Electronics and Computer Science Faculty
of Physical and Applied Sciences
University of Southampton

Alexandros-Pana Oikonomou
1^{st} May 2013

Solving the Conjugate Gradient Method in a SpiNNaker Machine

Project Supervisor: Jeff Reeve
Second Examiner: Marcus Breede

A project report submitted for the award of
BSc Computer Science

Abstract

SpiNNaker is an asynchronous, event-driven parallel architecture designed to simulate the human brain. It has been designed to operate as a large scale neural network in real-time using a System-on-Chip multi core system. Its architecture is different from usual parallel computers, since cores use spikes to communicate with each other. That way usual pitfalls of parallel computing, such as race conditions and deadlocks are avoided. So far the most prominent uses of this architecture have been in neuroscience and robotics. The aim of this project is to put into use SpiNNaker's architecture and bring it closer to classic computer science problems, while solving them optimally. The given algorithm to solve in this project is the conjugate gradient method, an iterative way of solving systems linear equations. The algorithm successfully runs on the simulator and reduces the time complexity of the most expensive operations of the algorithm.

Contents

Acknowledgments and Statement of Originality

I would like to thank my supervisor Jeff Reeve for his help and support throughout this project.

1 Introduction

1.1 Aim

The aim of this project is to correctly solve the Conjugate Gradient Method[7] on a SpiNNaker chip, thus using the massive parallelism that this machine offers to reduce the time complexity of the aforementioned algorithm. This is accomplished by reducing the time complexity of the most expensive operations of the algorithm which are matrix-vector multiplication and the scalar product of vectors. The complexity is reduced dramatically, due to the abundant number of cores provided from the architecture. This report explains this project and its constituents, any background research done to launch this project, along with designe and implementation choices.

1.2 Reasons and Justification

SpiNNaker is an architecture inspired by the biology of the human brain. Its optimal configuration has over a million cores[11], which have mainly been used to simulate the neurons of the human brain and in robotics. Examples of these would be using the SpiNNaker chip to simulate thousands of spiking neurons by using over four million synapses[15], or to simulate the neurons of a retina sensor[3].

However little work had been done into using the SpiNNaker architecture to solve classic computer science problems. That is why a problem such as the Conjugate Gradient Method had been proposed, which is a very common solution to optimization problems. In addition to that, the SpiNNaker architecture offers new parallel programming paradigms, that escape some common parallel programming pitfalls such as race conditions, deadlocks, mutual exclusion etc[16].

1.3 Overview

Give a brief overveiw of what each section contatins

2 Background

2.1 The neuron

To make the explanation of the SpiNNaker architecture easier, the design from which the SpiNNaker chip was inspired will be outlined. This is no other than the human neuron.

The human neuron is an electrically excitable cell that processes and transmits information through electrical and chemical signals. Its basic costitutes are the the soma, the dendrites and the axon. The soma is the body of the neuron. A dendrite receives signals from the soma of the neuron that it belongs to or other neurons. The dendrite extends for hundreds of micrometers and branches multiple times, thus forming a dendritic tree, which connects with other neurons axons. The axon is used to transmit signals to other neurons and it extends from the soma of the neuron to a dendrite. All human neurons have only one axon. Given the above analysis the dendrites could be described as the inputs of a neuron

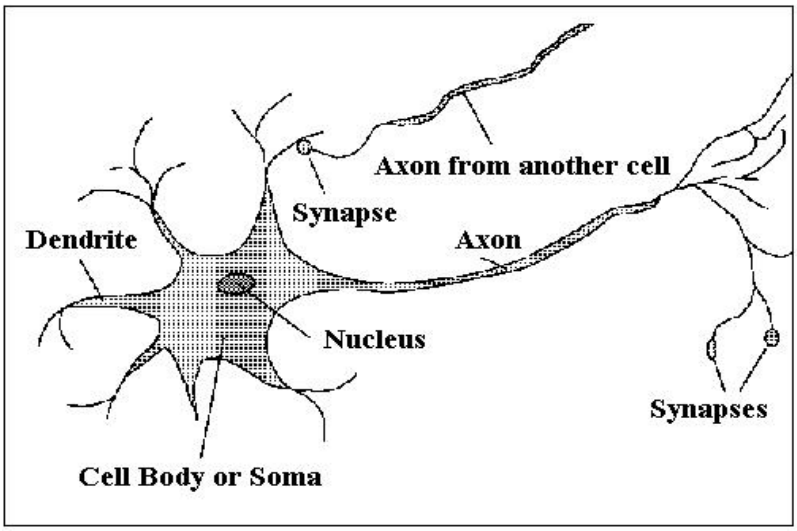

Figure 1: A neuron

One of the most important parts of the neuron structure is the synapse. The synapse is the contact between the axon of a neuron and the dendrite or the soma of another. It is where a information from one neuron is transmitted to the other. When a set of neurons are connected with each other through synapses, then they create a neural network.

Finally, the communication between neurons is accomplihed through spikes, which are either chemichal or electrical[6].

Given the terms in this section, the name of the SpiNNaker chip is detuctable. It stands for Spi(king)N(eural)N(etwork) architecture.

2.2 The SpiNNaker architecture

As mentioned before the SpiNNaker architecture is inspired by the biology of the brain, and more specifically, neurons. However, its architecture is not constrained by the biology of the brain, but many techniques to speed up computations are used. It differs from other supercomputers, which usually have a lot of strong processors with slow network capabilities. The design of the architecture was made with having as priority two concerns. The first one being MIPS(millions of instructions per-second) per mm^2. Namely how many instructions can be performed in an area of silicon. The second one was MIPS per watt, which means how many instructions per second can be performed given a fixed amount of energy. Its most optimal configuration will use a million cores and will be able to simulate over a billion neurons[4].

2.2.1 SpiNNaker chip Overview

The heart of the SpiNNaker architecture is the SpiNNaker chip. Its main components are 18 identical ARM cores, a router, a system NoC(Network-on-Chip) which connects to the router of the chip and a 128 MB SDRAM. At startup, in each chip a processor is selected to act as a Monitor processor, with the functionality of performing the management of the given chip's system. The remaining ARM cores perform the calculations for the given problem, with the exception of one which is reserved as a spare, in the emergency of another core having a malfunction[4]. Each core has the ability to hold 32KB of instructions and 64KB of data. Since this is not enough space, all other data is saved in the SDRAM which can hold up to 1GB of data[11].

2.2.2 The router

As mentioned before the architecture supports low-delay communication, with less powerful processors, than those used in most supercomputers. To accomplish that, the system uses asynchronous multicast packets to communicate between cores. This is done through the router which exists in each chip.

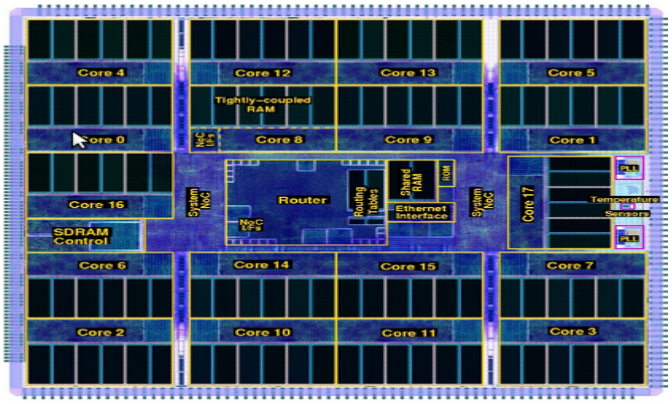

Figure 2: The SpiNNaker chip[8]

The router exists as the part of the NoC and its primary role is to direct packets that arrive and packets that are sent or need to be forwarded. The router has 20 ports for the ARM cores that are located in the given chip and is able to forward one packet at a time. The router works faster than a transmission port, which results into the router being most of the time lightly used. It is designed to support point-to-point communication, using small packets. Using multicast packets helps reduce the amount of packets that exist in any givne moment in the network. To help with that, the architecture supports default routing, which means that some connections do not need to be in some routing tables for them to be forwarded to their eventual destination. This concept will be explained in the next section.

The functionality of the router is pretty simple. When a packet arrives from the input port, then the router will try to send it to an output port. If there is a problem during transmission, then the router will keep trying to send it and after a while it will try the emergency route(also explained in the next section). If the emergency route still does not work and an amount of time is passed, then the packet will be dropped. To help with that, each packet has the time it was transmitted in their header.

To further understand this concept, the topology of the system needs to be considered.

2.2.3 Topology

In order to have a million cores available for processing a number of SpiNNaker chips need to get connected and work efficiently as an architecture. To reach that goal an effective topology is important. Considering Fig-

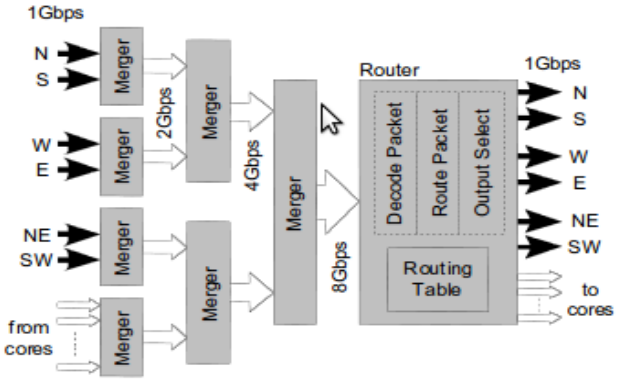

Figure 3: The router[11]

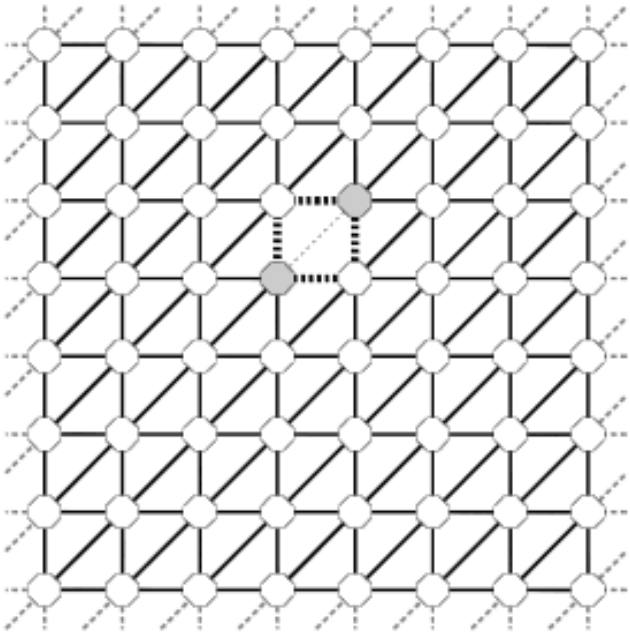

Figure 4: *The SpiNNaker topology[11]. This is an example of what an 8x8 would look like.Each circle is a SpiNNake chip and the lines between them are the connectivity.*

ure 4, it is visible that the topology is viewed by the system as a 2D mesh, with chips having a specific amount of neighbours and specific connections. Any chip is able to communicate directly to 6 other chips. This is also depicted in Figure 3, where the positions of these chips are shown. In general each chip directly communicates with chips that are North, South,

West, East ,NorthEast, and Southwest of its respective position. This however does not apply for chips that are in the perimeter of the mesh, since they in some of the positions described above chips will not exist.

Returning now to the default routhing mentioned in the previous section, if a packet arrives to a router, and the router does not have an entry for it, a then it will be forwarded to the chip opposite to the one it came from. For example, if a packet comes from the East and the aforementioned condition is met, then it will be forwarded to the West. This helps reduce the size of the routing tables in each chip[9].

In addition to the default routing, the topology also provides for two-hop routes aming neighbor chips, as shown in Figure 4. These routes are named emergency routes and their functionality is to bypass any faulty connections that might exist between connections.

2.3 The Conjugate Gradient Method

The conjugate gradient method is an algorithm for the numerical solution of systems of linear eqations of the type Ax=b, those whose matrix is symmetric and positive definite.

A *symmetric* matrix is a matrix which is equal to its transpose. If A is a symmetric matrix then A=A^T. The entries of the matrix are symmetric with respect to the main diagonal, so if an element of the matrix A is a, then a_{ij}=a_{ji}. A *positive definite* matrix M is a matrix which when multipled by any non-zero vector z and its transpose z^T, is always positive. In short the relationship that needs to be satisfied is zMz^T>0

It is an iterative method, which means it can be applied to sparse systems. A *sparse matrix* is a matrix which is populated primirily with zeros. Its opposite would be a *dense matrix*. It was developed by Magnus Hestenes and Eduard Stiefel and can be used to solve optimization problems[14].

2.3.1 The quadratic form

In order to explain why the Conjugate Gradient Method can solve problems whose matrix is positive-definite and symmetric an explanation of the quadratic form needs to be presented.

$$f(x) = \frac{1}{2}x^T Ax - b^T + c \tag{1}$$

The gradient $\nabla f(x)$ of the quadratic form is a vector field that points in the direction of the greatest increase of $f(x)$. If we were to take into account the i^{th} component of ∇f, then.

$$\begin{aligned} f(x+\psi e_i) &= \frac{1}{2}(x+\psi e_i)^T A(x+\psi e_i) - b^T(x+\psi e_i) + c \\ &= \frac{1}{2}(x^T Ax + \psi e_i^T Ax + x^T A\psi e_i) - b^T(x+\psi e_i) + c \end{aligned} \tag{2}$$

Note that e_i is the error affiliated with taking the i^{th} component of ∇f. Now from the definition of a derivative $f(x)$=$\frac{f(a+h)-f(a)}{h}$ and (2) we have.

$$\frac{f(x+\psi e_i) - f(x)}{\psi} = \frac{\frac{1}{2}(\psi e_i^T Ax + x^T A\psi e_i) - \psi e_i^T b}{\psi} \tag{3}$$

(3) is the same as the ith component of $\frac{1}{2}$(Ax+A^{T}x)-b So we have

$$\nabla f = \frac{1}{2}A^T x + \frac{1}{2}Ax - b \tag{4}$$

But if A is symmetric, then it is obvious that we have $Ax = b$ at the minimum of f. In addition if A is positive negative then f is concave up.

2.3.2 Conjugate Gradients

As mentioned before the matrix A must be symmetric and positive definite for the Conjugate Gradient Method to work. Given a quadratic function as defined in (1), then it turns out that $-\nabla f = b - Ax = r$.

The CGM, as well as most optimization methods(for example the method of the Steepest Descent[**?**]), look towards a specific direction everytime they are to move to a correct position and try to look for the best step given their gradient and direction

$$x_{k+1} = x_k + \alpha p_k \tag{5}$$

Given (5), α can be deduced from the fact that e_{k+1}(the error vector of each iteration) must be orthogonal to p$_k$, which also means that p$_k$ will never be met again as the solution progresses. Using this we can dedcuce

$$\begin{aligned} p_k^T e_{k+1} &= 0 \\ p_k^T (e_k + \alpha p_k) &= 0 \\ \alpha &= -\frac{p_k^T e_k}{p_k^T p_k} \end{aligned} \tag{6}$$

The equation at (6) however is not solvable. The value of e_k is needed in each iteration, which is not computable at that time. To help with that the search directions in each iteration(p_k) is defined as A-orthogonal in respect to the matrix A, instead of just orthogonal. The definition of A-orthogonal vectors is, given a matrix A, then:

$$p_k^T A p_k = 0 \tag{7}$$

Given (7), e_{k+1} and p_k must be A-orthogonal. If we wish to find the optimal value for α,then the gradient must be set to 0. This leads to the following equation.

$$0 = \frac{d}{d\alpha} f(x_{k+1}) = \nabla f(x_{k+1})^T \frac{d}{d\alpha} x_{k+1} = -r_{k+1}^T \frac{d}{d\alpha}(x_k + \alpha p_k) = -r_{k+1}^T p_k = p_k^T A e_{k+1} \tag{8}$$

From (6),(7) the equation for α when e_k and p_k are A-orthogonal is

$$\begin{aligned} \alpha &= -\frac{p_k^T A e_k}{p_k^T A p_k} \\ &= \frac{p_k^T r_k}{p_k^T A p_k} \end{aligned} \tag{9}$$

Another important equation that comes from analysing the error term is

$$p_k^T r_k = u_k^T r_k \tag{10}$$

In (10) u_k^T are u vectors that span the vector subspace of the vector field of p_k. They are also orthogonal to r_k and u_{k+1} is also orthogonal to all the previous u vectors.

Using (9) and (10) we are able to reach a more accurate value for α

$$\alpha = \frac{r_k^T r_k}{p_k^T A p_k} \tag{11}$$

Using (9) and (10) and a Gram-Schmidt conjugation we are able to reach and a value for β

$$\beta = \frac{r_{k+1}^T r_{k+1}}{r_k^T r_k} \tag{12}$$

2.3.3 The algorithm

Combining the optimal value of α(8) and β(9) and a good step direction for p_k the Conjugate Gradient Method is formed as follows

r_0=b-Ax_0
p_0=r_0
k=0
for k to 1000000 **do**
  $\alpha_k = \frac{r_k^T * r_k}{p_k^T * A * p_k}$
  x_{k+1}=x_k+α_k*p_k
  r_{k+1}=r_k-α_k*p_k*A
  if r_{k+1}^T*r_{k+1} is small enough **then**
    break
  end if
  $\beta_k = \frac{r_{k+1}^T * r_{k+1}}{r_k^T * r_k}$
  p_{k+1}=r_{k+1}+β_k*p_k
  k=k+1
end for

Some notes that can be made about this algorithm are that the axis is defined by the eigenvectors of A and that the algorithm takes a conjugate step closest to r. Finally, the algorithm guarantees convergeance in at most n steps.[14][17][13]

Ofcourse there is an error that exists in every step, as mentioned before, but it is minimized in each iteration. The error function would be:

$$\left|e_k\right| \leq 2(\frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1})^k \left|e_0\right|_A \tag{13}$$

In Equation 9 κ is the condition number.

Another interesting applicatoin of the Conjugate Gradient Method is that you can apply precondition on it in order to find the solution faster. However, preconditioning is outside of the scope of this project, since it adds a lot more calculation and would need a less general solution.

2.3.4 Parallel solutions of the CGM

Some parallel implementations of the CGM use blocks of the input vectors and matrix and assign them carefully to specific cores, thus letting each core produce output that will be used in the next iteration. These kinds

of solutions work better with vector processors, since it allows a processor to complete large vector operations. Depending on how many processors a machine has, parts of the algorithm can continue being sliced to blocks, until the optimal implementation is reached[12].

Some other parallel implementations try to use more specific preconditionings to achieve faster results, but again for the parallel part, they split the input into blocks to distribute it to various cores[2][1].

One of the most interesting implementations of the CGM, suggests an improved algorithm named ICGS(Improved Conjugate Gradient Squared), which is based in another already altered version of the CGM[10]. This implementation computes all vector-matrix multiplications and inner products concurrently, for each iteration. The communication time between these operations and vector updates has been organised efficiently, so that global communication drops significantly, thus dropping the run time of the algorithm as well[19].

Finally, there are many recent implementations which have used GPU's in order to solve the Conjugate Gradient Method efficiently. These pieces of work use CUDA and various other techniques to accomplish their goal, with some very good results. Ofcourse different GPU models are used, but that just enhances the generality of the solution. Generally though,as is for most parallel implementations of the CGM, the most used technique is reducing the time it takes to compute the matrix-vector multiplication and inner products, thus reducing the time of the algorithm overall[5][18].

References

[1] Loyce Adams. M-step preconditioned conjugate gradient method for parallel computation. 1983.

[2] Loyce Adams. M-step preconditioned conjugate gradient methods. *SIAM Journal on Scientific and Statistical Computing*, 6(2):452–463, 1985.

[3] Sergio Davies, Cameron Patterson, Francesco Galluppi, Alexander D Rast, David Lester, and Steve B Furber. Interfacing real-time spiking i/o with the spinnaker neuromimetic architecture. In *Proceedings 17th International Conference, ICONIP 2010.*, pages 7–11. Australian Journal of Intelligent Information Processing Systems, Vol. 11, No. 1,, 2010.

[4] Steve Furber and Steve Temple. Neural systems engineering. *Journal of the Royal Society interface*, 4(13):193–206, 2007.

[5] V Galiano, H Migallón, V Migallón, and J Penadés. Gpu-based parallel algorithms for sparse nonlinear systems. *Journal of Parallel and Distributed Computing*, 72(9):1098–1105, 2012.

[6] Wulfram Gerstner and Werner M Kistler. *Spiking neuron models: Single neurons, populations, plasticity*. Cambridge university press, 2002.

[7] Magnus Rudolph Hestenes and Eduard Stiefel. Methods of conjugate gradients for solving linear systems, 1952.

[8] http://apt.cs.man.ac.uk/projects/SpiNNaker/. Official website of the spinnaker project.

[9] MM Khan, DR Lester, Luis A Plana, A Rast, X Jin, E Painkras, and Stephen B Furber. Spinnaker: mapping neural networks onto a massively-parallel chip multiprocessor. In *Neural Networks, 2008. IJCNN 2008.(IEEE World Congress on Computational Intelligence). IEEE International Joint Conference on*, pages 2849–2856. IEEE, 2008.

[10] Muthucumaru Maheswaran, Kevin J Webb, and Howard Jay Siegel. Mcgs: a modified conjugate gradient squared algorithm for nonsymmetric linear systems. *The Journal of Supercomputing*, 14(3):257–280, 1999.

[11] Javier Navaridas, Mikel Luján, Jose Miguel-Alonso, Luis A Plana, and Steve Furber. Understanding the interconnection network of spinnaker. In *Proceedings of the 23rd international conference on Supercomputing*, pages 286–295. ACM, 2009.

[12] Dianne P O'Leary. Parallel implementation of the block conjugate gradient algorithm. *Parallel Computing*, 5(1):127–139, 1987.

[13] L. Olson. Lecture 21 better iterative methods; google and markov chains, 2009.

[14] William H Press, Saul A Teukolsky, William T Vetterling, and Brian P Flannery. *Numerical recipes 3rd edition: The art of scientific computing*. Cambridge university press, 2007.

[15] Thomas Sharp, Francesco Galluppi, Alexander Rast, and Steve Furber. Power-efficient simulation of detailed cortical microcircuits on spinnaker. *Journal of Neuroscience Methods*, 2012.

[16] Thomas Sharp, Luis A Plana, Francesco Galluppi, and Steve Furber. Event-driven simulation of arbitrary spiking neural networks on spinnaker. In *Neural Information Processing*, pages 424–430. Springer, 2011.

[17] Jonathan Richard Shewchuk. An introduction to the conjugate gradient method without the agonizing pain, 1994.

[18] Marcin Wozniak, Tomasz Olas, and Roman Wyrzykowski. Parallel implementation of conjugate gradient method on graphics processors. In *Parallel Processing and Applied Mathematics*, pages 125–135. Springer, 2010.

[19] Laurence Tianruo Yang and Richard P Brent. The improved conjugate gradient squared (icgs) method on parallel distributed memory architectures. In *Parallel Processing Workshops, 2001. International Conference on*, pages 161–165. IEEE, 2001.

[11] [15] [3] [16] [6] [14] [17] [13] [12] [2] [10] [19] [1] [5] [18] [7] [**?**] [4] [8] [9]