Accelerating Brain Circuit Simulations of Object Recognition with CELL Processors

Andrew Felch, Jayram Moorkanikara Nageswaran[1], Ashok Chandrashekar, Jeff Furlong[1], Nikil Dutt[1], Richard Granger, Alex Nicolau[1], Alex Veidenbaum[1]

Neukom Institute, Dartmouth College
Hanover, NH 03755, USA
E-mail:andrew.felch,ashok.chandrashekar,richard.granger@dartmouth.edu

[1]Centre for Embedded Computer Systems, University of California, Irvine
Irvine, CA 92697, USA
E-mail: jmoorkan,jfurlong,dutt,nicolau,alexv@ics.uci.edu

Abstract

Humans outperform computers on many natural tasks including vision. Given the human ability to recognize objects rapidly and almost effortlessly, it is pragmatically sensible to study and attempt to imitate algorithms used by the brain. Analysis of the anatomical structure and physiological operation of brain circuits has led to derivation of novel algorithms that in initial study have successfully addressed issues of known difficulty in visual processing. These algorithms are slow on uni-processor based systems, thwarting attempts to drive real-time robots for behavioral study, but as might be expected of algorithms designed for highly parallel brain architectures, they are intrinsically parallel and lend themselves to efficient implementation across multiple processors. This paper presents an implementation of such parallel algorithms on a CELL processor and further extends it to a low-cost cluster built using the Sony PlayStation 3 (PS3). The paper describes the modeled brain circuitry, derived algorithms, implementation on the PS3, and initial performance evaluation with respect to both speed and visual object recognition efficacy. The results show that a parallel implementation can achieve a 140x performance improvement on a cluster of 3 PS3s, attaining real-time processing delays. More importantly, we show that the improvements scale linearly, or nearly so in practice. These initial findings, while highly promising in their own right, also provide a new platform to enable extended investigation of large scale brain circuit models. Early prototyping of such large scale models has yielded evidence of their efficacy in recognition of time-varying, partially occluded, scale-invariant objects in arbitrary scenes.

1. Introduction

Processors have experienced tremendous progress (Moore's Law) and computer chips now have a million times more building blocks than they did 40 years ago. Historically, Intel et al. have attempted to use those resources (transistors) to increase the speed of already-existing programs by: (1) supporting higher instruction throughput (using pipelines, caches, branch prediction etc.,) and, (2) finding and executing multiple instructions simultaneously. After many years, both of these techniques are now facing severely diminishing returns, and in an extreme divergence from tradition the newest chips yielded by Moore's Law no longer speed up old programs. Instead, the additional transistors are used to fabricate multiple CPUs on a single computer chip. The unfortunate drawback is that few applications contain the parallelism necessary to significantly benefit from the additional CPUs.

In contrast, the mammalian brain has evolved circuits that lack any central processors or main memory but instead comprise billions of low-precision processing units (neurons) with distributed memory (synapses) stored within their interconnections. With such a simple computing fabric, how can humans still outperform computers at natural tasks such as visual object recognition? We propose that these brain circuit components are designed and organized into specific brain circuit architectures that perform atypical but quite understandable algorithms conferring unexpectedly powerful functions to the resulting composed circuits. As an example, humans recognize visual objects in less than a second, during which billions of neurons receive input from the visual scene, but due to slow neuron communication (milliseconds) only a few tens of serial operations are performed. Algorithms derived from the anatomical structure and physiological operation of these circuits similarly lack serial dependencies and are inherently parallel, thus poised to take advantage of parallel hardware such as multi-core processors.

In this paper we first present the components of visual brain circuit architecture, and an overview of visual object

recognition. We then show a parallel "brain derived vision" (BDV) algorithm derived from this, and we demonstrate its application to a particular visual recognition benchmark of known difficulty (the "Wiry Object Recognition Database" from CMU). Various kinds of parallelism existing in the BDV algorithm facilitate mapping it onto a variety of computing platforms. It is hoped that understanding the pros and cons of each platform will help in the design of a customized architecture for BDV and future algorithms. Some of the computing platforms considered for accelerating the BDV algorithm are FPGA, CELL processor [1], and Graphics Processing Unit (GPUs). In this paper we describe the programmed realization of the algorithm on the CELL multi-processor based PlayStation 3 gaming console and analyze the resulting findings while scaling a small cluster of PS3s from one to three nodes.

The overall flow of the paper is as follows. In Section 2 we briefly describe the background for this work. In Section 3, we describe the experimental setup and methodology for evaluating the BDV algorithm. In Section 4 we describe the architectures suitable for simulating the BDV algorithm. In Section 5 we describe the details of the BDV on the PS3 CELL processor and in Section 6 its relative performance on FPGA. Section 7 has the object recognition results followed by the conclusion.

2. Background

The thalamo-cortical system, which constitutes more than 70% of the human brain, is primarily responsible for all sensory processing as well as higher perceptual and cognitive processing. The photo-receptors (rods and cones) in the eye are activated by light and the information is sent electrically to the thalamocortical system. It has long been noted [2, 3] that this system of the brain operates hierarchically: Downstream regions receive input from upstream regions and in turn send feedback, forming extensive cortico-cortical loops. Early visual components have been shown to respond to simple constructs such as spots, lines, and corners [4]; these form the lower stages in the hierarchy of organization. Further downstream higher level constructs and complex types of shape are selectively activated in response to a cluster of low-level features, thus forming more stages in the hierarchy [4, 5, 20]. Also the neuron response becomes independent of the exact location or size of the object (translation and scale invariance) [6]. In our simulations each stage of the hierarchy responds to a particular feature which is composed of multiple line segments. In particular we present a computation which starts with three line segments ("line triples"). Though the organization is highly simplified, the architecture is shown to be very effective on difficult visual applications such as the CMU WORD database [7]. It is also hoped that implementations of further downstream areas will extend the work to more abstract perceptual and cognitive processing [8, 9].

In Figure 1, we illustrate the working of a simplified form of the BDV algorithm using 4 levels of hierarchy to detect the number '8'. The first level or layer involves detection of the line segments (line segment extraction). The second level involves converting the line-segment into line-segment triples and the detection of various known line-segment triples present in the image. The detection of specific line-segment triples is represented by the activation of a grid in level 2. Each individual grid in level 2 and higher represents a shape processor. Each shape processor is modeled to detect the same set of shapes and its variation. Level 2 elements are receptive to only a small field of the input image.

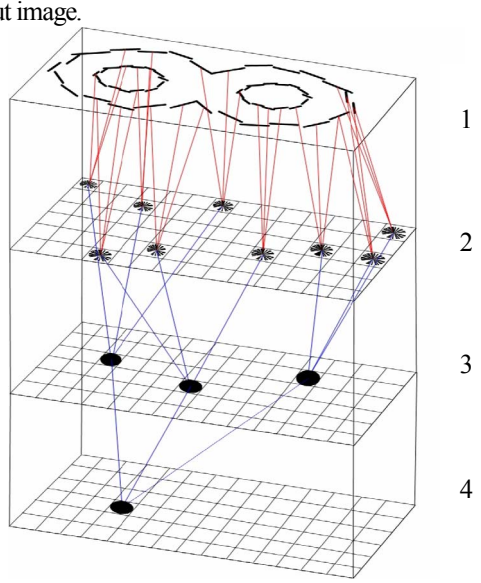

Figure 1: The hierarchical organization of shape detectors operating on the optical character number eight. "Bottom-Up" input flows from 1 to 2, 2 to 3 etc.

In Figure 1, we observe that some of the shape processors are activated for the given input image. Levels 3 and higher correspond to higher level constructs formed by a group of low level constructs. At the top-most levels an image region is classified into an appropriate type. Level 3 and 4 are under preliminary study and were implemented such that for the given object we know the relative activation pattern of other shape detectors in Level 2. A hierarchical organization as depicted in Figure 1 enables sharing of lower level shape processors by higher levels when recognizing other objects, and hence reducing the total memory requirements. For example, while detecting number '8' and '9', a number of shape processors in Level 2 will be shared. The exact identification will be done at higher levels.

We briefly explain some aspects of the BDV algorithm and its biological relevance. A given set of input neurons (pattern) activates a particular neuron and this set of inputs is loosely referred in our paper as *receptive fields* (RF). Any

two shapes are considered similar based on the number of activated neurons shared in their activation patterns. As a result of sparse population codes (SC) [10], most neurons are inactive; this concept is represented in a highly simplified form as sparse bit-vectors. The intrinsic random connectivity tends to select some areas of neurons to respond to some input features (RFs). These neurons train via increments to their synaptic connections, solidifying their connection-based preferences to specific input features. After a simulated "developmental" phase, synapses are either present or absent and each neuron's level of activation can be represented as the bit vector.

Neurons activate local inhibitory cells which in turn de-activate their neighbors; the resulting competition among neurons is often modeled as the K best (most activated) "winners" take all or K-WTA [8, 9, 11], which our model incorporates. These K winners activate a next set of neurons, termed RF2. As objects are viewed, these RF2 neurons are synaptically trained, becoming "recognizers" of classes of objects. RF2 activation can in turn be used in "top-down" or feedback processing, to affect the RF1 detectors based on what the RF2 cells "think" is being recognized. More details of the mechanism are shown in Figure 2.

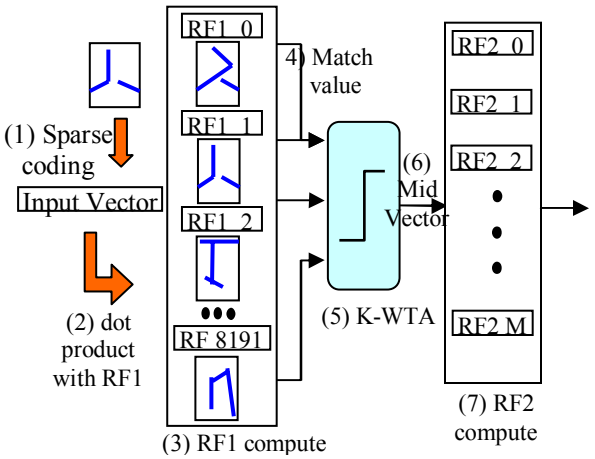

Figure 2: Simplified model of the B-U computation

Figure 2 depicts the process of converting line-triple representations into detected shapes, termed Bottom-Up (B-U) computation. The model described here uses 8192 neurons to represent the first set of input feature detectors (RF1) and 1024 neurons for RF2; other configurations exhibit comparable behavior. Intuitively, increasing the number of neurons can be used to increase the number of classes of objects that can be recognized. Step 1 in Figure 2 converts line triples into a somewhat scale invariant and translation invariant bit-vector representation based on the angular relations of line segment endpoints. The representation is a sparse encoding (SC), and slight changes in the orientation of the line segments, or movement of a line endpoint, lead to decreasing similarity between bit vectors derived from the normal and modified shapes. An example for generation of the line-segment triple is shown in Figure 3 and the encoding process is illustrated in

Figure 4. Step 3 depicts the approximately 8,192 vectors of 160 bits (RF1 vectors), each of which previously and maximally trained on a single input. Step 2 indicates that the dot-product will occur between the input vector and all RF1 vectors. In Step 4 the resulting matches (8,192 match values between 0 and 160 each) are generated. Step 5 depicts the application of a threshold for k-WTA operation, with K=512. This actually means that out of 8192 RF1 neurons only 512 best matching RF1 neurons will be triggered to output. Step 6 shows the 512-hot[1] of 8,192 bit-vector (called Mid Vector), which provides input into the next set of 1,000 vectors (RF2 vectors, 1024-2048 hot of 8,192). Step 7 indicates the dot-product operation between the inputs from Step 6 with all RF2 vectors. The output indicates the match values, ranging between 0 and 512, indicating how well each RF2 vector matched the input. See [14] for further details.

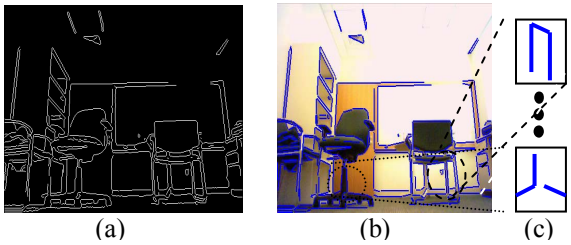

Figure 3: (a) shows the extracted edges in a picture. (b) shows the picture superimposed with the line segments (c) shows sample line-segment triples corresponding to different objects. These line segment triples are processed by the B-U engine for recognition.

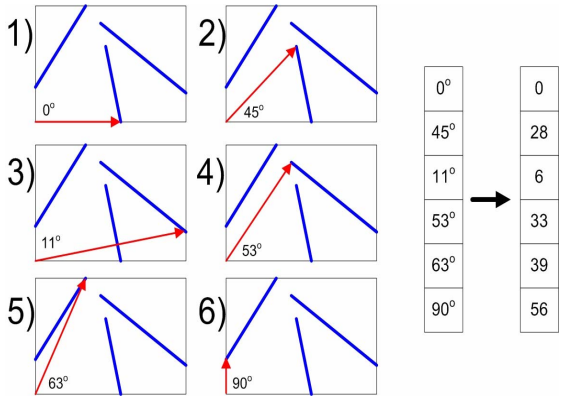

Figure 4: The process of converting a line-triplet into a vector of quantized angles. 0 corresponds to 0° and value 56 corresponds to 90°.

3. Experimental Setup

[1] In 1-hot coding only a single bit is 1. In N-hot coding N bits are 1 (may or may not be in consecutive positions).

The Wiry Object Recognition Database (WORD) [7] is used to provide a difficult dataset dependent on shape-based recognition. The database contains a series of videos, in which a barstool is placed in several different office environments. The goal is to determine the location of the barstool in the videos. The location of the stool in the video frame is judged correct if the bounding box had less than 25% area difference from the true bounding box.

Line segments are extracted using the Canny edge detector implementation [12] such that approximately 400-600 line segments are extracted per frame. The algorithm and parameters used in this work were chosen for their ease-of-use combined with the fact that humans often recognize objects from the extracted line segments alone. The performance impact of using other numbers (e.g., 100 or 1000 line segments) will be studied in the future.

The shape detectors modeled here are believed to learn their shape during child development, a process during which 50% of all synapses die off, and presumably only the strongest remain. Under these circumstances, synapses can be modeled as either present or absent, and each neuron's input weights can be modeled as a bit-vector. Thus computing the degree of match between one shape detector and an input shape is a matter of computing the bit-vector dot-product.

Each of the 8192 upstream neurons (RF1s) was trained to be maximally receptive to a single input line triple. This is in accordance with the previously discussed physiological studies that found upstream neurons to be simpler than downstream neurons. Each RF1's line triple was randomly selected from one of the 22 videos of WORD. To model the sparse activity of the brain with sparse encoding, the upstream layer (so called "RF1" vectors) was made to activate in a K-WTA fashion, with K = 512 (6% activity density) in all processing. Downstream neurons (so called "RF2" vectors) were trained on multiple line triples during the developmental period, randomly selected from those line triples in the videos with which no receptive field yet matched very well. All objects that need to be recognized are a combination of elements of RF2 and located at specific relative distances with respect to each other. To enable recognition, an early prototype of downstream brain regions was implemented, built to serve the very simple purpose of creating expected locations of particular shapes inside a recognizable object relative to other shapes within that object. Figure 1 depicts the full system with additional hierarchy (layers 3 and 4) operating over the line-triple shape detectors. For each training frame of video, representations of line triples built from members of the training object (a sitting-stool) were iteratively added to a hierarchy of the shapes and spatial relationships using the downstream layer's most active shape detector, along with its expected location (relative to other shapes already added to the hierarchy) and scale (standard deviation of line triple endpoints from their midpoint). Thus, when testing on a new frame of video, a highly active shape detector of a particular scale indicates the expected locations of other shape detectors of a learned object in the current frame. Testing whether the expected shapes are at their expected locations (within some maximum distance) computes the likelihood of the visual object. Thresholds are used to convert the matches between expected and actual shapes into recognition confidence, and confidence above a threshold indicates an actual guess of a recognized visual object. The guesses created bounding boxes, which were judged as in [13] for comparison.

4. Architectures for BDV

A salient feature of the BDV algorithm is the high degree of parallelism at various levels. The simplest inherent parallelism is the bottom-up computation for different line segment triples. In our experiments most pictures contain about 10,000 to 20,000 useful line segment triples. Hence the best matching RF2 for all of these line segment triples can be concurrently evaluated. The next level of parallelism is achieved by the shape processing mechanism. For each line segment triple we need to find a best matching shape from the given table of shapes (RF2 elements). This search can also be potentially parallelized to a higher degree with the only limitation being the communication overhead. The algorithm also exhibits large amounts of bit-level parallelism and SIMD parallelism. For example in the RF2 computation we need to evaluate an 8192-bit dot-product and population count on the result to estimate the degree of match between two RF2 vectors. This can be concurrently executed either at the bit, byte or at higher word levels.

Various kinds of parallelism in the BDV algorithm facilitate mapping it onto a variety of computing platforms. Also, understanding the pros and cons of each platform helps in the design of customized architectures for the BDV algorithm. Some of the computing platform choices for BDV algorithms are briefly discussed below.

(1) FPGA (Field Programmable Gate Array):

FPGA based solutions are suitable for specific customization of the architecture at bit-level and also have the ability to deliver high performance with low power requirements. Some of the disadvantages are high cost associated with high-performance FPGA and large application development time to achieve considerable performance. A detailed study of the trade-offs when mapping BDV on FPGA and the resulting performance achieved is described elsewhere [14].

(2) CELL / Play Station 3 (PS3):

STI (Sony, Toshiba, IBM) CELL Broadband Engine (CELL BE) is a high-performance, low-cost multi-processor targeting graphics and multimedia applications. The CELL BE contains eight specialized Synergistic processors (SPE) and one dual-threaded PowerPC Processor all operating at about 3.2 GHz. A detailed description of the chip is present in [1]. The recently released Sony Play Station3 (PS3) is

powered by the CELL processor and is available at about $500. The PS3 console offers a programmable PowerPC processor with 6 SPEs. It has an inbuilt Gigabit Ethernet, making it suitable for cluster computing. Thus, the PS3 offers impressive performance and programmability for mapping the BDV algorithm. Section 5 contains the details of the various trade-offs involved in mapping BDV on CELL/PS3 The main limitation is in developing efficient parallelization and optimization to exploit the capabilities of the CELL processor.

(3) General Purpose Computer Clusters and High Performance Parallel Architectures: This platform is suitable for large scale prototyping of BDV, and with parallel programming models like MPI & PVM, it is easier to map BDV on grids or clusters. The main disadvantages are the impact of communication overhead on overall performance and large system cost.

(4) Programmable Graphic Processing Units (GPU): Recent GPUs such as NVidia's CUDA and AMD's CTM, offer affordable high performance, parallel hardware. Increasingly these GPUs are becoming much more programmable and useful for general purpose applications.

(5) Application Specific Integrated Circuit (ASIC) or MPSoC (multi-processor system-on-chip): For very low power and small footprint, ASIC or MPSoC offers a good solution.

This paper discusses the implementation of the BDV algorithm on the CELL architecture. In our future work we will be looking into GPU and ASIC/MPSoC as a computing platform for BDV algorithms.

5. Mapping on CELL / Play Station 3 (PS3)

We now present the programming methodology and trade-offs involved in mapping BDV on CELL. We have used different levels of programmable parallelization available on CELL namely: (1) parallel execution of many CELLs (network or cluster of CELLs), (2) multiple programmable units executing simultaneously (namely 6 SPEs and 1 PPE in PS3) (3) concurrent computation and communication (DMA operation) by the SPEs, (4) instruction level parallelization in the SPE with two instructions executing simultaneously, and (5) SIMD parallelization using 128-bit data path (up to 16 single byte operations in one cycle). In the remainder of this section we examine the parallelization applied across the CELL clusters and within the CELL. Compiler assisted parallelization using IBM XL compiler [15] will be part of our future studies.

For mapping the application on clusters we use a client-server architecture consisting of a cluster of three PS3s controlled by a powerful desktop PC. This architecture fits well with our application computation requirements. The cluster (server) consists of multiple CELLs in charge of the bottom-up computations and the desktop PC (client) controls the overall flow of the application. For each line-triple request, the workflow follows the process of the desktop PC sending the initial bit-vectors as input to the bottom-up engine, and each PS3 sending the output data structure back over the network. The desktop PC uses this output data structure information to prune the top-down search and evaluate higher levels of constructs (beyond line triples, to conjunctions of line triples, etc.) present in the image to arrive at confidence values for object recognition at different locations in the image.

5.1 Parallelization within the CELL

In this section we describe the techniques used in parallelizing the application within the CELL. First we execute the bottom-up engine on a single processor and then evaluate the critical functions that need to be executed parallelly within the CELL. A simple functional model of the bottom-up computation is shown in Figure 5. The Sparse Coding (SC) block converts the input line triplets into a space and scale invariant code. The RF1 and RF2 process input block finds the best matching line triplets from the reference set. The RF1 Activate pattern block implements the k-WTA computation. The bottom-up (B-U) engine was executed on a 2.13 GHz Intel Core2 (E6400) CPU. A fractional breakdown of execution time into functional bottom-up code blocks is shown in Figure 6. Approximately 1.89ms was required to execute the B-U computation for a single line segment triple. This corresponds to a B-U computation throughput of about 526 line segment triples per second (526 LST/sec). In Figure 6 we can observe that RF1 Vector computations and RF2 Vector computations take more than 95 % of the overall execution time. These are the critical functions that need to be optimized and parallelized to increase the throughput of the B-U computation. From these results, the runtime of processing an entire video frame with 30,000 line-triples can be estimated at approximately one minute. To execute the BDV algorithm on interactive robots, the recognition time of humans must be achieved (approximately 500 ms per frame, thus requiring a speedup of about 120x i.e., from 526 LST/sec to about 60,000 LST/sec).

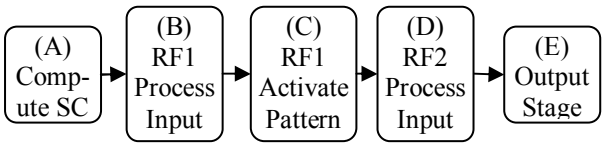

Figure 5: Functional Model of the BU Engine

The code size and the data size (both static and dynamic) need to be evaluated to effectively determine the memory footprint and the bandwidth requirements of the given application. Each SPE has a local store (LS) of 256 KB which can be used for both code and data. This small

memory size influences the way code and data for the applications are partitioned across the CELL. For many programs with a small code size, function overlaying and resident partition management [15] might not be necessary. The total code size for the B-U computation is about 57 KB and hence no function overlaying mechanism was used in our implementation.

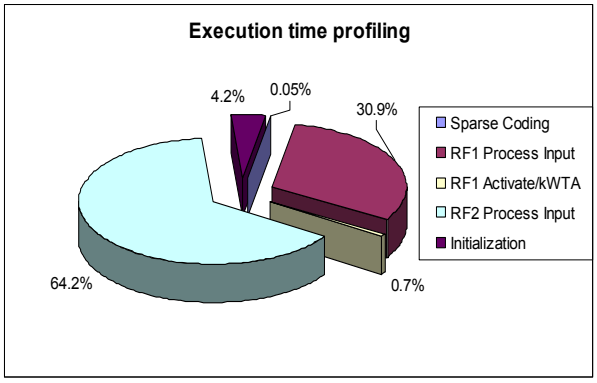

Figure 6: Fractional execution time of the important functions in the Bottom-Up engine on an Intel Core E6400 running at 2.13GHz

TABLE 1: Analysis of data structures used by BDV

Main data structures	Data Size	Data usage	Data Access Pattern	Accessing Tasks (Figure 5)
RF2 Vector Table	1.1 M	Partitionable	Linear	(D)
RF1 Vector Table	160 K	Partitionable	Linear	(B)
Sparse Code Table	125 K	Static/fixed	Random	(A)
SC Angle Table	48 K	Static/fixed	Random	(A)
popCount RF1	8 K	Static/fixed	Random	(B) & (C)
Histogram K-WTA	2 K	Static/fixed	Linear	(B) & (C)

With respect to data size, we evaluated the memory requirement of several large data structures used in our application (shown in Table 1). We further classified the data structures based on their usage as static/fixed and partitionable. If a particular function or loop is parallelized to run across different SPEs, we need to determine whether the data set is duplicated in each SPE (called static/fixed data set) or the data set gets divided across SPEs (called partitionable data set). This classification is important to decide what type of parallelization and data access mechanism should be used once the application is mapped onto the CELL. This information is also useful to determine the SPE bandwidth and the SPE LS memory requirements for a particular parallel model.

Furthermore, we need to determine how to map the data sets with size larger than the available SPE LS. To reduce this constraint, various techniques such as software cache, double buffering, pre-fetching [15], etc., can be used depending upon the data access pattern. For our application we have four large data structures (Table 1) namely RF1 Table, RF2 Table, Sparse Code (SC) Ordering Table and Angle Table. The SC Ordering and Angle tables are used by the PPE for input bit-vector generation. The remaining data sets can be accessed either through software cache or directly on SPE LS. Also, we observe from the algorithm that RF1 and RF2 table elements are accessed linearly (one after another in a specific sequence) during the comparison operation with the given input data. Hence if RF1 and RF2 do not fit into the SPE LS, then they can utilize either double buffering or software cache with data access optimization to allow efficient access to large data arrays.

Various kinds of generic parallel models can be developed from the functional model of the B-U engine. The possible models are: overlapped functional parallel model (OFP), data parallel model (DF), series-parallel model (SP), and overlapped series-parallel model (OSP). A model is termed overlapped if the communication and computation can happen concurrently, and hence the waiting time associated with communication can be mitigated.

In a fully overlapped functional parallel (OFP) model each functional block is mapped onto an SPE and hence the actual execution time of the model is dependant upon the execution time of the slowest functional block. Through this type of parallelization, the code size restriction can be reduced since each function, rather than the whole application, is mapped to a separate SPE. Load balancing issues, however, make this model difficult to implement. This model can be extended to process networks [16] or data flow networks [17] with APIs for communication and appropriate modeling methodology. Since the SPE LS is of a very small size, usage of these communication APIs will reduce the available memory resources even further.

The next logical choice for parallelization is a simple series-parallel model (SP). This model overcomes the load balancing limitation of a fully overlapped functional parallel model by splitting the sequential model into a series-parallel graph at loop boundaries either manually or by using the compiler (OpenMP primitives). If the data and computation is evenly partitioned across these loop boundaries, this kind of parallel model exhibits good load balancing, speedup, and reduced SPE LS requirements. A version of a SP model for the B-U engine is shown in Figure 7.

The serial portion can either be executed in the PPE or SPE, depending upon the complexity of the serial task. The main qualitative advantage of this model is lower data size requirement in each SPE, as well as reduced application latency and reduced SPE bandwidth. Two disadvantages are that the serial portion can affect the overall execution time and the presence of higher synchronization requirements

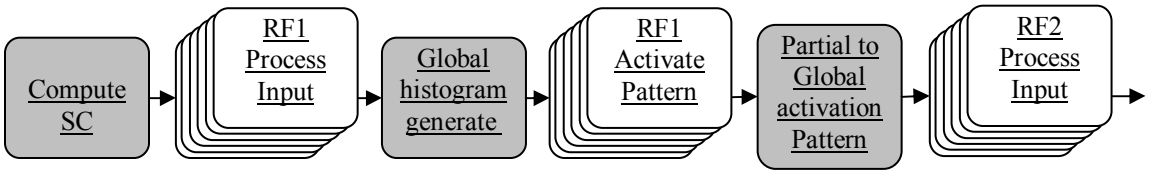

Figure 7: Parallel Model for Brain-derived Vision algorithm. Parallel part of the code is indicated by cascaded box

will result in increased waiting time for synchronization with serial portions of the execution, during which all the SPE cores will be waiting for new data. A series-parallel model can be extended to an overlapped series-parallel (OSP) model by overlapping computation and communication using either a double buffer or FIFOs. Thus instead of waiting for the serial portion to finish its operation and communicate the result to the SPE, the SPE performs the computation for the next input data. During this time the serial portion completes the execution and communicates the data by means of DMA so that the data is ready for the next cycle of SPE computation. Unfortunately, this approach does not solve the problem caused by potentially high-synchronization requirements between the serial and parallel portions.

For much higher performance the overlapped data parallel (ODP) model can be employed. In this type there is virtually no communication between the SPEs. Earlier models parallelize the bottom-up so that only part of the code or data is mapped on to the SPE. But in ODP model an SPE can be treated as a full processor and complete bottom-up computation for a line segment triple is mapped to a single SPE. This model requires large SPE bandwidth and large SPE LS because all the data and code for the execution of the application must be present or accessible by the SPE. The performance of this model is dependant upon the technique used to overcome the code and data size restrictions within the SPE LS. Table 2 gives a quick qualitative comparison of different kinds of parallel models.

5.2 Other kinds of optimizations

We now present specific optimizations carried out in the implementation of the B-U algorithm on CELL to exploit the low-level features which improves the concurrent execution of the code. More details of other kinds of programmer optimizations are available in [18].

- DMA alignment optimization: DMA operations in CELL can have a size of 1,2,4,8,16 bytes or multiples of 16 bytes. If a particular transaction's address crosses the 128 byte boundary, the results can be achieved through additional DMA transactions. Hence by means of careful alignment of important data that is communicated regularly, the overall communication bandwidth required by the application can be significantly reduced. If DMA alignment optimization is carried out on too many data sets, then it will result in significant wastage of precious SPE LS memory.
- Mailbox Vs signaling mechanism optimization: The CELL allows various means for synchronization, like regular DMA operations, mailboxes and signaling mechanisms. The mailbox mechanism allows 32 bit communication but takes about 0.5us (taking into account the setup and various application code overhead) for each 32 bit transaction. The signal communication mechanism allows 1-bit synchronization between the SPE and PPE at a much higher speed. Hence, the signaling mechanism was selected for synchronization between various tasks.
- Branch optimization: The SPE does not have a branch prediction unit and assumes a sequential program flow. Thus pipeline stalls due to mispredicted branches can be very costly (on the order of 18 to 19 cycles). Various techniques such as function inlining, loop-unrolling and branch hinting mechanisms were used to reduce the branch misprediction overhead.
- Compute intensive kernels in the BDV code (e.g., dot-product and population counting) can be significantly speeded-up by means of the various CELL SIMD instructions. After optimizing the kernels using special 128-bit SIMD instructions such as *absdiff, abs_sumb, spu_add, spu_cntb* [18], one RF1 inner loop computation takes only 31.2 cycles on an average (the desktop PC version takes 164 cycles) and RF2 inner loop takes about 246 cycles (the desktop PC version takes about 2879 cycles). Thus with superior 128-bit

Table 2: Comparison of different application parallel models on CELL

Model name	Application Latency	Bandwidth usage	Memory usage	Modeling effort	Performance
Functional parallel model	High	Low	Low	Low	Low-Medium
Series-parallel	Medium	Medium	Medium	Medium	Medium
Overlapped series-parallel	Low	Medium	Medium	Medium-High	Medium-High
Overlapped data-parallel	Medium	High	High	High	High

Table 3: Actual and Estimated Performance for BDV on Desktop PC and PS3 with a single CELL. The approach used to calculate the stages in an overlapped series-parallel model is shown in Figure 8.

Function name	Serial model (us)	series-parallel (SP) model (us)	Overlapped Data-parallel (ODP) model (us)	Overlapped functional parallel (OFP) model (us)	stage	Overlapped series-parallel (OSP) model (us)
(A) Sparse coding	0.89	0.80	0.80	0.80	1	0.80
(B) RF1 Process Input	603.53	13.78	82.66	82.66	2	13.78
(C) Partial to Global Histogram	0.00	2.71	0.00	0.00	3	13.78
(D) Generate Partial MidVector	13.34	2.48	3.87	3.87	4	2.71
(E) Partial to Global MidVector	0.00	1.91	0.00	0.00	5	2.48
(F) RF2 Process Input	1276.68	12.91	77.23	77.23	6	12.91
(G) Dump Output	0.60	0.75	0.75	0.75	7	12.91
					8	0.75
Performance Evaluation	Actual	Actual	Estimated	Estimated		Estimated
Effective time per LST	1895.04	35.34	27.55	41.33		30.06
Speedup w.r.t serial	1.00	53.63	68.78	45.85		63.04

SIMD instruction set and SPE instruction fine-tuning, impressive speedups are possible.

We implemented the series-parallel (SP) model on the CELL with various optimizations described above. On a single CELL present in PS3 we measured a speedup of 57 times using the series-parallel model compared to serial execution on a desktop CPU (E6400) running at 2.13 GHz. It should be noted that no special optimizations using SSE2 or MMX instructions were carried out on the desktop PC running Intel Core. From the performance of the series-parallel model we could estimate the performance of other parallel models (results are shown in Table 3). These estimations are very useful for obtaining approximate performance of various kinds of parallel models from only a single implementation. The execution time of an overlapped data-parallel model was estimated by evaluating the computation time for each function when it is completely mapped onto the SPE and assuming that we can overlap the computation and communication. Hence, for example, the execution time for RF1 processing in an overlapped data-parallel model is equal to the sum of the execution times of RF1 processing on each SPE in an unoverlapped series-parallel model. The execution time for an overlapped series-parallel model can be evaluated from a series-parallel model using the approach shown with an example in Figure 8. When it comes to estimating the overlapped function parallel (OFP) model, SC generation is mapped to the PPE and each function is mapped to a SPE (RF1 Process Input, RF1 Activate, and RF2 Process Input). Since we have 6 SPEs we can execute two overlapped function parallel models in a CELL. Hence the total execution time for an overlapped function parallel (OFP) model was obtained by taking the slowest execution time (namely RF1 Process Input) and multiplying by 2. It should also be noted that the two functions: Global histogram and Global MidVector (Table 3, functions (C) and (E)), take into account the serial computation part, and communication between the serial and parallel parts. This comes into the picture only in a series-parallel model.

6. Performance Comparison

We evaluated the performance of the BDV algorithm on other architectures like FPGA and clusters. A detailed description of mapping BDV on FPGA is presented in [14]. It was shown in [14] that a single Xilinx Virtex4 FPGA provided a 62x performance improvement and 2500x performance-per-watt improvement over a general purpose CPU for the B-U computation. The power dissipation was estimated to be around 1.63W and speedup per $1K was about 6.27.

We also evaluated the performance of the BDV algorithm by mapping the application on PS3 clusters. After eliminating some of the bottlenecks associated with Gigabit Ethernet communication (which requires all PPE cycles to achieve 500mbps) we scaled the implementation to a cluster of three PS3 and achieved a full round trip time of 1.24 seconds on a video frame with 93,267 line triples achieving a speedup over the desktop of 140x. The performance on PS3 cluster is shown in Figure 9. We are also currently evaluating the scalability of the BDV algorithm on large-scale PC cluster to find the potential scalability of the model.

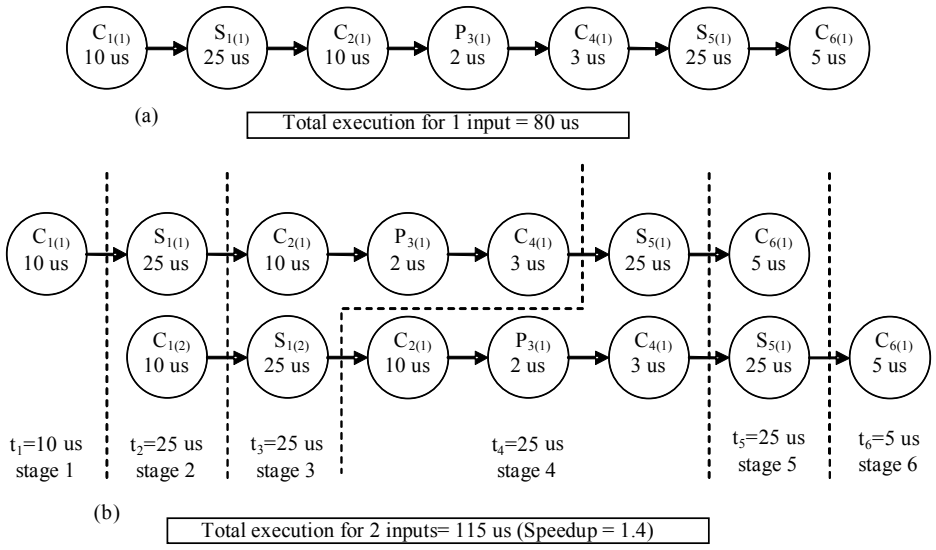

Figure 8: An illustrated example of the performance of (a) overlapped parallel model (b) un-overlapped parallel model. In the overlapped parallel model the SPE starts the computation for the second input data before proceeding to the next stage. $P_{1(2)}$ ($S_{1(2)}$) indicates the execution time for the 2nd iteration of task 1 on PPE (SPE), $C_{1(2)}$ indicate the communication time for data produced by task 1 in its 2nd iteration

Built from base model PS3s, and using two built-in gigabit network interfaces and one PCI gigabit network card, the cost of the cluster hardware was approximately $1500. Thus the speedup of more than 130 times was achieved at a cost similar, or cheaper than a top-of-the-line desktop machine, enabled by the intrinsically parallel nature of the derived brain algorithms.

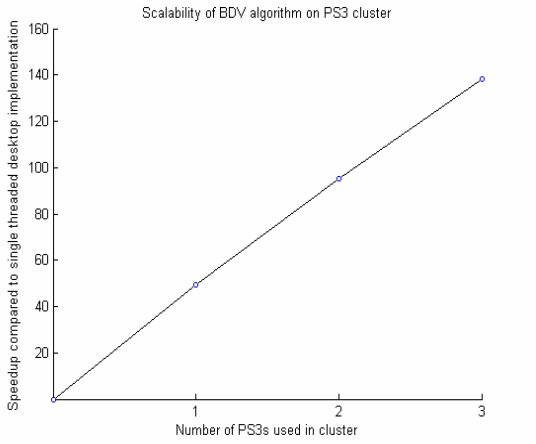

Figure 9: Scalability of BDV algorithm on PS3 Cluster

7. Object Recognition Experimental Results

Much work in computer vision and image retrieval has been performed using pattern recognition theory (e.g. [19]), which treats an image or image sequence as a vector, for which rigorous mathematical frameworks have been developed and on which machine learning algorithms such as traditional classifiers can operate. On the other hand, the models presented here share more in common with part-based image analysis, [3,9] and more specifically those systems that attend to the type of part being observed and the spatial relationships between them [21]. A difficult problem class in computer vision is that of shape based recognition, where the textures of contiguous pieces of an object are not sufficient for traditional algorithms to perform recognition. To address this difficult area of computer vision, we used videos from the Wiry Object Recognition Database [7] and compared our engine with the precision of the only other system reporting results on this dataset; namely the aggregation sensitive version of the cascade edge operator probes (EOP-AS) in [13]. In this dataset, the task is to find a sitting-stool in various cluttered office scenes.

Results of our experiment are given in Figure 10. The BDV model (8,448 RF1 vectors, 1020 RF2 vectors) was trained with first frames of videos 0 and 2 (Room A401 clips 5 and 6) and tested on 30 frames of video from different rooms (Room A408 clip 4 and Room sh201 clips 1 and 3). The activity supporting evidence threshold was set to 98.5%, such that a neuron's observed activity (using the previously described algorithm) was considered supporting evidence if the probability of the expected neuron occurring so active and close to the expected location was less than 1.5% (other thresholds could be selected and their impact on performance is currently under study). Stools were guessed in order of the most unique line segments part of some

supporting evidence. For comparison, results using the EOP-AS [13] were estimated from the reported accuracy on "other room" test sets of 22% (8.8 false positives per image) used as the probability of an arbitrary true positive guess. In Figure 10 the y-axis is the probability of finding the stool within the number of guesses defined by the x-axis. We can see that the neocortical model achieves similar performance to EOP-AS. As the system increases the number of guesses they make per image from 1 to 5, the probability of finding the sitting-stool in the image increases from 20% to 70%

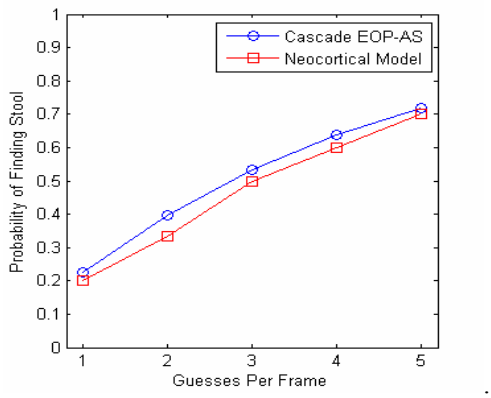

Figure 10: Comparison between our neo-cortical model and Cascade EOP-AS [13] model

8. Conclusion

The results indicate that computational models derived from brain circuitry are not only capable of delivering good performance on hard real-world tasks such as shape-based vision, but also carry with them the capacity for efficient implementation on highly parallel hardware. As the number of processors per CPU (or GPU) continues to increase, algorithms derived from brain circuitry are uniquely situated to take an advantage, resulting in impressive speedups over their single-threaded implementation. Future work includes accelerating the downstream brain regions that receive input from those accelerated here in order to support the real-time speeds necessary to drive interactive robots. These regions "recognize" relationships between objects and drive the object search by communicating Top-Down feedback of semantic context to earlier levels in the hierarchy. To this end, design of computer architecture for efficiently accelerating these and future brain-derived algorithms is also under study.

9. References

1. Gschwind M (2006), Hofstee H.P, et. al, "Synergistic Processing in CELL's Multicore Architecture", *IEEE Computer,* 2006 Volume 2, 10-2

2. Wallis, G., Rolls, E. (1997), "A model of invariant object recognition in the visual system", *Prog. Neurobiology,* 51, 167–194.

3. Wiskott, L. Fellous, J., Kruger, N., Malsburg, C. (1997), "Face recognition by elastic bunch graph matching", *Proc. 7th Intern. Conf. on Computer Analysis of Images and Patterns*, CAIP'97, 456–463

4. Hubel, D., Wiesel, T. (1965), "Receptive fields and functional architecture in two nonstriate visual areas (18 and 19) of the cat", *J. Neurophysiol*, 28, 229–289.

5. Hubel, D. & Wiesel, T. (1962), "Receptive fields, binocular interaction and functional architecture in the cat's visual cortex", *J. Physiol. (Lond.)* 160, 106–154.

6. Bruce, C., Desimone, R., et. al, "Visual properties of neurons in a polysensory area in the superior temporal sulcus of the macaque", in *J. Neurophysiol*., 1981, 46, 369–384.

7. Carmichael, O., Hebert, M, "WORD: Wiry Object Recognition Database", Carnegie Mellon University; *http://www.cs.cmu.edu/~owenc/word.htm*, retrieved December 20, 2006

8. Douglas, R., Martin, K. (2004), "Neuronal circuits of the neocortex", *Annu. Rev. Neurosci*. 27, 419–451.

9. Marr, D., Nishihara, H. (1978), "Representation and recognition of the spatial organization of three-dimensional shapes", *in RoyalP*, volume B-200, pg: 269–294.

10. Olshausen B.A, Field D.F, "Sparse coding of sensory inputs", *Current Opinion in Neurobiology*, 2004.

11. Coultrip, R., Granger, R., and Lynch, G. (1992), "A cortical model of winner-take-all competition via lateral inhibition", *Neural Networks*, 5: 47-54.

12. P. D. Kovesi, " MATLAB and Octave Functions for Computer Vision and Image Processing", School of Computer Science & Software Engineering, The University of Western Australia; Retrieved on December 2006 from *http://www.csse.uwa.edu.au/~pk/research/matlabfns/*.

13. Carmichael, O. (2003),"Discriminative Techniques For The Recognition Of Complex-Shaped Objects", *PhD Thesis*, *The Robotics Institute, Carnegie Mellon University. Technical Report* CMU-RI-TR-03-34

14. Furlong J, Felch A, Moorkanikara Nageswaran J, Dutt N, Nicolau A, Veidenbaum A, Chandreshekar A, Granger R. (2007), "Novel brain-derived algorithms scale linearly with number of processing elements", *Proc. Intl. Conf on Parallel Computing*, (parco.org) 2007

15. Eichenberger, A. O'Brien, et. al, (2005), "Optimizing Compiler for a CELL Processor", *IEEE PACT* 2005

16. de Cock, E. (2000), "YAPI: application modeling for signal processing systems", *DAC*-2000,

17. Lee, E., Parks, T. (1995), "Dataflow Process Networks", in *Proceedings IEEE*, May 1995

18. IBM (2005) CELL BE Programming Handbook and Programming Tutorial

19. Duda, R., Hart, P, Stork, D. (2000) in *Pattern Classification*, John Wiley & Sons, Inc.

20. Rodriguez, A., Whitson, J., Granger, R. (2004), "Derivation and analysis of basic computational operations of thalamocortical circuits", *Journal of Cognitive Neuroscience*, 16: 856-877.

21. Barrow, H., Popplestone, R. (1971), "Relational descriptions in picture processing", in *Machine Intelligence*, pg: 6:377–396.