Volume 14, November 2016

Science Highlight

A Scalable Parallel Simulated Annealing Algorithm for the Optimization of a Developmental Gene Regulatory Network

Zhihao Lou

The simulated annealing algorithm is a powerful optimization algorithm that relies only on the value of the cost function being optimized. This lack of requirement on a gradient enables great flexibility with respect to the models being studied. The inference of gene regulation networks from experimental data is one such application. It features a system of non-linear ODEs with 46 parameters. The model output is then compared to a collection of time series data obtained from experiments. An efficient parallel simulated annealing algorithm is necessary to explore the search space and find a high quality solution. Here we present a scalable parallel simulated annealing algorithm that works well with this production scale problem.

The simulated annealing algorithm mimics the physical annealing process by treating the value of the cost function as an energy E and sampling it according to the Boltzmann distribution at some artificial temperature T using the Metropolis algorithm. At each iteration, the algorithm generates a perturbed state (a "move") with energy E_{new} from the current state with energy E_{old}. If $E_{\text{new}} < E_{\text{old}}$, the new state is accepted. Otherwise the new state is accepted with probability $\exp[-(E_{\text{new}} - E_{\text{old}})/T]$. During the annealing process, T is decreased slowly and uphill moves become less and less likely. The algorithm stops when no more moves are possible. An implementation of the simulated annealing algorithm needs to specify the cooling schedule and the move generation scheme. For a parallel algorithm, it needs extra specification on how the states across the processor cores interact with each other.

Before we dive into the details of the algorithm, we need to first define how the performance will be measured. For a stochastic optimization problem, the quality of results in general depend on the number of iterations used. An accurate speedup must take into account changes in the quality of the answer that come from parallelization. To do this, we empirically measure the tradeoff relation, and conclude the number of iteration in serial $N_s(E_f)$ to be a function of the final energy E_f. The adjusted speedup S_p is then given by

$$S_P = \frac{N_S(E_P)}{N_P},$$

where E_P and N_P are the final energy and number of iteration of a parallel run on P cores. In addition to the speedup, we also use the success rate to characterize the performance of multiple annealing runs under same settings. The results of annealing runs tend to cluster together to show a two-peak distribution with a low-density valley in histogram, where one cluster is close to the true minimum and the other is far away from it. Such behavior reveals the structural barrier in the search space. In turn we use the valley in the histogram as the success threshold for calculating the success rate.

The details of our algorithm are as follows. The move generation we use is adapted from a scheme pioneered by Lam and Delosme. At each step, a proposed move is generated by drawing the perturbation from a Laplace distribution with mean 0 and variance $2\theta^2$. The average move size θ is adaptively adjusted based on the idea that on average a larger move is less likely to be accepted than a smaller move. Thus, at every I_{mc} steps, θ is adjusted by feedback move control so that

$$\ln \theta_{\text{new}} = K(\rho - 0.44) + \ln \theta_{\text{old}}$$

Training

Intro to Python
November 9th, 10AM
Room 240A, at the Computation Institute of the University of Chicago

Topics will include:

- Python on Beagle
- Datatypes, functions
- Modules, plotting, iteration
- Object orientation, debugging
- NumPy arrays
- SciPy
- Importing and cleaning data

Beagle2 Events Learn more about Beagle2 trainings.

Resources

- Beagle2 Wiki Get detailed usage information from the Beagle2 team
- Beagle2 Support Contact the Beagle2 experts for help
- Globus for file transfer Get started moving files to/from Beagle2 using this fast service
- Other CI resources Learn about other computing resources available at the Computation Institute

where K is a problem dependent proportional gain parameter and ρ is the acceptance ratio. In parallel, ρ is computed from data collected from all processors so that the average move size θ is the same across all cores. The cooling schedule used in our algorithm is the exponential cooling schedule, which specifies

$$T_{n+1} = (1 - \lambda)T_n$$

where λ is the cooling speed. For a parallel setup using P cores, the cooling speed is $\lambda_P = P\lambda_s$, so that it samples the same number of states over the same temperature span as in serial. To compensate for the rapid cooling in parallel, the states across the cores are resampled periodically. Every R steps, all the energies are pooled together. Each core independently adopts a target state with energy E_i from the pool with probability

$$P(E_i) = \frac{e^{-E_i/T}}{\sum_j e^{-E_j/T}}.$$

When the temperature is high, every state has roughly equal probability being adopted. As the system cools, a better state is more likely to be adopted by multiple cores and hence be replicated across the system.

The algorithm is implemented in C++ using the MPI library for communication. All the numerical experiments were conducted on Beagle2, a Cray XE-6 system at the Computation Institute of the University of Chicago. In particular, the acceptance ratio statistics and energy pools are collected using blocking global communication routines to take advantage of Beagle's superior interconnect infrastructure, while the resampling of states uses one sided communication routines specified in MPI v2 and later.

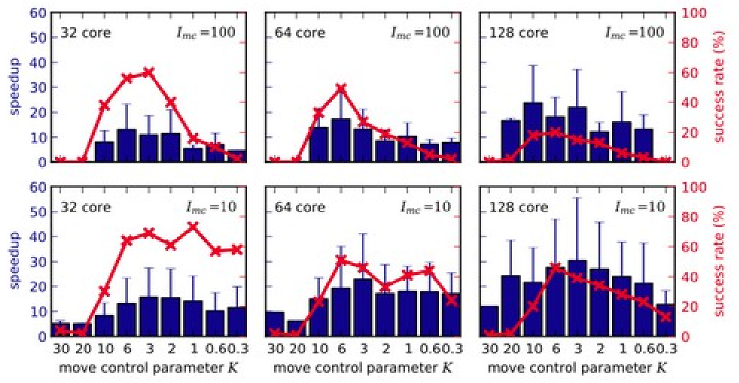

Figure 1. The effects of different move control interval and gain parameter K. The results presented are the success rate (red, right axis), and for successful runs, the mean and standard deviation of adjusted speedup (blue, left axis) for 32, 64, and 128 core runs over *Imc*=100 (top) and *Imc*=10 (bottom).

In parallel simulated annealing, the parameters I_{mc} and K in move control have a large impact on the performance. Traditionally, the move control interval I_{mc} is set to 100 to make sure there are adequate moves presented before a move control. When P is large, this large I_{mc} will cover a wide range of temperature and make the acceptance ratio ρ estimates inaccurate. At the same time, large P also causes more data about moves to be collected. That means keeping I_{mc} at 100 is no longer necessary. We compare the performance between $I_{mc} = 100$ and $I_{mc} = 10$ using a range of different K values (Figure 1), and conclude that the combination $I_{mc} = 10$ and $K = 3$ will give the best performance regardless of P.

To read the whole article go to: http://beagle.ci.uchicago.edu/science-at-beagle/

Beagle2 Related Publications

- Z. Lou, J. Reinitz
  Parallel simulated annealing using an adaptive resampling interval.
  Parallel Computing; 53: 23-31
- R.C. Cockrell, M. E. Stack, G. An
  Supercomputing ulcerative colitis-associated cancer simulations to bridge mechanism with epidemiology
  Digestive Disease Week, San Diego, CA, 5/21/2016
- R.C.Cockrell,G. An
  Characterizing the behavioral landscape of sepsis: supercomputing simulation of 40 million in silico patients
  Society of Critical Care Medicine Annual Congress, Orlando, FL 02/20/2016
- R.C.Cockrell,G. An
  Supercomputing sepsis simulations for in silico outcome prediction
  Academic Surgical Congress 2016, Jacksonville, FL 02/03/2016
- R.C.Cockrell,G. An
  Investigating the development of ulcerative colitis-associated cancers through an agent-based model
  Academic Surgical Congress 2016, Jacksonville, FL 02/03/2016