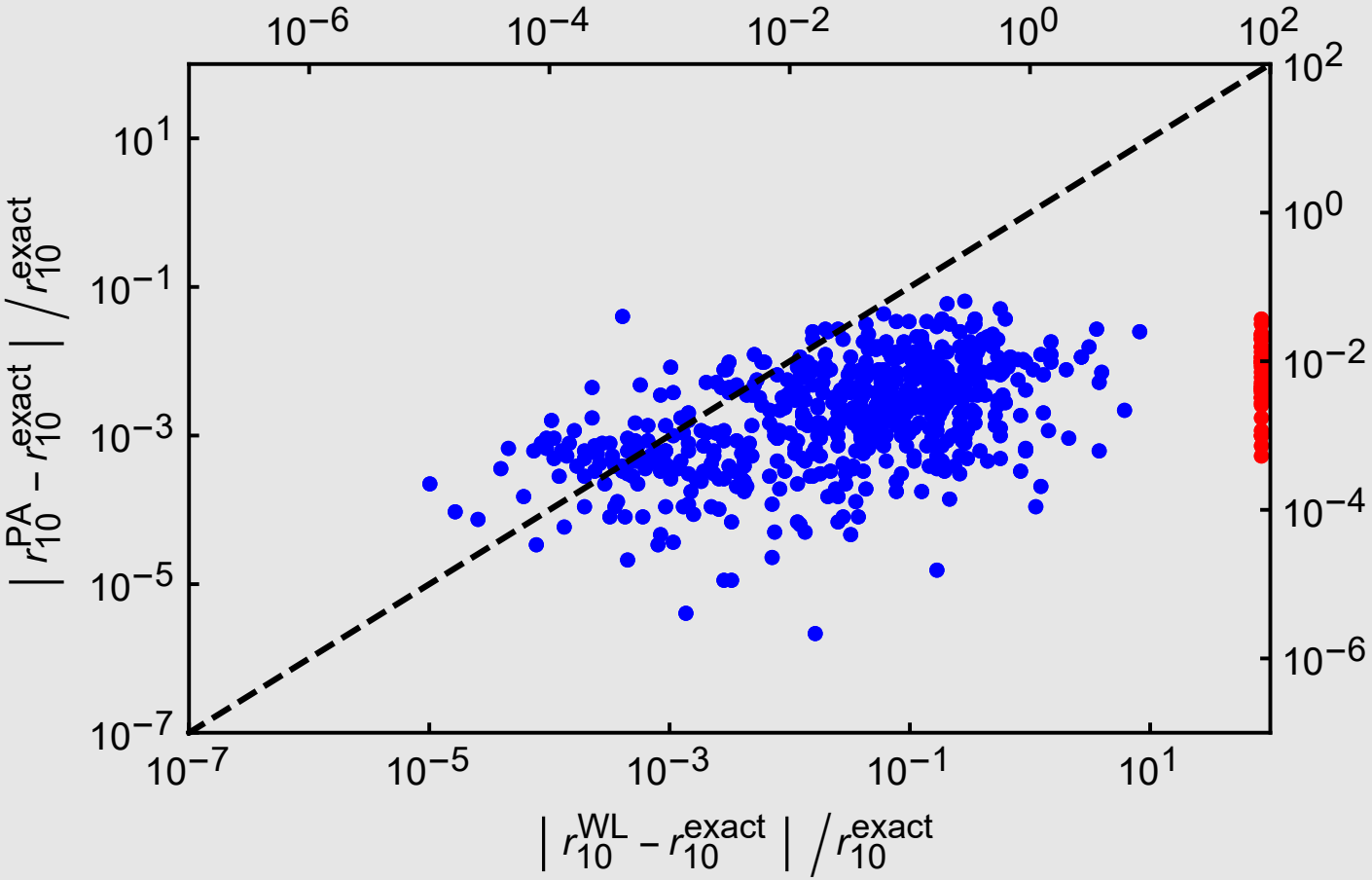
Sampling the density of states

Something that we normally think can only be done with multicanonical or Wang-Landau simulations.



Sampling the density of states (cont'd)

Estimate density of states of Chimera spin-glass samples with planted solutions.

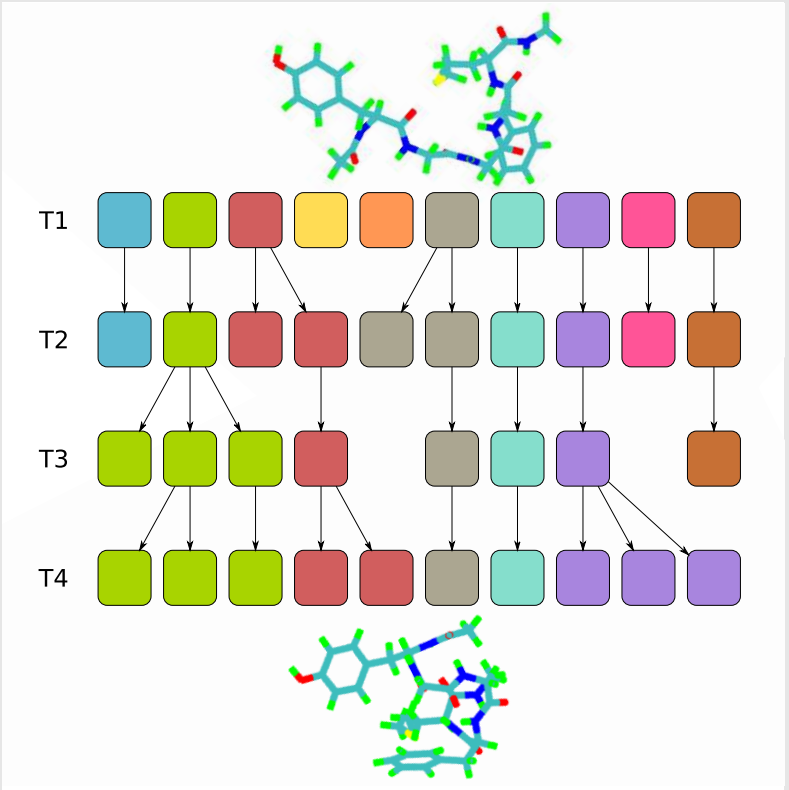


$$r_{10} = \frac{\Omega(E_1)}{\Omega(E_0)}$$

Population annealing molecular dynamics

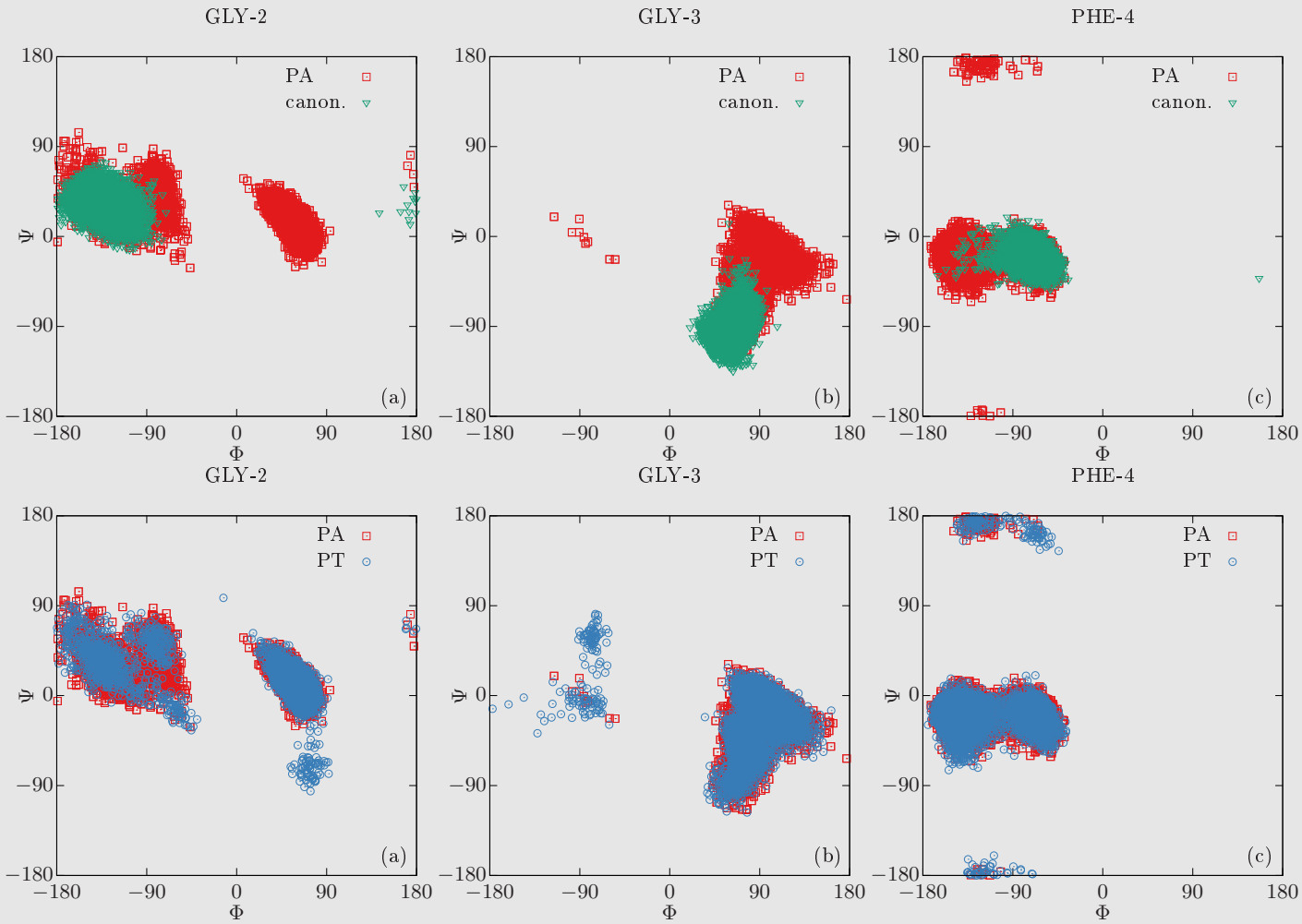
Population annealing as a meta algorithm can be combined with other types of underlying dynamics.

- simulate R systems with NVT MD in parallel
- need to use a stochastic thermostat
- resampling using the same rule as before
- can easily use existing MD code, for example OpenMM, Gromacs, NAMD, ...



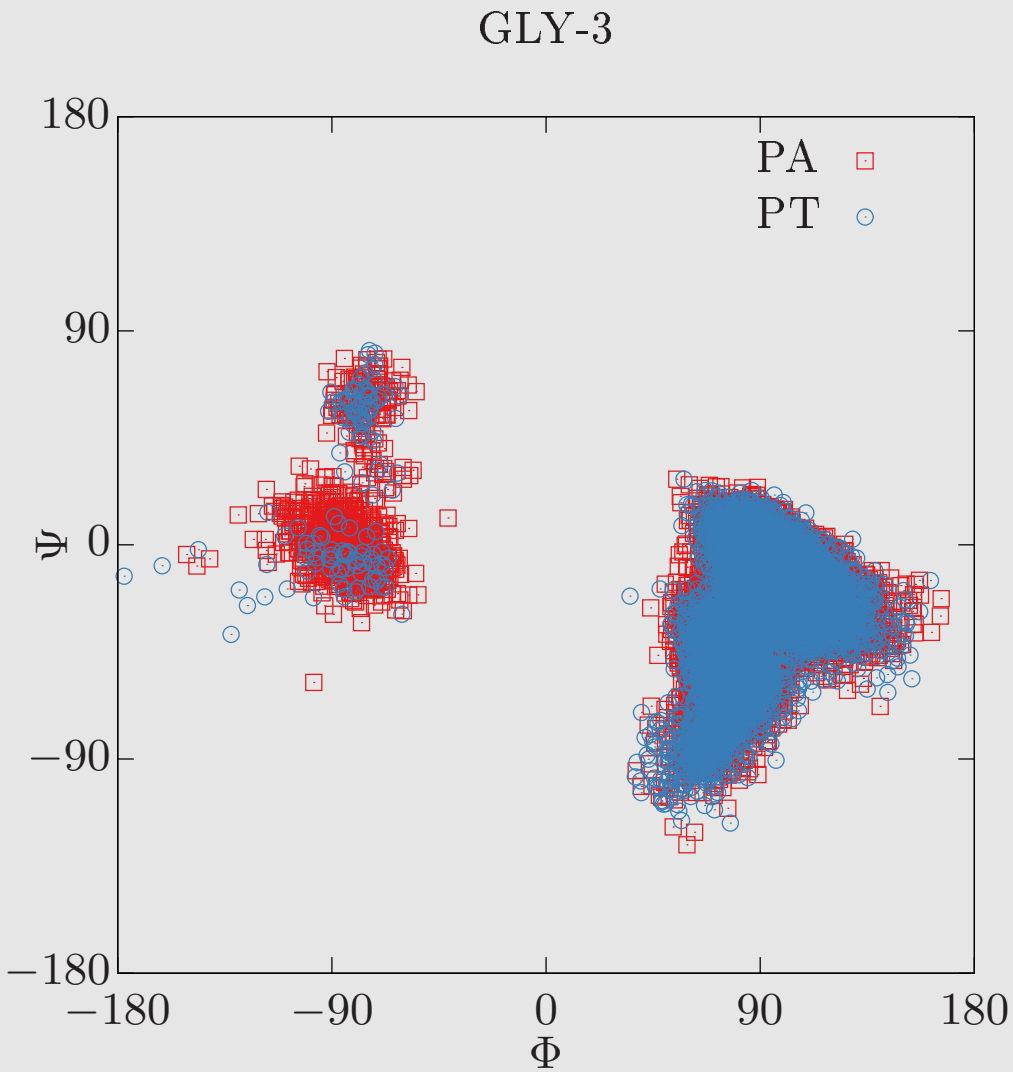
Population annealing molecular dynamics (cont'd)

Test for met-enkephalin in vacuo, a pentapeptide with sequence Tyr-Gly-Gly-Phe-Met.



Population annealing molecular dynamics (cont'd)

So the efficiency of PA for MD is on par with PT given the same computational resources, but PA scales to a practically arbitrary number of cores!



Summary

Conclusions

Parallel Monte Carlo:

- use domain decomposition for canonical updates of short-range models
- non-local updates possible (Swendsen-Wang and friends)
- very efficient on GPUs

Conclusions

Parallel Monte Carlo:

- use domain decomposition for canonical updates of short-range models
- non-local updates possible (Swendsen-Wang and friends)
- very efficient on GPUs

Multicanonical simulations:

- parallelize with multiple walkers
- of particular interest for systems with 1st order transitions
- super-linear speedup observed up to 70k walkers

Conclusions

Parallel Monte Carlo:

- use domain decomposition for canonical updates of short-range models
- non-local updates possible (Swendsen-Wang and friends)
- very efficient on GPUs

Multicanonical simulations:

- parallelize with multiple walkers
- of particular interest for systems with 1st order transitions
- super-linear speedup observed up to 70k walkers

Population annealing:

- theoretically perfect parallel scaling with R plus can combine independent runs
- free-energy estimator generalizes thermodynamic integration
- can be turned into a fully adaptive algorithm

Acknowledgements

Co-workers

Coventry group:
Ravinder Kumar, Jeffrey Kelling, Michal Borovský, Taras Yavors'kii
Lev Barash, Moscow
Jonathan Gross, Leipzig
Alexander Hartmann, Oldenburg
Wolfhard Janke, Leipzig
Helmut Katzgraber, College Station
Jon Machta, Amherst
Markus Manssen, Oldenburg
Lev Shchur, Moscow
Johannes Zierenberg, Göttingen

Acknowledgements

Co-workers

Coventry group:
Ravinder Kumar, Jeffrey Kelling, Michal Borovský, Taras Yavors'kii
Lev Barash, Moscow
Jonathan Gross, Leipzig
Alexander Hartmann, Oldenburg
Wolfhard Janke, Leipzig
Helmut Katzgraber, College Station
Jon Machta, Amherst
Markus Manssen, Oldenburg
Lev Shchur, Moscow
Johannes Zierenberg, Göttingen

Funding

- German Research Foundation (DFG, Emmy Noether group)
- Coventry University (Research Sabbatical Fellowship)
- European Commission (IRSES network DIONICOS)

Acknowledgements

Co-workers

Coventry group:
Ravinder Kumar, Jeffrey Kelling, Michal Borovský, Taras Yavors'kii
Lev Barash, Moscow
Jonathan Gross, Leipzig
Alexander Hartmann, Oldenburg
Wolfhard Janke, Leipzig
Helmut Katzgraber, College Station
Jon Machta, Amherst
Markus Manssen, Oldenburg
Lev Shchur, Moscow
Johannes Zierenberg, Göttingen

Funding

- German Research Foundation (DFG, Emmy Noether group)
- Coventry University (Research Sabbatical Fellowship)
- European Commission (IRSES network DIONICOS)

References

MW, *Monte Carlo methods for massively parallel computers*, in: "Order, disorder and criticality", Vol. 5, ed. Yu. Holovatch (World Scientific, Singapore, 2018), pp. 271–340, arXiv:1709.04394

H. Christiansen, MW, W. Janke, Preprint arXiv:1806.06016

J. Gross, J. Zierenberg, MW, and W. Janke, Comput. Phys. Commun. 224, 387 (2018)

L. Barash, MW, M. Borovský, W. Janke, L. Shchur, Comput. Phys. Commun. 220, 314 (2017)

L. Barash, MW, L. Shchur, W. Janke, EPJ Spec. Topics 226, 595 (2017)

Acknowledgements

Co-workers

Coventry group:
Ravinder Kumar, Jeffrey Kelling, Michal Borovský, Taras Yavors'kii
Lev Barash, Moscow
Jonathan Gross, Leipzig
Alexander Hartmann, Oldenburg
Wolfhard Janke, Leipzig
Helmut Katzgraber, College Station
Jon Machta, Amherst
Markus Manssen, Oldenburg
Lev Shchur, Moscow
Johannes Zierenberg, Göttingen

Funding

- German Research Foundation (DFG, Emmy Noether group)
- Coventry University (Research Sabbatical Fellowship)
- European Commission (IRSES network DIONICOS)

You

Thank you for your attention!

References

MW, *Monte Carlo methods for massively parallel computers*, in: "Order, disorder and criticality", Vol. 5, ed. Yu. Holovatch (World Scientific, Singapore, 2018), pp. 271–340, arXiv:1709.04394

H. Christiansen, MW, W. Janke, Preprint arXiv:1806.06016

J. Gross, J. Zierenberg, MW, and W. Janke, Comput. Phys. Commun. 224, 387 (2018)

L. Barash, MW, M. Borovský, W. Janke, L. Shchur, Comput. Phys. Commun. 220, 314 (2017)

L. Barash, MW, L. Shchur, W. Janke, EPJ Spec. Topics 226, 595 (2017)