



Real-Time Mutual-Information-Based Linear Registration on the Cell Broadband Engine Processor

*Moriyoshi Ohara, Hiroshi Inoue and Hideaki Komatsu, IBM Tokyo
Research Laboratory*

*Hangu Yeo, Frank Savino, Vadim Sheinin, Giridharan Iyengar and
Leiguang Gong, IBM T. J. Watson Research Center*

Shahrokh Daijavad, IBM Systems and Technology Group

Bradley Erickson, Mayo Clinic and Foundation

Contents	
1. <i>Introduction</i>	2
2. <i>Mutual-information-based linear registration algorithm</i>	3
3. <i>Accelerating the registration process</i>	4
3.1 <i>Task parallelization and partitioning</i>	4
3.2 <i>SIMD pipeline optimization</i>	5
3.2.1 <i>Image down sampler</i>	5
3.2.2 <i>Gradient computation</i>	6
3.3 <i>Optimized data partitioning: Localized sampling and speculative packing</i>	7
4. <i>Experimental results</i>	10
4.1 <i>Overall Registration Time</i>	10
4.2 <i>Performance Gain</i>	11
4.2.1 <i>Computation time</i>	11
4.2.2 <i>Memory traffic</i>	13
5. <i>Discussions and conclusions</i>	14
6. <i>Acknowledgement</i>	15
<i>Footnotes</i>	16

The emerging generation of multi-core processors can accelerate medical imaging applications by exploiting the parallelism that is available in their algorithms. We have implemented a mutual-information-based three-dimensional (3D) linear registration algorithm on the Cell Broadband Engine™ (Cell/B.E.) processor, which has nine processor cores on a chip and a 4-way Single Instruction, Multiple Data (SIMD) unit for each core. This implementation parallelizes the code for multiple cores and organizes the data structure for reducing the amount of memory traffic. With two Cell/B.E. processors, our algorithm can compute the mutual information and register a pair of 256 x 256 x 30 3D images, about 33 million pixel pairs, in one second, which is significantly faster than a conventional implementation on a traditional microprocessor or even a custom hardware implementation.

This paper describes our implementation techniques with a focus on code optimization for the SIMD pipeline and data partitioning. The code optimization accelerates the application by 4.5 times on average. The data partitioning reduces the amount of the memory traffic by 82%.

1. Introduction

Image registration is a process that aligns two sets of images, typically obtained at different times or from different imaging devices, and it plays an increasingly important role in clinical applications.¹ Mutual-information-based image registration is known to be robust and effective, but it is computationally very expensive. Therefore, custom hardware approaches and supercomputer-based approaches for accelerating the registration process have been proposed in the past.²

Our approach implements a mutual-information-based linear registration algorithm on a Cell/B.E. processor. This is an asymmetric, high-performance multi-core processor that combines eight synergistic processing elements (SPEs) and a Power Processing Element (PPE), which is a general-purpose IBM Power PC® core.³ Each SPE has a 4-way SIMD engine, a high-speed local store and a direct memory access (DMA) engine. A high speed bus (for example, an

IBM BladeCenter® QS20 that can run 16 SPEs in parallel) connects two of these processors (Figure 1).

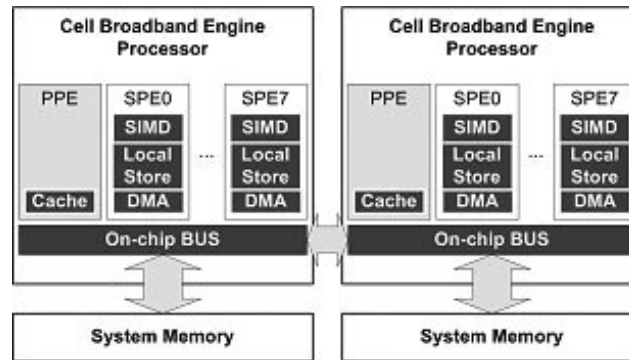


Figure 1 The high-level structure of a dual Cell/B.E. system

The 4-way SIMD unit of each SPE can perform a floating or integer operation on four data elements at every clock tick. Unlike conventional microprocessors, each SPE does not have a hardware cache to manage its small on-chip local store. Therefore, this architecture should be viewed as a distributed memory multiprocessor with a very small local memory, under software control, attached to a larger shared memory.

To accelerate the image registration on this processor, it was necessary to optimize the program to exploit the parallelism at both the task and SIMD instruction levels and to use the available memory bandwidth efficiently. Because of the unique characteristics of the Cell/B.E. architecture, additional techniques can further accelerate the application, namely task partitioning and code optimization for the SPE SIMD pipeline. This paper describes our implementation with a focus on code optimization and data partitioning techniques, which significantly improve the utilization of the SIMD pipeline and the memory bandwidth, respectively.

2. Mutual-information-based linear registration algorithm

We used a Mattes mutual-information-based multi-resolution algorithm implemented in the National Library of Medicine Insight Segmentation and

Registration Toolkit (ITK) as a base.⁴ This algorithm treats the registration problem as an optimization problem. It uses the spatial transformation that aligns one image (the moving image) to another image (the fixed image) by traversing the parameter space of the spatial transformation to find the position that maximizes the mutual information between the two images. To limit the parameter space to be traversed and accelerate the process, our implementation uses multi-resolution and random sampling. The multi-resolution method starts by registering lower-resolution images to estimate the transformation parameter. Then, it uses the estimate as the initial position in the parameter space for registering higher-resolution images. The random sampling method, furthermore, reduces the computation time by computing the mutual information only for a small subset of sample pixels from the fixed image and packs them into a contiguous memory area for improving their spatial locality. It repeats using the same set of samples at each resolution.

Our algorithm uses four different resolutions (original, 1/4, 1/16 and 1/64). It computes the mutual information 100 times using sample pixels for each resolution except for the original one (200 times for the original resolution). The number of sample pixels is the same as 1% of the pixels at the original resolution.

3. Accelerating the registration process

To obtain high performance from this algorithm, it must be mapped into the highly parallel architecture of the Cell/B.E. processor. This involves task parallelization and portioning, SIMD pipeline optimization and optimized data partitioning.

3.1 Task parallelization and portioning

Parallel programming techniques for shared-memory-based multi-core processors are generally applicable to the Cell/B.E. processor.⁵ Because of the rich parallelism at the data level, it is relatively straightforward to parallelize the code for 16 SPEs. We offloaded parallelizable sections to SPEs as two sets of SPE tasks to make the code size smaller and to allocate more space for data. The PPE performs sequential sections: the file read/write and the optimizer.

The SPE does not have a hardware-managed cache, and therefore the code and data must fit in its 256 KB local store. Parallel programs in a traditional single program, multiple data (SPMD) model often partition the data into small pieces explicitly to make them fit in the data cache, but they typically rely on an instruction cache to manage the code on the on-chip memory. The SPE lacks a cache memory, however, so the code must be portioned explicitly to fit in the local store. This can typically be done by portioning SPMD tasks into multiple sets. This divides the code into multiple pieces and more space can be allocated in the local store for data.

3.2 SIMD pipeline optimization

The SPE is highly pipelined and optimized for SIMD operations. Because a program with frequent branches or scalar operations does not fully exploit the high performance of the SPE, it is important not only to perform multiple operations using SIMD instructions, but also to optimize the code to exploit the architecture. We optimized the registration code by using intrinsics, which allow the programmer to state SIMD instructions explicitly in C/C++ programs. Such restructuring is called SIMDization.

For example, a branch operation can take 18 to 19 clock cycles when the branch prediction fails. Because of this, translating conditional branches to conditional moves can improve the performance. Our optimization techniques using two examples from our code are described in the sections that follow.

3.2.1 Image down sampler

The image down sampler performs a two-dimensional (2D) 5 x 5 Gaussian convolution filter. A typical implementation uses two stages of one-dimensional (1D) convolution filters.⁶ Using SIMD intrinsics, we implemented an optimized one-stage 2D convolution filter, which is approximately 8.7 times faster than a two-stage version in C. This performance gain is a combination of parallel arithmetic operations with SIMDization, load/store operations aligned with the SIMD vector size (16 bytes) and a better use of the large Cell/B.E. register file (128 vector registers).

3.2.2 Gradient computation

The mutual-information computation includes several loops with a small loop bound. One example is a gradient computation, which obtains the spatial gradient of the pixel value (Example 1). The scalar version determines whether the pixel index is within a valid range for each dimension. If so, the code computes the gradient value by dividing the difference of two adjacent pixel values with their spatial distance. If it is not in a valid range, the code assigns zero for the gradient value of that dimension.

Example 1 Pseudo code to compute a gradient of pixel values: Original scalar code

```
1  /*
2  index[dim](dim=0..2) holds the pixel index.
3  Limit[dim](dim=0..2) holds the maximum value of the valid index.
4  Pixels[i](i=0..7) holds the pixel value of eight corners of a cube
5  Space[dim](dim=0..2) holds the length of the perimeter of the cube
   for each dimension
6  */
7  i=1;
8  for (dim=0;dim<3;dim++){
9      if (index[dim]<1 || index[dim]>limit[dim]){
10         gradient[dim]=0.0;
11     } else {
12         gradient[dim]=(pixels[i] - pixels[0])/space[dim];
13     }
14     i *= 2 ;
15 }
```

Because of the length of the SPE pipeline, the scalar code severely penalizes performance. To overcome that limitation, we converted this code with SIMD intrinsics (Example 2). The SIMDized code uses a conditional move instruction (`spu_sel` on line 14 in Example 2) to implement conditional operations. This instruction uses the third parameter (`mask3`) to select the first vector (`v2`) or the second vector (`c0`) for each vector element. This type of conversion can accelerate SPE performance significantly by removing branch penalties. For this particular example, the performance was 4.2 times higher.

Example 2 Pseudo code to compute a gradient of pixel values: Optimized code with SIMD intrinsics (shown in bold)

```
1  /*
2  vector float pixel0 hold the value of pixels[0] in the first three
   elements
3  vector float pixel1 holds pixel[1], pixels[2] and pixels[4] in the
   first three elements
4  vector float inverse_space holds the inverse of the
   space[dim](dim=0..2) in the first three elements
5  The fourth element of these vector variables is not used
6  */
7  constant vector signed int c1 = {1,1,1,1};
8  const vector float c0 = {0.0,0.0,0.0,0.0};
9  vector float v1 = spu_sub(pixels1, pixels0);
10 vector float v2 = spu_mul(v1, inverse_space);
11 vector unsigned int mask1 = spu_cmpgt(c1, index);
12 vector unsigned int mask2 = spu_cmpgt(index, limit);
13 vector unsigned int mask3 = spu_or(mask1, mask2);
14 vector float gradient = spu_sel(v2, c0, mask3);
```

3.3 Optimized data partitioning: Localized sampling and speculative packing

To exploit the Cell/B.E. processor even further, we partitioned the data so that the available memory bandwidth could be used more efficiently. Because the original algorithm randomly samples pixels, a naive implementation accesses the memory at scattered locations, which can result in very poor bus utilization.

A “brick” caching scheme has been proposed to improve memory usage for the field-programmable-gate-array-based 3D image processor.⁷ This scheme partitions the fixed image into a set of cuboids. To compute the mutual information for a cuboid, it maps the cuboid to the moving image space and computes its axis-aligned bounding box. The pixels must be fetched in two cuboids—one for the fixed image and the other for the bounding box—to compute the mutual information for the fixed-image cuboid. The bounding box can be fetched from the memory much more efficiently than individual moving-image pixels.

The brick scheme has two limitations when the pixels for computing mutual information are randomly sampled. First, the original algorithm randomly samples pixels from the entire fixed image and packs them in a contiguous memory space before computing the optimization loop to maximize the mutual

information, which prevents the direct gathering of the samples within a cuboid. Second, because this scheme fetches all pixels in a cuboid, it wastes memory when the sampling ratio (the number of samples per pixel) is small. For example, our sample is 1% of the fixed-image pixels in our algorithm. Typically, eight moving image pixels per fixed-image pixel are used for interpolation, so at least 92% of the fetched moving-image pixels can be wasted.

We designed and implemented two strategies to access fixed and moving-image pixels: localized sampling and speculative packing. The sampling ratio determines which strategy the registration dynamically selects. When the ratio is high (64% or higher), the registration selects the localized sampling scheme. It partitions the fixed-image cuboid into a set of stripes (rectangles), samples pixels for each stripe and packs them in a contiguous memory space (Figure 2). We chose to sample pixels per stripe instead of cuboid because available memory is used most efficiently when the volume size of the bounding box is maximized.⁸ The size of the bounding box is a function of the spatial transformation parameter and the size of the fixed-image cuboid; however, the spatial transformation parameter must be obtained before the size of the fixed-image cuboid can be determined to avoid an overflow of the bounding box by the local store. Localized sampling allows us to adjust the height of the fixed-image cuboid (that is, the number of stripes) dynamically every time the spatial transformation parameter changes because samples of each stripe are packed separately.

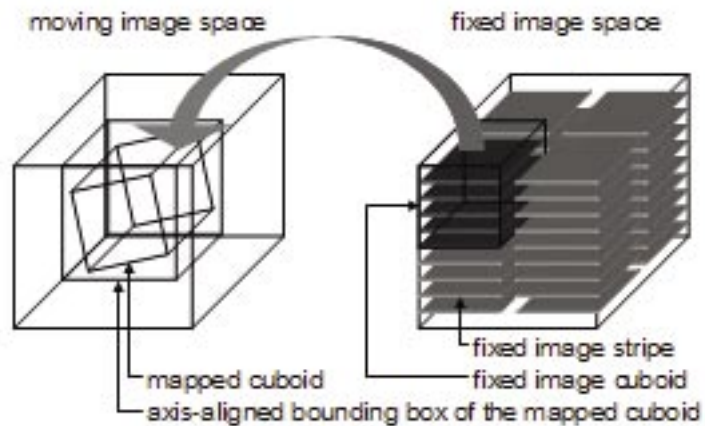


Figure 2 The localized sampling scheme

When the sampling ratio is low (less than 64%), the registration process uses speculative packing. For the first computation of the mutual information at each resolution, the process fetches scattered moving-image pixels and packs them into a contiguous memory space. Although this is a time-consuming process, it amortizes the cost because it is done only once for each resolution. For the second computation of the mutual information, the moving-image pixels can be fetched efficiently with DMA operations because they are in the contiguous memory space. However, the transformation parameter between the fixed and moving-image spaces changes as the registration proceeds and the packed moving-image pixels do not necessarily represent the set of pixels that must be accessed. More specifically, if a fixed image is mapped within the same $2 \times 2 \times 2$ pixel cube in the moving-image space as the previously mapped cube (called hit), we use the previously packed pixels to interpolate the pixel value at the mapped position, in the same way that the original algorithm does (Figure 3).

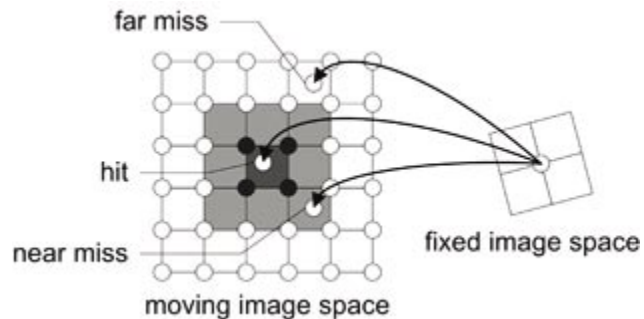


Figure 3 The speculative packing technique when the near-miss threshold is one, illustrated in a 2D space for simplicity

If the fixed image is mapped far from the $2 \times 2 \times 2$ pixel cube (called far miss), the correct cube is fetched from the system memory. If a fixed image is mapped outside the $2 \times 2 \times 2$ pixel cube but nearby (called near miss), the process extrapolates the pixel value from the previously used cube. Therefore, when a near miss occurs, this scheme can save the extra memory accesses but can also affect the result. The near-miss threshold then becomes the maximum discrepancy of the pixel position in each dimension where a pixel can be classified as a near-miss.

4. Experimental results

We ran our program on an IBM BladeCenter QS20, which employs two Cell/B.E. processors at 3.2 GHz with 1 GB memory. We used SDK 1.1 compilers (xlc for SPE) and libraries. We also ran a sequential version of our program on one core of an Intel® Xeon® 5160 processor at 3.0 GHz with 4 GB memory. We used an Intel compiler (ICC 9.1) with the “-fast” option, which automatically uses the SIMD unit (SSE3) on the Xeon 5160. This code was not tuned for the SIMD unit or the memory hierarchy of Intel Xeon. We compared the performance of our program on the two platforms only to show the performance gain that we can achieve by optimizing a naive implementation for the Cell/B.E. architecture.

In our experiments, we used a series of 97 deidentified image sets, which are clinical MRI images that were collected at the Mayo Clinic after Institutional Review Board approval, including T1, T2 and fluid-attenuated inversion recovery (FLAIR) images of the brain, with a matrix size of 256 x 256 or 512 x 512 pixels in plane and between 12 and 379 slices in the Z-dimension. Examinations consisted of at least three contrast types, and at least two examinations separated by 2 months were included. All registration pairs were from a single individual.

4.1 Overall Registration Time

Figure 4 shows the overall registration time on the two CBE processors, which include the file I/O. The registration time is roughly proportional to the number of slices in the fixed image. This is because the number of pixels in the fixed image is the primary factor to determine the dominant component in the registration, which is the mutual information computation. For fixed images with 256x256 slices, the longest execution time is 1.6 seconds when the number of slices is 56. For those with 512x512 slices, the longest execution time is 20 seconds when the number of slices is 379.

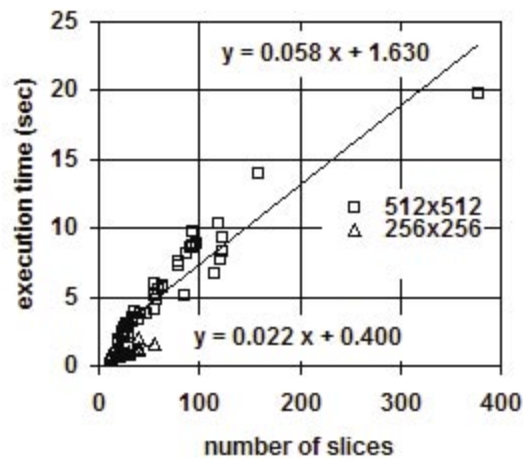


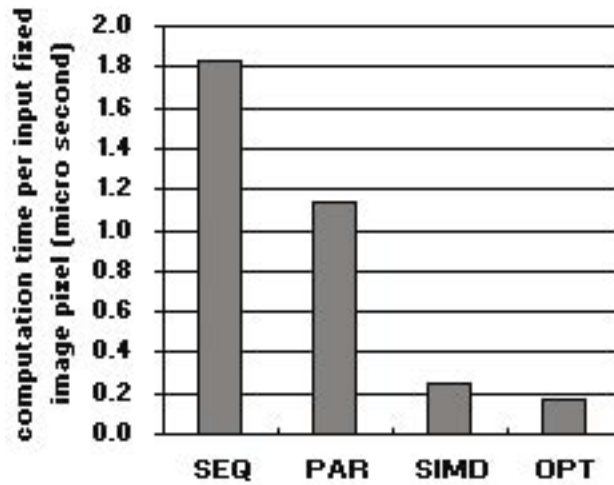
Figure 4 The total execution time on two CBE processors

4.2 Performance Gain

4.2.1. Computation time

The graph in Figure 5 shows the computation time (excluding the I/O time) per fixed-image pixel for four sets of experiments. We gathered the total computation time to perform a registration for all 97 pairs and divided it by the total of input fixed-image pixels. In the graph, the bar labeled SEQ corresponds to the Xeon 5160, which performs the registration sequentially by using one processor core. The remaining bars show the registration time with two Cell/B.E. processors: PAR corresponds to a parallelized version for 16 SPEs; SIMD corresponds to a SIMDized version in addition to the parallelization; and OPT corresponds to a version with optimized data partitioning in addition to the SIMDization and the parallelization. A function for computing the mutual-information dominates the computation time.

The graph does not include the file I/O time. In our environment on the QS20, the average I/O time per input fixed-image pixel was $0.17 \mu\text{s}$. Because that average is about the same as the computation time for the most optimized case, overlapping I/O operations with the computation during the back-to-back processing of multiple data sets hide the bulk of the I/O time.



Fixed and Moving Image Size	Computation Time (sec)		Speed-up Ratio
	SEQ	OPT	
256x256x30	3.43	0.59	5.8
512x512x60	31.85	2.45	13

Label	Processor	Restructured/Optimized Level
SEQ	Xeon 5160 3.0GHz	Sequential (single thread)
PAR	Cell/B.E.x2 3.2GHz	Parallelized
SIMD	Cell/B.E.x2 3.2GHz	Parallelized and SIMDized
OPT	Cell B.Ex.2 3.2GHz	Parallelized, SIMDized and optimized Data Portioning (Localized Sampling and Speculative Packing)

Figure 5 The computation time results. The graph shows the computation time per input-fixed image pixel in microseconds (the lower is the better). The first table shows the absolute execution time of SEQ and OPT cases for two examples. With parallelization, SIMDization and optimized data partitioning, OPT is about 11 times faster than SEQ. The speed-up ratio generally increases as the input size increases.

Parallelizing the code results in an acceleration of the computation time that is only 1.6 times that of a sequential version (Xeon 5160). SIMDizing the code improved the performance an additional 4.5 times. When we optimized the data partitioning, we improved the performance by 1.5 times. In total, our registration program on two Cell/B.E. processors at 3.2 GHz runs about 11 times faster than a sequential version on the Xeon 5160 at 3.0 GHz. For each individual run, the larger the image size, the larger the speed-up ratio becomes. This is because our static partitioning of the workload decreases the number of times that SPEs synchronize with each other.

4.2.2 Memory traffic

Figure 6 shows the memory traffic reduction that is a result of our localized sampling and speculative packing techniques. The amount of the memory traffic is shown as the number of cache lines transferred between the system memory and the local stores for the entire execution of the program and is normalized with the base case.

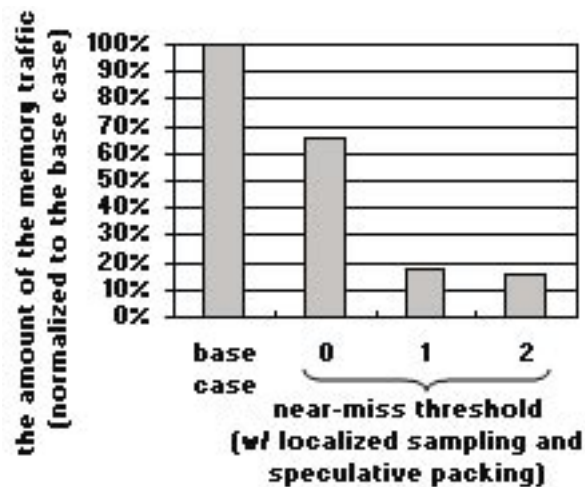


Figure 6 The memory traffic reduction with localized sampling and speculative packing techniques. The base case shows the memory traffic when neither is used, and the others show the memory traffic for three different near-miss thresholds.

When we increased the threshold for speculative packing, the amount of memory traffic decreased because the number of far misses decreased. When the threshold is 1, the amount of the traffic is reduced by 82%. When we increase the threshold beyond 1, however, the amount of the memory traffic does not decrease significantly. This indicates that, for all dimensions, most of the fixed-image pixels are mapped to a position within one pixel pitch from the initial position that was given by the initial transformation parameter for the resolution level. Therefore, the registration process at each resolution level obtains a good estimate of the transformation parameter for the next higher resolution level. At the first resolution level, this estimate is not necessary because the localized sampling technique is used, and it does not rely on the initial estimate to reduce the amount of memory traffic.

We also compared the consistency distance between two registration results: those with and without our localized sampling and speculative packing techniques. The consistency distance indicates the consistency between the transformation parameters obtained from a forward registration and ones obtained from a backward registration. We computed the consistency distance by applying the forward and backward transformations to the eight corners of the fixed-image space and by averaging the distance between the original and transformed positions for the eight corners. The result of our computations indicated that our optimization techniques for the memory bandwidth do not affect the statistical characteristics of the consistency distance significantly.

5. Discussions and conclusions

We have shown that exploiting the multiple processing cores of the Cell/B.E. processor can accelerate the image registration algorithm significantly. To achieve high performance, however, it is critical to restructure the program to utilize the SIMD unit and the available memory efficiently. Performance improved 4.5 times when we did this. Our localized sampling and speculative packing techniques further reduced the amount of the memory traffic by 82%. As a result, our optimized code on two Cell/B.E. processors at 3.2 GHz is about 11 times faster than a naive sequential code on the Xeon 5160 at 3.0 GHz.

A more thorough study is necessary to compare the performance of our implementation against different processor architectures, such as shared-memory-based multi-cores and graphics processing units (GPUs). Although a study has shown that the Cell/B.E.-based approach outperforms GPU-based approaches for a backprojection algorithm, the performance comparison for registration is still an open question.⁹

Mutual information is also applicable to deformable registration algorithms (non-rigid registration algorithms). We would also like to explore the applicability of the Cell/B.E. architecture, our programming technique and our data-partitioning techniques—localized sampling and speculative packing—to deformable registration algorithms and other architectures.

6. Acknowledgement

The authors wish to thank Liqin Wang at Mayo Clinic for providing us a registration program that we used as a base.



¹ B. J. Erickson, J. W. Patriarche, C. P. Wood, N. G. Campeau, E. P. Lindell, V. Savcenko, N. Arslanlar and L. Wang, "Image Registration Improves Confidence and Accuracy of Image Interpretation." Accepted in Cancer Informatics; B. J. Erickson, J. Mandrekar, L. Wang, J. W. Patriarche, B. J. Bartholmai, C. P. Wood, E. P. Lindell, A. Sykes, G. F. Harms, R. M. Lindell and K. Andirole, "Effect of Automated Image Registration on Radiologist Interpretation." To appear in Journal of Digital Imaging.

² C. R. Castro-Pareja, J. M. Jagadeesh and R. Shekhar, "FAIR: A Hardware Architecture for Real-Time 3-D Image Registration," IEEE Transactions on Information Technology in Biomedicine. Vol. 7, No. 4 (Dec. 2003): 426-434; L. Jianchun, R. Shekhar and C. Papachristou, "A 'Brick' Caching Scheme for 3D Medical Imaging," 2004 IEEE International Symposium on Biomedical Imaging: Macro to Nano. Vol. 1 (Apr. 2004): 563-566; T. Rohlfing and C. R. Maurer, "Nonrigid Image Registration in Shared-Memory Multiprocessor Environments with Application to Brains, Breasts, and Bees," IEEE Transactions on Information Technology in Biomedicine. Vol. 7, no. 1 (Mar. 2003): 16-25; S. K. Warfield, F. Jolesz and R. Kikinis, "A High Performance Approach to the Registration of Medical Imaging Data," Parallel Computing. Vol. 24, no. 9-10 (1998): 1345-1368.

³ J. A. Kahle, M. N. Day, H. P. Hofstee, C. R. Johns, T. R. Maeurer and D. Shippy, "Introduction to the Cell Multiprocessor," IBM Journal of Research & Development. Vol. 49, no. 4/5 (2005): 589-604.

⁴ D. Mattes, D. R. Haynor, H. Vesselle, T. K. Lewellen and W. Eubank, "PET-CT Image Registration in the Chest Using Free-Form Deformations," IEEE Transactions on Medical Imaging, Vol. 22, no. 1 (Jan. 2003): 120-128; L. Ibanez, W. Schroeder, L. Ng and J. Cates, The ITK Software Guide, Second Edition, New York: Kitware Inc., 2005.

⁵ G. Amit, Y. Caspi, R. Vitale and A. T. Pinhas, "Scalability of Multimedia Applications on Next-Generation Processors," 2006 IEEE International Conferences on Multimedia & Expo, 17-20.

⁶ Kahle, Day, Hofstee, Johns, Maeurer and Shippy, "Introduction," 589-604.

⁷ Jianchun, Shekhar, Papachristou, "Brick," 563-566.

⁸ Jianchun, Shekhar, Papachristou, "Brick," 563-566.

⁹ Marc Kachelriess, Michael Knaup, Olivier Bockenbach, "Hyperfast Parallel-Beam Backprojection," 2006 Nuclear Science Symposium Medical Imaging Conference, M14-168, Oct. 2006.

© 2007 International Business Machines Corporation

IBM Systems and Technology Group
Route 100
Somers, NY 10589
U.S.A.

Produced in the United States of America
05-07
All Rights Reserved.

IBM, the IBM logo, BladeCenter and Power PC are the trademarks of IBM Corporation.

Intel and Intel Xeon are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

Cell Broadband Engine is a trademark of Sony Computer Entertainment Inc. in the United States, other countries, or both.

Other company, product, or service names may be trademarks or service marks of others.

References in this publication to IBM products or services do not imply that IBM intends to make them available in all countries in which IBM operates.

The IBM home page on the Internet can be found at **ibm.com**