RC24138 (W0612-055) December 13, 2006
Computer Science

IBM Research Report

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

Moriyoshi Ohara[1], Hangu Yeo[2], Frank Savino[2], Giridharan Iyengar[3], Leiguang Gong[3], Hiroshi Inoue[1], Hideaki Komatsu[1], Vadim Sheinin[2], Shahrokh Daijavad[4], Bradley Erickson[5]

[1]IBM Tokyo Research Laboratory
Yamato, Kanagawa 242-8502
Japan

[2]IBM Research Division
Thomas J. Watson Research Center
P.O. Box 218
Yorktown Heights, NY 10598

[3]IBM Research Division
Thomas J. Watson Research Center
P.O. Box 704
Yorktown Heights, NY 10598

[4]IBM Systems & Technology Group
Hawthorne, NY 10532

[5]Mayo Clinic and Foundation
Rochester, MN 55905

Research Division
Almaden - Austin - Beijing - Haifa - India - T. J. Watson - Tokyo - Zurich

REAL-TIME MUTUAL-INFORMATION-BASED LINEAR REGISTRATION ON THE CELL BROADBAND ENGINE PROCESSOR

Moriyoshi Ohara[1], Hangu Yeo[2], Frank Savino[2], Giridharan Iyengar[3], Leiguang Gong[3], Hiroshi Inoue[1], Hideaki Komatsu[1], Vadim Sheinin[2], Shahrokh Daijavad[4], and Bradley Erickson[5]

[1]IBM Tokyo Research Laboratory, Yamato, Kanagawa 242-8502 Japan
[2]IBM T.J. Watson Research Center, Yorktown Heights, NY 10598 USA
[3]IBM T.J. Watson Research Center, Hawthorne, NY 10532-1596 USA
[4]IBM Systems & Technology Group, Hawthorne, NY 10532-1596 USA
[5]Mayo Clinic and Foundation, Rochester, Minnesota 55905 USA

ABSTRACT

Emerging multi-core processors are able to accelerate medical imaging applications by exploiting the parallelism available in their algorithms. We have implemented a mutual-information-based 3D linear registration algorithm on the Cell Broadband Engine™ (CBE) processor, which has nine processor cores on a chip and has a 4-way SIMD unit for each core. By exploiting the highly parallel architecture and its high memory bandwidth, our implementation with two CBE processors can compute mutual information for about 33 million pixel pairs in a second. As a result, it can register a pair of 256x256x30 3D images in less than one second by using a multi-resolution method. This implementation is significantly faster than a conventional one on a traditional microprocessor or even faster than a previously reported custom-hardware implementation. This paper describes our implementation with a focus on localized sampling and speculative packing techniques, which reduce the amount of the memory traffic by 82%.

Keywords: Image registration, Biomedical image processing, and Parallel processing

1. INTRODUCTION

Image registration is a process to align two sets of images, typically obtained at different times or by using different imaging devices, and plays an increasingly important role in clinical applications [1]. While mutual-information-based image registration is known to be effective, it is computationally expensive. Thus, to accelerate the registration process, custom hardware approaches [2][3] and supercomputer-based approaches [4][5] have been proposed.

We have implemented a mutual-information-based linear registration algorithm on the Cell Broadband Engine (CBE) processor [6]. This is an asymmetric multi-core processor with a high peak performance; it combines eight synergistic processing elements (SPEs) and a Power Processing Element (PPE), which is a general-purpose IBM® PowerPC® core. Two of these processors can be connected via a high speed bus (e.g. IBM BladeCenter® QS20) as shown in Figure 1, where 16 SPEs can run in parallel. Each SPE, furthermore, has a SIMD unit, which can perform a floating or integer operation on four data elements at every clock.

To accelerate the image registration on this processor, one has to optimize the program to exploit the parallelism at both task and instruction levels and to use the memory bandwidth efficiently. First, at a task level, one needs to partition the program into multiple tasks that fits in the local store on each SPE. Unlike conventional microprocessors, each SPE does not have a hardware cache memory to manage a small on-chip local store. Thus, one can view this architecture as a distributed memory multiprocessor with a very small local memory attached to a large shared memory.

At an instruction level, moreover, one typically needs to restructure frequently executed sections of the task by using intrinsics, which allow the programmer to state SIMD instructions explicitly in C/C++ programs. Such restructuring is called SIMD-ization.

Finally, for utilizing the memory bandwidth efficiently, one has to partition the data in such a way that each task can transfer it in a large block, such as one or multiple cache lines (128B per cache line), by using a DMA engine. Pixel-wise memory accesses, for example, result in a very poor usage of the memory bandwidth.

2. MUTUAL-INFORMATION BASED LINEAR REGISTRATION ALGORITHM

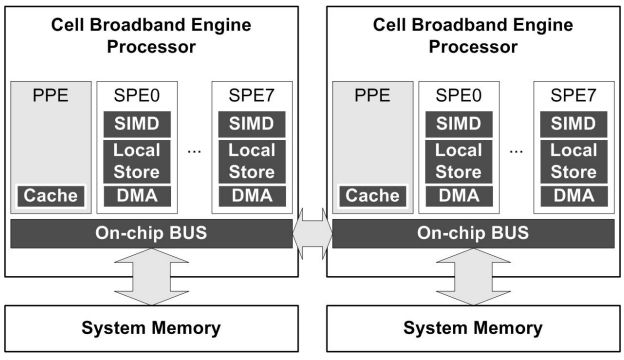


Figure 1: The high-level structure of a dual CBE system. Each synergistic processing element (SPE) has a SIMD engine, a high-speed local store, and a DMA engine.

We used a Mattes's mutual-information-based multi-resolution algorithm [7] implemented in ITK [8] as a base. This algorithm solves the registration problem as an optimization problem to find the spatial transformation that aligns one image (the moving image) to another image (the fixed image); it traverses 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, it uses a multi-resolution method. That is, it starts from 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. To reduce the computation time, furthermore, this algorithm computes the mutual information only for a small subset of sample pixels from the fixed image. 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 the 1% of the pixels at the original resolution.

This algorithm is implemented in two components: preprocessor and registrator. The preprocessor includes a pyramid filter to compute lower-resolution images to be used for the multi-resolution registration method. It also randomly samples fixed image pixels for each resolution of the image, which the registrator uses for computing the mutual-information. The registrator tries to find the spatial transformation between the two images that maximizes the mutual-information value by iteratively changing the transformation parameters. We parallelized both of the preprocessor and registrator to exploit the 16 SPEs. We also SIMD-ized frequently executed sections of the code, such as the sections for affine transformations, linear interpolations, and Gaussian filters.

3. OPTIMIZED DATA PARITIONING: LOCALIZED SAMPLING AND SPECULATIVE PACKING

To exploit the CBE processor, it is critical to partition the data to utilize the memory bandwidth efficiently. Since the original algorithm randomly samples pixels, a naive implementation accesses the memory at scattered locations, which can result in very poor bus utilization.

Jianchun, et al. proposed a "brick" caching scheme to improve the efficiency in the memory bandwidth usage for FPGA-based 3D image processor [3]. 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 the bounding box of the mapped cuboid. One needs to fetch the pixels in the two cuboids – one for the fixed image and the other for the bounding box – to compute the mutual information for the fixed image cuboid. One can fetch the bounding box from the memory much more efficiently than individual moving image pixels that correspond to each fixed image pixel.

This scheme is, however, limited in the following two ways when pixels for computing mutual information are randomly sampled. First, since the sample pixels are packed in a contiguous memory space at each resolution level during the preprocessing, during the main computation, one cannot directly address them by using the physical position of the cuboid. Second, since this scheme fetches all pixels in a cuboid, it wastes the memory bandwidth when the sampling ratio (the number of samples per pixel) is small. It can be for instance 1% of the fixed image pixels in our algorithm. Since we typically uses eight moving image pixels per fixed image pixel for interpolation, at least 92% of the fetched moving image pixels can be wasted when the moving and fixed images have the same resolution.

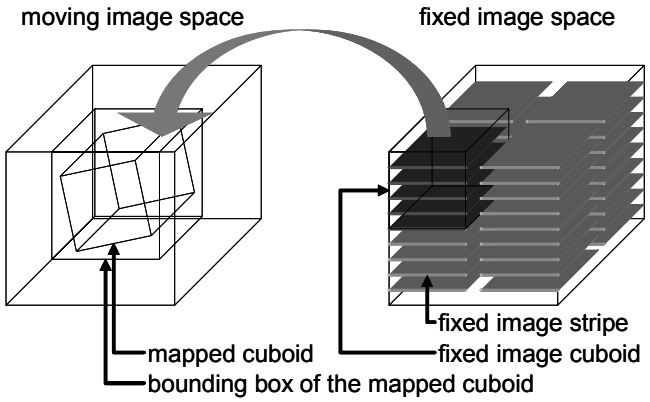


Figure 2: The localized sampling scheme. We sample pixels for each stripe. This allows us to adjust the number of stripes in the fixed image cuboid to make the bounding box fit in the local store.

We designed and implemented two strategies to access fixed and moving image pixels: (1) localized sampling scheme and (2) speculative packing scheme. The registration system dynamically selects one strategy depending on the sampling ratio. When the sampling ratio is high, the localized sampling scheme is selected which partitions the fixed image into a set of stripes (i.e. rectangles) as shown Figure 2. This scheme samples pixels for each stripe and packs them in a contiguous memory space. We chose to sample pixels per stripe instead of cuboid because of the following reason. It is known that we can utilize the memory bandwidth most efficiently by maximizing the volume size of the bounding box [3]. Since the size of the bounding box is a function of the spatial transformation parameter and the size of the fixed image cuboid, however, we cannot determine the size of the fixed image cuboid to avoid overflowing the bounding box from the local store until we obtain the spatial transformation parameter. The localized sampling scheme allows us to adjust the height of the fixed image cuboid (i.e. the number of stripes) dynamically every time when the spatial transformation parameter changes because samples of each stripe are packed separately.

When the sampling ratio is low, on the other hand, the speculative packing scheme is used. For the first computation of the mutual information at each resolution, we compute the mutual information by fetching scattered moving image pixels. Then, we pack them in a contiguous memory space. While this is a time consuming process, we can amortize the cost since we do this only once for each resolution. When we compute the mutual information for the second time, we can efficiently fetch moving image pixels with DMA operations because they are densely packed in a contiguous memory space. Since the transformation parameter between the fixed and moving image spaces changes as the registration proceeds, however, the packed moving image pixels do not necessarily represent the set of pixels we need to access. More specifically, as shown in Figure 3, if a fixed image is mapped within the same 2x2x2 pixel cube in the moving image

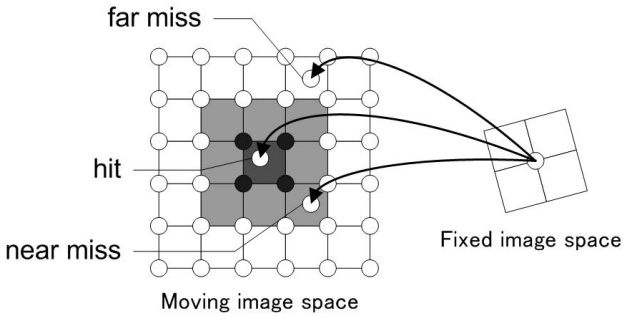


Figure 3: The speculative packing technique when the near-miss threshold is one, illustrated in a 2D space for simplicity. Black pixels in the moving image were packed previously. If a fixed image pixel is mapped to a position within the near-miss threshold from the packed pixels, we use them to interpolate or extrapolate the pixel value at the mapped position. If a fixed image pixel is outside the near-miss area, we fetch correct pixels from the memory and replace the previously packed pixels with them.

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. If a fixed image is mapped outside the 2x2x2 pixel cube but close enough (called *near miss*), we use the previously used cube and extrapolate the pixel value. If the fixed image is mapped far from the 2x2x2 pixel cube (called *far miss*), however, we simply fetch the correct cube from the system memory. Thus, when a near miss occurs, this scheme can save the extra memory accesses but can also affect the result. We define the near-miss threshold as the maximum discrepancy of the pixel position in each dimension that we allow a pixel to be classified as a near-miss. We will discuss the tradeoff between the performance gain and the accuracy in the next section. In our experiments, we used the localized sampling when the sampling ratio is 64% and used the speculative packing when the ratio is lower than that.

4. EXPERIMENTAL RESULTS

We ran our program on an IBM BladeCenter QS20, which employs two CBE processors at 3.2GHz with 1GB 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 Intel Xeon™ 5160 processor at 3.0GHz (Woodcrest) with 4GB memory. We used Intel® compiler (ICC 9.1) with "-fast" option, which automatically utilizes the SIMD unit (SSE3) on Woodcrest. Note that this code is not tuned for the SIMD unit or the memory hierarchy of Intel Xeon. We compare the performance of our program on the two platforms only to show the performance gain we can achieve by optimizing a naive implementation for the CBE architecture.

In our experiments, we used a series of 98 image sets, which are clinical MRI images collected at Mayo Clinic after IRB approval, consisting of T1, T2, and Fluid-attenuated inversion recovery (FLAIR) images of the brain, with a matrix size of 256x256 pixels in plane and between 30 and 48 slices in the Z-dimension. Exams consisted of at least 3 contrast types, and at least two examinations separated by 2 months were included. All registration pairs were from a single individual.

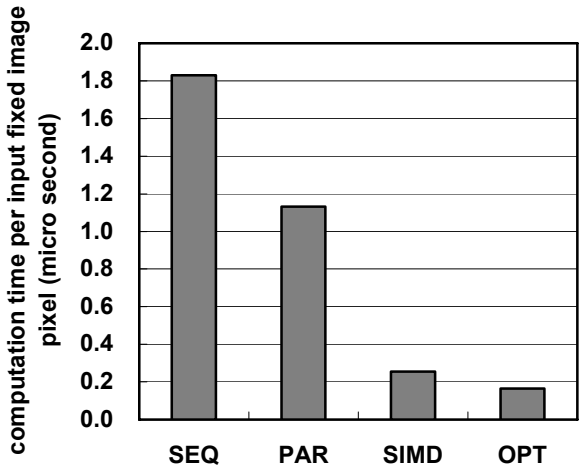
4.1. Computation Time

Figure 4 shows the computation time per fixed image pixel for four sets of experiments. We gathered the total computation time to perform a linear registration for all the 98 pairs and divide it by the total of the number of input fixed image pixels.

The left most bar (SEQ) corresponds to Woodcrest, which performs the registration sequentially by using one processor core. The right three bars show the registration time with 2 CBE processors. The third from the right (PAR) corresponds to a parallelized version for 16 SPEs. The second from the right (SIMD) corresponds to a SIMD-ized version in addition to the parallelization. The right most bar (OPT) corresponds to a version with optimized data partitioning in addition to the SIMD-ization and the parallelization. A function to compute the mutual-information dominates the computation time.

Note that this graph does not include the file IO time since we did not focus on it in this study. In our experimental environment on QS20, the averaged IO time per input fixed image pixel was 0.17 micro seconds. Since it is about the same as the computation time for the most optimized case (OPT), we should be able to hide the bulk of the IO time by overlapping IO operations with the computation when we process multiple data sets in a back-to-back fashion.

As shown in Figure 4, by parallelizing the code, we can accelerate the computation time only by 1.6X with 16 SPEs over a sequential version (Woodcrest). By SIMD-izing the code, we improved the performance additionally by 4.5X. We obtained this performance gain by optimizing the code with SIMD intrinsics in



Label	Processor	Restructured/Optimized Level
SEQ	Woodcrest 3.0GHz	Sequential (single thread)
PAR	CBEx2 3.2GHz	Parallelized
SIMD	CBEx2 3.2GHz	Parallelized and SIMD-ized
OPT	CBEx2 3.2GHz	Parallelized, SIMD-ized and Optimized Data Portioning (Localized Sampling and Speculative Packing)

Figure 4: The computation time per input fixed image pixel in micro second (the lower is the better). The registrator dominates the total computation time. With parallelization, SIMD-ization and optimized data partitioning, we have reduced the computation time to 0.17 micro seconds per input fixed image pixel.

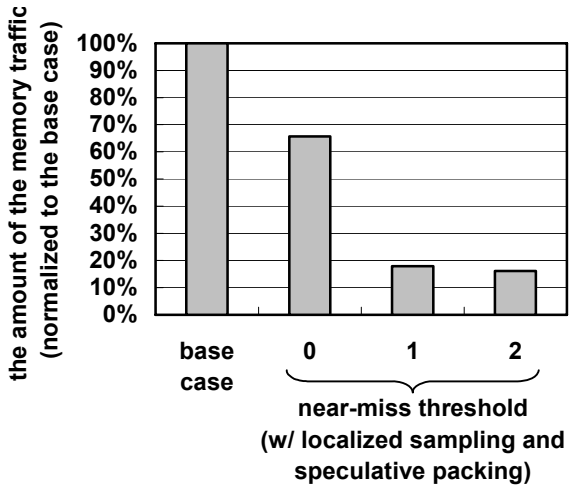


Figure 5: The memory traffic reduction with localized sampling and speculative packing techniques. The left most bar (base case) shows the memory traffic without them, and the three right bars show the memory traffic with them for three different near-miss thresholds. When the near-miss threshold is one, our techniques reduce the amount of the traffic by 82% from the base case.

various ways. For example, we converted some conditional branch operations to conditional move operations to eliminate the branch overhead. By optimizing the data partitioning, further more, we improved the performance by 1.5X. In total, our registration program on two CBE processors at 3.2GHz runs about 11X faster than a sequential version on Woodcrest at 3.0GHz.

4.2. Memory Traffic

Figure 5 shows the memory traffic reduction due to 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. As we increase the threshold for speculative packing, the amount of the memory traffic decreases since the number of *far misses* decreases. When the threshold is one, the amount of the traffic is reduced by 82%. When we increase the threshold beyond one, however, the amount of the memory traffic does not decrease significantly. This indicates that most of the fixed image pixels are mapped to a position within a one pixel pitch for all dimensions from the initial position given by the initial transformation parameter for the resolution level.

We 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 resignation 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. While not shown here due to the space limitation, our experimental result indicates that our optimization techniques for the memory bandwidth do not affect the statistical characteristics of the consistency distance significantly.

5. FUTURE WORK

A more thorough study is necessary to compare the performance characteristics of different processor architectures. We would also like to explore the applicability of the CBE architecture to non-rigid registration algorithms.

6. CONCLUSION

We have shown that the CBE processor can accelerate the image registration algorithm significantly by exploiting its multiple processing cores. To achieve the high performance, it is critical to restructure the program to utilize the SIMD unit and the memory bandwidth efficiently. We obtained 4.5X performance gain by restructuring the program for utilizing the SIMD unit. Our localized sampling and speculative packing techniques furthermore reduced the amount of the memory traffic by 82%. As a result, our optimized code on two CBE processors at 3.2GHz is about 11X faster than a naive sequential code on Woodcrest at 3.0GHz.

8. REFERENCES

[1] B.J. Erickson, J.W. Patriarche, C.P. Wood, N.G. Campeau, E.P. Lindell, V. Savcenko, N. Arslanlar, L. Wang, "Image Registration Improves Confidence and Accuracy of Image Interpretation," Accepted in *Cancer Informatics*.

[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, pp. 426-434, Dec. 2003.

[3] 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, pp. 563-566, Apr. 2004.

[4] T. Rohlfing, 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, pp. 16- 25, Mar. 2003.

[5] 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, pp. 1345–1368, 1998.

[6] 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, pp. 589–604.

[7] 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, pp. 120-128, Jan. 2003.

[8] L. Ibanez, W. Schroeder, L. Ng, J. Cates, *The ITK Software Guide Second Edition*, Kitware Inc., New York, 2005.