Метод отжига популяции и его реализация на СКК НИУ ВШЭ

Lev Shchur
Landau Institute for Theoretical Physics,
and
NRU Higher School of Economics

Метод отжига популяции и его реализация на СКК НИУ ВШЭ

Соавтор - Александр Русcков, НЦЧ РАН, Черноголовка

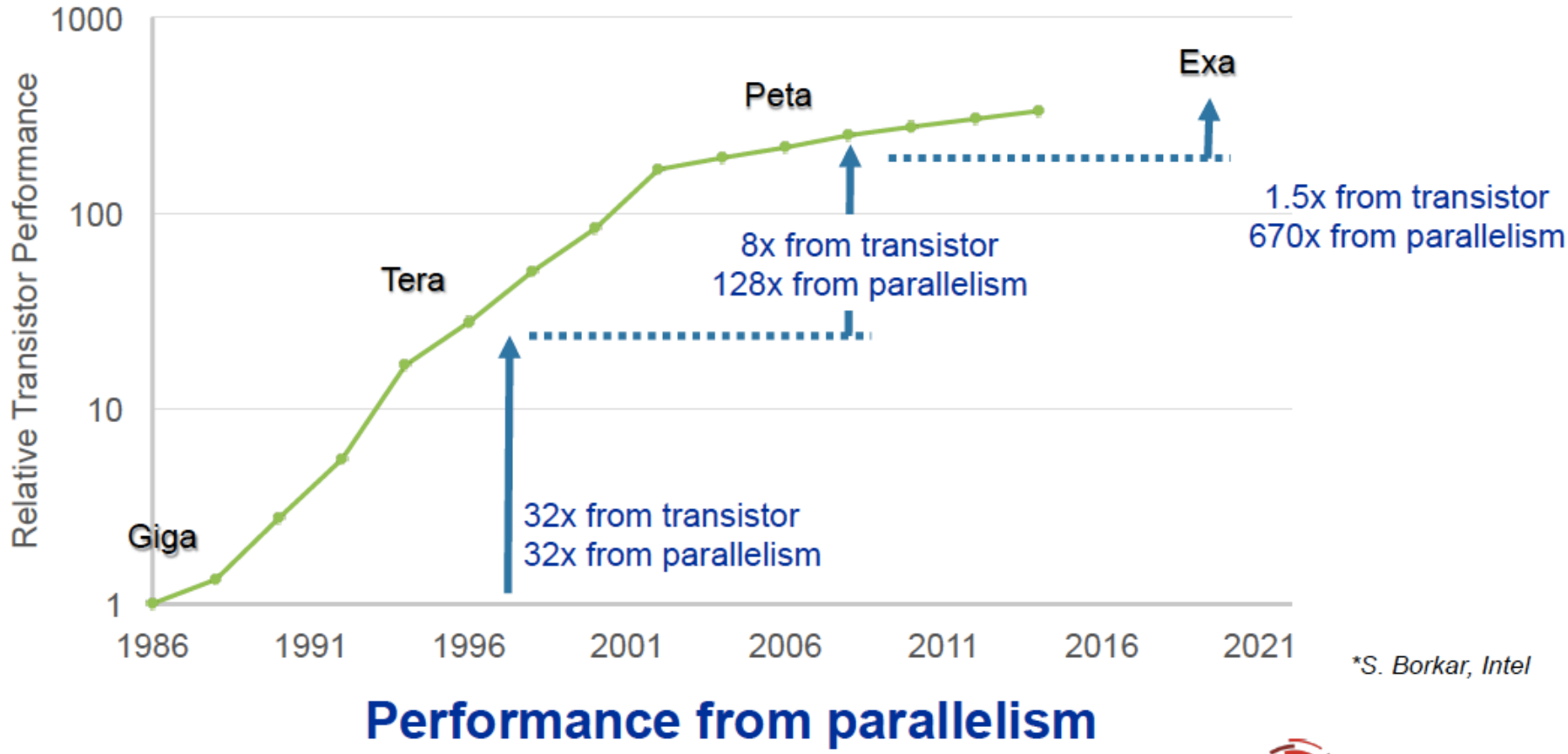
Содействие – П.А. Костенецкий, НИУ ВШЭ
Техническая помощь – Роман Чулкевич, НИУ ВШЭ

Метод отжига популяции и его реализация на СКК НИУ ВШЭ

Outline:

- *Motivation*
- *Population annealing*
- *CUDA+MPI*
- *Scalability*
- *Conclusion*

From Giga to Exa, via Tera & Peta*



*S. Borkar, Intel

From J. Messina, 31 July, 2016 (ECP director, ANL)

Метод отжига популяции и его реализация на СКК НИУ ВШЭ

The main goal

The big challenge for scientific computing is to develop algorithms and computational frameworks that use parallel hardware efficiently.

There are two alternatives:

- Develop a universal potentially fully scalable framework
- Invent an algorithm that is efficient for the particular albeit broad set of problems

Population annealing algorithm

1. The population annealing (PA) algorithm could be suitable for study systems with rough free-energy landscapes (spin glasses, optimization problems).
2. PA combines the power of the known efficient algorithms - simulated annealing, Boltzmann weighted differential reproduction, and sequential Monte Carlo process.
3. Bring the population of replicas to the equilibrium even in the low-temperature region.
4. It gives a perfect estimation of the free energy.
5. Efficient parallel implementation – many replicas.

K. Hukushima, Y. Iba, AIP Conf. Proc. **690** 200 (2003)
J. Machta, Phys. Rev. E **82** 026704 (2010)

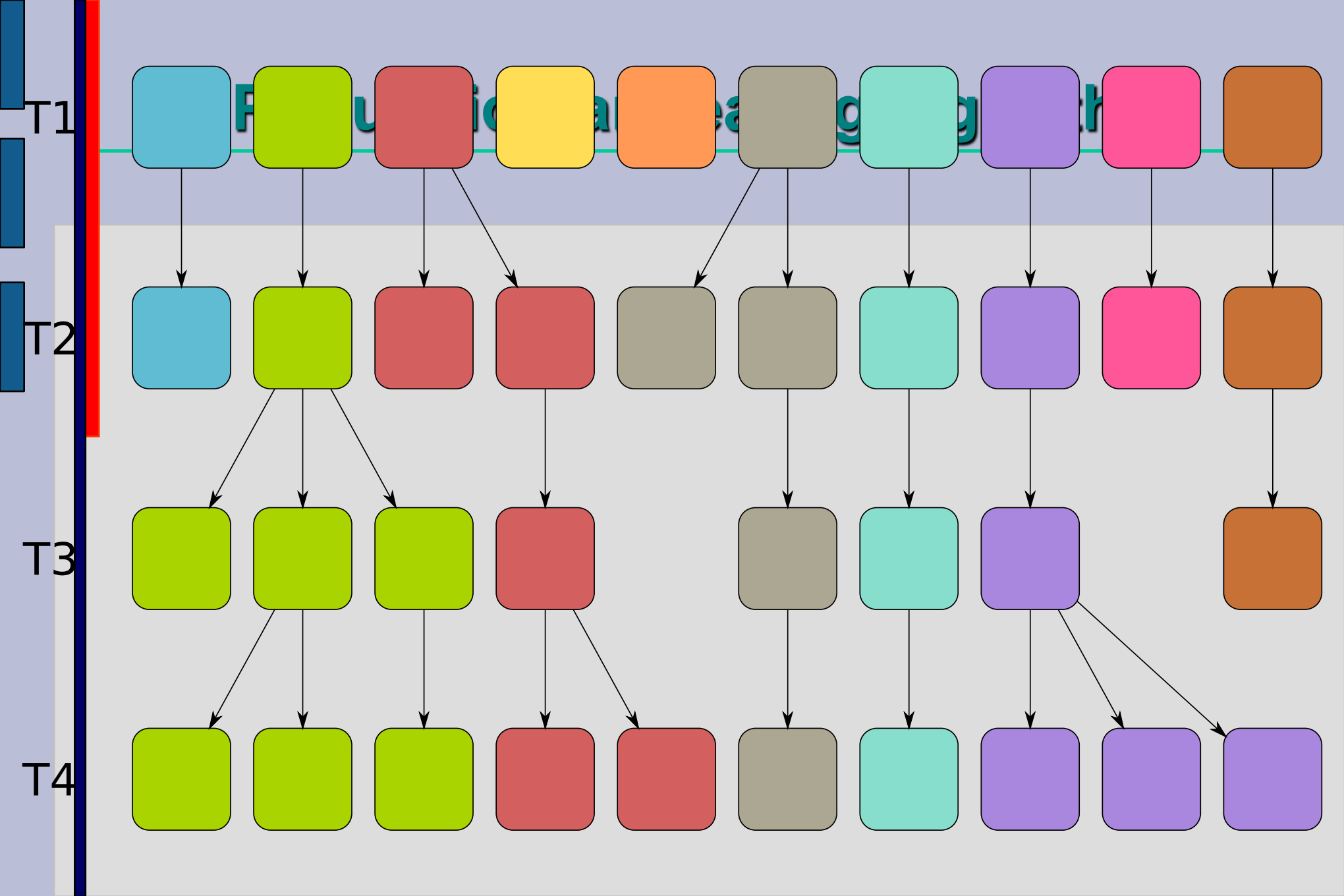
Population annealing algorithm

1. Set up an equilibrium population of $R_0 = R$ independent copies (replicas) at inverse temperature β_0.
2. Propagate the population to the inverse temperature β_i $(i = 1, 2, \ldots)$: Resample replicas $j = 1, \ldots, R_{i-1}$ with their normalized Boltzmann weights $\hat{\tau}_i(E_j) = (R/R_{i-1}) \exp\left[-(\beta_i - \beta_{i-1})E_j\right]/Q_i$, where

$$Q_i = \sum_{j=1}^{R_{i-1}} \frac{e^{-(\beta_i - \beta_{i-1})E_j}}{R_{i-1}}. \tag{1}$$

3. Update each replica by θ sweeps of the chosen MCMC algorithm at inverse temperature β_i.
4. Calculate estimates for observables $\mathcal{O}$ as population averages $\sum_j \mathcal{O}_j/R_i$.
5. Goto step 2 unless the target temperature β_f has been reached.

Machta 2010; LS, Barash, Weigel, Janke 2019


T1
T2
T3
T4

Population annealing algorithm

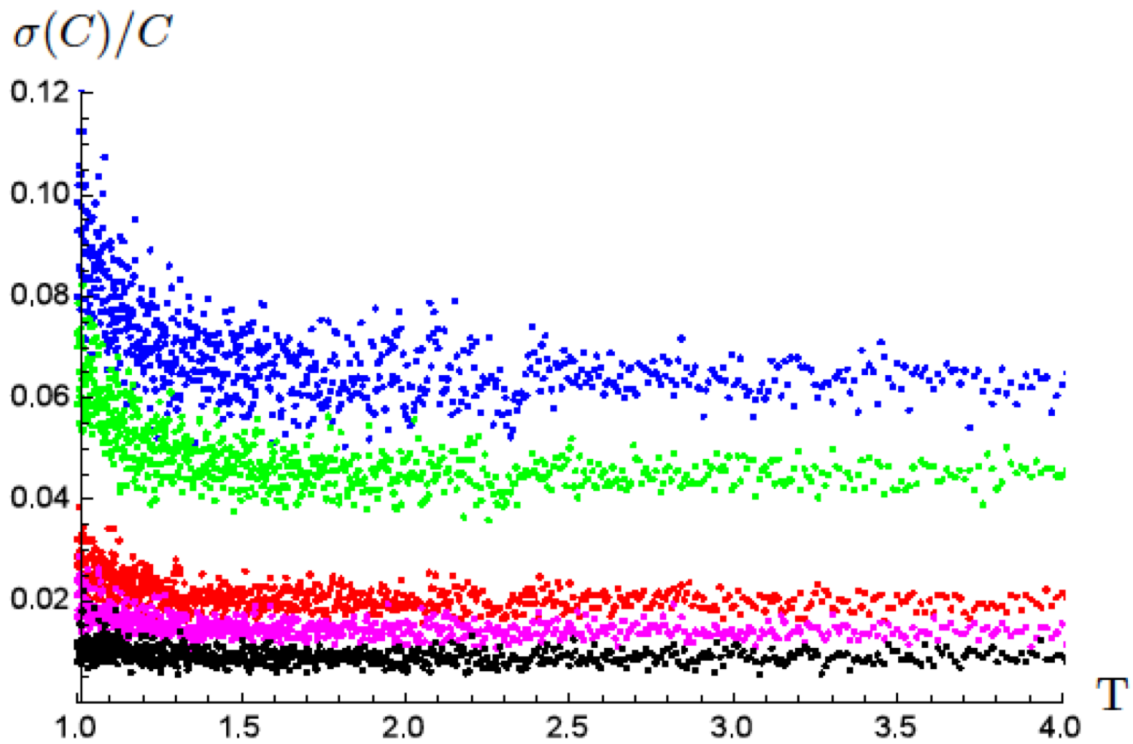


FIG. 5: Relative specific heat errors, $\sigma(C(T))/C(T)$, as a function of the temperature for a different number of replicas R, $R = 500$ - blue, $R = 1000$ - green, $R = 5000$ - red, $R = 10000$ - pink, and $R = 25000$ - black.

Barash, Weigel, Janke, LS – 2017

Population annealing algorithm

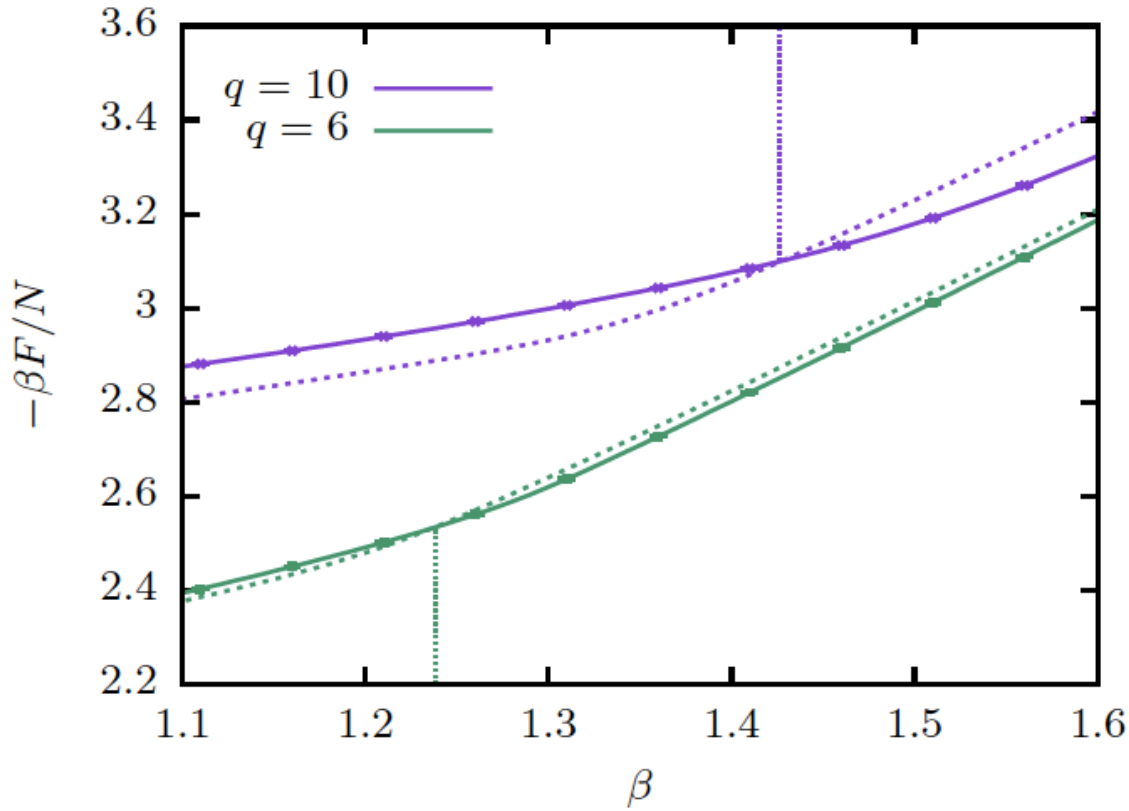
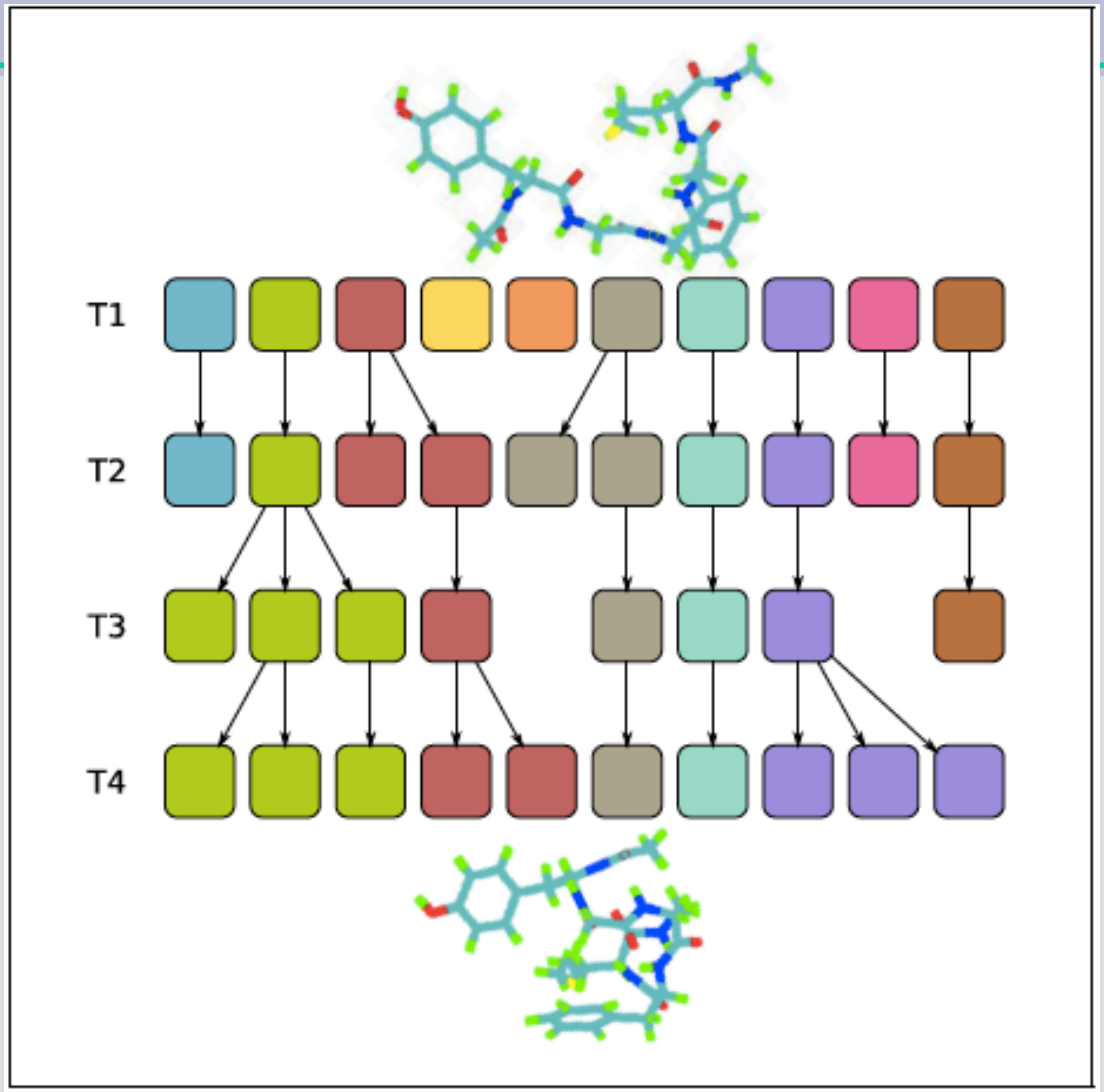


Fig. 2. Metastable free energy for the q-state Potts model with $q = 6$ (green) and $q = 10$ (magenta) for a cooling (solid lines) and a heating (dashed lines) cycle. The vertical dotted lines indicate the locations of the transition points β_t. The lattice size is $L = 32$, and the PA parameters are $\theta = 10$, $R = 10000$, and $\Delta\beta = 0.01$. Taken from Ref. [27].

Barash, Weigel, LS, Janke – 2017

Population Annealing - Simulations of Biopolymers



Population Annealing - Simulations of Biopolymers

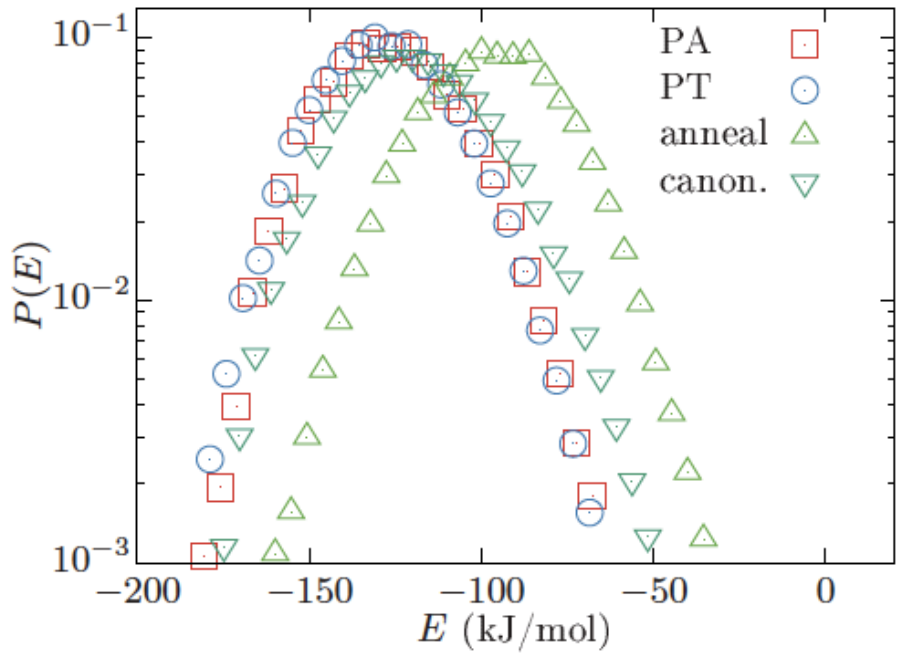


Figure 3: Energy histograms at the lowest temperature, $T = 200$ K, as obtained from the population annealing (PA), parallel tempering (PT), population annealing without resampling ("anneal"), and canonical ("canon.") simulations, respectively.

GPU accelerated PA algorithm

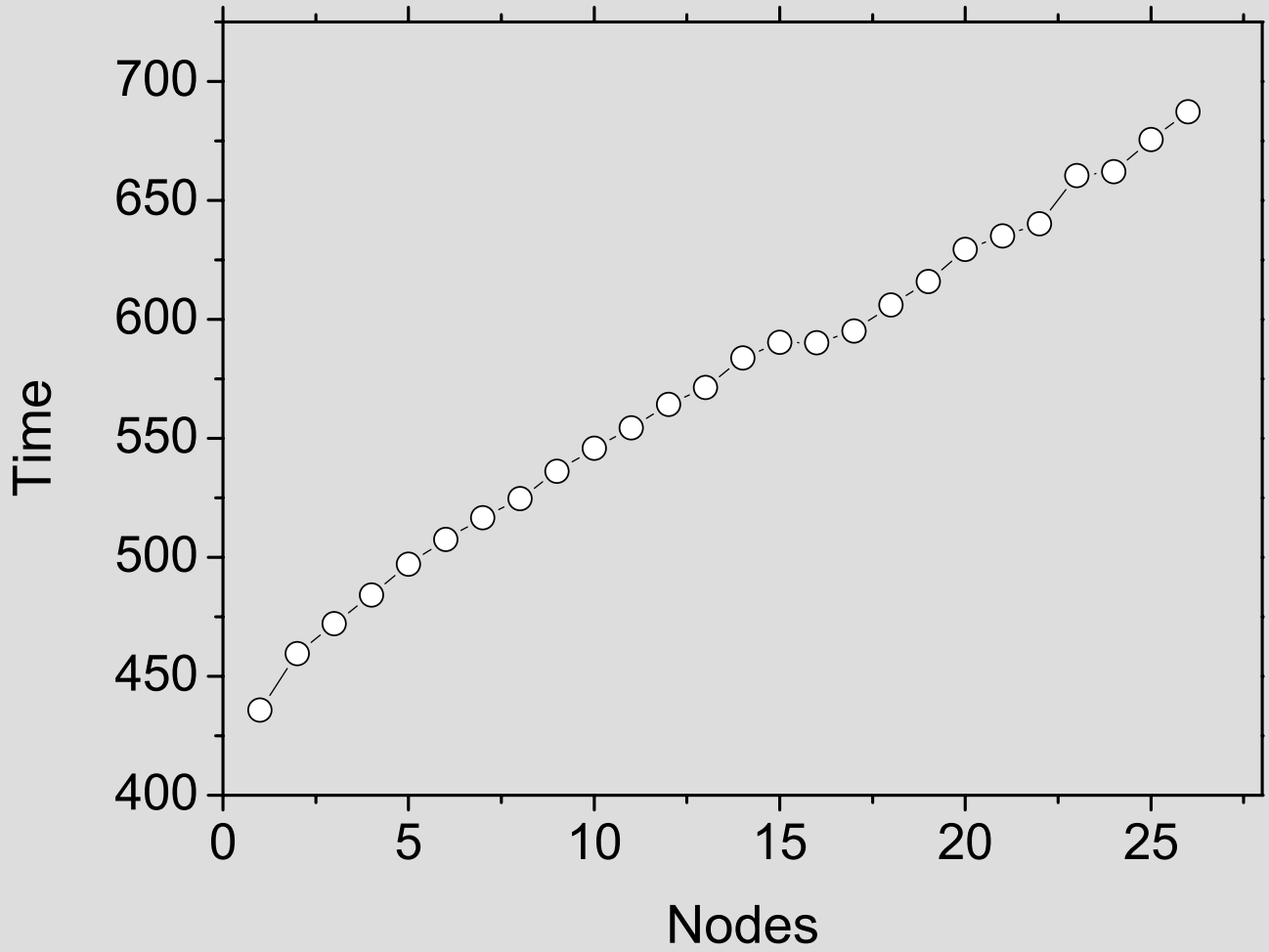
(1) initialization of the population of replicas (kernel `ReplicaInit`)
(2) equilibrating MCMC process (kernel `checkKerALL`)
(3) calculation of energy and magnetization for each replica (kernel `energyKer`)
(4) calculation of $Q(\beta, \beta')$ (kernel `QKer`)
(5) calculation of the number of copies n_i of each replica i (kernel `CalcTauKer`)
(6) calculation of the partial sums $\sum_{i=1}^{j} n_i$, which identify the positions of replicas in the new population (kernel `CalcParSum`)
(7) copying of replicas (kernel `resampleKer`)
(8) calculation of observables via averaging over the population (kernel `CalcAverages`)
(9) calculation of histogram overlap (kernel `HistogramOverlap`)
(10) updating the sum of energy histograms $\sum_{i=1}^{K} P_{\beta_i}(E)$ for the multi-histogram reweighting (kernel `UpdateShistE`)

GPU accelerated PA algorithm

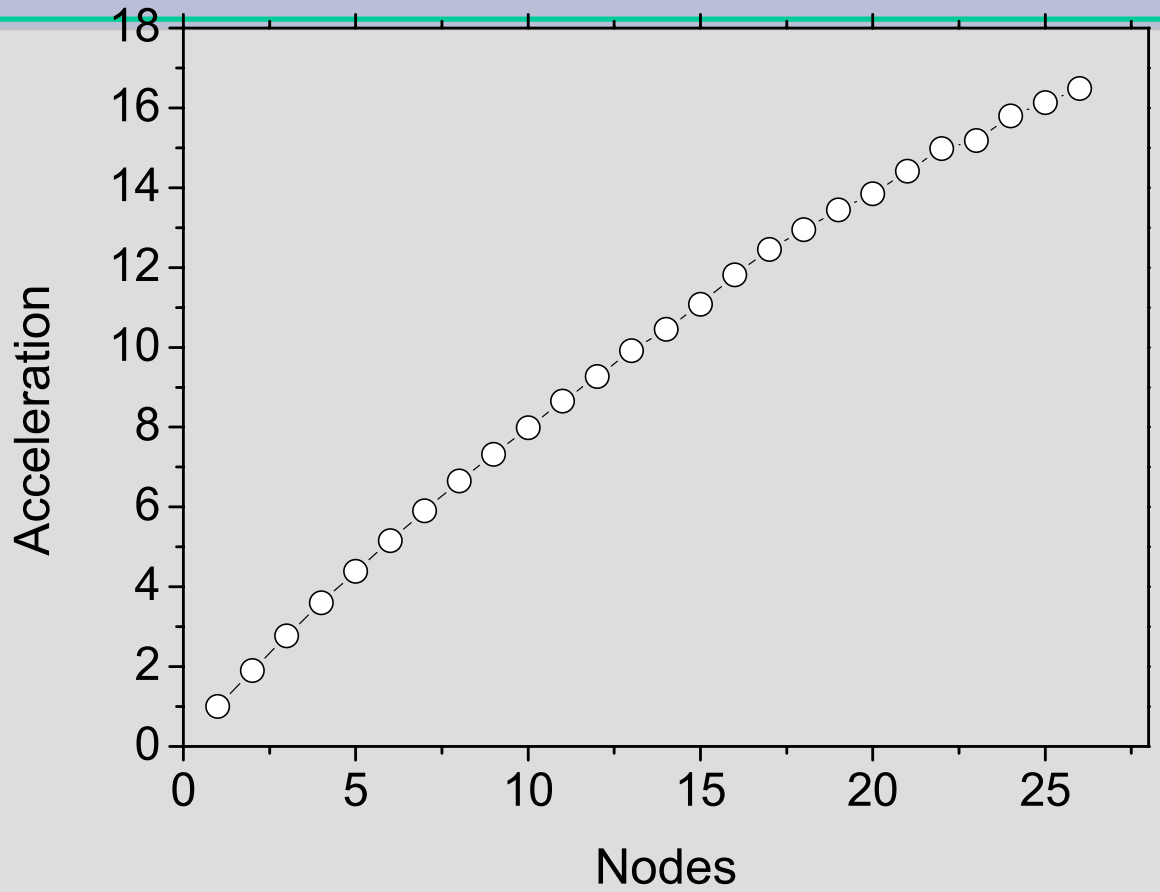
Table 1. Peak performance of the CPU and GPU PA implementations in units of the total run time divided by the total number of spin flips performed, for different system sizes. The best GPU performance is achieved for large θ, and here $\theta = 500$ was chosen for a population of $R = 50\,000$ replicas. GPU performance data are for the Tesla K80 (Kepler card) and GeForce GTX 1080 (Pascal card). The sequential CPU code was benchmarked on a single core of Intel Xeon E5-2683 v4 CPU running at 2.1 GHz.

	time per spin flip (ns)		
	CPU	GPU (Kepler card)	GPU (Pascal card)
$L = 16$	23.1	0.094	0.038
$L = 32$	22.9	0.092	0.034
$L = 64$	22.6	0.092	0.036
$L = 128$	22.6	0.097	0.039

CUDA+MPI accelerated PA algorithm



Simulations on the full-scale of SCC NRU HSE

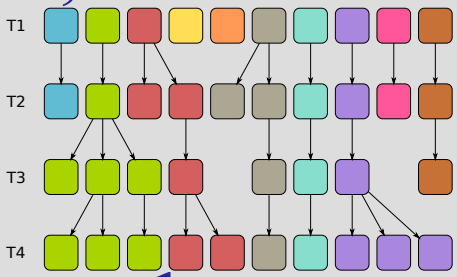


26 nodes x 4 V100 = 104 GPGPU

MPI/CUDA PA implementation

Technical problems solved:

- Load balancing of all GPU nodes with an approximately same number of replicas;
- Blocks of replicas = 2000;
- 10 blocks of replicas per GPU;
- Minimization of the extensive memory exchange between nodes;
- Approx. 2.5 million replicas in one run
  - technically impossible with the smaller clusters!



Conclusion

- *Population annealing* approach is the promising tool for the number of problems:

• "complex" ground state,

• the rough energy landscape,

• optimization problems.

- The *Population annealing* algorithm is the natural candidate for large-scale and *fully scalable simulations* with the *heterogeneous parallelism*.

- One can use *Population annealing* algorithm for the full-scale simulations on SCC NRU HSE.