1 Introduction

SpiNNaker is an architecture inspired by the biology of the human brain. Its optimal configuration can use over a million cores[2], which can be used to simulate neurons in the human brain. It is a System-on-Chip machine that provides massive parallelism to the user, which escapes most norms of parallel computing, without having its main pitfalls such as race conditions, deadlocks, mutual exclusion etc. [3]. The cores in the system communicate with each other with spikes(packets), in the same fashion that neurons communicate. The system is event-driven, which means that computations can occur in case of certain events.
The Conjugate Gradient Method is an iterative way of solving symmetric and positive definite systems of linear equations. Its iterative property makes it easy to even solve sparse systems, that other methods might have problems solving.
The aim of this report is to explain in more detail the aforementioned concepts, as well as give an account of the background research done to launch the project. It will also try to sketch out some aspects of the final design of the system, as well as the reasons why certain choices in the design were made. Finally, it will show the tasks completed so far, along with a plan for the remaining work.

2.0 Project Description

2.1 Neuron functionality

In order to explain the SpiNNaker architecture some basic functionality of the human neuron should be presented. A standard human neuron can be divided into three parts, the dendrites, the soma and the axon. Each neuron can be connected to a number of different other neurons, throughout the entire human body. The dendrites are the parts of the neuron, which receive information from other neurons. The axon is the part that delivers information to other neurons and the soma is the processing unit of the neuron.[4]
Each time a neuron wants to send information it does so by using spikes through its axon and the targeted neuron's dendrite. The part of the connection where the dendrite connects to the axon is called synapse. Figure 1 demonstrates a typical neuron.

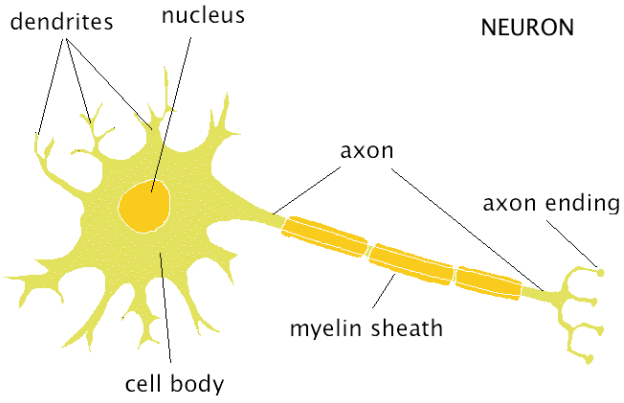

Figure 1: A neuron

2.2 SpiNNaker Architecture

As mentioned before SpiNNaker is a massive parallel computer able to host over a million cores. At the heart of the machine the SpiNNaker chip can be found .

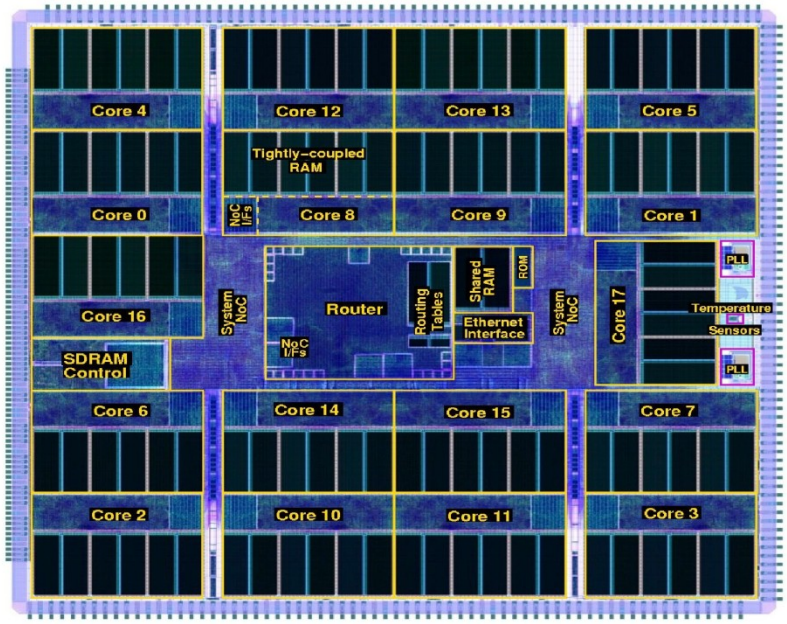

Figure 2: The SpiNNaker chip[5]

As can be seen in Figure 2, the main components of a SpiNNaker chip are 18 ARM cores, a router and an SDRAM. At start-up, after each core finishes testing its functionality, a Monitor core is selected, with a responsibility of monitoring the other cores. Another core is used as a backup core and all the other are used for computational purposes[6]. The official name for a monitor core is “scamp” and for an application core “sark”. There is an additional core in the system named “ybug”, which handles I/O interactions, but also communicates with monitor cores[5].

A core in the chip uses spikes to communicate with other cores. Each spike is relatively small (70 bits at most), thus following the example of neurons[5], which use low energy spikes to propagate their messages. Each core has immediate connectivity to 6 other cores, in the west, east, north, south, northeast and southwest[2]. However with some configuration each core can be made to communicate with all other cores in the system. To make this communication possible, each chip incorporates a router that distributes these messages to other cores. Its main purpose is to direct the messages sent by its cores to other cores, but also receive messages and send them to the corresponding cores of the chip. To achieve that the router supports multi cast communication, which helps the network run faster by reducing the number of packets sent[2]. The multi cast packets have unusual configuration, since no information about the destination is stored in the packet, but rather only information from the source(id value etc.)is stored. If for some reason a packet fails to send, or fails to arrive, the system has the ability to replicate this message and resend it.

As mentioned before, the programming model of a SpiNNaker machine is event-driven. That means that a computation can only happen following an event. That event can be a packet having sent or arrived, a DMA access, or a timer event. There is no way for an application to control the flow of a program. The programmer can only design event handlers that will be executed when an event occurs and assign them a certain priority[3]. To execute these callbacks, as they are named, a scheduler and a dispatcher is used. The scheduler puts these callbacks in a queue according to their priority and upon the occurrence of an event the dispatcher executes them according to these priorities. There can also be special callbacks that are non-queueable and will not be placed in the queue at all. Rather they will precede any other action of either the scheduler or the dispatcher.

2.3 Conjugate Gradient Method

As explained before, the Conjugate Gradient Method(CGM) is an iterative way of solving systems of linear equations, whose matrix is symmetric and positive definite. The equation it needs to solve is of the form A*x=b. Its pseudo code can be seen below.[8]

r_0=b-Ax_0
p_0=r_0
f=0
for f to 1000000 **do**
  $alpha_f = \frac{r_f^T * r_f}{p_f^T * A * p_f}$
  x_{f+1}=x_f+alpha$_f$*p_f
  r_{f+1}=r_f-alpha$_f$*p_f*A
  if r_{f+1}^T*r_{f+1} is small enough **then**
    break
  end if
  $beta_f = \frac{r_{f+1}^T * r_{f+1}}{r_f^T * r_f}$
  r_{f+1}=r_{f+1}+beta$_f$*p_f
  f=f+1
end for

Pseudocode 1: Conjugate Gradient Method

Based on Pseudocode 1, the CMG needs storing of many variables, along with many matrix-vector multiplications. That makes the algorithm by default resource and computationally heavy. The most expensive calculation of the algorithm seems to be the vector-matrix multiplication, which has complexity O(N^2).

2.4 Final Description and Goals

To sum up, based on the description of the constitutes of the problem, the problem itself has as a goal not only to solve the CGM in a SpiNNaker machine successfully, but also to reduce the time needed to solve the problem by doing computations concurrently. A further goal would be to reduce the complexity of the algorithm and its time expensive operations.

3 Background Research and Literature

To start off the project some background reading about the main aspects of it was done in the beginning of the year. The first pieces of information about the project came from studying the contents of the website of the SpiNNaker project[5] and thus learning the basics of the SpiNNaker architecture.
After studying the website's contents some more general papers were studied in order to establish better foundations for the project[1][6][13]. To acquire further understanding of architecture of the SpiNNaker chip, papers which describe in more detail certain aspects of the project were studied. These would describe the connection and communication between cores and different SpiNNaker chips[2] and others would describe the process of creating event handlers[3].
Furthermore, some papers were studied to view the uses of the SpiNNaker chip so far. One example would be using a set of SpiNNaker chips to simulate thousands of spiking neurons by using over four

million synapses[11]. Another one would be using the SpiNNaker chips to simulate the neurons of a retina sensor with the result of making it possible for a robot to calculate the place it is in and navigate[12].
Finally some research was done in order to understand the CGM. This was accomplished by studying several sources, but the main one being a book[8].

4 Final Design

In order to reduce the time complexity of the algorithm the design of the program needs to be done correctly. To accomplish that, every element of the matrix A and the vectors b and x will be placed in a node, each of which will be assigned to a core. That way each time nodes send information to other nodes, the computation can be accomplished atomically for that node only. Further nodes will be created to help with the computation. Figure 3 explains this property more explicitly.

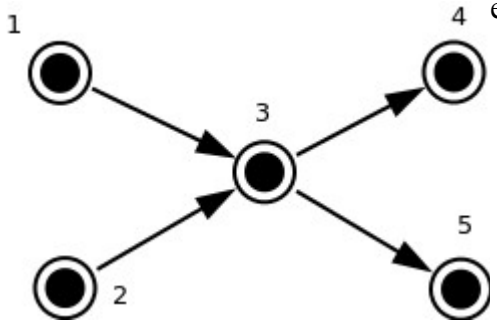

Figure 3: Basic communication

Figure 3 shows the basic communication of the nodes in the program. Each node(3), will receive information by one or more nodes(1,2) and after processing this information, it will send it to one or more nodes(4,5).

The above functionality is displayed in the context of the CGM problem in Figure 4. The first multiplication of the CGM will be shown(A*x), while choosing A to be a 3x3 matrix and x a 3-element vector for simplicity purposes.

As can be seen in Figure 4, the values of x are flooded to the nodes of the A matrix. Each value of x goes to the corresponding

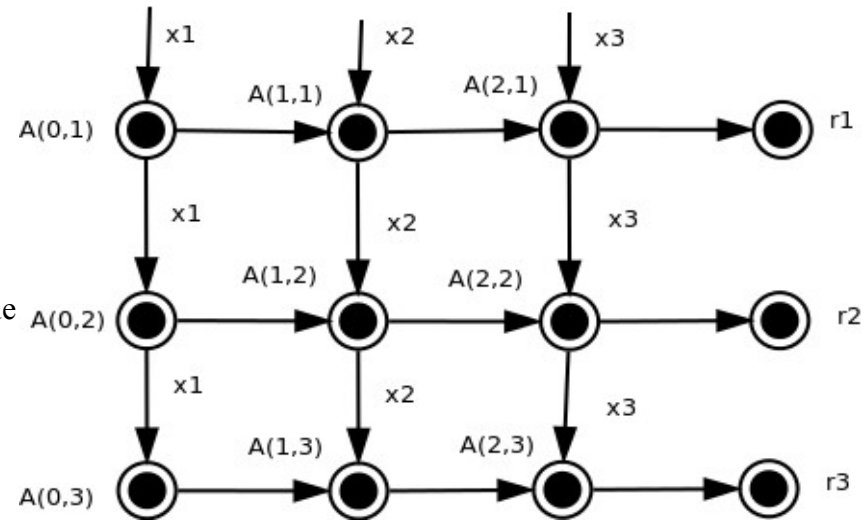

*Figure 4: Matrix Multiplication(A*x)*

node of A(the one needed for multiplication). So for example, x1 is sent to all the A nodes of the first column, x2 is sent to the A nodes of the second column and x3 to all the nodes of the third column. Once the computation is completed in each node, the result is sent to the corresponding r node, where all the values will be added to produce the final r values, which will correspond to an element of the multiplication result. Figure 4 illustrates that each value is transferred to the node to the right, but that is done only for clarity purposes. For each calculation, there exists an event handler that will be triggered as soon as the corresponding event happens. So for example, when x1 arrives to A(0,1) an event handler will be triggered that multiplies these two values. In the same fashion, when all the multiplication results arrive to an r node, a handler will be triggered to add the values.

Following the example of Figure 4, the node format for A,x and r nodes will be shown.

X node

ID value

A node

ID value x_ID last_result

r node

ID value A_ID*

All nodes contain an ID, so as to identify the node, and a value to be updated during the program's run time. In contrast to x-nodes, A-nodes also contain an x_ID and a last_result field. The x_ID is there, so that when X nodes multi cast their values, each A node picks up the correct X value. During the creation of the nodes, the corresponding ID of x should be placed to the X_ID of each A node. The last_result field exists in order to save the result of the multiplication and send it to r nodes. R nodes are more or less the same with A nodes with the difference being that instead of having one ID, it stores an array of A nodes IDs. The reasoning behind this design is that r nodes are the place where the addition for the completion of the multiplication is done, so all the nodes of one row of A, will send their last_result fields to the corresponding r nodes.

This format has been chosen to agree with the architecture of SpiNNaker, which states that packets only contain information about their source.

The format of the remaining nodes is more or less the same, with the exception that some nodes will have more fields in order to intercept information sent from more node.

For the design of the project, a flow diagram will be presented, which will display the sequence of the program.

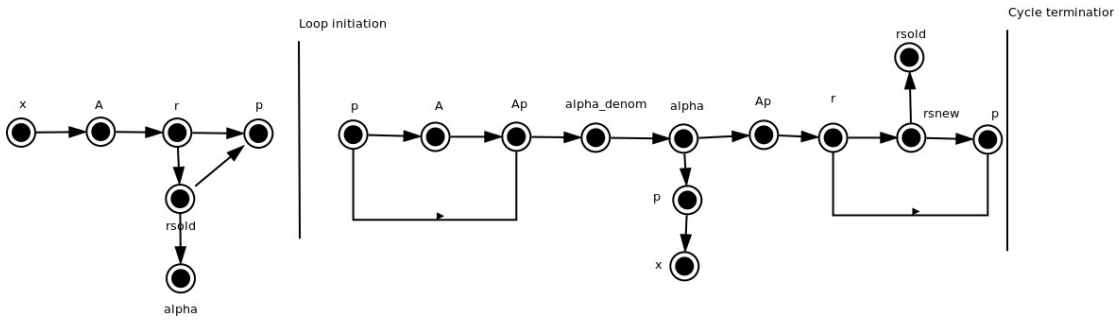

Figure 5: Flow diagram

Figure 5 displays the flow of the diagram. It has been designed according to Pseudocode 1, with three differences. However it should firstly be noted that each node in the figure represents a set of nodes i.e the x node represents all the nodes of x., with the exception of some nodes where only one node is represented. This is the case with rsold, alpha and rsnew. Moving to the differences between the figure and the pseudo code, the Ap nodes have been created to bring the functionality closer to the specifications of SpiNNaker. Also beta as a variable does not exist, but it is calculated in the last node before the cycle termination by supplying p with rsold and rsnew. Lastly, it is noticeable that a b node for the elements of vector b is missing in the diagram. The reason for that is that b is only needed in the beginning of the algorithm to calculate r. To reduce redundancy the elements of the b vector are inserted as initial values in the r nodes. Another milestone for Figure 5 is the loop initiation and the cycle termination. The loop initiation is when the loop starts and the cycle termination is when a cycle ends. At that point the sequence of the packet sending will again start from the loop initiation line. The check for loop termination will be done from an event handler at the r-node, three nodes before the cycle termination line. The communication of the nodes is handled in a similar way as in Figure 4.

Figure 4 displays the progression of the first 3 nodes in Figure 5. X nodes send to A nodes and they send their results to r. Sometimes a node sends information to more than one node as stated in Figure 3. The reason for that is that this information needs to be processed somehow before sent to particular nodes, but also needs to be sent somewhere else without being processed. That is the case for when the r nodes before the loop initiation send their values to both p nodes and to rsold. At other times, the process would be faster if the information would be sent to many nodes simultaneously. For example, when alpha sends its value to the nodes of Ap and p, the process is accelerated, since the new values of x are calculated at the same time as the new values of r.

Finally, it is worthy to mention that even though the design discussed above has been presented as the final design for the project, it is subject for further iteration and improvement in the future.

5.0 Justification for the Approach

The reasons for picking parts of the design were diverse. Each decision had as its main focus to adapt the design into the SpiNNaker architecture. The justification for these decisions is made more specific in this section.

Having one node for each element was inspired by the design of a sample program written for the SpiNNaker chip supplied from this project's supervisor[10].

Picking the design seen in Figure 3 was done due to the nature of the problem. For example, in a matrix-vector multiplication the elements of the vector were needed in all the elements of the

corresponding columns of the matrix. So it was apparent that nodes would have to send input to many other nodes.

The design on Figure 4 was inspired by the design of the matrix multiplication seen in the Interim report for solving the LU Decomposition algorithm in a SpiNNaker machine[10].

Finally, the design in Figure 5 was inspired by trying to solve the CGM using the SpiNNaker architecture. As mentioned before it is subject to improvement, but the main idea behind it was to introduce some kind of flow to the algorithm, as well as having information in the nodes, without duplication and high complexity.

6 Completed Work

The completed work so far can be seen in the form of a Gantt chart in Appendix 10.1.

The introduction to the project has already been completed. The SpiNNaker and the Conjugate Gradient Method have been understood in a fairly good level. Furthermore, the CGM was coded in Java to enhance the understanding of its complexity.

The task that is underway at the moment is the design of the project. Its completion stands at 60% because in addition to the design that has been demonstrated above, the implementation of the nodes has also been added there. The reason for that is that the implementation of the nodes has helped constructing a more realistic design not just for the flow diagram in Figure 5, but also for the nodes design. So far the implementation of A,x,r, and p nodes have been completed in C++.

7 Plan for remaining work

The remaining work needed to be done can also be displayed in the form of a Gantt chart, which is demonstrated in Appendix 10.2.

The main work to be done at this stage is to actually implement the algorithm for a SpiNNaker machine. Roughly a month has been allocated for each main task, so as to allow time for any design changes that occur. The tasks that fall into this category are implementation and testing of the MCTables and the Cores classes and the implementation and testing of the event handlers. Some time has been allocated to merge the aforementioned components and test them, both in a SpiNNaker simulator, but also in an actual SpiNNaker machine. Finally, enough time has been allocated for report writing and after the submission of the report some time for preparation for the viva

8 Conclusion

In conclusion, this project needed quite a lot of quality reading in order to understand concisely the architecture of SpiNNaker, as well as the project concept as a whole.The hardware and ways to produce software is very diverse in comparison to usual computer systems, which is probably the reason for a lenghty introduction to the system.

So far the design of the system for the project has been covered thouroughly. Progress thus far has been satisfactory and the deadlines introduced in the Gantt charts have been met.In the remaining time left until the final report deadline, the focus for the project will be implementing the algorithm and writing the report.