Genet Program Evolvable Mach (2014) 15:245–274
DOI 10.1007/s10710-014-9217-1

Hardware architecture of the Protein Processing Associative Memory and the effects of dimensionality and quantisation on performance

Omer Qadir · Alex Lenz · Gianluca Tempesti · Jon Timmis · Tony Pipe · Andy Tyrrell

Received: 11 January 2014 / Revised: 21 February 2014 / Published online: 2 April 2014

Abstract The Protein Processor Associative Memory (PPAM) is a novel hardware architecture for a distributed, decentralised, robust and scalable, bidirectional, hetero-associative memory, that can adapt online to changes in the training data. The PPAM uses the location of data in memory to identify relationships and is therefore fundamentally different from traditional processing methods that tend to use arithmetic operations to perform computation. This paper presents the hardware architecture and details a sample digital logic implementation with an analysis of the implications of using existing techniques for such hardware architectures. It also presents the results of implementing the PPAM for a robotic application that involves learning the forward and inverse kinematics. The results show that, contrary to most other techniques, the PPAM benefits from higher dimensionality of data, and that quantisation intervals are crucial to the performance of the PPAM.

Electronic supplementary material The online version of this article (doi:10.1007/s10710-014-9217-1) contains supplementary material, which is available to authorized users.

O. Qadir (✉)
Nordic Semiconductor, Otto Neilsens v12, Trondheim, Norway
e-mail: oq500@ohm.york.ac.uk

A. Lenz · T. Pipe
University of West of England, Bristol, UK

G. Tempesti · J. Timmis · A. Tyrrell
University of York, York, UK
e-mail: gt512@york.ac.uk

J. Timmis
e-mail: jt517@ohm.york.ac.uk

A. Tyrrell
e-mail: amt@ohm.york.ac.uk

Keywords Protein processing · PPAM · FPGA · Associative memory · BERT2 · Inverse kinematics · Dimensionality · Quantisation · Non-standard computation

1 Introduction

The work presented here stems from research into parallel architectures targeted towards problems in the domain of Artificial Intelligence (AI) and is an attempt to explore possible alternate non-standard architectures for computation. Although, many architectures exist that attempt solutions for problems in this domain (for example [1, 2, 14, 25, 27]), most are based on Arithmetic and Logic Units (ALUs) and often are simply traditional processors in parallel, thereby suffering from the well known Von Neumann bottleneck [3]. Some recent attempts have been made with neuromorphic hardware using mixed signal design, however, these tend to treat the analogue portion as an add-on to the digital *master* [7], while others are attempts to design systems for mimicking biological experiments rather than for AI [16]. Systemic Computation architectures [22] are yet another alternative, designed using standard digital logic techniques (FSM, instruction decoders). Like other traditional processor based architectures, SC hardware also uses mathematical operations as the basis of computation.

An AI application may be defined as one that tries to achieve behaviour that will be displayed by intelligent beings like humans [29]. Some of the desired features of such applications are adaptability (in the presence of new stimuli), robustness (in the presence of errors), real-time results and scalability. In any case, most will agree that the hallmark of an AI application is complexity. The traditional computation paradigm (Von Neumann or otherwise) is to implement simple (mathematical) operations in ALUs, which are accessed by instructions to a control unit. This is further accessed through (potentially multiple) software/firmware wrapper layers, which may be an Operating System or a Virtual Machine or both. Therefore, implementing complex AI applications on traditional machines not only involves mapping an individual complex operation to multiple simple mathematical operations, but also, each operation passes through multiple software layers to reach a hardware unit (ALU) that is accessed through yet another hardware wrapper (the control unit). Traversing all these layers is a severe handicap for an application that intends to operate in real-time. In Qadir et al. [17] we proposed a novel architecture called the Protein Processor Associative Memory (PPAM), that used hetero-associative recall to achieve AI, and explored the effect of moving computation into memory. Various facets of the PPAM have been explored in the past, including comparison with more traditional computation methods [18], hardware implementability [19], comparison with other techniques on a robotic task [20], and fault-tolerance abilities [21].

In this current paper, we present the complete details of the hardware architecture. We provide the reasoning for the various design decisions including why analogue communication is preferred over both serial or parallel digital communication. We also present results from new experiments performed on a real robot to learn forward and inverse kinematics. The rest of this paper is structured as follows. Section 2 presents the motivation and introduces the principles of protein processing. Section 3 describes the hardware architecture in detail, while Sect. 4 describes the new experiments conducted and the results obtained using the Bristol Elumotion Robot Torso 2 (BERT2) to learn the forward and inverse kinematics. Section 5 discusses the results and presents a comparison with the state of the art. Section 6 summarises, concludes and discusses future directions.

2 Protein Processing Associative Memory

Johnson [12] define six properties of classical computation. It follows therefore, that a solution that challenges even one of these six would be non-standard. However, in order to perturb the existing solution (classical computation) by a large amount, it might be preferable to challenge as many of the properties as possible. To summarize, the main focus in this work was to *challenge the use of mathematical operators as the fabric of computation and to test if this brings any advantage for AI applications*.

2.1 Principles of protein processing

Let V1 and V2 be two variables that are different (in content and also size). Then auto-associative memories store V1 and can retrieve V1 upon presentation of a sub-portion of V1. Hetero-associative memories, store (V1, V2) and can recall V2, upon presentation of a sub-part of V1. Bidirectional hetero-associative memories can perform hetero-associative recall in both directions—therefore they can recall V2 using a part of V1 and recall V1 using a part of V2. As shown by Coppin [5, Chapter 11], the human brain implements bidirectional, hetero-associative memory. The PPAM is designed to be such a memory.

The PPAM takes inspiration from the biological Neural Network and also the biological Genetic Regulatory Network (GRN). It is composed of a large number of simple nodes, which use the Hebbian principle [9] to determine if two pieces of data are related and a GRN inspired model decides how (and in which node) to store the data so that it can be recalled quickly and correctly, even in the presence of faults. A detailed description of the PPAM may be found in Qadir et al. [18, 19] and only a brief summary is reproduced below, followed by a worked example.

The PPAM associates 2 variables ($V1$ and $V2$) with any number of dimensions in each variable: $V1 = (A_1, A_2, A_3, \ldots A_d)$ and $V2 = (B_1, B_2, B_3, \ldots B_D)$ where $D \neq d$. The PPAM can then be defined as:

$$\begin{aligned}
PPAM &::= (V1Nodes, V2Nodes)\\
V1Nodes &::= (ConNodesV1, DimNodesV1)\\
V2Nodes &::= (ConNodesV2, DimNodesV2)\\
ConNodesV1 &::= (P_{v1}1, P_{v1}2, P_{v1}3, \ldots P_{v1}C)\\
DimNodesV1 &::= (P_{v1}1, P_{v1}2, P_{v1}3, \ldots P_{v1}D)\\
ConNodesV2 &::= (P_{v2}1, P_{v2}2, P_{v2}3, \ldots P_{v2}c)\\
DimNodesV2 &::= (P_{v2}1, P_{v2}2, P_{v2}3, \ldots P_{v2}d)
\end{aligned}$$

where C and c are the number of “Conflict-Resolving nodes” in variable 1 and variable 2 respectively, and P is a “Protein Processor” whose operation is summarized using pseudo-code 1. Note, that there are D nodes in *DimNodesV*1, which is the number of dimensions for V2.

PseudoCode 1: Protein Processor (from perspective of V1 nodes)

```
Input: inStatePP::= outStatePP from V2 neighbouring node
Input: inVarData::= input data ($A_x$ ) of V1 that is to be associated or to be used
       for associative recall
Output: outStatePP::= state of this node; used during associative recall
Event eventEnv::= input change from Environment (change in inVarData)
Event eventOVN::= input change from Other-Variable node (change in inStatePP)

eventEnv → actionUpdateLUT // event eventEnv triggers actionUpdateLUT
eventOVN → actionUpdateDout // event eventOVN triggers actionUpdateDout

Action actionUpdateLUT
    if (LUT contains inStatePP) or (LUT location "inVarData" is not empty) then
        pass data amongst conflict resolving nodes to determine how many nodes
        have the same conflict
        if (current node is among the first half of the nodes with the same conflict)
        then
            remove old conflict generating data from LUT
            write value "inStatePP" to address "inVarData" in the LUT
        end
    end
    else
        write value "inStatePP" to address "inVarData" in the LUT
    end
EndAction

Action actionUpdateDout
    if eventEnv then
        outStatePP ← inVarData
    end
    else
        if (LUT contains inStatePP) then
            outStatePP← address of LUT location storing inStatePP
        end
        else
            outStatePP← 0
        end
    end
EndAction
```

2.2 Worked example

A sample dataset is used here to illustrate the operation of the PPAM. Details about the dataset can be found in Online Resource 1, but a subset of the data is summarised in Table 1. $V1 = (X, Y), V2 = (R, \theta)$ and $D = d = 2$. For simplicity, assume $C = c = 4$. Therefore, there are $2C$ nodes (8 nodes) in each of X, Y, R and θ, resulting in a total of 32 nodes in the PPAM. The PPAM structure is shown in Fig. 1. Each node in Fig. 1a (e.g. X) is expanded into multiple other nodes as shown

Table 1 Cartesian polar dataset

Cartesian coordinates (V1)		Polar coordinates (V2)	
X-code	Y-code	R-code	θ-code
0x1	0x5	0x4	0xA
0x1	0x6	0x4	0x9
0x1	0x7	0x3	0x9

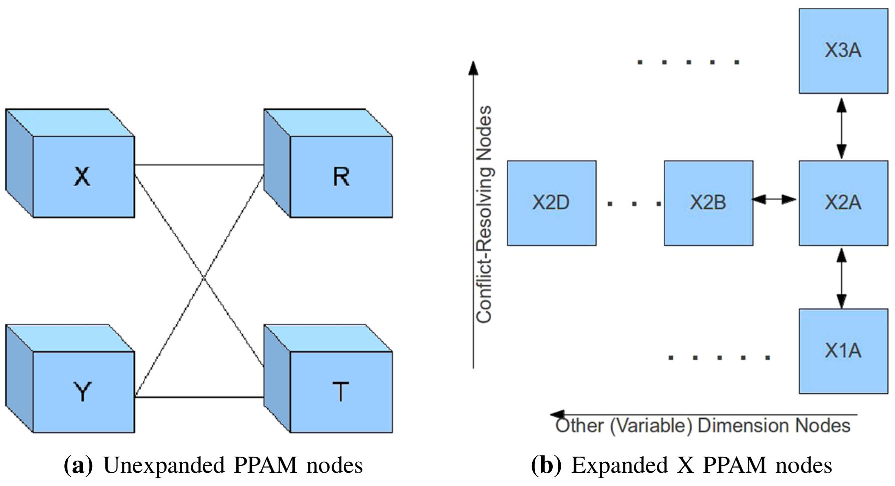
X
R
Y
T
(a) Unexpanded PPAM nodes
Conflict-Resolving Nodes
X3A
X2D
X2B
X2A
X1A
Other (Variable) Dimension Nodes
(b) Expanded X PPAM nodes

Fig. 1 Abstract view of the PPAM nodes (T represents θ). **a** Unexpanded PPAM nodes, **b** expanded X PPAM nodes

in Fig. 1b. The following works through the dataset for the *Y* nodes which should be sufficient to illustrate the operation of the PPAM. It uses the convention in Fig. 1b to identify nodes—i.e. columns (for dimensions) are identified using letters *A* and *B*, while rows (for Conflict-Resolving nodes) are identified using numbers $1, 2, \ldots C$

Let all *Y* nodes in column *A* (*YiA* nodes) have *R* nodes as neighbour-nodes, and let all column *B* nodes (*YiB* nodes) have θ nodes as neighbour-nodes. Then, when the first tuple is applied to *YiA* nodes inStatePP = 0x4 and inVarData = 0x5 and so they update their memory as shown in time step 1 of Table 2.

Similarly, *YiB* nodes observe inStatePP = 0xA and inVarData = 0x5 and so update their memory as shown in time step 1 of Table 3. Upon presentation of the second tuple of the dataset (in Table 1), *YiA* nodes fire with 0x6 and observe 0x4 from their neighbour-nodes. They detect a conflict as inStatePP = 0x4 again but inVarData $\neq$ 0x5. To resolve this, half of the nodes with conflict (which in this case is all of the *YiA* nodes) autonomously update to the new tuple, while the other half autonomously retain the old value. At the same time, *YiB* nodes fire with 0x6 and observe 0x9 from their neighbour-nodes, which is not a conflict and therefore, none of the *YiB* nodes *object* to updating to the new tuple. However, only those *YiB* nodes

Table 2 Memory contents in *YiA* nodes during the worked example

Time-step	Mem-address	Y1A	Y2A	Y3A	Y4A
1	0x5	0x4	0x4	0x4	0x4
	0x6				
	0x7				
	0x8				
2	0x5			0x4	0x4
	0x6	0x4	0x4		
	0x7				
	0x8				

Table 3 Memory contents in *YiB* nodes during the worked example

Time-step	Mem-address	Y1B	Y2B	Y3B	Y4B
1	0x5	0xA	0xA	0xA	0xA
	0x6				
	0x7				
	0x8				
2	0x5			0xA	0xA
	0x6	0x9	0x9		
	0x7				
	0x8				

update to the new tuple whose corresponding *YiA* nodes indicated that they can update to the new tuple as well. Thus, the *Yij* nodes update as shown in time step 2 of Tables 2 and 3.

A similar case (with two differences) occurs when the next tuple from the dataset is presented. Firstly, *YiB* nodes detect a conflict instead of *YiA* nodes. Secondly, only *Y*1*B* and *Y*2*B* nodes have a conflict and not *Y*3*B* and *Y*4*B*. This is because these nodes have already *differentiated*. In order to resolve the conflict, half of the nodes with the conflict update to the new conflict-generating tuple. *Y*3*B* and *Y*4*B* do not have a conflict, so they don't *object* to updating to the new tuple. All of the *YiA* nodes indicate that they are willing to update to the new tuple. However, only those *YiA* nodes update to the new tuple whose corresponding *YiB* nodes indicated that they can update to the new tuple as well. Thus, the memory values in *Yij* nodes is as shown in time step 3 of Tables 2 and 3. To see updates for further time steps, refer to Online Resource 1.

After 2 time steps, if the V1 input is removed, and a V2 input of $R \rightarrow$ 0x4 and $\theta \rightarrow$ 0xA (first tuple in the dataset) is provided, the PPAM should recall 0x5 from the *Y* nodes. Nodes *Y*1*A* and *Y*2*A* fire with 0x6 while nodes *Y*3*A* and *Y*4*A* fire with 0x5. Nodes *Y*1*B* and *Y*2*B* do not generate any output since these nodes do not have 0xA in their memory. Conversely, nodes *Y*3*B* and *Y*4*B* fire with 0x5, thus reinforcing the outputs of *Y*3*A* and *Y*4*B*. In this way, the correct value of 0x5 is recalled from the *Y* nodes in the PPAM.

3 Hardware architecture

This section details the hardware architecture. A sample digital logic implementation highlights the implications of using existing techniques for hardware design. First however, it presents some trade-offs in order to justify some of the major design decisions.

3.1 Trade-offs in synchronisation

The components of most modern machines and systems operate in synchronization with one (or more) central clocks. The converse are asynchronous circuits, which present many theoretical advantages, but suffer from many practical hurdles. A full review of this is outside the scope of this paper, but a brief summary is presented in Online Resource 3.

The PPAM combines *Delay insensitive* and *Burst mode* asynchronous design techniques [8] with a synchronous-write RAM. Thus memory recall is an asynchronous operation, while learning is synchronous to avoid the requirement of asynchronous RAMs in hardware. The use of these asynchronous methods means that the circuit operates at less than the theoretical maximum speed. Note however, that the PPAM is envisioned as a system that operates on real-time inputs obtained from the real world. Since sensory data from the real world is typically slow—in the range of KHz, rather than MHz—this reduced speed does not reduce the performance of the system. On the contrary, it means that the system can meet the data rates while at the same time not waste power on a clock that is operating faster than the data rates require. Section 3.3 explains the theoretical design and presents circuit diagrams and schematics, while Sect. 3.4 presents an adapted implementation that only uses digital logic elements available in existing FPGAs. As an aside, note that Sect. 3.4 also shows that using a small number of user-defined constraints specifying false-paths and multi-cycle paths, this asynchronous logic can be synthesised using a standard commercial synthesis tool (Xilinx ISE) which is not specifically designed for implementing asynchronous logic. This opens an interesting avenue for further research exploring synchronous design tools for asynchronous design.

3.2 Trade-offs in parameters for parallel systems

The PPAM is a parallel architecture since it is composed of nodes operating in parallel. However, unlike other parallel architectures, nodes are not based on *traditional processors*, where traditional processors are defined as containing:

- an ALU that calculates results,
- a control unit that directs execution based on a program that is fetched from memory and
- a memory that stores results (data) and instructions.

Most existing parallel architectures are simply networks of such traditional processors. The usual motivation for parallel systems is to increase processing

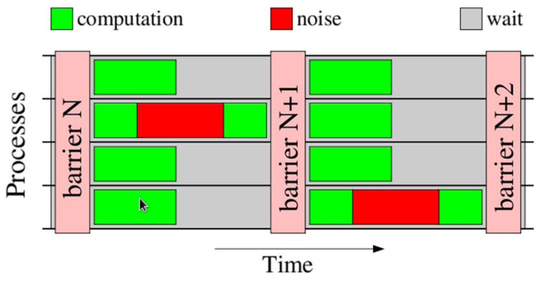

Fig. 2 Effect of Operating System noise on parallel arrays. Even when tasks are spawned at the same time (Barrier N), noise in one process can force all other processes to wait (Barrier N + 1) [28]

power and decrease processing time. In the case of architectures for AI, quite often the objective is to achieve real-time processing.

The issue of accessing the hardware through potentially multiple layers of hardware abstraction was mentioned earlier (in Sect. 1). This is the overhead of using a general purpose processor. This overhead is further enhanced in parallel arrays and is known as the operating system (OS) *noise* or *jitter* (Fig. 2). Note that the nodes in the clusters would typically run a complete operating system, like a flavour of Windows, or Linux. A detailed review of parallel systems is outside the scope of this paper, however a brief overview is presented in the Online Resource 4.

The PPAM does not execute a program or need a configuration bit-stream. This allows it to circumvent the parallel programming issues faced by parallel architectures based on traditional processors. However, it must be noted that the PPAM compromises by not being Turing Complete. Nonetheless, assuming a problem *can* be solved by an associative memory, the PPAM should be applicable. As will be seen from Sect. 3.3 some parameters of the circuit are dependent upon the characteristics of the dataset being associated. Therefore, one fundamental assumption for flexibility of the PPAM is that it is implemented either on:

- a reconfigurable fabric like an FPGA: This is less desirable because of reasons which will be explained in Sect. 3.3
- or an ASIC which includes certain dynamic routing capabilities—either through packet switching routers, virtual connections, or physical multiplexers to dynamically redirect packets. It must be noted however, that this dynamic routing capability is not required in real-time since it will happen only once at the beginning of the association when the PPAM is informed of the characteristics of the dataset.

Due to cost and time considerations, the ASIC option was not actually implemented as part of this research, but Sect. 3.4 does present details of implementing on a reconfigurable fabric.

3.3 Circuit design and schematic

The PPAM is a distributed system where each node makes its own decision about updating its memory contents. To reduce hardware connections, DimNodesV1 (and DimNodesV2) nodes have pipelined sequential connections, while ConNodesV1 (and ConNodesV2) nodes communicate in parallel. Therefore, V1 nodes update in $D + 1$ steps, where D is the number of dimensions in V2 (and vice versa). This means the PPAM needs $D + 1$ clock edges to store each tuple of data. In each step, the Conflict-Resolving nodes transmit an address and data to each other informing each other about their own memory contents. They use this information to generate a threshold which is later used to determine if the current node should update its value or maintain the old one. This process will hereafter be called "threshold and compare". The specifics of its implementation are crucial to the performance of the PPAM. This is discussed in more detail in Sects. 3.3.1 and 3.3.2.

If a node observes a conflict and decides to update, two (or more) tuples may have to be updated, as one tuple has to be removed while another has to be entered. In the first D cycles, nodes transmit information about tuples that will need to be removed (if any). In the last cycle, nodes transmit the new tuple information. In each cycle, nodes use threshold and compare operations to make their decisions.

Figure 3 illustrates the overall schematic for a node in the PPAM. Events eventEnv and eventOVN are generated by using standard *pull-down* resistors for inputs (sample circuit in Figure R2.4 in Online Resource 2).

The combinatorial cloud labelled "match all locations" uses N XNOR gates to compare all the locations of the neighbour-node LUT with the value from the other variable; where N is the size of the memory in bits. The circuit for this block is shown in Fig. 4. Conflicts are detected using a small set of gates as shown in the bottom right section of the schematic in Figure R2.1. The conflict resolving circuit is based on the "threshold and compare" logic. Two methods for implementing the "threshold and compare" circuit are considered. The first is an analogue implementation which is deemed better suited for the operation but requires mixed signal design techniques. The second is entirely digital logic which has the advantage of being easier to implement with greater tool-chains support. The advantages and disadvantages of both are considered, however only the digital logic design was actually implemented, due to the limitations of current FPGA architectures (which only allow digital logic design).

3.3.1 Analogue threshold and compare

The threshold and compare operates as follows. A Protein Processor node receives address and data values from its Conflict-Resolving neighbours as an analogue voltage. The node compares these value with its own by *subtracting* the 2 voltages. A regenerative amplifier is used at the output so that if the result is small, it tends to zero. The output is then *inverted*. This is repeated for all Conflict-Resolving neighbours and the results are *added* together. *Half* of the sum is *stored*. At the clock edge, a node decides to update, if the current result of the summed values is *greater than* the stored threshold. Note that it is essential that the summation circuit should respond faster than the clock skew between Conflict-Resolving nodes.

The italicised words represent the operations that need to be performed in the analogue domain. All of these operations have standard, well-studied analogue circuits, and so these aren't presented within this paper. The interested reader may refer to Online Resource 2 for sample circuits.

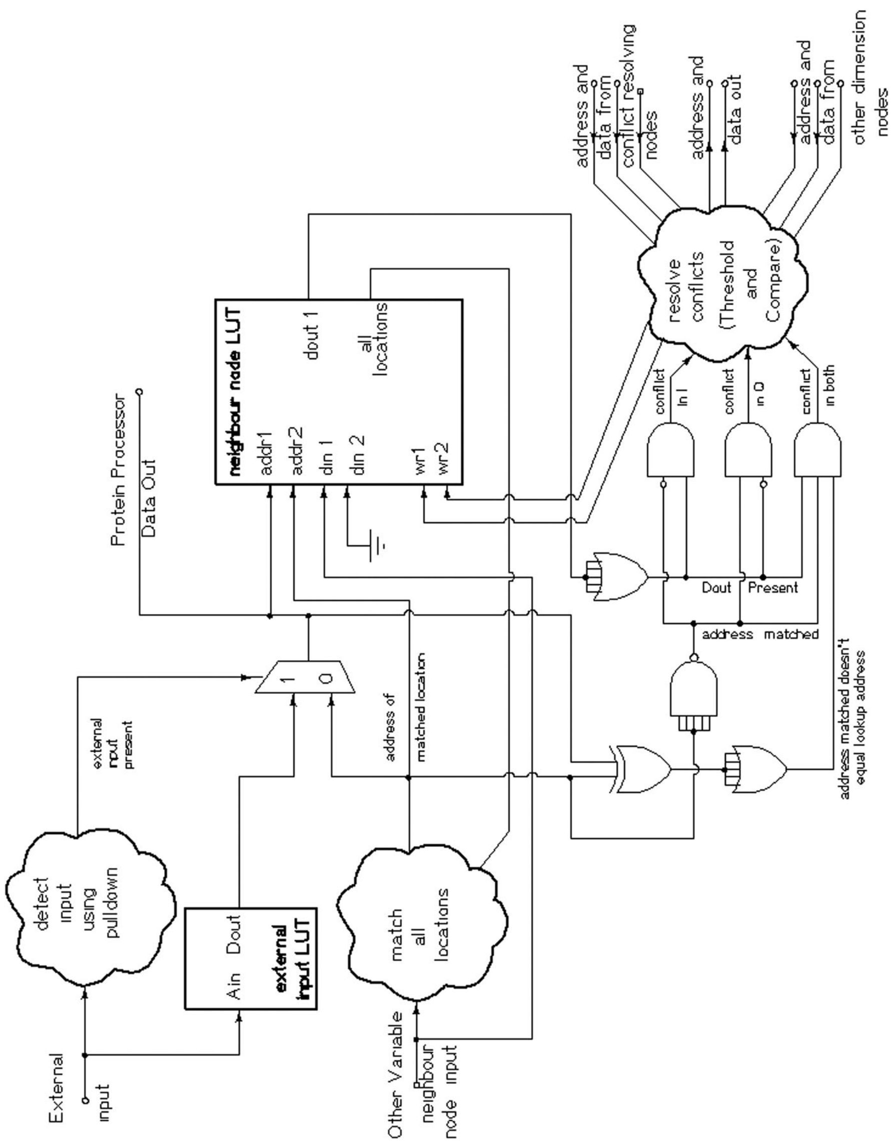
External input
detect input using pulldown
Ain
Dout
external input LUT
external input present
1
0
Protein Processor Data Out
neighbour node LUT
addr1
addr2
din 1
din 2
wr1
wr2
dout 1
all locations
address of matched location
match all locations
Other Variable neighbour node input
resolve conflicts (Threshold and Compare)
conflict in 1
conflict in 0
conflict in both
Dout
Present
address matched
address matched doesn't equal lookup address
address and data from conflict resolving nodes
address and data out
address and data from other dimension nodes

Fig. 3 Node schematic

The thresholding is based on the well studied and frequently used sample-and-hold circuit and the voltage adder. The voltage adder output increases as the voltages are summed, which is similar to the way neuronal synapses accumulate chemical charge. On the other hand, a digital implementation would use a binary representation with each bit using the same voltage to represent logic levels. Only by observing all these bits together, is the digital adder seen to be performing the

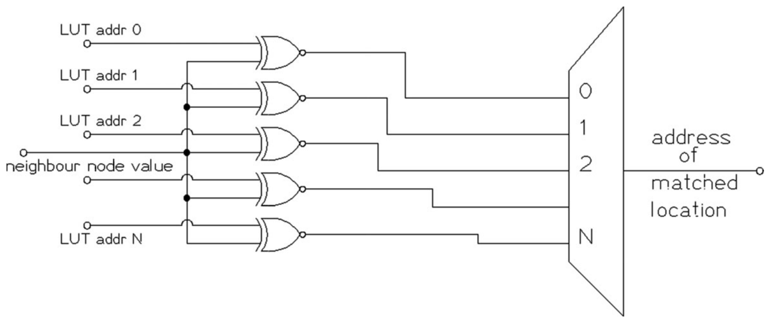

Fig. 4 Match all locations

same operation. Therefore, the analogue voltage adder better mimics its biological counterpart. The analogue circuit also uses fewer transistors and far fewer physical connections. It must be noted however that analogue transistors are much larger than their digital counterpart. Coupled with the extra hardware required for mixed signal design, this usually precludes analogue designs. In the case of the PPAM, the number of connections between nodes outweighs other considerations, particularly as the network is scaled. Although an N-bit symbol may be transmitted using N parallel lines, this becomes infeasible as the number of connections per node becomes prohibitive with any realistically sized network. Serial transmission is an alternative, but it significantly increases the complexity and size of the hardware and also the time required to transmit data, particularly as the network is scaled.

Complexity and size Serial communication involves the use of registers (typically shift registers) to store the data as it is received and a state machine (SM) to control the overall communication. The number of flip-flops in the registers and the SM are dependent on the width of the symbols; 16-bit symbols are quite conceivable. PPAM nodes communicates simultaneously with C Conflict-Resolving nodes and one Other-Dimension node. Therefore, given that symbols are represented using N-bits, each node must be composed of $C + 1$ serial communicators, each of which would be composed of an N-bit shift register and a $(\log_2 N)$-bit counter.

Timing and power Serial communication is dependent upon "clocking in" data as it is received. Assuming 16-bit symbols with 1 start-bit and 1 stop-bit, the clock would now have to be at least 18 times faster to be able to meet the same timing requirements. Also, power dissipation is directly related to signal transition. Using serial communication would therefore have a major impact on this as well—the exact value is dependent on the number of transistors, the length of interconnects, capacitance and other implementation specific factors. Finally, at eventOVN, nodes use inStatePP to recall the associated value. If this communication is serial, a clock would be required for recall operations as well. This is a major divergence from the synchronous store and asynchronous recall objectives of the PPAM. For these reasons, the digital alternative, though possible, is considered less desirable.

3.3.2 Digital threshold and compare

The digital logic threshold and compare is based on counters. Parallel communication was chosen to closely mimic the analogue circuit behaviour and ensure asynchronous recall. Conflict-Resolving nodes update in a staggered manner so that they have time to *register* updates from other nodes. The PPAM uses the clock skew inherent in digital logic designs to implement this staggered update (Fig. 5). Traditionally, clock skew is seen as an unwanted side-effect of large designs. As designs are scaled, considerable effort and expense is required to remedy this. The PPAM, on the other hand, embraces clock skew thereby making it more easily scalable. This opens interesting possibilities for future work exploring designs that use clock skew to their advantage.

3.4 Digital logic synthesis and implementation results

Including full schematics of the hardware architecture or a listing of the HDL code would not necessarily add to the understanding of the reader, therefore these have been excluded from this paper. However, a fully parametrized HDL (verilog) code including the test bench is available online.[1] Although the verilog is parametrized, Xilinx ISE constraints need to be tweaked manually. The following sections present results from the synthesis of this code.

3.4.1 Implementation on FPGAs

The HDL code for PPAM was synthesized for two different Xilinx FPGA architectures. The first was the older XCS10XL, attempted due to legacy reasons. A single node was synthesized per FPGA, with $C = 7$, and symbols were encoded in 2 bits. In addition, each node implemented 16 locations deep memory. The device utilisation summary is presented in Table 4. Reducing C to 3 allowed upto 2 PPAM nodes per FPGA.

To test the operation on the sample dataset in Sect. 2.2, the entire PPAM was implemented on one Virtex 5 FPGA (XUPV5LX110T[2] development board). $C = 6$, symbols were encoded in 4 bits and the LUTs were 16 locations deep.

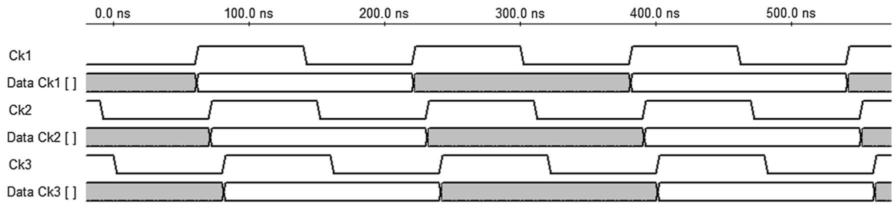

Fig. 5 Utilizing clock skew to implement staggered update

[1] http://www.eEvolved.info/blog.html#PPAMverilogHDL.

[2] http://www.xilinx.com/univ/xupv5-lx110t.htm.

Table 4 XCS10XL device utilisation summary

Elements	Used	Percentage
External IOBs	62 out of 112	55
Flops	4	
Latches	0	
IOBs driving global buffers	One out of eight	12
CLBs	163 out of 196	83
Latches	0 out of 392	0
CLB flops	36 out of 392	9
4 input LUTs	306 out of 392	78
3 input LUTs	34 out of 196	17
BUFGLSs	Two out of eight	25
STARTUPs	One out of one	100

The device utilisation summary is presented in Table 5a. The resource utilisation is shown in Table 5b. The Xilinx LogiCORE Clock Generator IP was used to generate staggered clocks and the Xilinx LogiCORE Distributed Memory Generator IP was used to generate asynchronous read, synchronous write RAM. For this implementation, the Clock Generator used a clock with period 10 ns and slowed it to generate an output clock of period 200 ns. The staggered clocks had a phase shift of approximately 18 ns.

3.4.2 Implementation on the Unitronics fabric

The PPAM was also implemented on the Unitronics fabric being developed by the project partners at UWE Bristol as part of the SABRE[3] project. The Unitronics (UX) fabric is a cellular architecture, with mechanisms inspired by the processes that occur in colonies of unicellular life; it is somewhat analogous to a custom FPGA. For full details about the fabric, its features and capabilities, refer to Samie et al. [23].

4 Experiments: testing the effects of quantisation

In order to test whether the PPAM is capable of extracting relationship information, the PPAM was used to learn the forward and inverse kinematics for a hand-eye coordination task. A robotic arm places objects in (and retrieves them from) its *reach space* (V1), while a vision system tries to identify the location of the object in its *vision space* (V2). The task was to associate the vision space with the reach space so that when one is known, the other can be *recalled*. The experiments were performed in the Bristol Robotics Laboratory[4] (BRL) with the help of the SABRE

[3] Self-healing cellular Architectures for Biologically-inspired highly Reliable Electronic systems.

[4] www.brl.ac.uk.

Table 5 5VLX110T implementation results

(a) 5VLX110T device utilisation summary

Elements	Used	Percentage
Slice logic utilization		
Slice registers	3,648	5
Slice LUTs	14,330	20
Used as logic	14,330	20
Slice logic distribution		
LUT flip flop pairs used	14,330	
With an unused flip flop	10,682	74
With an unused LUT	0	0
Fully used LUT-FF pairs	3,648	25
Unique control sets	774	
IO utilization		
IOs	314	
Bonded IOBs	312	48
Specific feature utilization		
BUFG/BUFGCTRLs	12	37
PLL_ADVs	1	16

(b) 5VLX110T resource utilisation summary

Elements	Used
BELS	14,703
GND	1
LUT2	3,356
LUT3	634
LUT4	2,339
LUT5	2,490
LUT6	5,511
MUXF7	371
VCC	1
FlipFlops/latches	3,648
FDC (FF Async clear)	480
FDC_1 (negedge FDC)	96
FDE (FF Clock Enable)	3,072
Clock buffers	12
BUFG	12
IO buffers	312
IBUF	19
IBUFG	1
OBUF	292
PLL_ADV	1

project partners. The BERT2 and the Vicon motion capture system were used for these experiments. Section 4.1 describes experiments to test the effect of quantisation on a forward/inverse kinematics task, while Sect. 4.2 presents the results. Forward/inverse kinematics experiments to compare the PPAM with state of the art techniques have also been conducted and results of this have been published in the past. Nonetheless, these results are summarised in Sect. 5.3 for completeness.

4.1 Experimental setup

In essence, the task is one of *learning* the kinematics (forward[5] and inverse[6]) of a robotic arm. This is by no means a difficult task for the state of the art today. The traditional approach is to use mathematics, usually trigonometry and geometry, to solve such tasks. On the other hand, rather than *calculating* the forward or inverse kinematics of the robotic arm, the PPAM attempts to *learn* kinematics from the dataset. Such a task is well within the domain of AI and as such is a good application to test the behaviour and performance of the PPAM. Lenz et al. [15] provides a detailed description of the hardware platform used, the relevant points of which are summarised here.

BERT2 (Fig. 6) is the second version of the upper-body humanoid torso at the Bristol Robotics Laboratory. The torso has four joints and 2 arms. Each arm has seven degrees of freedom. There is significant overlap in the movement, such that many different orientations of the arm can be used to reach the same location. Humeral and wrist movement were considered irrelevant to the task and these motors were not used. In order to simplify the shape of the reach space, only hip rotation was used from amongst the four torso joints. This meant that the reach space could be defined using a four-dimensional variable, with each dimension

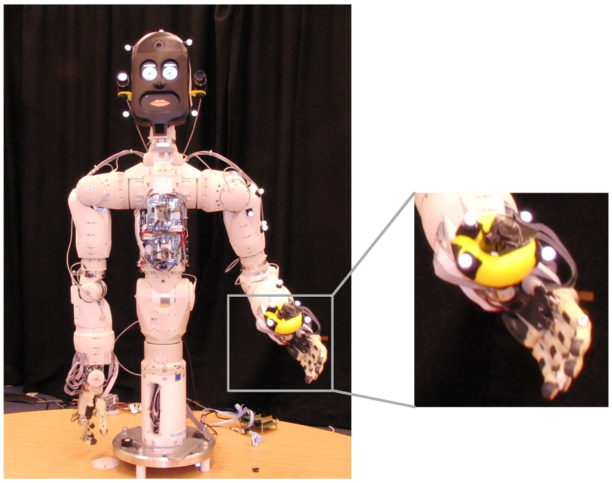

Fig. 6 Bristol Elumotion Robot Torso 2. Note the toroid object attached to the *left thumb*

[5] Calculate the position (and orientation) of a robotic arm from its joint angles.

[6] Calculate the joint angles from the position (and orientation) of a robotic arm.

being an angle value for each motor used—namely, hip rotation, shoulder flexion, shoulder abduction, and elbow flexion.

Vision is provided by the three-dimensional Vicon[7] Motion Capture system. The proprietary Vicon system detects and localises objects in 3-D space. Proprietary software developed at BRL [15] was used to retrieve Cartesian coordinates of objects in real-time, which formed the 3-dimensional variable representing vision space.

The following objectives were identified for this experiment.

- Determine the PPAM configuration required to store and accurately recall the dataset.
- Test whether the PPAM is able to extract underlying relationship information by placing an object in a new location in the 3-D space and using the object coordinates to reach for the object.

An important aspect of these experiments was to explore the effect of varying quantisation levels on the performance of the PPAM.

4.1.1 Generating the dataset

The range of motor movements was restricted according to Table 6a. These values were chosen empirically by observing the movements of the robot which defined a complex 3-D region. To simplify this region, the object coordinate space was limited as specified by Table 6b, which is a subset of the region in Table 6a. Table 6b defines a cuboid with base $b = 75$ cm, length $l = 96$ cm and height $h = 73$ cm, with a total volume of 5.256×10^5 cubic centimetres ($b \times l \times h$).

The dataset was produced by generating 500 evenly distributed random values for each of the four dimensions of the BERT2 Motor movement variable (500×4 random values). The BERT2 was placed in the pose corresponding to each of these 500 values, while an object was attached to its hand. Then the coordinates of the object were read back. If the coordinates were within the range described by Table 6, the data-point was accepted and added to the final dataset; otherwise it was discarded. The final dataset consisted of 430 real-valued tuples. This dataset was encoded into integer symbols to be used by the PPAM.

4.1.2 Determining the optimal configuration

The real-valued data was encoded into b-bit wide integers, which are abstract symbols from the perspective of the PPAM. This is the quantisation process and b is inversely proportional to the noise introduced by quantisation.

Experiments with $b = (4, 5, 6, 7, 8)$ were performed while keeping quantisation levels equal. This meant that each symbol encodes $(M - m)/(2^b - 1)$ real-values; where M is the maximum real-value and m is the minimum real-value. The divisor is $2^b - 1$ instead of 2^b because the symbol 0 is reserved to indicate that external data is not present. For an 8-bit symbol size, each Cartesian coordinate X-symbol encodes

[7] www.vicon.com.

Table 6 BERT2 experiment variables' ranges

(a) V1: BERT2 motor angular movement range (°)

Motor	Min angle	Max angle
Hip rotation	−45.0	45.0
Shoulder flexion	−30.0	30.0
Shoulder abduction	0.0	45.0
Elbow flexion	−30.0	30.0

(b) V2: Vicon object coordinate range (in cm)

Axis	Min value	Max value
X	−45	30
Y	−21	75
Z	14	87

Table 7 Effect of symbol size

Symbol size (*b*)	Volume (cm^3)	Memory depth (per node)
4	155.733	16
5	17.643	32
6	2.102	64
7	0.257	128
8	0.032	256

(from Table 6), (30 − (−45))/255 = 0.294 cm. Similar calculations for Y-symbols (0.376) and Z-symbols (0.286) show that the Cartesian coordinate variable encoded with 8-bits means a single 3-dimensional value encodes a cubic volume of 0.032 cubic centimetres. Volumes for all the symbol sizes are shown in Table 7 and the effect of this, along with results of the experiments using the different symbol sizes, are presented in Sect. 4.2.

The memory depth shown in Table 7 is the absolute theoretical maximum number of locations in each node, whereas the actual number of locations used in each node may be less (as shown by the previous results from Qadir et al. [19] and [20]). The total number of memory locations in the entire PPAM is dependent upon the number of Conflict-Resolving nodes (C) in the network. Experiments were conducted to find the optimal configurations for each of the symbols sizes listed in Table 7 and results are presented in Sect. 4.2.

4.1.3 Extracting relationships

To test if the PPAM can extract information from the training dataset about the relationship of the underlying dataset, previously unseen values need to be presented. Two measures were used to test this. Firstly, the training dataset was divided into 2 parts with the first 420 elements being part of the training set and the last 10 elements being used to test the PPAM in the presence of previously unseen data. This unbalanced ratio was used because the dataset already provided only a very sparse coverage and decreasing the number of training elements meant that the coverage would decrease even further. However, these experiments do not take into account the noise inherent in any real-world system interacting with the physical world. This includes such things as sensor reading errors, motor movement granularity, and play or flexibility in joints and motors. A second set of experiments were conducted to incorporate this. These were conducted using the physical BERT2 platform described in Sect. 4.1. Objects were placed randomly within the arena and the object coordinates in the vision space were retrieved from the Vicon system. These coordinates were presented to the PPAM and recalled values were used to move the BERT2 arm. The measure for success in this case was simply whether the arm would be able to successfully touch the object. Results are presented in Sect. 4.2

4.2 Results

4.2.1 Determining the optimal configuration

To determine the optimal configuration for storing the training dataset, experiments were conducted with varying numbers of Conflict-Resolving nodes (C) for each of the five different symbol sizes (b). Figure 7 shows the results. The PPAM generates a *confidence* value along with the recalled result. If the value with the highest confidence is correct, this is defined as a successful recall. If the PPAM did recall the correct value, but indicated a higher confidence for a different value, then this is a *Partial Error*. *Errors* are defined as the case where the correct value was not recalled at all. For each configuration (value of C), the 430 element training dataset was stored; next the 430 Cartesian coordinate values were presented and corresponding motor movement values recalled; and finally 430 motor movement values were presented and corresponding Cartesian coordinates recalled. Errors and partial errors were recorded for each dimension of data. The maximum possible errors are $(430 \times 4) + (430 \times 3) = 3{,}010$. Figure 8 plots the score for each symbol size as C is varied. Score is calculated as $2E + e$, where E is errors and e is partial errors, so that the objective is to minimise the score.

The number of errors in Fig. 7 reflect the storage capacity of the memory. Therefore, past trends [19, 20] can be seen to be repeated, namely memory capacity increases rapidly with C but with diminishing returns. Furthermore, from Fig. 8 it can be seen that larger symbol sizes have fewer errors for the same C, with the exception of $b = 5$. Therefore, $b = 8$ achieves zero errors and zero partial errors for $C = 24$, while $b = 7$ achieves the same for $C = 36$ and so on. Symbol sizes of $b = 4$ and $b = 5$ are exceptions in a couple of ways. Firstly, for both these cases, although zero errors were achieved, zero partial errors were not achieved for any configuration. The minimum for $b = 4$ was with a configuration of C = 60 while that for $b = 5$ was observed for $C = 69$. Secondly, $b = 5$ has more errors for the

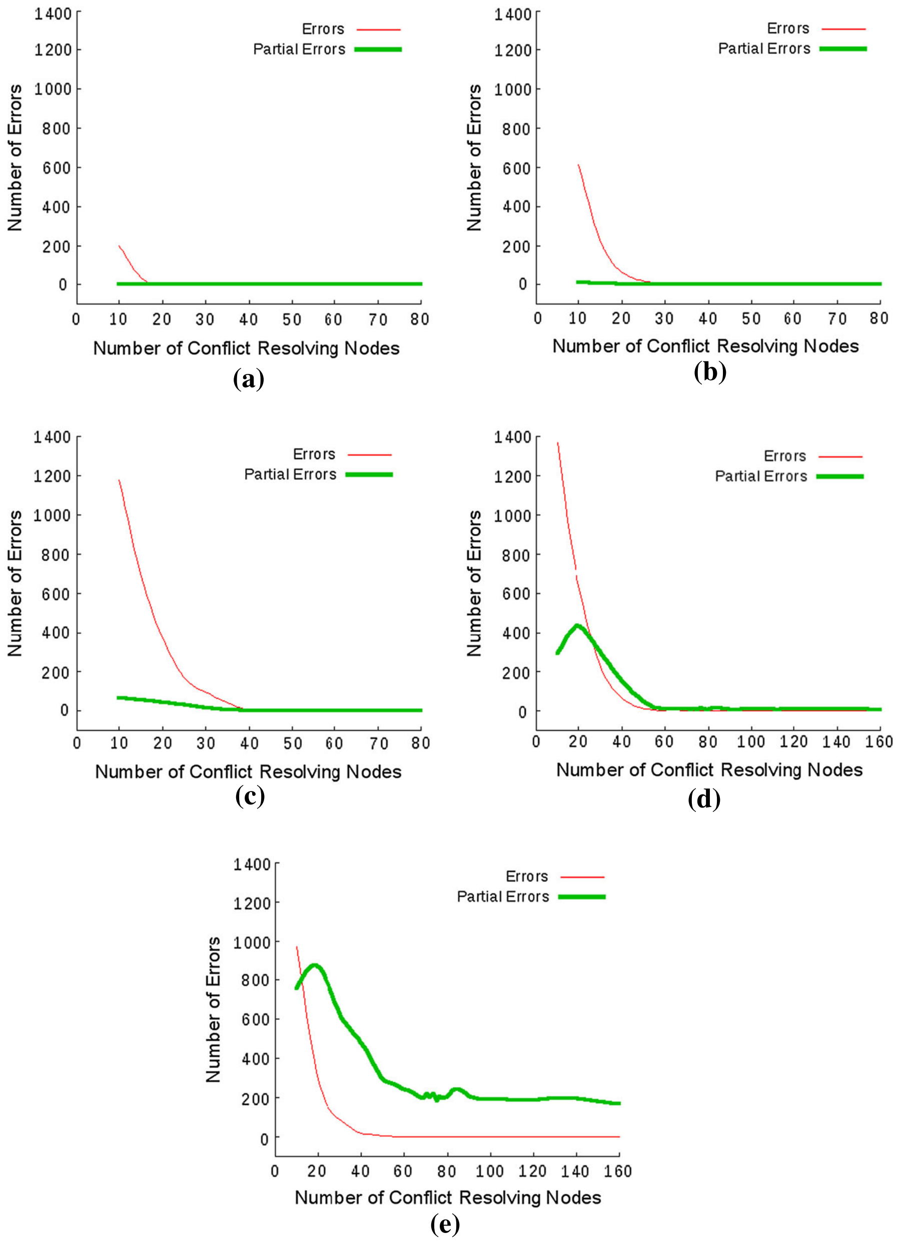

Fig. 7 For each symbol size (b), the number of Conflict-Resolving nodes (C) is varied and errors and partial errors are plotted to find the optimal configuration for the BERT2 dataset. (**a**) $b = 8$, (**b**) $b = 7$, (**c**) $b = 6$, (**d**) $b = 5$, (**e**) $b = 4$

same value of C which is in opposition to the trend for the rest of the symbol sizes. Note, however, that the number of partial errors for $b = 5$ approaches zero (though it does not attain it) while that for $b = 4$ remains quite high (around the 200 mark).

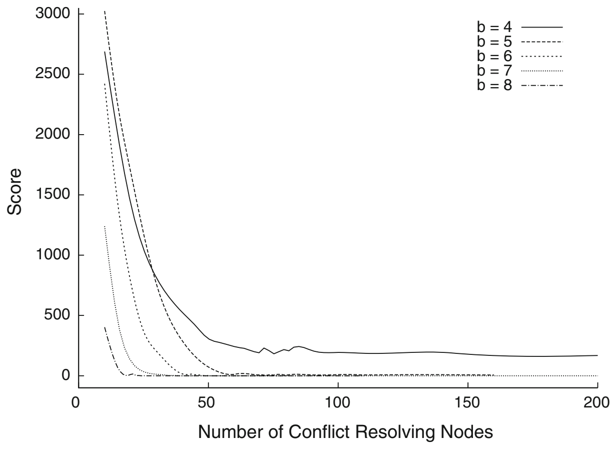

Fig. 8 Score for each symbol size while searching for the optimal C

Therefore, although the curve for $b = 4$ starts below the curve for $b = 5$, it ends up above it. In this, it does follow the overall trend.

Both these anomalies can be explained because of the effect symbol size has on the training dataset. As the symbol size is reduced to a sufficiently small value, the nature of the relationship of the two variables being associated also changes. As shown in Table 10, $b = 4$ and $b = 5$ both have significantly fewer unique number of coordinates. However, the number of unique motor commands remain at 430 for $b = 5$ and drops just a little to 428 for $b = 4$. Therefore, whereas for $b = 6$ and onwards the dataset has a one-to-one relationship (or almost that for $b = 6$), the dataset when $b = 4$ and $b = 5$ has a one-to-many relationship. This means, that when a particular value is presented for recall, there are multiple correct answers. As may be expected, in such a situation the PPAM can get confused.

4.2.2 Extracting relationships

For each configuration (value of C), the 420 element training dataset was stored. Next the 10 Cartesian coordinate values were presented and corresponding motor movement values recalled; and finally 10 motor movement values were presented and corresponding Cartesian coordinates recalled. Errors and partial errors were recorded for each dimension of data. The maximum possible errors are $10 \times 4 + 10 \times 3 = 70$. Figure 9 plots the errors for each symbol size as C is varied, while Fig. 10 plots the score for each symbol size as C is varied. As in Sect. 4.2.1, score is calculated as $2E + e$, where E is errors and e is partial errors, so that the objective is to minimise the score.

From the plots in Fig. 9, it can be seen that varying C does not have quite as much effect on the memory capacity of the PPAM as it did in Fig. 7. In fact, from Fig. 9a, it may be seen that $b = 8$ performs quite poorly in the case of previously unseen data, with errors being very close to the worst case value (as is the

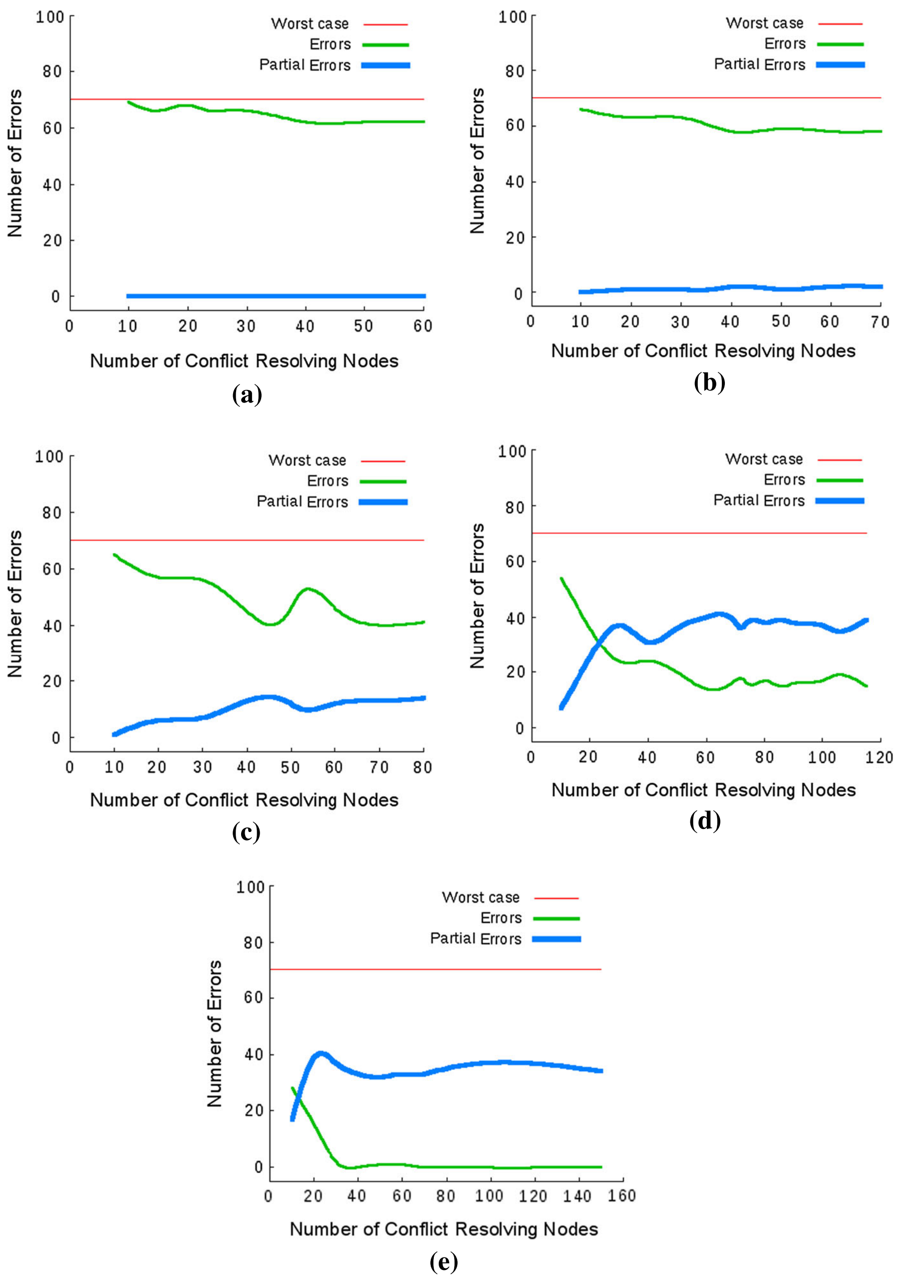

Fig. 9 For each symbol size (*b*), the number of Conflict-Resolving nodes (*C*) is varied and previously unseen data is presented to test whether the PPAM was able to extract relationship information about the underlying dataset. (**a**) $b = 8$, (**b**) $b = 7$, (**c**) $b = 6$, (**d**) $b = 5$, (**e**) $b = 4$

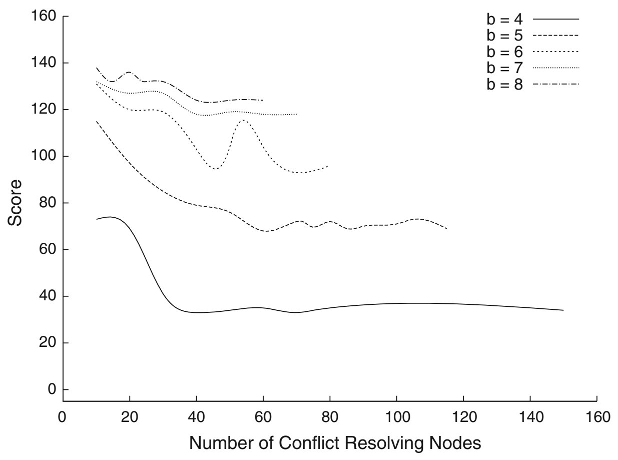

Fig. 10 Score for each symbol size while testing on previously unseen data

corresponding score in Fig. 10). The performance of $b = 7$ (Fig. 9b) is quite similar to that of $b = 8$ and its only with $b = 5$ that errors fall below 50 %. $b = 4$ is the only case where errors are reduced to zero and this trend is quite obvious from Fig. 10 which shows the score for $b = 4$ being far better than the rest. This effect is related to the sparsity of data and the effect of quantisation discussed in Sect. 5.2.

For the second set of experiments using the physical BERT2 robot, only symbol sizes of 4, 5 and 6 were attempted, since the simulations with $b = 8$ and $b = 7$ indicated a high error rate. An experiment was considered a *complete success* if the index finger or thumb of the hand touched the object. It was considered a *partial success* if any part of the robot's arm touched the object. 10 objects were placed randomly in the environment and results of these experiments are shown in Table 8 and discussed in Sect. 5.

5 Discussion

5.1 Tuning Conflict-Resolving nodes (C)

Table 9 collects the information from Figs. 10 and 8 to show the values of C for which the score converges. These results support previous results [19, 20] indicating

Table 8 BERT2 experiments using previously unseen object locations

Symbol size (b)	Complete success	Partial success
4	0	3
5	0	1
6	0	0

Table 9 Convergence of C for unseen data and the entire training set

Size (b)	Conflict Resolving nodes (C)	
	Unseen data	Entire training dataset
4	40	60
5	60	72
6	40	50
7	35	38
8	15	24

that a PPAM with fewer than optimal nodes displays almost optimal performance for unseen data.

5.2 Unseen data in the physical space

After training BERT2 on 430 data points, in none of the 30 tests was the BERT2 able to successfully reach an object placed in a new location (Table 8). Only 4 of the 30 cases resulted in partial success (any part of the hand touching the object). At first glance, these poor results may seem to indicate that the PPAM is unable to extract relationships, but an analysis in the light of the sparsity of the dataset presents some interesting points.

Part of Table 7 is reproduced in Table 10. Let N be the total number of discrete points possible in a 3-D volume. If n is the number of discrete data points in a dataset, then the sparsity of the dataset can be defined as n/N. Each point in the 430 element dataset represents a region of space whose volume is shown in Table 10. Volumes of points could overlap, therefore the actual coverage of the 3-D space depends upon the training dataset itself. However, maximum coverage of the volume would be achieved if the volume of points do not overlap with others. Assuming this, generates the least sparse dataset. The values in the "Total Volume Covered" column are generated using this assumption, therefore they are simply the total number of unique coordinates in the dataset multiplied by the volume of a data-point in that dataset. The total volume of the entire region is 5.256×10^5 cubic centimetres (Sect. 4.1). Therefore, the percentage coverage column (sparsity) is the

Table 10 Sparsity of coverage in the dataset

Symbol size (b)	Number of unique coordinates	Volume per symbol (cm^3)	Total volume covered (cm^3)	Percentage coverage
4	291	155.733	45,318	8.622
5	410	17.643	7,234	1.376
6	428	2.102	900	0.171
7	430	0.257	110	0.021
8	430	0.032	14	0.003

"Total Volume Covered" as a percentage of the total volume of the entire region. As can be seen, even with 4-bit wide symbols, only 8.622 percent of the total volume of space is covered. Note that a 4-bit wide symbol represents 155.733 cubic centimetres. Alternately, using 8-bit wide symbols, only 0.003 percent of the entire region is covered. For low values of *b*, the PPAM would be more likely to recall a value corresponding to the input, as data presented would be more likely to be previously observed data. However, the values recalled would suffer from a large quantisation error when mapped back to the real world. Alternately, for high values of *b*, the PPAM would be less likely to recall a value corresponding to the input, even though the values that are recalled would have a high accuracy when mapped back to the real world. Therefore, increasing the size of the symbols (*b*) has the following effects:

1. Increases the width of the memory in the nodes as each node needs to store *b* bits wide data in memory.
2. Increases the depth of the memory in the nodes as the data of one variable is used as the address in the other variable.
3. Changes the number of conflicts in the dataset as varying the quantisation levels modifies the number of real-valued points encoded in each symbol. The nature of the change (increase or decrease in the number of conflicts) is dependent upon the actual dataset.
4. Increases the accuracy when mapping back to real-valued data as quantisation noise is reduced.
5. Decrease the ability to extract relationship information and predict values when presented with previously unseen data as more and more data is previously unobserved.

From this, it is evident that the performance of the PPAM hinges not only on the number of Conflict-Resolving nodes (*C*), but also on the bit-width (*b*) that effects the quantisation.

5.3 Comparison with other techniques

5.3.1 Comparing hardware resources

Existing solutions for hetero-associative memory are typically software implementations on traditional ALU-based processors. A comparison between a software solution and a hardware solution (e.g. PPAM) is not entirely fair. Due to the novelty of the PPAM architecture, there would always be some imbalance in the comparison. Nonetheless, a comparison with existing architectures is essential.

Although a mixed signal VLSI implementation of the PPAM would be ideal, due to time and cost considerations, only FPGA implementations were attempted. PPAM implementations on FPGAs are handicapped because of their inability to customise RAM blocks (in the way required by the PPAM) and also because of the lack of analogue resources. Regardless, experiments with various configurations indicate that the Virtex 5 LX110T FPGA can fit approximately 200 nodes with 32 memory locations in each node (Sect. 3.4). Qadir et al. [18] shows that a 31 node

PPAM learns the complete dataset to successfully operates a mobile robot (ePuck) on an object avoidance task. Qadir et al. [18] also shows that an optimally trained BAM [13] even with 85 neurons is only able to learn 98 % of the same dataset. Since the SoftTOTEM [2] is a commercially produced neural network hardware architecture for the Xilinx Virtex XCV600E FPGA, this is a good candidate for comparison. The SoftTOTEM boasts 32 neurons with 8-bit weights, operating at 40 MHz. Note that this maximum number of neurons (32) is inclusive of all layers in the neural network and is insufficient to implement the BAM for the ePuck object avoidance task. To put this further into perspective, consider that Chellapilla and Fogel [4] evolved feed-forward ANNs with 2 hidden layers to play checkers in what later became the famous Blondie24 program [6]. The ANNs used for Blondie24, had 40 neurons in the first hidden layer and 10 neurons in the second one. Yang et al. [31] described the application of a feed-forward ANN with back-propagation to the problem of image recognition to differentiate between crops and weeds. It uses one hidden layer and, in its various experiments, implements up to 300 neurons in the hidden layer. Wang et al. [30] use up to 150 nodes in ANNs used to associate 8×7 pixel images. Sudo et al. [24] use a variant of the Bidirectional Associative Memory using 99 neurons in the competitive layer to associate the 7×7 pixel IBM PC CGA characters dataset (capital letters versus small letters). Tirdad and Sadeghian [26] use 100 neurons in a Hopfield network to generate a pseudo-random number generator that passes the 15 critical tests identified by the National Institute of Standards and Technology. None of these solutions would fit on the SoftTOTEM.

5.3.2 Comparing computation method and performance

The task described in Sect. 4.1 is to learn the forward/inverse kinematics of a robotic arm. Experiments were conducted to evaluate and compare the performance of the PPAM against a regression-based solution applied to another forward/inverse kinematics task. The *catcher-in-the-rye* project was conducted at the University of Aberystwyth, and defined the same forward/inverse kinematics task as is specified in Sect. 4.1, using a robotic arm and a stereo vision system. Details of the dataset and results of the regression-based solution are presented in Hülse et al. [10]. The PPAM results and comparison with Hülse et al. [10] were originally published in Qadir et al. [20]. These are summarized below since a comparison with other techniques provides an essential benchmark.

Let T be a set of vector pairs $T = \{X^{(k)}, Y^{(k)}\}_{k=1,\ldots,N}$. Hülse et al. [10] stores selected associative pairs from the dataset and uses Euclidean distance to recall the closest matching pair using the *Catcher-in-the-rye* algorithm (or catcher algorithm). Each pair is tested to see if it should be added to the list of stored pairs and old data that hasn't been refreshed in a specified number of cycles is removed. Complete details of the algorithm (including pseudo-code) are presented in the Online Resource 5.

The catcher experiments use polar coordinates (d, α) that range as follows: $30 \leq d \leq 60$ cm; $-1.4 \leq \alpha \leq 1.4$ radians.

$$D = \sqrt{[d_1 \cdot sin\alpha_1 - d_2 \cdot sin\alpha_2]^2 + [d_1 \cdot cos\alpha_1 - d_2 \cdot cos\alpha_2]^2} \quad (1)$$

Distance D between points can be measured according to Eq. 1. Benchmarks for calculating trigonometric functions vary greatly, depending upon the hardware architecture and the method used to compute the function. Intel architectures can be used as the standard, particularly since they provide hardware support for calculating trigonometric functions using native instructions (like *fsin, fcos* and *fsincos*). Latencies and throughput are shown in Table 11. Here clock latency is defined as the number of cycles to complete execution of the instruction and throughput is defined as the number of cycles to wait before the instruction can be issued again. The exact number of cycles is dependant upon the range of the input data and the processor model. This estimate does not include the overhead from the branch instructions or the load/store instructions required to manipulate the data. Furthermore, the calculations are performed on floating point numbers, thus requiring a Floating Point Unit.

Hülse et al. [10] uses a training set of 300 elements ($N = 300$), and shows that best results are achieved if the entire training set is maintained in memory. Therefore, recalling an arbitrary associative pair requires, at most, N iterations of the loop (with trigonometric calculations). On the other hand, the PPAM required 29 conflict resolving nodes to fully store and recall the entire dataset and the maximum number of memory locations in any node was 160, with the average and quartiles being much lower [20]. Table 12 compares the number and types of operations for the catcher-in-the-rye with a PPAM configuration of H nodes that achieves the same association. It does not consider the branch operations implicit in any software implementation of the algorithm, since these would be removed if the algorithm was implemented in hardware. The catcher-in-the-rye implementation is assumed to be optimized such that calculations are not repeated and results are considered to be stored in temporary memory (CPU registers). Memory access (read/write) does not include access to these temporary locations or to any other *working* memory. Furthermore, memory requirements (or operations) for stimulating nodes or providing input is not included. For the catcher-in-the-rye, $X^{(k)} \in \mathbb{R}^N$ and $Y^{(k)} \in \mathbb{R}^M$, while for the PPAM $X^{(k)}$ and $Y^{(k)}$ are encoded values of the same. Note that each of the H operations execute in parallel in each of the H nodes in the PPAM. Further note that all memory variables in the catcher-in-the-rye are real values and therefore all catcher-in-the-rye calculations require floating point arithmetic. The details for calculating the values in Table 12 are published in Qadir et al. [20] and can also be found in the Online Resource 5.

Table 11 Clock latency and throughput for trignometric operations on Intel architectures [11]

Operation	Latency in clock cycles	Throughput in clock cycles
fsin	160–200	130
fcos	180–280	130
fsincos	170–250	140

Table 12 Comparison of PPAM with standard techniques

	Catcher-in-the-rye	PPAM
Trigonometric	Up to $N \times 4$	0
Multiplications	Up to $N \times 6$	0
Square root	Up to N	0
Additions	Up to N	0
Subtractions	Up to $N \times 2$	0
$>$Comparisons	0^{a} or up to N^{b}	0
$\leq$Comparisons	Up to N	0
$==$ Comparisons	0	Up to $N \times H$
Memory reads	Up to $N \times 3^{a}$ or up to $N \times 4^{b}$	$N \times H$
Memory writes	1^{a} or up to $N \times 5 + 6^{b}$	0^{a} or up to H^{b}

[a] For recall

[b] For store

The catcher algorithm uses error thresholds to control errors in the recalled values. As shown by Hülse et al. [10] the algorithm accumulates uncertainty. For the PPAM, the equivalent of this error threshold is performed implicitly through encoding the real values into integer symbols. If more values are placed in the same *bin* such that a larger range of values are encoded using the same symbol, the error threshold (or *quantisation error*) is higher. In Qadir et al. [20] we show that, unlike the catcher-in-the-rye, increasing this quantisation error did not have a detrimental effect on the performance. In this paper we explored the effects of quantisation in a more controlled manner and discovered that although quantisation does not directly reduce performance, it must be tuned along with C to optimise performance.

6 Conclusion and future work

This paper presented the hardware architecture for the PPAM which is a novel bidirectional hetero-associative memory. It also presented the results of an experiment to associate the reach space of the BERT2 robot with the vision space of the Vicon system and to test if the PPAM was able to extract relationship information about the underlying dataset. The effect of quantisation on the performance of the PPAM was explored. Quantisation was emulated by varying the bit-width (b) of symbols encoding real-values. Section 4.2.1 shows that as b increases, the relationship between the variables being associated becomes more well-defined and requires fewer number of Conflict-Resolving nodes (C). Section 4.2.2 shows that, although increasing b sharpens the relationship between the variables, it also reduces the coverage of the overall underlying dataset. This in turn means that extracting relationship information becomes harder. Therefore, although the PPAM can extract relationship information, it is a function of the quantisation parameter (b) which needs to be tuned along with the number of Conflict-Resolving nodes (C) to maximise performance. Being able to analytically

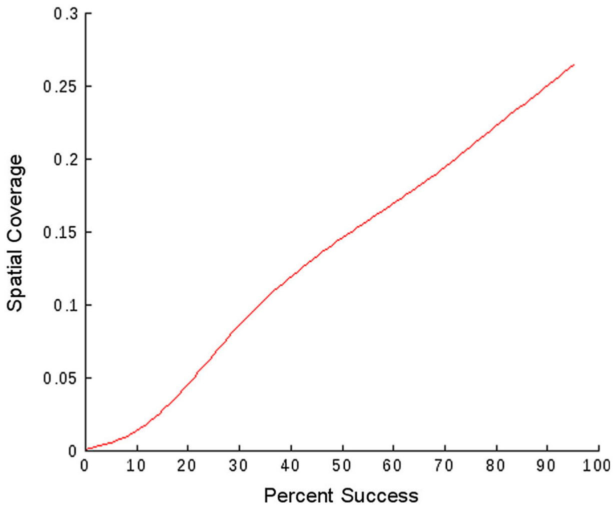

Fig. 11 Projected coverage requirements to improve success rate (generated using linear regression)

determine a value for C to provide a given level of accuracy and fault tolerance would be quite helpful, though it is uncertain whether such a (analytical) method exists.

It remains to discover the relationship between spatial coverage and the accuracy of recall; in particular, what is the level of coverage at which the PPAM can provide 95 percent accuracy? The logical next step to the BERT2 experiments would be to conduct more experiments in order to generate a higher coverage of the space and evaluate the performance of the PPAM in this situation. However, due to limited time and access to the shared resource (BERT2) at the Bristol Robotics Laboratory, it was not possible to gather more data. An imprecise projection can nonetheless be made from the current data. Figure 11 uses linear regression to predict how coverage varies with the success rate. It plots the Percentage Coverage from Table 10 against the score obtained using the data from Table 8. Spatial coverage is normalised to range between 0 and 1.

It must be noted, however, that in terms of the more abstract aim of discovering an alternate computational paradigm that could be applied more widely to a large range of AI applications, the results are less favourable. The results of Sect. 4 which analyse the effect of quantisation indicate that the ability of the PPAM to extract information about the underlying dataset hinges on the quantisation level used, and the coverage of the dataset. This leads to the philosophical discussion of what is a *reasonable* amount of coverage (by the training dataset) that is necessary before an architecture can be evaluated.

Although it is an advantage that quantisation errors can be utilised to increase performance, incorrectly chosen quantisation levels (or b) could drastically reduce the performance of the PPAM. In terms of generalisation from a sample dataset, whereas other techniques suffer from *the curse of dimensionality*, the PPAM *requires* higher dimensionality in data to generalise. Association of a 1-dimensional variable with a 1-dimensional variable reduces to a simple lookup. However, the larger the dimensions, the greater would be the ability of the PPAM to generalise, since the likelihood of the PPAM having previously observed a portion of the input data increases.

Acknowledgments The research was funded by the EPSRC funded SABRE (Self-healing cellular Architectures for Biologically-inspired highly Reliable Electronic systems) project under Grant No. FP/F06219211.

References

1. J. Amaral, J. Ghosh, An associative memory architecture for concurrent production systems. in *Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics*, vol. 3 (1994), pp. 2219–2224. doi:10.1109/ICSMC.1994.400194.
2. G. Anzellotti, R. Battiti, I. Lazzizzera, G. Soncini, A. Zorat, A. Sartori, G. Tecchiolli, P. Lee, Totem: a highly parallel chip for triggering applications with inductive learning based on the reactive tabu search. Int. J. Mod. Phys. C **6**(4), 555–560 (1995)
3. J. Backus, Can programming be liberated from the Von Neumann style? A functional style and its algebra of programs. Commun. ACM **21**(8), 613–641 (1978). doi:10.1145/359576.359579
4. K. Chellapilla, D. Fogel, Evolving neural networks to play checkers without relying on expert knowledge. IEEE Trans. Neural Netw. **10**(6), 1382–1391 (1999). doi:10.1109/72.809083
5. B. Coppin, *Artificial Intelligence Illuminated*, 1st edn. (Jones and Bartlett Publishers Inc, Sudbury, MA, 2004)
6. D.B. Fogel, *Blondie24: Playing at the Edge of AI* (Morgan Kaufmann Publishers Inc., San Francisco, CA, 2002)
7. A. Grübl, *VLSI Implementation of a Spiking Neural Network*. PhD thesis (University of Heidelberg, Heidelberg, 2007)
8. S. Hauck, Asynchronous design methodologies: an overview. Proc. IEEE **83**(1), 69–93 (1995). doi:10.1109/5.362752
9. D.O. Hebb, *The Organization of Behavior: A Neuropsychological Theory* (Wiley, New York, 1949)
10. M. Hülse, S. McBride, M. Lee, Fast learning mapping schemes for robotic hand-eye coordination. Cogn. Comput. **2**, 1–16 (2010). doi:10.1007/s12559-009-9030-y
11. Intel (2011) Intel 64 and IA-32 Architectures Optimization Reference Manual. Intel, 248966th edn
12. C.G. Johnson, The non-classical mind: cognitive science and non-classical computing. in *Intelligent Computing Everywhere, chap 3*, ed. by A. Schuster (Springer, Berlin, 2007), pp. 45–59. doi:10.1007/978-1-84628-943-9_3
13. B. Kosko, Bidirectional associative memories. IEEE Trans. Syst. Man Cybern. **18**(1), 49–60 (1988). doi:10.1109/21.87054
14. S.W. Lee, J.T. Kim, H. Wang, D.J. Bae, K.M. Lee, J.H. Lee, J.W. Jeon, Architecture of RETE network hardware accelerator for real-time context-aware system. in *KES (1)*, vol. 4251 ed. by B. Gabrys, R.J. Howlett, L.C. Jain (Springer, LNCS, Berlin, 2006), pp. 401–408
15. A. Lenz, S. Skachek, K. Hamann, J. Steinwender, A. Pipe, C. Melhuish, The bert2 infrastructure: an integrated system for the study of human-robot interaction. in *IEEE-Humanoids-2010* (2010), pp. 346–351. doi:10.1109/ICHR.2010.5686319
16. T. Pfeil, A. Grübl, S. Jeltsch, E. Müller, P. Müller, M.A. Petrovici, M. Schmuker, D. Brüderle, J. Schemmel, K. Meier, Six networks on a universal neuromorphic computing substrate. Front. Neurosci. **7**, 11 (2013)
17. O. Qadir, J. Liu, J. Timmis, G. Tempesti, A. Tyrrell, Principles of protein processing for a self-organising associative memory. in *Proceedings of the IEEE CEC 2010* (2010)
18. O. Qadir, J. Liu, J. Timmis, G. Tempesti, A. Tyrrell, From bidirectional associative memory to a noise-tolerant, robust self-organising associative memory. Artif. Intell. **175**(2), 673–693 (2011). doi:10.1016/j.artint.2010.10.008

19. O. Qadir, J. Liu, J. Timmis, G. Tempesti, A. Tyrrell, Hardware architecture for a bidirectional hetero-associative protein processing associative Memory. in *Proceedings of the IEEE CEC 2011*, (New Orleans, 2011)
20. O. Qadir, J. Timmis, G. Tempesti, A. Tyrrell, The protein processor associative memory on a robotic hand-eye coordination task. in *6th International ICST Conference on Bio-Inspired Models of Network* (Information and Computing Systems (Bionetics2011), York, 2011)
21. O. Qadir, J. Timmis, G. Tempesti, A. Tyrrell, Profiling the fault tolerance for the adaptive protein processing associative memory. in *NASA/ESA Conference on Adaptive Hardware and Systems (AHS2012)* (Nuremberg, Germany, 2012)
22. C. Sakellariou, P.J. Bentley, Describing the fpga-based hardware architecture of systemic computation (haos). Comput. Inform. **31**(3), 1001–1021 (2012)
23. M. Samie, G. Dragffy, A. Popescu, T. Pipe, C. Melhuish, Prokaryotic bio-inspired model for embryonics. in *Proceedings of AHS 2009* (IEEE Computer Society, Washington, DC, AHS '09, 2009), pp. 163–170. doi:10.1109/AHS.2009.45
24. A. Sudo, A. Sato, O. Hasegawa, Associative memory for online learning in noisy environments using self-organizing incremental neural. Network **20**(6), 964–972 (2009). doi:10.1109/TNN.2009.2014374
25. J. Teich, Invasive algorithms and architectures (Invasive Algorithmen und Architekturen). IT—Inf. Technol. **50**(5), 300–310 (2008)
26. K. Tirdad, A. Sadeghian, Hopfield neural networks as pseudo random number generators. in *Fuzzy Information Processing Society (NAFIPS), 2010 Annual Meeting of the North American* (2010), pp. 1–6. doi:10.1109/NAFIPS.2010.5548182
27. T. Toffoli, CAM: a high-performance cellular automaton machine. Phys. D **10**, 195–204 (1984)
28. D. Tsafrir, Y. Etsion, D.G. Feitelson, S. Kirkpatrick, System noise, OS clock ticks, and fine-grained parallel applications. in *Proceedings of the 19th Annual International Conference on Supercomputing* (ACM, New York, NY, USA, ICS '05, 2005), pp. 303–312. doi:10.1145/1088149.1088190
29. A. Turing, Computing machinery and intelligence. Mind **59**, 433–460 (1950)
30. L. Wang, M. Jiang, R. Liu, X. Tang, Comparison bam and discrete hopfield networks with cpn for processing of noisy data. in *Proceedings of ICSP2008* (2008), pp. 1708–1711. doi:10.1109/ICOSP.2008.4697466
31. C.C. Yang, S. Prasher, J.A. Landry, H. Ramaswamy, A. Ditommaso, Application of artificial neural networks in image recognition and classification of crop and weeds. Can. Agric. Eng. **42**(3), 147–152 (2000)