Implementing a multi-core algorithm on a massively parallel low-power system

Andrew D. Gait
andrew.gait@manchester.ac.uk

Juan Yan, Michael Hopkins, Andrew Rowley, Steve Furber

School of Computer Science, University of Manchester

1 Abstract

SpiNNaker [1] is a massively parallel low power supercomputer designed to model large spiking neural networks in real time through e.g. the use of PyNN [2], consisting of roughly 1 million ARM cores, each with a low instructional memory (ITCM) limit of 32K. However, applications for the machine are not limited to neuromorphic models; for example, code for Markov Chain Monte Carlo (MCMC) inference has been designed and found in most useful cases to well exceed the ITCM limit on a single core. This requires the use of multiple cores to each perform different parts of the algorithm, and the communication of (short) messages and the sharing of data between a subset of cores in order to perform the calculations required at each timestep. Results show comparisons between an implementation of parameter estimation using MCMC for an ARMA (auto-regressive moving average) time series model working with wind power data [3], on both SpiNNaker (Python host code / C machine code) and on a single machine in MATLAB.

2 SpiNNaker Machine

SpiNNaker is designed to be a low-power, highly interconnected machine for running neural networks, using multicast packets to efficiently communicate from one place to many others. However, its use is not limited to this case; due to its power efficiency of 1W per chip (containing 18 cores) and 1kW per sub-rack (containing 24 48-chip boards), it is also viable as an alternative to larger and more conventional supercomputers.

Figure 1: Machines: (a) 4-node board, (b) 48-node board, (c) 600 boards

3 Markov Chain Monte Carlo

MCMC can be used to provide Bayesian posterior inference for a time-series forecasting problem: in this case, wind-power generation. The time series ARMA(p, q) model is used:

$$z(t) = a_1 z(t-1) + a_2 z(t-2) + ... + a_p z(t-p) + e(t) - b_1 e(t-1) - ... - b_q e(t-q) + \mu \quad (1)$$

where z is the time series variable and e is a white noise sequence, and where the parameters provided are $\{a_1, ..., a_p, b_1, ..., b_q, \mu, \sigma\}$ (σ is the standard deviation of the noise signal). The coefficients are solved for by solving for the roots using the characteristic equations of 1. The log-likelihood function used is:

$$\log L = \log \prod_{i=1}^{N} \frac{1}{\sqrt{2\pi\sigma^2}} \exp(-\frac{err^2}{2\sigma^2}) \quad (2)$$

After setting the intial posterior P for a random starting point θ_0, a matrix SM of size N*$(p+q+2)$, and initialise a random walk step function using a standard t distribution with $p+q+2$ elements, for each subsequent iteration:

- Update $\theta_{new} = \theta_{old} + \Delta\theta$ where $\Delta\theta$ is a random walk using a step function
- Calculate (new) posterior P_{new} as prior * likelihood for the solution θ_{new}
  - The log-prior is calculated by solving for the roots of equation 1, with a value of 0 if all roots lie outside the unit circle and $\sigma > 0$, or -1000 otherwise.
  - The log-likelihood is calculated using equation 2.
- Test new posterior against old posterior using $if P_{new}/P_{old} > \rho$, where ρ is a random number in $(0, 1)$
- If true, accept and set $\theta_{old} = \theta_{new}$, otherwise keep θ_{old}
- If SM now has N completed rows, then
  - Perform a Cholesky decomposition of $cov(SM)$ to get adaptive rotation matrix R
  - Set the step function to be R multiplied by the t distribution vector, and reset the SM matrix to be empty
- else, append θ_{old} as the next row of matrix SM.
- θ_{old} is saved on the shared memory of the chip at a user-specified thinning interval, once a user-specified burn-in period has completed.

4 Implementing the algorithm on SpiNNaker

A simple MCMC problem requires a single core (a coordinator, C) per board to read the data from the Host and inform the other cores (V) where to read it from. The Host sends each of the cores the (initial) calculation parameters.

For the ARMA problem, where the prior probability requires the finding of the complex roots of two high-degree polynomials, an extra core (Rt) is required for this calculation. The transition jump function is also modified by a Cholesky decomposition every N timesteps, and this algorithm also requires an extra core (Ch). Thus, the basic algorithm is as follows:

- Send the data to be analysed from Host to the coordinator C, and the parameters required from Host to worker cores V
- Cores V perform the calculations based upon data and parameters:
  - Calculate prior probability using Rt cores
  - Calculate likelihood (on V cores)
  - Calculate transition jump ahead of next time-step using Ch cores
  - V cores record a sample at every user-specified thinning interval
- Once the calculation reaches a user-specified number of time-steps, the V cores return their recorded samples to the Host.

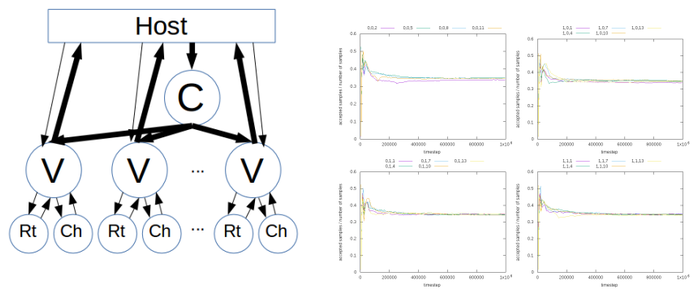

Figure 2: (a) ARMA-MCMC application graph, (b) Acceptance rates for samples using a 4-chip SpiNNaker board

5 Results

Results are shown in figure 3 for parameters when running ARMA-MCMC algorithm using MATLAB (black), and a 4-chip SpiNNaker board (colours), collecting 20k samples for each set of multi-cores used in the run (one every 50 timesteps of a 1 million-timestep run). The full run takes roughly one day in all cases; MATLAB returns a single set of results and there are 19 sets of results from the SpiNNaker board. Increasing the size of the SpiNNaker machine used will increase this runtime slightly due to the overheads involved in loading the data onto and collecting the results from the machine, but will result in more (and consequently, better) results. Figure 2 (b) shows the sample acceptance rates for each set of three cores running the algorithm on a 4-chip SpiNNaker board.

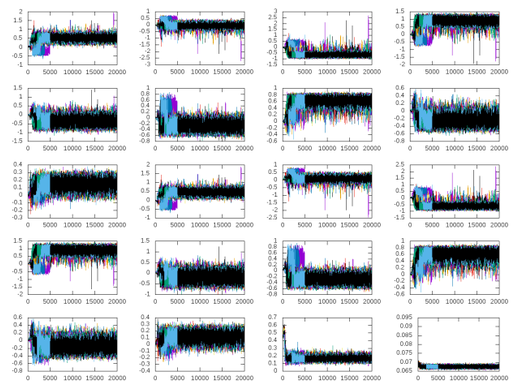

Figure 3: From top left to bottom right, results for nine AR polynomial coefficients, nine MA polynomial coefficients, mean and standard deviation

References

[1] Steve Furber et al. (2013). *Overview of the SpiNNaker System Architecture*

[2] Andrew P. Davison et al.(2008). *PyNN: A Common Interface for Neuronal Network Simulators*

[3] George Papaefthymiou and Bernd Kloeckl *MCMC for Wind Power Simulation*

[4] AD Gait, AGD Rowley, J Yan, M Hopkins et al. (2018) *MarkovChainMonteCarlo 4.0.0*, https://github.com/SpiNNakerManchester/MarkovChainMonteCarlo

The University of Manchester
Human Brain Project
EPSRC Engineering and Physical Sciences Research Council
erc