2014 International Joint Conference on Neural Networks (IJCNN)
July 6-11, 2014, Beijing, China

Accelerating Pattern Matching in Neuromorphic Text Recognition System Using Intel Xeon Phi Coprocessor

Khadeer Ahmed, Qinru Qiu, Parth Malani, Mangesh Tamhankar

Abstract— **Neuromorphic computing systems refer to the computing architecture inspired by the working mechanism of human brains. The rapidly reducing cost and increasing performance of state-of-the-art computing hardware allows large-scale implementation of machine intelligence models with neuromorphic architectures and opens the opportunity for new applications. One such computing hardware is Intel Xeon Phi coprocessor, which delivers over a TeraFLOP of computing power with 61 integrated processing cores. How to efficiently harness such computing power to achieve real time decision and cognition is one of the key design considerations. This paper presents an optimized implementation of Brain-State-in-a-Box (BSB) neural network model on the Xeon Phi coprocessor for pattern matching in the context of intelligent text recognition of noisy document images. From a scalability standpoint on a High Performance Computing (HPC) platform we show that efficient workload partitioning and resource management can double the performance of this many-core architecture for neuromorphic applications.**

I. Introduction

Cognitive computing is an emerging field made possible due to the advancement in High Performance Computing (HPC) domain. There is a special interest in cognitive computing because the challenges which are being faced today have no definite way of deriving a solution and hence cannot be coded in the traditional style. There are many examples which fall under this category, including natural language understanding, text image recognition, autonomous unmanned vehicle controls, user preference suggestion, etc.

Neuromorphic computing systems refer to the computing architecture inspired by the working mechanism and massive parallel structure of human brains. A neuromorphic information processing model is presented in [1]. The model consists of a simple but massive parallel pattern matching layer that generates fuzzy results retaining rich information (sometimes referred as ambiguity) and a powerful information inference layer that removes the ambiguity by statistical inference. The event-driven computation in neuromorphic engines loosely represents the integrate-and-fire behavior of neurons, leading to high computation efficiency. The hierarchical architecture mimics the primary sensory cortex and association cortex in brain's sensory processing area [2][3][4]. The model is applied for intelligent text recognition in document image processing, where meaningful sentences are extracted from camera captured documents and road signs. These are non-trivial problems as the images are captured with perspective distortion, angular distortion, warping or other general noises. Reliable performance is achieved even when the system is exposed to new experiences. As their experience gets richer and richer for every new exposure, their performance improves.

The state-of-the-art multi-core computer architecture enables large-scale implementation of neuromorphic models. One of such computer system is Intel's Xeon Phi coprocessor, where more than 61 X86 compatible cores with 4-way multi-threading capability are integrated on a single die. Each core has its own L1 and L2 cache and can access coherent L2 caches of any other core [5]. This architecture delivers over a TeraFLOP computing bandwidth. How to harness such computing power to achieve real-time cognition and decision is an urgent research problem.

In this work, we accelerate the performance of the neuromorphic model for *Intelligent Text Recognition System* (ITRS) using the hybrid Xeon - Xeon Phi coprocessor setup. According to Amdahl's law, the most effort in performance optimization should be spent on the most common operation, which is identified to be pattern matching in ITRS. Pattern matching forms the bottom layer of ITRS. It is implemented using an auto-associative memory model called *Brain-State-in-a-Box* (*BSB*). BSB is a simple nonlinear, energy minimizing neural network [6][7][8], whose convergence speed is proportional to the similarity between the input image and the stored pattern. The bottom layer of the ITRS consists of large number of independent BSB models, which are ideal candidates for parallel implementation.

A scalable platform capable of handling images with different resolutions for higher fidelity pattern matching is developed on Intel's Xeon Phi coprocessor. One of the reasons to select Xeon Phi over GPGPU is that the applications can run natively on Xeon Phi compared to the offload model of GPGPU. This is particularly an important capability as it frees up the CPU to handle more control intensive functions of ITRS [1].

The rest of the paper is organized as follows: Section 2 provides insight into related work, section 3 provides a background of the entire system, section 4 provides the implementation setup details, section 5 presents the optimization work, section 6 presents the results, section 7 has performance analysis of the implementation and finally section 8 gives the conclusions and future works.

II. Related Work

The first stage of a text recognition system is image

Khadeer Ahmed, Qinru Qiu are with Syracuse University, Syracuse NY 13244 USA e-mail: {khahmed, qiqiu}@syr.edu.

Parth Malani, Mangesh Tamhankar are with Intel Corporation, 2200 Mission College Blvd. Santa Clara, CA 95054 USA e-mail: {parth.malani, mangesh.tamhankar}@intel.com

978-1-4799-1484-5/14/$31.00 ©2014 IEEE

processing, which feeds the Optical Character Recognition (OCR) modules. A wide variety of OCRs for printed text recognition were developed over the years [9]. There is a heavy focus on improving their accuracy by employing various image processing and classifying techniques including neural network based learning techniques [10][11][12]. In ITRS, the intelligence for recognition and error correction has been pushed to upper layers, where word and sentence contexts are considered. Hence it affords a simple fuzzy pattern-matching layer, which generates more than one likely matches for a given pattern. This is also why performance optimization rather than accuracy optimization is focused in this work.

There are prior attempts to implement the pattern matching layer of ITRS on IBM Cell processors [13][14]. However, the limited size of on-chip memory of Cell processor prevents us from processing higher resolution images in reasonable time. Attempts have been made to have a hardware solution for processing these BSB neural networks using memristor crossbar arrays [15]. However, no real hardware has been implemented yet. GPGPU based optimizations have been implemented for several similar neural network models [16][17][18]. They need special attention from the CPU to move computation data from main memory to GPU memory. Furthermore, combining CUDA with Message Passing Interface (MPI) is not an ideal option [19].

Efficient use of Single Instruction Multiple Data (SIMD) architecture is the key to achieving high performance with Xeon Phi coprocessor. Several SIMD vectorization techniques are proposed in [20]. A comprehensive guide on optimization for the coprocessor is given by [21][22]. Work on optimizing a data intensive application running natively on the coprocessor is presented in [23]. Its authors have tried to reduce inter-thread dependency to minimize thread syncing overhead. Efficient parallelization of batch pattern training algorithm for the coprocessor and other HPC platforms is proposed in [24]. It uses MPI to perform communication and reduction operations at the same time.

III. BACKGROUND

In this section, the computation model and software architecture of ITRS are presented, followed by a brief introduction of hardware architecture of Xeon Phi coprocessor.

A. Brain-state-in-a-box model

BSB model is a simple, auto-associative, nonlinear, energy minimizing neural network [6][7][8]. A common application of the BSB model is to recognize a pattern from the given noisy input. It can also be used as a pattern recognizer that employs a smooth nearness measure and generates smooth decision boundaries.

There are two main operations in a BSB model, Training and Recall. Here the focus is on BSB recall operation. The mathematical model of a BSB recall operation can be represented in the following form:

$$x(t+1) = S(\alpha * A * x(t) + \lambda * x(t) + \gamma * x(0)) \qquad (1)$$

Where:

- x is an N dimensional real vector
- A is an NxN connection matrix
- $A * x\ (t)$ is a matrix-vector multiplication operation
- α is a scalar constant feedback factor
- λ is an inhibition decay constant
- γ is a nonzero constant if there is a need to maintain the input stimulation
- $S\ ()$ is the "squash" function defined as follows:

$$S(y) = \begin{cases} 1 & if \quad y > 1 \\ y & if \quad -1 \le y \le 1 \\ -1 & if \quad y < 1 \end{cases} \qquad (2)$$

Note that in our implementation, we choose λ to be 1.0 and γ to be 0.0. But they can be easily changed to other values. Given any input pattern $x(0)$, the recall process executes equation (1) iteratively to reach convergence. A recall converges when all entries of $x(t+1)$ are either "1.0" or "-1.0".

The BSB model is selected in the ITRS for two reasons. First, it is simple to operate compared to other complex neural network models [4]. Although it has lower accuracy, the error can be corrected by the upper layer information processing. Second, its convergence roughly indicates the similarity between the input and the stored pattern. It is pointed out by [4] that the average convergence time of the BSB model increases as the input goes further away from the attractor. Such property enables the racing behavior in character recognition, which is discussed in confabulation sub section.

B. Cogent confabulation

Cogent confabulation is a connection-based cognitive computing model. It captures correlations between objects (or features) at the symbolic level and stores this information as a knowledge base [1]. Given an observation, familiar information with high relevancy will be recalled from the knowledge base.

Based on the theory, the cognitive information process consists of two steps: learning and recall. During learning, the knowledge links are established and strengthened as symbols are co-activated. During recall, a neuron receives excitations from other activated neurons. A "winner-takes-all" strategy takes place within each lexicon. Only the neurons (in a lexicon) that represent the winning symbol will be activated and the winner neurons will activate other neurons through knowledge links. At the same time, those neurons that did not win in this procedure will be suppressed.

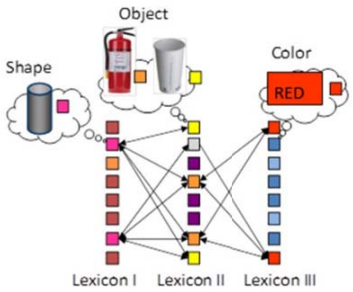

Fig. 1. A simple example of lexicons, symbols, and knowledge links

Fig. 1 shows an example of lexicons, symbols, and know-

ledge links. The three columns in Fig. 1 represent three lexicons for the concept of shape, object, and color with each box representing a neuron. Different combinations of neurons represent different symbols. For example, the pink neurons in lexicon I represent the cylinder shape, the orange and yellow neurons in lexicon II represent a fire extinguisher and a cup, while the red neurons in lexicon III represent the red color. When a cylinder shaped object is perceived, the neurons that represent the concepts "fire extinguisher" and "cup" will be excited. However, if a cylinder shape and a red color are both perceived, the neurons associated with "fire extinguisher" receive more excitation and become activated while the neurons associated with the concept "cup" will be suppressed. At the same time, the neurons associated with "fire extinguisher" will further excite the neurons associated with its corresponding shape and color and eventually make those symbols stand out from other symbols in lexicons I and III.

C. *Software Architecture of ITRS*

ITRS is developed to extract meaningful sentences from document images. It has two information processing layers, a BSB model based pattern-matching layer and a confabulation model based statistical inference layer. The salient feature of ITRS is that it provides contextually correct sentence reconstruction even if there are illegible characters or words in the document image. This is enabled by a trained knowledge base, which captures the statistical information among building components in English language, from letters and words to phrases and part-of-speech tagging.

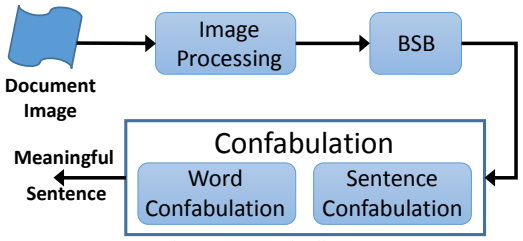

Fig. 2. ITRS Pipeline

The information processing in ITRS has several stages, which can be arranged as a pipeline shown in Fig. 2. After simple image processing which corrects image distortion, skew and warping, the character images are segmented from document image and forwarded to BSB, where fuzzy pattern matching is performed. The pattern matching results will be processed by word level and sentence confabulation for inference based error correction and association.

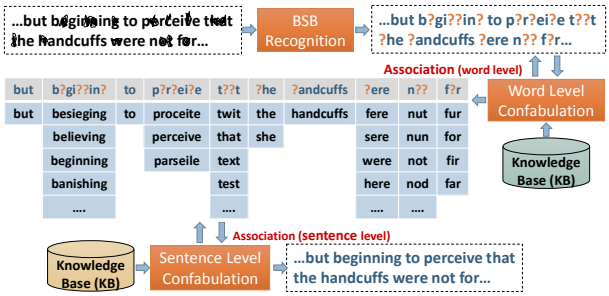

Fig. 3. ITRS cognitive model

A working example of ITRS is shown in Fig. 3. Given a noisy document image, the BSB provides best effort pattern matching for each character images. Each question mark in the figure represents all 26 possible alphabets. The word confabulation layer forms all possible words based on these matching alphabets and the sentence confabulation layer selects the words that forms the most meaningful sentence. It is easy to see that, for each sentence, one sentence confabulation task and multiple word confabulation tasks must be executed, along with even more number of BSB pattern matching tasks. Information is passed across layers. The computation tasks in the same layer are independent to each other and hence can be implemented in parallel.

The pattern-matching layer (BSB) is trained on clean font image. The word level confabulation is trained by reading a dictionary. And the size of word level knowledge base is about 200 MB. The sentence level knowledge base is trained by reading multiple classic literatures. The size of this knowledge base is 6 - 12 GB.

D. *Intel Xeon Phi Coprocessor*

Intel Xeon Phi coprocessor is based on Intel Many Integrated Core (MIC) Architecture. It has more than 50 modified x86 cores integrated on a single die [5]. The X86 compatible architecture allows many existing software to be supported with very few modifications, while the multi-core architecture provides huge performance boost. The 7110P Intel Xeon Phi coprocessor used in our experiments has 61 cores with 8GB RAM, each core runs at 1.1GHz and supports 4 hardware threads.

The cores on Intel Xeon Phi coprocessor are simplified in-order execution cores, hence they are small and power efficient. Huge performance boost is obtained by exploiting the massive parallelism and by hiding memory latency. This architecture is a very good example of MIMD (multiple instruction, multiple data). It supports a total of 244 threads running independent of each other. Thus fine grained parallelism and good load balancing can be achieved.

There is a 512-bit wide vector processing unit (VPU) on every core providing Single Instruction Multiple Data (SIMD) capable instruction set. The VPU can execute 16 single-precision or 8 double-precision operations per cycle. It also supports Fused Multiply-Add (FMA) instructions and hence can execute 32 SP or 16 DP floating-point operations per cycle. Vectorization can be assisted by Intel Compiler or by manually inserting intrinsics and language extensions.

Every core on Xeon Phi has a local L1 and L2 cache. The L2 cache is coherent and its capacity is 512KB. Cache coherence is achieved through a ring interconnect which is 64 bytes wide. Hence each core has the access to a total of 30MB shared L2 cache.

IV. Implementation Flow

The hardware system used for this project has a Xeon based host processor and the 7110P Xeon Phi coprocessor. We started with a sequential implementation of the ITRS software, followed by 3 development phases. The first phase is to restructure the ITRS software to support multiple parallel BSB threads, and thus to parallelize the pattern matching layer. We first implemented the BSB threads and the rest of

the ITRS software on a standalone CPU. Each BSB can handle 2 character image sizes, with 15x15 or 30x30 pixels per character image. They correspond to 256 bytes wide and 1024 bytes wide input vectors respectively. This is a crucial step as both CPU and the coprocessor are based on similar x86 architecture, hence initial code development and testing becomes smoother. Once the initial code is tested on CPU, it can also be compiled and run on Xeon Phi.

Naturally the next phase is to move the BSB code to Xeon Phi coprocessor, and port the image processing and confabulation models on the host Xeon processor of our system. Message Passing Interface (MPI) is used for data communication between confabulation and BSB. In this way we separated the pattern-matching layer from the rest of the ITRS and allocate more computing resources to it. Image processing module groups the character images in to workloads and issues them to BSB through MPI. Each workload consists of 96 characters, which is an input to the BSB module. The size is chosen to provide a good balance between communication and computation time. BSB compares each image in the workload against 93 trained patterns. The pattern set consists of lower case and upper case English alphabets, numbers and some common symbols & punctuations. The output is the set of matching patterns, which are called letter candidates, and their convergence speed. These results are used by the confabulation module to generate meaningful sentences.

The third and final phase is to optimize the BSB code for Xeon Phi coprocessor. The methods utilized and the avenues explored for optimization are described in the next section.

V. Optimization

Two progressive steps are taken to optimize the pattern-matching layer to exploit the resource on Xeon Phi. The first step is to restructure the software for efficient resource management and workload balance. The second step is to tweak compiler options to investigate different auto optimization/vectorization techniques and performance benefits.

The core computation of BSB model is matrix vector multiplication as shown in equation (1). This is repeated for a maximum of 50 iterations or until the results converge whenever an input image is compared against a stored pattern. It is worth to note that the Intel Math Kernel Library (MKL) provides an optimized parallel implementation for matrix-vector multiplication, where the original matrix and vector are segmented and loaded to different cores for distributed processing. The results will be merged at the end. However, the performance of such fine-grained workload partition and parallelization is severely limited by Amdahl's law. Unless the size of the matrix and vector is sufficiently large, the performance gain from parallel computing is not enough to offset the overhead of communication and synchronization [25]. Furthermore, to get best performance, MKL allocates the maximum resources i.e. all the cores, for one matrix-vector operation. Hence we have to serialize the bottom layer of ITRS and run the pattern-matching tasks one by one. It was observed that such globally serial and locally parallel (GSLP) implementation is not efficient for ITRS.

In contrast to GSLP, we adopt a globally parallel and locally serial approach. OpenMP threads and pthreads are created and distributed across Xeon Phi to handle multiple pattern-matching tasks independently. They are referred as *solver threads*. All threads run in parallel. Their synchronization is handled by thread safe blocking queues, which have critical sections defined for accessing the queue and blocks the thread if the queue is empty. This allows the architecture to be inherently load balancing as each computation thread can pick up workload from the queues whenever it finishes processing the current task. The compute threads have data exchange only with the thread-safe queue. The communication with the rest of the ITRS system, which runs on the host CPU, is handled by another thread. By decoupling the compute thread from MPI communication, we keep them busy for maximum duration.

A. Software architecture optimizations

The first step of optimization is to the find the best software architecture for efficient resource management and workload balancing. For any combination of input image and stored pattern for comparison, a pattern-matching task is created. The set of pattern-matching tasks for all of the 96 input images forms the workload. Based on how the workload is partitioned and distributed, three different resource management schemes are tested and their performance is compared.

Multiple comparison patterns to multiple OpenMP threads (MPMT): In this architecture 244 solver threads are created using OpenMP. This is the number of logical cores available on the coprocessor card. Any pattern-matching job can be assigned to any of the available solver threads. Fig. 4 shows how the workload is created and assigned.

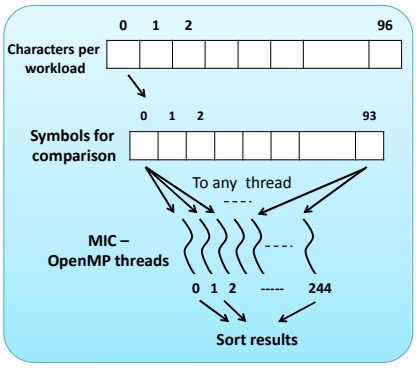

Fig. 4. MPMT architecture

Because the threads can work on any available job at any time, this approach has excellent load balancing ability. However, the BSB models for different patterns have different weight matrices. Due to the limited cache size, every time a new pattern-matching job is started, a new weight matrix of the BSB model (corresponding to the pattern to be compared) must be shuttled/read into the target core's local cache. When 244 threads running simultaneously, large amount of data transfer is created, which causes bus contention. Explicit data management to preserve data locality can improve the performance significantly. Using this resource management scheme, it takes 18.41 seconds to process 96 input images with 30x30 resolution. The performance analysis from VTune Amplifier confirms that there were huge

memory stalls in the compute section of the solver thread. This indicates that the performance of the pattern-matching layer is bounded by memory performance.

Specific comparison pattern to specific pthread (SPST): To get finer control over data, thread creation and destruction, pthreads are used instead of OpenMP threads as shown in Fig. 5. A set of 93 pthreads are created during initialization and destroyed only when the ITRS terminates. There are 3 other threads, which take care of MPI communication and cleanup. We also divide the entire workload into 93 sets. Each set of workload contains pattern-matching tasks between all input images and one of the 93 stored patterns. A specific set of workload is assigned to a specific pthread. Because each pthread always compares the input with the same stored pattern, the weight matrix of the corresponding BSB model can stay in the cache. Therefore, the data contention problem in MPMT is relieved.

Since the solver threads are alive for the entire duration of the program and each thread works on a specific symbol, there is better utilization of cache. Compared to MPMT, the work distribution of SPST may not be uniform but the performance gained due to efficient memory utilization outperforms load balancing overhead

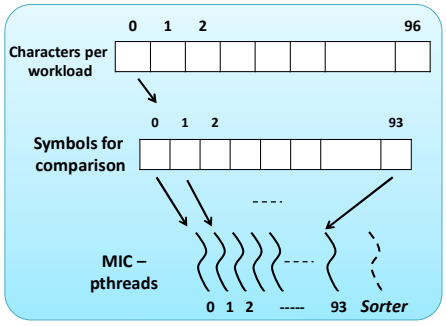

Fig. 5. SPST architecture

Performance can also be tweaked by changing the thread affinity. Specifying affinity of a particular thread makes it run on the specified logical core. Three affinity settings were tried out as described below:

Compact affinity: Each thread was assigned to adjacent core. The run time was 18.5 seconds. This is not very ideal setting as every physical core (and its local cache) is shared by 4 pthreads.

Affinity for alternate logical cores: In this case each physical core will run no more than two threads. This is an improvement over the previous case because only 2 threads share a physical core and the cache. The runtime was 11.3 seconds.

Scatter affinity: By default the micro OS running on the coprocessor scatters the threads among the 244 logical cores. This is the best configuration, as it tries to minimize cache sharing among threads. The runtime in this case was 10.37 seconds.

Performance analysis by VTune Amplifier shows that the optimized solver code was not getting steady stream of data due to sustained memory bandwidth limitations and smaller local cache size for the required data. We were able to achieve 176.58 GB/s memory bandwidth utilization.

Specific pattern to specific pair of pthreads (SP2T): The architecture in Fig. 6 was developed to improve weight matrix data retention in cache. It is similar to SPST, however, each weight matrix was split into halves and distributed to a solver thread and its companion thread. While issuing characters for comparison a duplicate copy was also issued to the companion thread. Hence one workload now needs 93*2=186 threads as shown in Fig. 6.

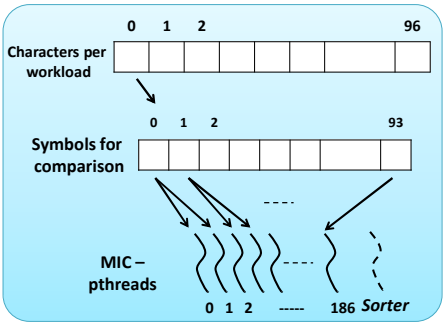

Fig. 6. SP2T architecture

Although the core computation in this case took the same time as SPST, new overhead for syncing the computation between the thread pairs is added. The runtime had now increased to 13.92 seconds.

B. Compiler based optimizations

Compiler switches and corresponding pragmas, language extensions and appropriate coding styles were employed to assist in auto optimization, based on the guidelines provided by the Intel compiler. Loop unrolling, vectorization, prefetching, streaming stores and Inter Procedural optimization (IPO) were evaluated.

Streaming stores and IPO had limited boost in performance due to the nature of the BSB algorithm. Since the software architecture was already refined the compiler generated optimized loops without the need for additional guidance for loop unrolling.

Prefetching: The coprocessor does not have out of order execution and has to solely depend on prefetch techniques to keep the pipeline full. It is accomplished by issuing prefetch instructions interleaved between other instructions before the actual need for the specified data/instruction. These instructions don't stall the processor and the data/instructions will be available ready in cache by the time they are actually needed.

The coprocessor supports both hardware prefetch and software prefetch. It relies more on software prefetching than on hardware prefetch. Hardware prefetching is enabled by default. The BSB algorithm did not benefit from the hardware prefetcher due to the nature of data access pattern required by the algorithm. However software prefetching had significant impact on the runtime and is enabled by default. For the case of 30x30 character BSB run needed about 16 seconds without software prefetching and after enabling, it took about 10 seconds for the same run.

An experiment was carried out by manually adding prefetch intrinsics and hints for prefetch distances to C++ code. In comparison the compiler optimized code provided better performance as it was able to compute optimum prefetch distances. As an observation in this particular case; manually

adding prefetch is probably best suited if the coding style is at assembly level or for intrinsic heavy coding format.

Vectorization: The Xeon Phi compiler performs vectoization, which converts scalar operations (operations on one set of data) to vector operations (same operation is performed on multiple data). One vector instruction operates on multiple operands hence significantly reducing the effective runtime compared to scalar implementation.

The coprocessor has vector processing units with 512 bit vector registers. These VPUs help in greatly reducing the code runtime. In fact vectorization provided the largest boost to the overall performance. All the results presented in this section are obtained with vectorized code.

Vectorization was achieved by recoding computation loops in a specific pattern [21], along with the use of language extensions (restrict) specific to the Intel compiler and respective compiler switches. Also the data had to be memory aligned to benefit from these optimizations. The code was fine-tuned and the required level of optimization achieved was confirmed through specific compiler reports.

VI. Results

After applying the SPST resource management with scatter affinity, and with the help of compiler based optimization options, we were able to optimize the pattern-matching layer for MIC architecture and achieved 1.8x performance gain on Intel's Xeon Phi coprocessor over MPMT as shown in Fig. 7. The runtimes plotted are for one workload of 96 characters at 30x30 resolution.

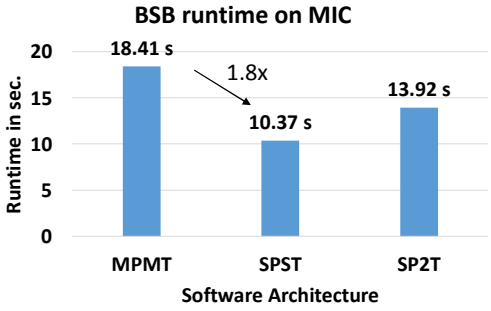

Fig. 7. BSB Runtime Optimization

For fair comparison, the same pattern-matching layer is also implemented on a standalone CPU and IBM Cell processor based PlayStation 3® setup. The CPU used in this experiment has 16 physical cores, each supporting 2-way simultaneous multi-threading. Each core has 512KB L1 cache, 2MB L2 cache and 20MB L3 cache. Since there are more threads than logic cores, the workload balancing is done by OS. Each PS3 processor has 6 Synergetic Processing Elements (SPE) and one PowerPC processor. Each SPE handles one SPST threads.

We consider the performance of the serial version of BSB algorithm running on CPU as our base reference and set its performance to 1. Fig. 8 gives the normalized performance of the pattern matching layer implemented on CPU, Xeon Phi and PS3. Because the same software architecture is implemented, the comparison measures the performance gained by upgrading the hardware to the MIC architectures. There is no Cell processor implementation data for 30x30 resolution case, as it was not feasible to run at this resolution due to limited memory. As we can see, Xeon Phi is able to provide 1.35x performance gain over the PS3 for 15x15 images and 1.94x performance gain over optimized CPU implementation for 30x30 images.

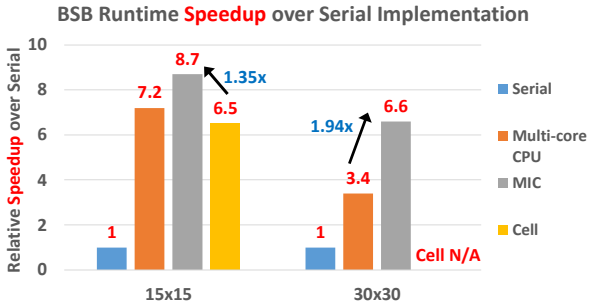

Fig. 8. Runtime comparison for standalone case

Fig. 9 shows the normalized performance comparison considering the communication interface with the rest of the ITRS. Again, Xeon Phi is able to provide 1.46x performance gain over the PS3 for 15x15 images and 1.9x gain over the optimized CPU implementation for 30x30 images.

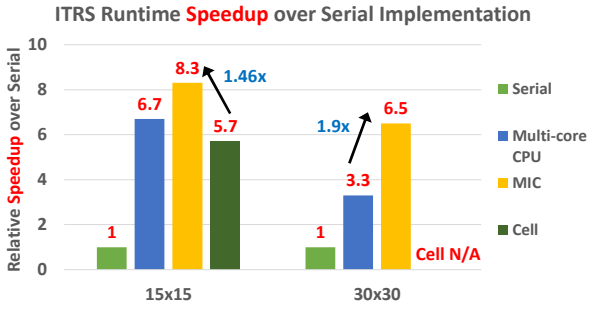

Fig. 9. Runtime comparison with confabulation

VII. Analysis

The pattern matching workload for each input character image is a matrix multiplication, for 30x30 character it is [1024 x 1024] * [1024 x 1]. Each element in these matrices is a float data, hence the size of the weight matrix is 4MB and the size of the input vector is 4KB. The result of the matrix vector multiplication generates 4KB of new data. For each input image, the number of times this multiplication is performed is: 50*93 = 4650, where 50 is the maximum number of iterations allowed for convergence. The above number of iterations is repeated for all the characters in the workload i.e. 96 times.

Total data requirement per iteration is ~4MB. Total cache on the coprocessor is about 30 MB. This cache is coherent and can be accessible through any core. Each core has 32 KB L1 cache and 512 KB L2 caches.

The data required per iteration is significantly greater than the per core cache size hence there is memory spill over per iteration. This is clearly the bottleneck which is holding back the overall performance. The nature of core compute part of the algorithm is like stream read. This application is data intensive and the achieved bandwidth is 176.58 GB/s (peak)

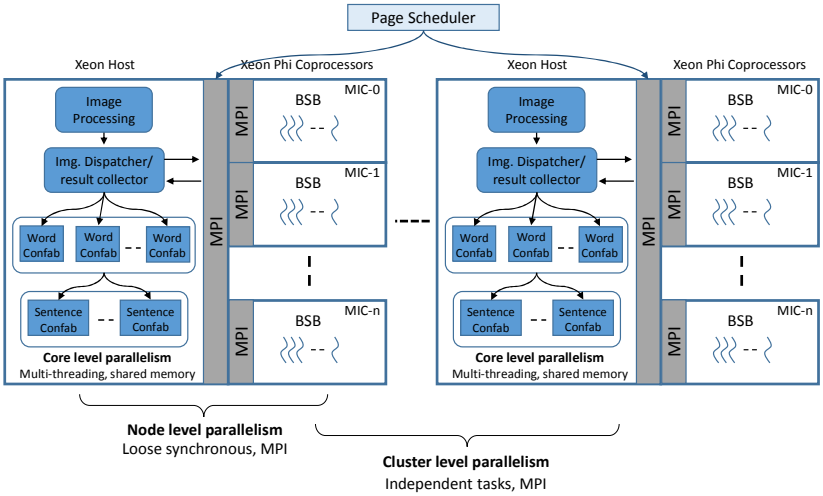

Fig. 10. Multi-level scaling of hybrid ITRS architecture

which is almost same as Stream Memory Benchmark which peaks at 181 GB/s [26].

The BSB runs natively on the coprocessor hence the Xeon host is dedicated solely for confabulation. This is especially beneficent as the BSB can run independently by receiving workload requests and send results through MPI without stalling any other module. Hence as many BSBs can run as the system can support.

The software architecture where a specific symbol is issued to a specific pthread as shown in Fig. 5 is found to be ideal for MIC architecture. The scaling capabilities are described based on this software architecture. This architecture is very flexible and allows for testing the behavior in terms of scaling at the workload level with in the Xeon Phi coprocessor. For one workload configuration 3 + 93 + 1 = 97 threads are required. The number of threads for two simultaneous workload configuration is 3 + (93 + 1) * 2 = 191 threads. Going beyond this is not advisable as number of threads will exceed the available hardware resources.

The BSB also scales at the node level and cluster level along with ITRS as shown in Fig. 10. The BSBs communicate directly with the host Xeon processor and not with each other. This kind of partitioning helps in maintaining simplicity and low MPI communication delays. Using multiple coprocessor cards in a single node provides linear performance scaling of BSB. The same scaling benefit can be achieved at the cluster level as well.

VIII. Conclusion and Future Work

We started off with a goal of upgrading the processing capability and accelerating BSB by having an efficient and optimized platform which can scale up to a cluster level. We parallelized and optimized the serial version of BSB for Xeon Phi coprocessor. During optimization we explored several avenues on the software architecture side and tweaked auto-optimization features available. We explored the effectiveness of using OpenMP and pthreads for the BSB algorithm. Both 15x15 and 30x30 resolution images were experimented on and found that 30x30 case is now feasible. Overall we were able to achieve a speed up of ~2x with our ITRS hybrid Xeon – Xeon Phi coprocessor implementation. This architecture is scalable at the core level, node level and at the cluster level. Every BSB (coprocessor) added to the system provides linear scaling in overall performance of all BSBs combined.

Our future work on this problem is to optimize the architecture for memory bandwidth. Also we will be reformulating the algorithm such that the computation can be split on multiple cores and span multiple iterations, to improve data retention on the coprocessor.

Acknowledgment

This work was partially supported by the National Science Foundation under Grants CCF-1337198, CCF-1337300, and the Air Force Research Laboratory, under contract FA8750-12-1-0251. We also thank Intel Corporation for supporting this work through an internship.

References

[1] Qinru Qiu; Qing Wu; Bishop, M.; Pino, R.E.; Linderman, R.W., "A Parallel Neuromorphic Text Recognition System and Its Implementation on a Heterogeneous High-Performance Computing Cluster,"*Computers, IEEE Transactions on* , vol.62, no.5, pp.886,899, May 2013 doi: 10.1109/TC.2012.50

[2] R. Wray, C. Lebiere, P. Weinstein, K. Jha, J. Springer, T. Belding, B. Best, and V. Parunak, "Towards a Complete, Multi-level Cognitive Architecture," *Proc. of the International Conference for Cognitive Modeling*, 2007.

[3] R. S. Swenson, "*Review of clinical and functional neuroscience*," Educational Review Manual in Neurology, Castle Connolly Graduate Medical Publishing, 2006.

[4] J. A. Anderson, "*An Introduction to Neural Networks*," The MIT Press, 1995.

[5] George Chrysos, Intel Corporation Intel® Xeon Phi™ Coprocessor - the Architecture http://software.intel.com/en-us/articles/intel-xeon-phi-coprocessor-codename-knights-corner

[6] J. Park, Y. Park, "An Optimization Approach to Design of Generalized BSB Neural Associative Memories," *Neural Computation, MIT Press Journals*, Vol. 12, No. 6, Jun. 2000, pp. 1449-1462.

[7] Y. Park, "Optimal and Robust Design of Brain-State-in-a-Box Neural Associative Memories," *Neural Networks, Elsevier*, Volume 23, Issue 2, Mar. 2010, pp. 210-218.

[8] A. Schultz, "Collective recall via the brain-state-in-a-box network," *Neural Networks, IEEE Transactions on*, vol. 4, no. 4, pp. 580–587, 1993.

[9] S. Mori, C.Y. Suen, and K. Yamamoto, “Historical Review of OCR Research and Development,” *Proc. IEEE*, vol. 80, no. 7, pp. 1029-1058, July 1992.
[10] Jianchang Mao, "A case study on bagging, boosting and basic ensembles of neural networks for OCR," *Neural Networks Proceedings, 1998. IEEE World Congress on Computational Intelligence. The 1998 IEEE International Joint Conference on* , vol.3, no., pp.1828,1833 vol.3, 4-9 May 1998
[11] Blando, L.R.; Kanai, J.; Nartker, T.A., "Prediction of OCR accuracy using simple image features," Document Analysis and Recognition, 1995., *Proceedings of the Third International Conference on* , vol.1, no., pp.319,322 vol.1, 14-16 Aug 1995
[12] Peng Ye; Doermann, D., "Learning features for predicting OCR accuracy," *Pattern Recognition (ICPR), 2012 21st International Conference on* , vol., no., pp.3204,3207, 11-15 Nov. 2012
[13] Qing Wu; Mukre, P.; Linderman, Richard; Renz, T.; Burns, D.; Moore, M.; Qinru Qiu, "Performance optimization for pattern recognition using associative neural memory," *Multimedia and Expo, 2008 IEEE International Conference on* , vol., no., pp.1,4, June 23 2008-April 26 2008
[14] Taha, T.M.; Yalamanchili, P.; Bhuiyan, M.A.; Jalasutram, R.; Mohan, S.K., "Parallelizing two classes of neuromorphic models on the Cell multicore architecture," *Neural Networks, 2009. IJCNN 2009. International Joint Conference on* , vol., no., pp.3046,3053, 14-19 June 2009
[15] Miao Hu; Hai Li; Qing Wu; Rose, G.S.; Yiran Chen, "Memristor crossbar based hardware realization of BSB recall function," *Neural Networks (IJCNN), The 2012 International Joint Conference on* , vol., no., pp.1,7, 10-15 June 2012
[16] Honghoon Jang; Anjin Park; Keechul Jung, "Neural Network Implementation Using CUDA and OpenMP," *Digital Image Computing: Techniques and Applications (DICTA)*, 2008 , vol., no., pp.155,161, 1-3 Dec. 2008
[17] Billconan and Kavinguy. A neural network on gpu http://www.codeproject.com/Articles/24361/A-Neural-Network-on-GPU
[18] Nere, A.; Hashmi, A.; Lipasti, M., "Profiling Heterogeneous Multi-GPU Systems to Accelerate Cortically Inspired Learning Algorithms," *Parallel & Distributed Processing Symposium (IPDPS), 2011 IEEE International* , vol., no., pp.906,920, 16-20 May 2011
[19] Diaz, J.; Munoz-Caro, C.; Nino, A., "A Survey of Parallel Programming Models and Tools in the Multi and Many-Core Era," *Parallel and Distributed Systems, IEEE Transactions on* , vol.23, no.8, pp.1369,1386, Aug. 2012
[20] Xinmin Tian; Saito, H.; Preis, S.V.; Garcia, E.N.; Kozhukhov, S.S.; Masten, M.; Cherkasov, A.G.; Panchenko, N., "Practical SIMD Vectorization Techniques for Intel® Xeon Phi Coprocessors," *Parallel and Distributed Processing Symposium Workshops & PhD Forum (IPDPSW), 2013 IEEE 27th International* , vol., no., pp.1149,1158, 20-24 May 2013
[21] David Mackay, Optimization and Performance Tuning for Intel® Xeon Phi™ Coprocessors - Part 1: Optimization Essentials http://software.intel.com/en-us/articles/optimization-and-performance-tuning-for-intel-xeon-phi-coprocessors-part-1-optimization
[22] Shannon Cepeda, Optimization and Performance Tuning for Intel® Xeon Phi™ Coprocessors, Part 2: Understanding and Using Hardware Events http://software.intel.com/en-us/articles/optimization-and-performance-tuning-for-intel-xeon-phi-coprocessors-part-2-understanding
[23] Gao Tao; Lu Yutong; Suo Guang, "Using MIC to Accelerate a Typical Data-Intensive Application: The Breadth-first Search," *Parallel and Distributed Processing Symposium Workshops & PhD Forum (IPDPSW)*, 2013 IEEE 27th International , vol., no., pp.1117,1125, 20-24 May 2013
[24] Turchenko, V.; Bosilca, G.; Bouteiller, A.; Dongarra, J., "Efficient parallelization of batch pattern training algorithm on many-core and cluster architectures," *Intelligent Data Acquisition and Advanced Computing Systems (IDAACS), 2013 IEEE 7th International Conference on* , vol.02, no., pp.692,698, 12-14 Sept. 2013
[25] Intel® Math Kernel Library http://software.intel.com/en-us/intel-mkl
[26] Karthik Raman Optimizing Memory Bandwidth on Stream Triad http://software.intel.com/en-us/articles/optimizing-memory-bandwidth-on-stream-triad