Accelerating Brain Circuit Simulations of Object Recognition with a Sony PlayStation 3

Jayram Moorkanikara Nageswaran[1], Andrew Felch[2], Ashok Chandrashekar[2], Jeff Furlong[1], Nikil Dutt[1], Alex Nicolau[1], Alex Veidenbaum[1], and Richard Granger[2,*]

[1] University of California, Irvine, CA 92697, USA
E-mail: { jmoorkan, jfurlong, dutt, nicolau, alexv}@ics.uci.edu

[2] Dartmouth College, Hanover, NH 03755, USA
E-mail: {Andrew.Felch, Ashok.Chandrashekar, Richard.Granger}@dartmouth.edu

*Corresponding author: 6207 Moore Hall, Dartmouth College, Hanover NH 03755
Richard.Granger@Dartmouth.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, successfully address issues of known difficulty in visual processing. These algorithms are slow on conventional uni-processor based systems, 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 demonstrates the potential to support the massive parallelism inherent in these algorithms by exploiting the CELL's parallel instruction set and by further extending it to low-cost clusters built using the Sony PlayStation 3 (PS3). The paper describes the modeled brain circuitry, derived algorithms, implementation in the PS3, and initial performance evaluation with respect both to speed and visual object recognition efficacy. The results show that a parallel implementation can achieve about 140x performance improvement on a cluster of 3 PS3 consoles. More importantly, we show that the improvements scale linearly with added processing elements. These results provide a new platform enabling extended investigation of much larger-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 clock speeds (pipelines, branch prediction, cache) and, 2) finding and executing parallelizable 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 CPU's on a single computer chip. The unfortunate drawback is that few applications contain the parallel algorithms necessary to significantly benefit from the additional CPU's.

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 local memory (synapses) and rare connections between them. With such an inferior 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 thus poised to take advantage of parallel hardeware such as multi-core processors.

In this paper we first present the components of a 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). We describe programmed realization of the algorithm on the PS3 CELL multi-core processor and analyze the resulting findings while scaling a small cluster of PS3 from one to three nodes. The overall flow of the paper is pictorially represented below in Figure 1.

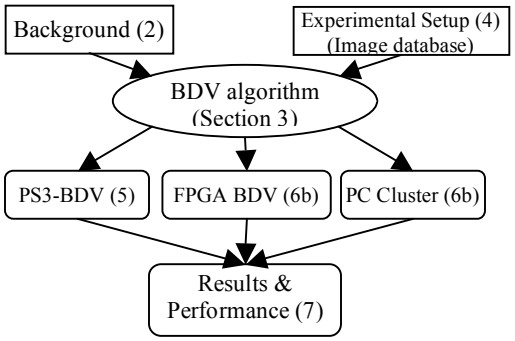

Figure 1: Paper organization. Section number is indicated in bracket

2. Background

Neurons in the eye are activated by light, and send information electrically to the thalamocortical system, which is the focus of our work. Thalamo-cortical circuits, constituting more than 70% of the human brain, are primarily responsible for all sensory processing as well as higher perceptual and cognitive processing. It has long been noted that the brain operates hierarchically [20, 21]: downstream regions receive input from upstream regions and in turn send feedback, forming extensive cortico-cortical loops. Proceeding downstream, neural responsiveness gains increasing complexity in types of shapes and constructs recognized, as well as becoming independent of the exact location or size of the object (translation and scale invariance)[3]. Early visual components have been shown to respond to simple constructs such as spots, lines, and corners [13]; simulations of these capabilities have been well-studied in machine vision. Thus further downstream higher level constructs are selectively activated in response to a cluster of low-level features, thus the whole system is organized into a hierarchy [12, 14, 22]. In our simulations each stage of hierarchy responds to a particular feature which is composed of multiple line segments. In particular we present computation which starts with three line segments ("line triples"). This highly simplified architecture is shown to be very effective on difficult visual applications such as the CMU WORD database [6]. It is hoped that implementations of further downstream areas will extend the work to more abstract perceptual and cognitive processing [17, 9].

In Figure 1, we illustrate the working of a simplified form of the algorithm to detect the character number. At the first level, line segments are detected similar to the early visual areas. The second level is the portion of the model most studied in this work, which involves converting line triples into bit-vectors via a sparse coding mechanism using RF1s and RF2s. Layers 3 and 4 are under preliminary study and were implemented such that viewing a recognizable object allowed the activity of shape detectors in level 2 to be predicted from the activity of other shape detectors in level 2. Each individual grid square represents a shape processor. Only a few are depicted as active in this example, but it could be possible for all of them to be active simultaneously, performing a high degree of parallel processing which is responsible for the fast visual recognition by the humans. Each shape processors in level 2 was modeled as detecting the same set of shapes. A hierarchical organization as depicted in Figure 1 enables sharing of lower layers elements by higher layers, and hence reducing the total memory requirements.

We now explain the key components in the implemented algorithm. Though the algorithm is explained specifically with respect object recognition, the algorithm is generic enough and is being used for other pattern classification applications like speech recognition etc. A given set of inputs activates a particular neuron and this set of inputs is loosely referred as receptive fields (RF) in our paper. Thus each shape processor is receptive to a specific shape only. Any two shapes are as similar as the number of activated neurons shared in their activation patterns. As a result of sparse population codes, 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 to respond to some input features (RFs). 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.

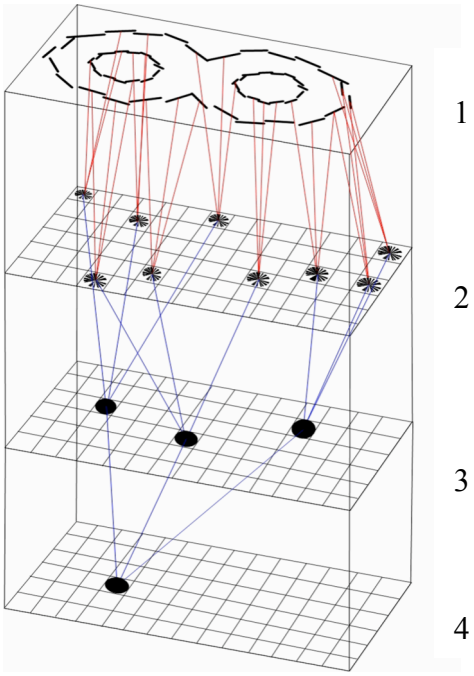

Figure 2: The hierarchical organization of shape detectors operating on the optical character number eight.

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 [6, 17, 8, 9], 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.

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. The algorithm modeled is described in Figure 3.

1. Initialize:
// load RF1 with 8192 entries of size 160 bits
// load RF2 with 1024 entries of size 8192 bits (RAM)
2. Capture Video Frame:
// extract 400 to 600 line segments (32 bits each)
// group 20,000 to 100,000 line triples (30 bits each)
// depending on scene complexity
3. Calculate sparse-code:
// determines the sparse code for the input line segment
// triples
4. RF1 Compute:
// Computes the similarity or match value between the
// RF1 neurons with a input sparse vector by means of
// vector dot-product;
5. K-WTA Compute (K=512):
// computes a threshold so that about 512 RF1 neurons // match value can cross the threshold
// Computes a vector called midVector indicating what
// RF1 neurons are receptive
6. RF2 Compute:
// Computes dot products between the resultant
// midVector and all RF2 neurons;
// Outputs the RF2 index if the population count of the
// dot product is over certain threshold
7. Evaluate Learning:
// input results from RF2 Compute
// analyze highly receptive neurons to determine if
// they should learn
modify RF2 neurons (to be similar to frame data) based on learning
8. RF2 Update:
// if applicable, send updates to RF2 as a result of
// learning

Figure 3: Pseudo-code for the BDV algorithm

Figure 3 depicts the process of converting line-triple representations into detected shapes, with vectors depicted as circles and operations as horizontal lines with a centered symbol. Step 1 converts line segments into a somewhat scale invariant and translation invariant bit-vector representation based on angle of each line with respect to origin. The representation is a sparse encoding, 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. Row 3 depicts the approximately 8,192 vectors of 160 bits (RF1 vectors), each of which was previously and maximally trained on a single input. Row 2 indicates that the dot-product will occur between the input vector and all RF1 vectors. Row 4 is the resulting matches (8,192 match values between 0 and 160 each). Row 5 depicts the application of a threshold for k-WTA operation, with K=512. Row 6 shows the 512-hot of 8,192 bit-vector, which provides input into the next set of 1,000 vectors (RF2 vectors, 1024-2048 hot of 8,192, Row 8, trained on multiple inputs and performing more complex shape detection). Row 7 indicates the dot-product operation between the input in Row 6 with all RF2 vectors. Row 9 indicates the match values, ranging between 0 and

512, indicating how well each RF2 vector matched the input.

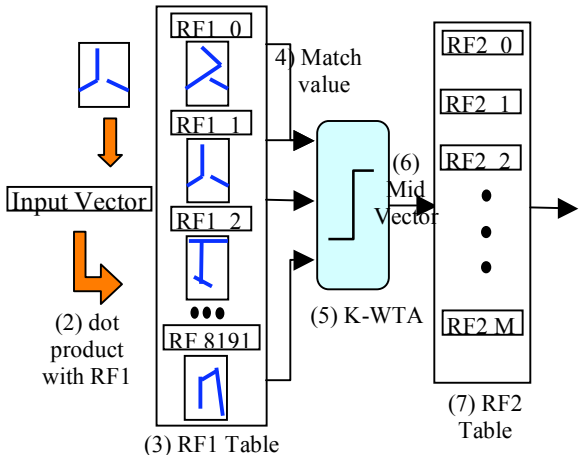

Figure 4: Simplified model of the BU computation

3. Experimental Setup

The Wiry Object Recognition Database (WORD) [6] was 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 where in the videos the barstool is located. Successful identification requires the bounding box to enclose the location of the stool in the video frame, within 25% of the stool's actual area.

Line segments were extracted using the Canny edge detector [16], implemented such that approximately 400-600 line segments were extracted per frame. While other line segment extraction algorithms could be used, and their sensitivities adjusted to extract 100 or 1,000 line segments per frame (and indeed the performance impact of this decision provides a subject for future study), the algorithm and parameters used in this work were chosen for their ease-of-use and the fact that humans could often recognize objects from the extracted line segments alone.

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.

In accordance with the previously discussed physiological studies that found upstream neuron receptive fields (inputs for which a neuron activates) to be simpler than downstream neurons, each of the approximately 8,192 upstream neurons was trained and became maximally receptive to a single input line triple 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. A strict K was not defined for the downstream layer, but rather the downstream layer was made to return the bit-vector dot-product (net activation) of each neuron. This was done so that the effects of various K values could be studied and statistics collected without complete recomputation of dot-products in the upstream and downstream layer.

Because the receptive fields modeled here are somewhat translation sensitive, the modules just described are replicated many times across the field-of-view and only one module activates for shapes at a particular location. The same set of shape detectors were used in each replicated module, decreasing memory requirements of the implementation while enabling the model to process many shapes in parallel, similar to the brain.

To enable recognition, an early prototype of further downstream regions with bottom-up and top-down processes 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 an additional two-level hierarchy 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 [5] for comparison.

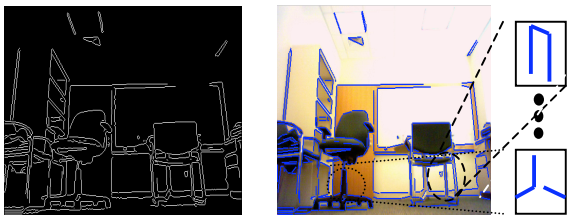

Figure 5: An example for line segmentation and line triple extraction from given frame

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 due to 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 high degree, limited only by the communication overhead. The algorithm also exhibits large number of bit-level parallelism and SIMD parallelism. For example in the RF2 compute stage we need to evaluate an 8192-bit dot-product and population count on the result to estimate the degree of match between two vectors. This can be concurrently executed either at 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 architecture 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 even 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 [23].

(2) IBM Cell / Play Station 3 (PS3):
IBM CELL 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 3.5GHz. A detailed description of the chip is present in [11]. The recently released Sony Play Station3 (PS3) is powered by the IBM CELL processor and is available at about $500. The PS3 console offers a programmable PowerPC processor with 6 SPEs. It has a 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 associated with efficient parallelization and optimization to exploit the capabilities of the architecture.

(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 impact of communication overhead on overall performance and large system cost.

(4) Programmable Graphic Processing Units (GPU): Recent GPUs such as NVidia CUDA and AMD 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 the best 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 BDV to IBM CELL / Play Station 3 (PS3)

We now describe in detail the programming methodology and trade-offs involved in mapping the BDV algorithm to the CELL architecture. Different levels of parallelization can be acheived on CELL, namely: (1) parallel execution of BDV on multiple CELL's (using a cluster of PS3s), (2) multiple programmable units executing simultaneously (namely 6 SPEs and 1 PPE in PS3) on a single node. (3) Concurrent computation and communication (DMA operation) by the SPE. (4) Instruction level parallelization in the SPE with two instructions executing simultaneously, and (5) 128-bit data path allows SIMD parallelization.

The IBM CELL has compiler support for parallel programming across its SPE and PPE [10]. Application programmers can use OpenMP programming directives to describe parts of the code which can execute in parallel. This compiler based approach facilitates faster

mapping of applications on CELL. However for this paper we did not use the user-directed compiler approach for parallelization for several reasons: the restricted nature of the type of parallelization done by the IBM XL compiler does not allow parallelization and pipelining to be used in the application code (will be explained further in this section), and also the available compiler is a prototype and is still under heavy development and not widely released.

We describe a systematic methodology and experiments which were performed to exploit these levels of parallelization on the CELL for our brain derived application. Our approach is sufficiently generic to be employed for mapping other kinds of application models to the CELL.

Mapping on CELL Clusters

We use a client-server architecture consisting of a cluster of three PS3s controlled by a powerful desktop PC, since this fits well with our application computation requirements. The cluster consists of PS3s serving as engines for the bottom-up computations described earlier. The client consisted of the desktop PC controlling the top-down part of the activity flow. The workflow consists of the desktop server sending line triples as input to the bottom-up engine, with each PS3 sending the computed results back over the network. The client uses the results obtained to prune the top-down search and evaluate higher level of constructs present in the image to arrive at confidence values for object recognition at different locations in the image.

Parallelization within a PS3 node

In this section we describe a systematic approach for examining the various modes of parallelization and mapping of the application onto the CELL processor. A simple functional model of the CELL is given in Figure 6 (See the pseudocode in **Error! Reference source not found.** for the detailed description of each stage).

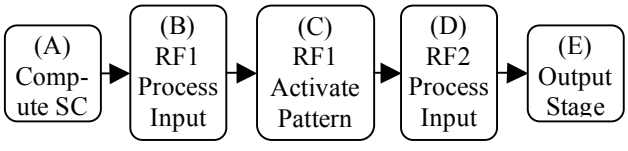

Figure 6: Functional Model of the Bottom-Up Engine

The bottom-up (BU) 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 7. Approximately 1.89ms was required to execute the BU computation for a single line segment triple. This corresponds to a BU computation throughput of about 526 line segment triples per second (526 LST/sec). In Figure 7 we can observe that RF1 Vector computations and RF2 Vector computations take more that 95 % of the overall execution time. These are the critical functions which need to be optimized and parallelized to increase the throughput of the BU computation. From these results, the runtime of processing an entire video frame with 30,000 line-triples can be estimated at approximately one minute, crucially limiting the experiments that can be run on the system and restricting study of the downstream brain areas. To enable experiments using interactive robots the recognition time of humans would need to be approached (approximately 500ms, thus requiring a speedup of about 120x).

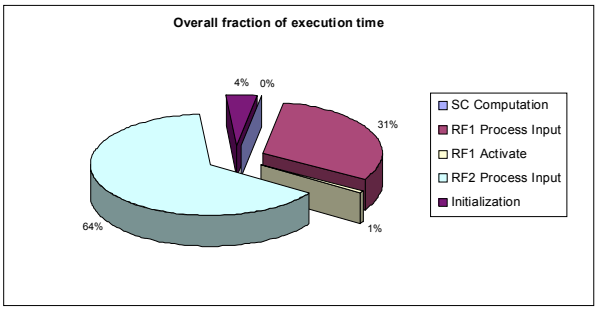

Figure 7: Performance comparison between different function blocks of the bottom-up engine on an Intel Core E6400 running at 2.13GHz

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 in size 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, overlaying and resident partition management [10] 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.

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 classify the data size and data sets based on their usage into static/fixed and partitionable. If a particular function or loop is parallelized to run across different SPE, 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 SPE (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.

TABLE 1: Analysis of data structures used in BDV application

Main data structures	Data Size	Data usage	Data Access Pattern	Accessing Tasks (Figure 6)
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)

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, etc. [10], can be used depending upon the data access pattern. For our application we have around four large data sets (Table 1) namely RF1 Table, RF2 Table, Sparse Code (SC) Ordering Table and Angle Table. The SC Ordering and Angle tables are used to avoid the costly arctangent functions when computing the angles for input bit-vector generation from the original three line segments. 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 bottom-up engine. The possible models are: a fully overlapped functional parallel model, data parallel model, un-overlapped series-parallel model, or overlapped series-parallel model. An overlapped model is one in which communication and computation can happen concurrently, so the waiting time associated with communication can be mitigated. In a fully overlapped functional parallel 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 [8] or data flow networks [15] 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 series-parallel model. 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 (similar to the SIMD paradigm). If the data and computation is evenly partitioned across these loop boundaries, this model exhibits good load balancing, speedup, and reduced SPE LS requirements. A version of this model for the BU engine is illustrated in Figure 8. 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 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. An overlapped series-parallel model overcomes the waiting time problem encountered in an un-overlapped series-parallel 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 keeps the data 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 model can be employed. In this type there is virtually no communication between the SPEs. Each SPE can be treated as a full processor and complete bottom-up computation for a line segment triple 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 in each SPE or should be otherwise 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.

Along with the higher level parallel models, the performance improving programming optimizations that we implemented on CELL [14] are as follows:

- 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 0.1us (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.
- After the above optimization and using special 128-bit SIMD instructions such as *absdiff, abs_sumb, spu_add, spu_cntb*, See [14], 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 SIMD instruction set and SPE instruction fine-tuning impressive speedups are possible. It should be noted that no special optimizations using SSE2 or MMX instructions were carried out on the desktop PC running Intel Core.

We implemented the unoverlapped series-parallel model on the CELL with various optimizations described above. From the performance of this model we could estimate the performance of other parallel models (results are shown in Table 3). These estimations are a very useful approach 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 assumes 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 an unoverlapped series-parallel model using the approach shown with an example in Figure 9. When it comes to estimating the overlapped function parallel 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 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 RF2 pattern, take into account the serial computation part, and communication between the serial and parallel parts. This comes into the picture only in an un-overlapped series-parallel model. On a single CELL present in PS3 we measured a speedup of 54 times using the unoverlapped series-parallel model compared to serial execution on a desktop CPU (E6400).

6. Object Recognition Experimental Results

Much work in computer vision and image retrieval has been performed using pattern recognition theory (e.g [4]), 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, [17,21] and more specifically those systems that attend to the type of part being observed and the spatial relationships between them [2,22]. 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 [6] 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 [5]. In this dataset, the task is to find a sitting-stool in various cluttered office scenes. The results are shown in Figure 10. We can see that the neocortical model achieves similar performance to EOP-AS. As the systems increase how many guesses they make per image from 1 to 5, the probability of finding the sitting-stool in the image increases from 20% to 70%.

Additional Performance Results

After eliminating bottlenecks associated with GigE communication (which requires PPE throughput to achieve 500mbps), we scaled the implementation to a cluster of three PS3 and achieved a full round trip time of 1.25 seconds on a video frame with 93,267 line triples achieving a speedup over the desktop of 138x (Figure 8). This is a scaling efficiency of approximately 94%. Investigation into the small performance drop discovered that TCP/IP buffer overheads were to blame, and that state-of-the-art inifiniband technology would likely achieve greater than 99%.

Built from off-the-shelf PS3s, and using two built-in gigabit network interfaces and one PCI-E gigabit network card, the cost of the cluster hardware was approximately $1500. Thus the speedup of about 140x was achieved at a cost similar, or lesser than a top-of-the-line desktop machine, enabled by the intrinsically parallel nature of the brain dervived algorithms.

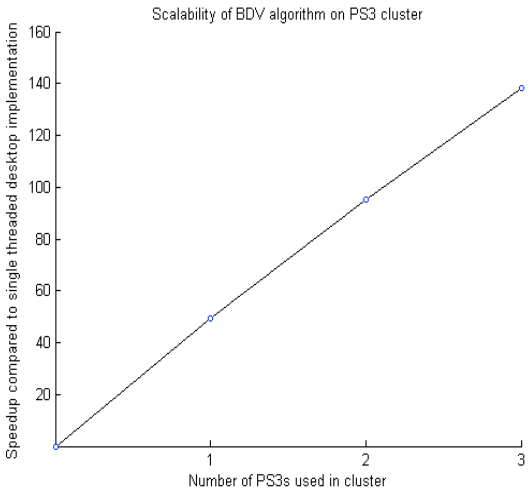

Figure 8: Scalability of BDV algorithm on PS3 cluster.

7. Discussion and 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.

Acknowledgments: The research described herein was supported in part by the Office of Naval Research and the Defense Advanced Research Projects Agency.

8. References

1. Ambros-Ingerson, J., Granger, R., Lynch, G. (1990). Simulation of paleocortex performs hierarchical clustering. Science, 247: 1344-1348.

2. Barrow, H., Popplestone, R. (1971) Relational descriptions in picture processing. In Machine Intelligence, 6:377–396.

3. Bruce, C., Desimone, R., Gross, C. (1981). Visual properties of neurons in a polysensory area in the superior temporal sulcus of the macaque. J. Neurophysiol. 46, 369–384.

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

5. 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

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

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

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

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

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

11. Gschwind M (2006), Hofstee H.P, et. al, "Synergistic Processing in CELL's Multicore Architecture", IEEE Computer 2006, 10-24

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

13. 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.
14. IBM (2005) CELL BE Programming Handbook and Programming Tutorial
15. Lee, E., Parks, T. (1995) Dataflow Process Networks, Proceedings of the IEEE, May 1995
16. P. D. Kovesi. MATLAB and Octave Functions for Computer Vision and Image Processing. School of Computer Science & Software Engineering, The University of Western Australia; http://www.csse.uwa.edu.au/~pk/research/matlabfns/ retrieved December 2006.
17. Marr, D., Nishihara, H. (1978). Representation and recognition of the spatial organization of three-dimensional shapes. In Proceedings of the Royal Society, B-200, 269–294.
18. Perrett, D., Oram, M. (1993). Neurophysiology of shape processing. Imaging Vis. Comput. 11, 317–333.
19. Rodriguez, A., Whitson, J., Granger, R. (2004). Derivation and analysis of basic computational operations of thalamocortical circuits. J Cognitive Neuroscience, 16: 856-877.
20. Wallis, G., Rolls, E. (1997). A model of invariant object recognition in the visual system. Prog. Neurobiol. 51, 167–194.
21. Wiskott, L. Fellous, J., Kr¨uger, N., Malsburg, C. (1997). Face recognition by elastic bunch graph matching. In G Sommer, K Daniilidis, J Pauli (Eds), Proc. 7th Intern. Conf. on Computer Analysis of Images and Patterns, CAIP'97, Kiel, 1296: 456–463, Heidelberg. Springer-Verlag.
22. Zhang, D.-Q. (2005). Statistical Part-Based Models: Theory and Applications in Image Similarity, Object Detection and Region Labeling. PhD Thesis, Columbia University
23. Furlong J, Felch A, 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.

TABLE 2: Comparison for different parallel models on CELL

Model name	Application Latency	Bandwidth usage	Memory usage	Modeling effort	Performance
Full overlapped parallel mode	High	Low	Low	Low	Low-Medium
Unoverlapped 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 9

Function name	**Serial model (us)**	**Unoverlapped series-parallel model (us)**	**Overlapped Data-parallel model (us)**	**Overlapped functional parallel model (us)**	**stage**	**Overlapped series-parallel model (us)**
SC	0.89	0.80	0.80	0.80	1	0.80
RF1 Process Input	603.53	13.78	82.66	82.66	2	13.78
Partial to Global Histogram	0.00	2.71	0.00	0.00	3	13.78
Generate Partial MidVector	13.34	2.48	3.87	3.87	4	2.71
Partial to Global MidVector	0.00	1.91	0.00	0.00	5	2.48
RF2 Process Input	1276.68	12.91	77.23	77.23	6	12.91
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

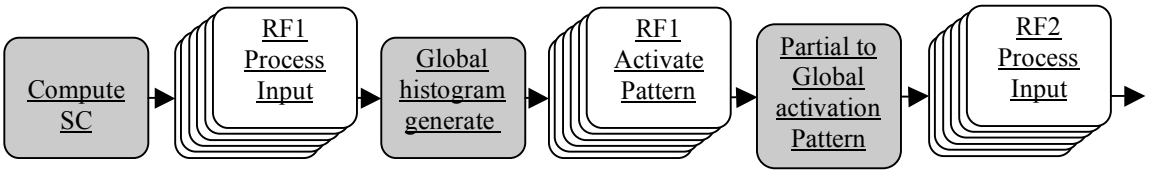

Figure 8: Parallel Model for Brain-derived Vision algorithm

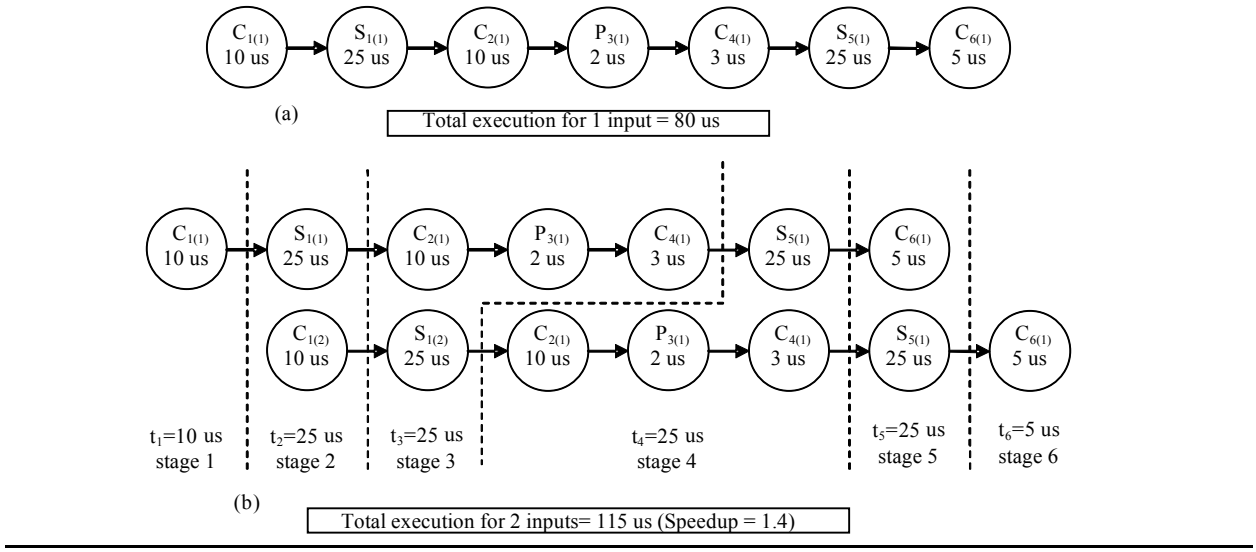

Figure 9: An illustrated example of the performance of (a) overlapped parallel model (b) un-overlapped parallel model. In the overlapped parallel model the SPE starts the computation for the second input data before proceeding to the next stage. $P_{1(2)}$ ($S_{1(2)}$) indicates the execution time for 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

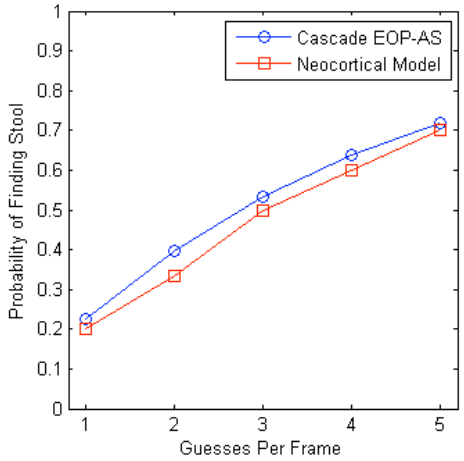

Figure 10: The neocortical model was trained with first frames of videos 0 and 2 (Room A401 clips 5 and 6) and tested on 30 frames of video from different rooms (Room A408 clip 4 and Room sh201 clips 1 and 3). The activity supporting evidence threshold was set to 98.55%, such that a neuron's observed activity (using the previously described algorithm) was considering to be supporting evidence if the probability of the expected neuron occuring so active and close to the expected location was less than 1.45% (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 aggregation sensitive cascade edge operator probes of Carmichael 2003 were estimated from the reported accuracy on "other room" test sets of 22% (at a false positive rate of 8.8 per image) used as the probability of an arbitrary true positive guess. The x-axis is the probability of finding the stool within the number of guesses defined by the y-axis. Location guesses were judged correct if the bounding box had < 25% area difference from the true bounding box. 8,448 RF1 vectors and 1020 RF2 vectors were used