A Multi-Answer Character Recognition Method and Its Implementation on a High-Performance Computing Cluster

Qing Wu, Morgan Bishop, Robinson Pino, Richard Linderman
Information Directorate
US Air Force Research Laboratory
Rome, New York, USA
{Qing.Wu, Morgan.Bishop, Robinson.Pino, Richard.Linderman}@rl.af.mil

Qinru Qiu
Department of Electrical Engineering & Computer Science
Syracuse University
Syracuse, New York, USA
qinru.qiu@gmail.com

***Abstract* — In this paper, we present our work in the implementation and performance optimization of a novel multi-answer character recognition method on a high-performance computing cluster. The main algorithm used in this method is called the Brain-State-in-a-Box (BSB), which is an auto-associative neural network model. We applied optimization techniques on different parts of the BSB algorithm to improve the overall computing and communication performance of the system. Furthermore, the proposed method adopts a new way to train, recall, and organize the BSB models for different characters, in order to provide a sorted (based on recall convergence speed) list of candidates for a given character image.**

Keywords – character recognition, brain-state-in-a-box, neural network, performance optimization

I. INTRODUCTION

In the past four decades, a significant amount of research work [7][8] has been performed on optical character recognition (OCR) for printed characters. OCR for printed text represents an important application of pattern recognition and image processing. As of today, there are many software OCR tools available, such as the open-source Tesseract-OCR [9].

Although OCR for printed text is regarded as a largely solved problem, its reliability and robustness are not guaranteed when the input text image is either noisy or severely occluded. The main limitation of the existing OCR tools is that they are trying to provide a single answer for each individual character. When a character image is severely occluded, most OCR algorithms will make a "best guess" and give one deterministic answer, which does not provide a "second-thought" candidate. On the other hand, the human cognition process looks at not only the individual character image, but also its context such the word and the sentence. When a character is hard to recognize, a human will first make multiple guesses and cross-reference them with context, then decide which guess fits the word best, which word fits the sentence best, etc.

The goal of our research at AFRL is to develop a high-performance text recognition software tool that is highly reliable and robust for noisy or occluded text images. The multi-answer character recognition method introduced in this paper is an essential component of the project. Other major components of the project include a word-level language model and a sentence-level language model [12], which are not in the scope of this paper. The overall software architecture makes these neuromorphic algorithms work together to produce improved text recognition results.

The text recognition software is implemented on a high-performance computing (HPC) cluster that consists of almost 70,000 processor cores and provides a massive peak computing power of 500 trillion floating-point operations per second. To take full advantage of the available HPC resources, the algorithm must be highly scalable so that the overall performance increases linearly with the number of processor cores used. Furthermore, HPC resources are most effective when performing regular and continuous floating-point operations. Through optimization efforts, we found the Brain-State-in-a-Box (BSB) neural network model [1][2][3] to fit the HPC efficiently.

In order to provide multiple answers to a given character image, we designed a new "racing" mechanism when performing pattern recognition using the BSB models. Basically, through a training process we build different BSB models for different characters. Any input character image will be sent to all the BSB models for recall (pattern recognition). When all recalls are completed, a set of candidates is selected based on the convergence speed. This process forms the proposed multi-answer character recognition method.

The remainder of the paper is organized as follows. In Section II we provide background information on the system architecture, the BSB model, and processor architecture. Section III describes the details of optimizing the BSB algorithm on the IBM Cell-BE processor. Section IV discusses the implementation of the "racing" mechanism to generate multiple recognition candidates, as well as the simulation results.

II. BACKGROUND

A. Massively Parallel Computing System

Modeling and simulation of human cognizance functions involve large-scale mathematical models, which demand a

 ISBN: 978-1-61208-154-0

high performance computing platform. We desire a computing architecture that meets the computational capacity and communication bandwidth of a large-scale associative neural memory model. In particular, we are interested in processing pages of text at real-time rates of up to 50 pages per second.

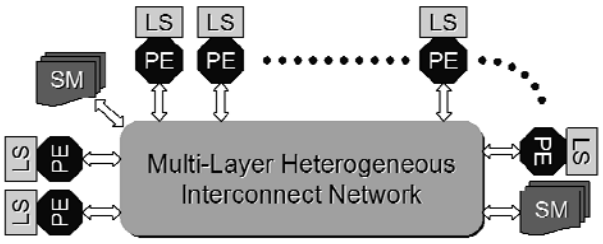

Figure 1. High performance cognitive computing system.

Figure 1 shows the targeted architecture of a high performance cognitive computing system. This system consists of multiple Processing Elements (PEs) interconnected with a multi-layer heterogeneous interconnect network. All PEs have access to a local memory called the Local Store (LS) and Shared Memory (SM) connected to the interconnect network.

B. Brain-State-in-a-Box Model

The BSB model is a simple, auto-associative, nonlinear, energy-minimizing neural network [1][2][3]. A common application of the BSB model is to recognize a pattern from a given occluded version. It can also be used as a pattern recognizer that employs a smooth nearness measure and generates smooth decision boundaries3.

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

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

where:

- **x** is an *N* dimensional real vector
- **A** is an *N*-by-*N* 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:*

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

Note that in the proposed algorithm, we choose λ to be 1.0 and γ to be 0.0. But they can be easily changed to other values during the implementation. Given an input pattern **x**(0), the recall process computes Equation (1) iteratively to reach convergence. A recall converges when all entries of **x**(*t*+1) are either "1.0" or "-1.0". In our implementation, it usually takes more than ten iterations for recall to converge.

C. Cell Broadband Engine Architecture

The IBM Cell Broadband Engine (Cell-BE) architecture [4][5][6] is a multi-core architecture (shown in Figure 2) designed for high performance computing. The architecture features nine microprocessors on single chip. A Power architecture compliant core called the *Power Processing Element* (PPE) and eight other attached processing cores called *Synergetic Processing Elements* (SPEs) are interconnected by a high bandwidth *Element Interconnect Bus* (EIB). This heterogeneous architecture with high performance computing cores is designed for distributed multicore computing.

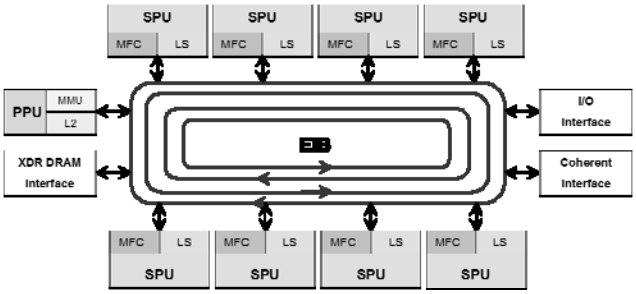

Figure 2. Cell-BE processor block diagram.

III. Performance Optimization of the BSB Algorithm on the Cell-BE Processor

In this section we will describe the implementation and performance optimization of the recall operation on a single Cell-BE processor. One of the major challenges in implementing the recall operation in software is the high computational demand. For one 128-dimensional BSB model recall we need a matrix-to-vector multiplication that involves 16384 floating-point multiplications and 16256 floating-point additions. In addition, we also need 128 floating-point multiplications for the feedback factor and 256 comparisons. Our final large-scale associative neural model would involve a large number of BSB models (up to 500,000). We need a parallel distributed computational model which can perform a massive number of BSB recall operations in parallel.

The Cell-BE processor with high performance computing cores is an ideal platform for distributed computing. We can run one BSB recall on each SPE. Next, we will describe the implementation details of a 128-dimensional BSB recall operation on one SPE.

A. Matrix-Vector Multiplication

Multiplication of a 128x128 matrix with a 128x1 vector can be represented as follows. We are showing these rather

 ISBN: 978-1-61208-154-0

simple operations in detail, in order to elaborate the floating-point operations involved in the computation.

$ax_0 = a_{0,0}*x_0 + a_{0,1}*x_1 + a_{0,2}*x_2 + \ldots + a_{0,127}*x_{127}$
$ax_1 = a_{1,0}*x_0 + a_{1,1}*x_1 + a_{1,2}*x_2 + \ldots + a_{1,127}*x_{127}$
$ax_2 = a_{2,0}*x_0 + a_{2,1}*x_1 + a_{2,2}*x_2 + \ldots + a_{2,127}*x_{127}$
..
..
$ax_{127} = a_{127,0}*x_0 + a_{127,1}*x_1 + a_{127,2}*x_2 + \ldots + a_{127,127}*x_{127}$

The scalar implementation of the above equations can be written in C as:

```
float ax[128], x[128], a[128*128];
int row,col;

for(row=0;row<128;row++){
   ax[row]=0;
   for(col=0;col<128;col++) {
        ax[row]+=x[col]*a[row*128+col];  }}
```

We can improve the computational performance by using the Single Instruction Multiple Data (SIMD) model of the SPE. Most of the instructions in SPEs operate on 16 bytes of data and also the data fetching from local store is 16-byte aligned. To implement the scalar computation using SIMD instructions, the compiler has to keep track of the relative offsets between the scalar operands to get the correct results. This implementation ends up having additional rotation instructions added, which is an overhead. To get better performance and the correct result, it is wise to handle the data as vectors of 16 bytes (or four single-precision floating-point numbers) each.

The matrix multiplication using SIMD instructions can be implemented as below.

```
float ax[128] __attribute__((aligned (128)));
float x[128] __attribute__((aligned (128)));
float a[128*128] __attribute__((aligned (128)));
int row,col;
vector float  *x_v, *a_v;
vector float temp;
x_v = (vector float *)x;
a_v = (vector float *)a;

for(row=0;row<128;row++){
  temp = (vector float){0.0,0.0,0.0,0.0};
  for(col=0;col<128/4;col++) {
    temp=spu_madd(x_v[col],a_v[row*32+col],temp);
  }
  ax[row] =
(spu_extract(temp,0)+spu_extract(temp,1)+
 spu_extract(temp,2)+spu_extract(temp,3));
}
```

spu_madd and *spu_extract* are intrinsics which make the underlying Instruction Set Architecture (ISA) and SPE hardware accessible from the C programming language. *spu_madd* is the C representation for the multiply and add instruction. *spu_extract* returns the vector member value specified by the offset. Compared to the scalar implementation, SIMD code reduces 16384 multiplication operations to 4096 vector multiplications. The above implementation still performs scalar addition to get the final result. We can further improve the performance by rearranging the matrix so that we can apply SIMD instructions on all the matrix-vector multiplication operations.

We divide the entire matrix into smaller 4x4 matrices. Elements of each of these 4x4 matrices are shuffled according to a specific pattern as shown in Figure 3.

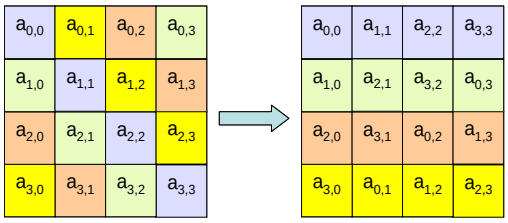

Figure 3. Matrix shuffling from original order to SIMD order.

The shuffled matrix is multiplied with the X vector as shown in Figure 4. Note that each row of the shuffled matrix is a vector of four single-precision floating-point numbers. And (x_0, x_1, x_2, x_3) is the other vector in the SIMD operation.

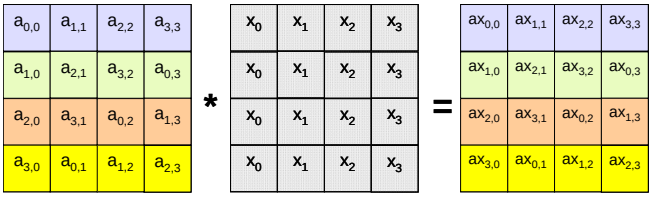

Figure 4. Shuffled matrix multiplication.

To obtain the final result we need to rotate the some of the elements to align them back to their original offset and add all the rows, as shown in Figure 5.

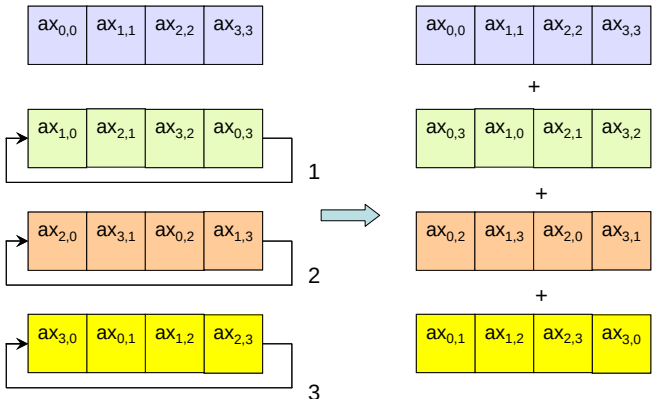

Figure 5. Product alignment and accumulation.

By shuffling the input matrix we have effectively replaced the 3*128 scalar additions from the previous implementation with 3*32 rotations and also we reduced the overhead added by the compiler to perform scalar addition.

For every four rows of the 128x128 matrix, we will have 32 4x4 matrices. The above shuffle-multiply-rotate-accumulate operation is repeated for each one of them, and

 ISBN: 978-1-61208-154-0

at the end we will be able to obtain the four elements of the resulting 128x1 vector. For 128 rows, we need to repeat the above procedure 32 times and obtain the complete result. In our current program implemented on the PS3, we assume that the 128x128 matrix has already been shuffled before being stored into the main memory.

B. Other operations in BSB recall

From Equation (1) we know that $\mathbf{A}*\mathbf{x}(t)$ has to be multiplied with feedback constant α and the result added with $\mathbf{x}(t)$. This multiplication and addition can be performed by using the *spu_madd* intrinsic, which multiplies two vectors and adds the result to the third vector. This step requires 32 *spu_madd* intrinsics.

The final operation in recall is the squash function. As given in Equation (2) this operation needs two comparisons to check whether the result of the previous computation is >1 or <-1. To perform this kind of compare and assign operation on vector data, the SPE provides special instructions.

spu_cmpgt is an intrinsic which performs element-wise comparisons on two given vectors. If an element in the first vector is greater than corresponding element in the second vector then all the entries of the corresponding element in the result vector are set to '1', else '0'. This result can be used as a multiplexer select: '1' assigns input_1 to the output and '0' assigns input_0 to the output. An intrinsic called *spu_sel* is used to perform the selection. *spu_sel* takes two input vectors and a select pattern. For each entry in the 128-by-1 vector, the corresponding entries from either input vector-0 or input vector-1 are selected.

C. Balancing computation and communication

One of the important factors affecting the performance of the SPE is the data transfer time. Due to limited local store size it is not possible to get all the data required for the computation at once. Therefore the SPE has to initiate direct memory access (DMA) transfers whenever it requires additional data from the main memory, which takes a significant amount of time. However we can leverage the communication-computation concurrency provided by the Cell-BE's asynchronous DMA model by performing computation while data for future computations is being fetched. This is called the *double buffering* method. Figure 6 shows the computational flow with and without double buffering. In the regular communication model, the weight matrix required for the recall is fetched and ten recall iterations are performed. Then the DMA request for the weight matrix of the next recall is initiated. This induces gaps in the computational flow that reduce the effective throughput. In the double buffering implementation, when the computation of the current recall starts, a DMA request for the weight matrix of the next recall is initiated in parallel. The memory controller of the SPE can work in background to fetch the data from the memory. By the time ten iterations of the current recall are completed, the data for the next recall will be available, and then the SPE can continue with its computational flow.

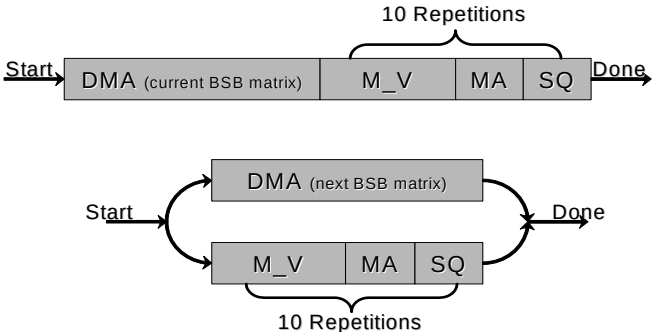

Figure 6. Algorithmic flow without (top) and with (bottom) the double buffering method.

IV. Multi-Answer Implementation and Experimental Results

In this section, we first describe the "racing" mechanism that we use to implement the multi-answer character recognition process. Then, we present and discuss the experimental results when we apply this method on character images with different amounts of added noise.

A. Multi-answer character recognition using BSB models

Without loss of generality, assume that the set of characters we want to recognize from images consists of 52 characters, which are the upper and lower case characters of the English alphabet.

$$S = \{'a', 'b', \dots, 'z', 'A', 'B', \dots, 'Z'\}$$

We also assume that for each character in *S*, there are *M* typical variations in terms of different fonts, styles, and sizes. For example, the set of images of character '*a*' with different variations can be represented as:

$$S_a = \{a_1, a_2, \dots, a_M\}.$$

In terms of pattern recognition, there are a total of 52**M* patterns to remember during training and to recognize during recall. If we follow the traditional application approaches of the BSB models, the solution is to train one BSB model to remember all the 52**M* patterns. During recall, given an input image, this model will eventually converge to one of the remembered patterns (attractors) that represent the recognition result. The shortcomings of this approach is that firstly it requires a BSB model with large dimensionality (*N* the dimension of vector *X* in Equation 1) to remember all the patterns. This increases the complexity ($\propto N^2$) of the computation and also reduces the scalability when implemented on parallel computing architectures. Secondly, this approach only provides one answer to the input image. The BSB recall process does not return the second or third closest attractor for the image. For recognizing corrupted texts, providing only one answer is not adequate for the low-level pattern recognition model to work with high-level language models.

 ISBN: 978-1-61208-154-0

Therefore, in our implementation, the primary goal is to design a process that provides multiple candidates for an input image. And the secondary goal is to have reasonably-sized BSB models to have good scalability and keep computation complexity under control.

The solution we designed is to use one BSB model for each character in *S*. Therefore there will be a set of 52 256-dimensional BSB models, that is:

$$S_{BSB} = \{BSB_a, BSB_b, \ldots, BSB_z, BSB_A, BSB_B, \ldots, BSB_Z\}.$$

Each BSB model is trained for all variations of a character. For example, BSB_a is trained to remember all the variable patterns in S_a, BSB_b will remember patterns in S_b, so on so forth. If we define the procedure "Recall(*A*, *B*)" as the recall process using model A with input image *B*, which returns the number of iterations it takes to converge, the recall and candidate selection process can be described as follows.

```
Input: character image X.
1. For each trained BSB model BSB_i in S_BSB
        Conv[i] = Recall(BSB_i, X);

2. Sort Conv[i] from low to high to form
   sorted list Conv_s[j];

3. Pick the first K in Conv_s[j] as
   recognition candidates, if it satisfies
   both conditions listed below:

   a. Conv_s[j] <= Th_1;
   // Convergence speed threshold
   b. Conv_s[j] – Conv_s[j-1] <= Th_2;
   // Separation threshold
```

In this algorithm, {*K*, Th_1, Th_2} are adjustable parameters based on overall reliability and robustness needs.

Generally speaking, in our multi-answer implementation, we utilize the BSB model's convergence speed to represent the "closeness" of an input image to the remembered characters (with variations). Then we pick up to *K* "closest" candidates (that satisfy conditions 3a and 3b) to work with high-level language models to determine the final output. In a HPC platform consisting of many (up to 1,700) IBM Cell-BE processors, our implementation was able to execute the recall operations in parallel. Because each BSB model is small enough to fit on a single Cell-BE processor, the overall performance scales linearly with the number of Cell-BE processors used.

B. Simulation results

In our current implementation, the BSB models are trained for 93 characters, with up to six different fonts, five different sizes and two different styles, i.e., up to 60 variations per character. The complete set covers all the characters that can be found on a computer keyboard.

To better demonstrate the general trends of our approach, we will present the results when we use 256-dimensional BSB models trained for the first 52 characters in the set, with two fonts, two sizes and one style (four variations for each character). The input character image is a 15-by-15, 225-pixel grayscale or black-and-white bitmap. Therefore the dimensionality of the BSB models must be at least 225. We chose 256 because it is the next powers-of-two number. In addition, for the given number of the character variations it needs to remember, a 256-dimensional BSB model should have enough attractors.

Table 1 shows the top-three candidates generated by our program when non-occluded character images are given to the models. In the table, the "C" columns show the first, second and third fastest converging BSB models, and the "N" columns show the number of recall iterations before convergence. A "*" mark means that the BSB model has not converged by the limit (Th_1 = 75) to be selected as a candidate. The cells highlighted in the diagonal-line pattern represent the "correct" candidates. We can see that for non-occluded images, the "correct" BSB models always converge first. We also see a 2x to 3x margin in terms of the "N" value between the first and second candidates.

Table 2 and Table 3 show the top-three candidates when the input images are occluded by 1-pixel-strike-through and 3-pixel-strike-through black bars, respectively. Please note the characters shown in the "Input" columns only indicate the damaged characters, but are not the actual images. For a 1(3)-pixel-strike-through, we added one (three) horizontal black bar(s) in the middle of the input image, from the left edge to right edge. From Table 2 we can see that for moderate occlusion, the "correct" BSB models still converge the fastest, although slower as compared to Table 1. However for significant occlusion as in Table 3, we can see that two of them did not converge first, while five of them (highlighted in red) are not among the top-three candidates.

Overall, we have shown that the "closeness" of the input pattern to the remembered ones can be measured by the speed of convergence of the BSB recall process, which we can use to select multiple candidates for an input character image. To deal with the situation when the "correct" BSB is not in the candidate list, we can lower the iteration threshold (Th_1) to, for example, 30. This change will result in "no candidate" for some occluded input, which also means that the high-level language model will receive a candidate list with all characters in it.

Working as a component of the text recognition software developed by AFRL/RI, the absolute accuracy of the proposed character recognition method is not as important as it may be in other character recognition software tools. Our text recognition software has unique language models and algorithms [10][11][12] to work with the BSB outputs. The overall integrated approach achieves better text recognition accuracy, particularly when the character images are damaged.

V. CONCLUSION

We have presented work in the implementation and performance optimization of a novel multi-answer character recognition method on a high-performance computing

 ISBN: 978-1-61208-154-0

cluster. We applied optimization techniques on different parts of the BSB algorithm to improve the overall computing and communication performance of the system. Furthermore, the proposed method adopts a new way in training, recalling, and organizing the BSB models for different characters, in order to provide a list of candidates for a given character image. By offering multiple answers, this character recognition algorithm was able to be integrated with high-level language models to achieve more reliable and robust text recognition capabilities.

ACKNOWLEDGMENT OF SUPPORT AND DISCLAIMER

Received and approved for public release by AFRL on 04/14/2011, case number 88ABW-2011-2178.

The contractor's work is supported by the Air Force Research Laboratory, under contract FA8750-09-2-0155.

Any Opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of AFRL or its contractors.

REFERENCES

[1] J. A. Anderson, J. W. Silverstein, S. A. Ritz, and R. S. Jones, "Distinctive features, categorical perception, probability learning: Some applications of a neural model," Neurocomputing; Foundations of Research, J. A. Anderson and E. Rosenfeld, Editors, The MIT Press, 1989, ch. 22, pp. 283–325, reprint from Psychological Review 1977, vol. 84, pp. 413–451.

[2] M. H. Hassoun, Editor, "Associative Neural Memories: Theory and Implementation," Oxford University Press, 1993.

[3] A. Schultz, "Collective recall via the Brain-State-in-a-Box network," IEEE Transactions on Neural Networks, vol. 4, no. 4, pp. 580–587, July 1993.

[4] T. Chen, R. Raghavan, J. Dale, and E. Iwata, "Cell Broadband Engine Architecture and its first implementation," IBM, 2011, http://www.ibm.com/developerworks/power/library/pa-cellperf/.

[5] "IBM Cell Broadband Engine Architecture," https://www-01.ibm.com/chips/techlib/techlib.nsf/techdocs/1AEEE1270EA2776387257060006E61BA/$file/CBEA_v1.02_11Oct2007_pub.pdf, 2011.

[6] "IBM Cell Broadband Engine resource center," http://www-128.ibm.com/developerworks/power/cell/, 2011.

[7] J. Mantas, "An Overview of Character Recognition Methodologies," Pattern Recognition, 19(6); 425-430, 1986.

[8] G. Nagy, "Optical Character Recognition – Theory and Practice," Handbook of Statistics, Vol. 2, 621-649, 1982.

[9] "Tesseract-OCR", http://code.google.com/p/tesseract-ocr/, 2011.

[10] Q. Qiu, Q. Wu, D. Burns, M. Moore, M. Bishop, R. Pino, and R. Linderman, "Confabulation Based Sentence Completion for Machine Reading," Proceedings of the IEEE Symposium Series on Computational Intelligence, Paris, France, April 2011.

[11] Q. Qiu, Q. Wu, M. Bishop, M. Barnell, R. Pino, and R. Linderman, "An Intelligent Text Recognition System on the AFRL Heterogeneous Condor Computing Cluster," to appear, Proceedings of the DoD High Performance Computing Modernization Program Users Group Conference, Portland, Oregon, June 2011.

[12] Q. Qiu, Q. Wu, and R. Linderman, "Unified Perception-Prediction Model for Context Aware Text Recognition on a Heterogeneous Many-Core Platform," to appear, Proceedings of the International Joint Conference on Neural Networks, San Jose, California, July 2011.

Table 1. Top-three candidates generated when given clean character images.

Input	Top-3 Candidates						Input	Top-3 Candidates					
	1st		2nd		3rd			1st		2nd		3rd	
	C	N	C	N	C	N		C	N	C	N	C	N
a	a	12	I	31	O	31	A	A	12	I	36	z	38
b	b	11	L	28	t	31	B	B	11	z	30	F	30
c	c	12	o	25	e	28	C	C	11	I	27	E	34
d	d	12	I	31	g	32	D	D	12	E	28	f	32
e	e	12	B	35	C	35	E	E	11	F	27	z	29
f	f	11	t	27	E	28	F	F	11	E	22	L	26
g	g	12	I	28	q	30	G	G	11	I	30	D	33
h	h	11	L	26	f	29	H	H	12	L	27	f	28
i	i	11	l	25	I	27	I	I	11	l	22	z	30
j	j	11	v	31	l	33	J	J	12	I	26	g	31
k	k	11	L	26	E	29	K	K	11	L	28	N	28
l	l	11	I	24	z	30	L	L	11	b	29	z	30
m	m	11	f	28	n	28	M	M	11	Y	32	F	35
n	n	12	L	30	b	34	N	N	12	U	26	E	27
o	o	12	c	27	e	29	O	O	11	I	28	G	30
p	p	12	L	28	z	32	P	P	11	L	26	z	32
q	q	11	g	23	I	29	Q	Q	11	I	28	z	32
r	r	12	l	27	I	33	R	R	11	B	30	f	32
s	s	12	I	31	a	32	S	S	11	I	28	C	33
t	t	11	L	25	b	30	T	T	11	l	23	I	23
u	u	12	n	25	D	33	U	U	12	b	24	z	34
v	v	11	I	33	n	36	V	V	11	I	35	n	39
w	w	12	j	35	n	35	W	W	12	f	32	E	32
x	x	11	k	32	n	33	X	X	12	Z	30	v	34
y	y	11	E	31	F	32	Y	Y	11	I	29	R	33
z	z	11	I	29	H	31	Z	Z	11	I	29	n	35

 ISBN: 978-1-61208-154-0

Table 2. Top-three candidates generated when given 1-pixel-strike-through character images.

Input	Top-3 Candidates						Input	Top-3 Candidates					
	1st		2nd		3rd			1st		2nd		3rd	
	C	N	C	N	C	N		C	N	C	N	C	N
a	a	23	R	32	I	33	A	A	23	e	34	z	37
b	b	23	L	28	j	32	B	B	22	F	30	L	30
c	c	24	e	24	o	27	C	C	24	I	30	J	37
d	d	23	g	33	J	34	D	D	23	E	28	f	33
e	e	22	B	35	C	35	E	E	22	F	27	N	29
f	f	23	t	28	E	30	F	F	22	E	25	L	28
g	g	23	q	30	I	31	G	G	23	O	32	H	33
h	h	23	L	28	H	29	H	H	22	f	28	L	29
i	i	23	l	26	I	29	I	I	23	l	24	z	30
j	j	23	a	33	e	33	J	J	23	I	28	R	31
k	k	23	L	29	H	32	K	K	23	L	29	N	29
l	l	23	I	26	z	31	L	L	23	E	27	z	30
m	m	23	f	30	n	31	M	M	23	F	35	U	35
n	n	23	L	30	e	33	N	N	23	F	28	U	29
o	e	24	o	24	c	27	O	O	24	I	31	s	35
p	p	23	L	31	B	33	P	P	23	L	28	r	34
q	q	24	g	25	I	31	Q	Q	23	I	31	R	33
r	r	23	l	29	J	33	R	R	22	f	31	B	33
s	s	22	B	33	a	34	S	S	24	I	30	C	34
t	t	23	L	28	b	33	T	T	23	l	25	I	25
u	u	23	n	28	G	35	U	U	23	b	26	L	34
v	v	24	k	37	B	37	V	V	23	B	40	I	42
w	w	26	j	36	N	38	W	W	28	L	35	f	36
x	x	23	k	35	n	35	X	X	23	Z	34	v	35
y	y	23	H	34	n	36	Y	Y	23	I	34	R	34
z	z	23	I	29	L	34	Z	Z	24	I	29	R	36

Table 3. Top-three candidates generated when given 3-pixel-strike-through character images.

Input	Top-3 Candidates						Input	Top-3 Candidates					
	1st		2nd		3rd			1st		2nd		3rd	
	C	N	C	N	C	N		C	N	C	N	C	N
a	a	26	e	36	B	41	A	A	26	e	36	y	38
b	b	35	t	35	L	39	B	B	28	P	28	L	32
c	c	29	o	36	G	46	C	G	34	e	36	C	37
d	d	31	J	36	K	39	D	D	27	E	32	O	32
e	c	37	G	38	m	45	E	E	29	F	30	L	32
f	f	27	t	33	e	37	F	F	30	f	31	E	31
g	q	36	I	42	J	44	G	G	27	I	43	a	44
h	h	28	H	30	f	33	H	H	29	h	33	i	35
i	l	31	J	36	e	38	I	I	27	i	29	l	29
j	j	30	J	35	K	38	J	J	27	I	35	R	37
k	k	26	L	35	i	37	K	K	26	f	36	i	36
l	l	27	i	29	I	33	L	L	27	E	33	a	37
m	m	26	f	39	n	41	M	M	26	A	44	R	44
n	n	31	o	36	L	39	N	N	26	i	38	E	38
o	o	34	c	35	E	42	O	s	32	G	34	O	34
p	p	26	D	33	e	37	P	f	31	L	33	a	34
q	q	29	e	42	p	42	Q	Q	26	e	42	h	43
r	r	29	l	37	e	43	R	R	25	f	33	i	35
s	s	29	R	37	m	40	S	b	39	J	39	I	40
t	t	27	e	36	I	36	T	T	27	I	31	l	32
u	u	28	w	36	G	39	U	U	27	L	38	f	40
v	v	29	y	41	P	42	V	V	26	y	37	P	40
w	N	36	w	37	q	43	W	W	29	w	40	G	45
x	x	27	w	43	E	45	X	X	28	Z	39	v	40
y	y	26	P	38	K	40	Y	Y	27	e	38	M	41
z	z	29	A	35	H	36	Z	Z	27	I	34	e	38

 ISBN: 978-1-61208-154-0