ELSEVIER

Parallel Computing 24 (1998) 1369–1383

PARALLEL COMPUTING

MIMD vs. SIMD parallel processing: A case study in 3D medical image registration

Gary E. Christensen [1,2]

Department of Electrical and Computer Engineering, The University of Iowa, Iowa City, IA 52242, USA

Received 15 January 1998; revised 15 April 1998

Abstract

This paper compares and contrasts the issues involved with implementing computationally intensive medical imaging algorithms on a multiple-instruction, multiple-data (MIMD) and single-instruction, multiple-data (SIMD) parallel processing computers. Implementation issues and timing analysis are presented for a 3D medical image registration algorithm implemented on a 16 processor SGI Challenge MIMD computer and on a 128×128 processor MasPar SIMD computer. The MIMD implementation is shown to have nearly N-times performance improvement with respect to a single processor when using N processors. It is also shown that the MIMD implementation is a minimum of four times faster than the SIMD implementation.

Keywords: Parallel processing; Global shape models; Imaging registration; 3D imaging; High speed computing

1. Introduction

Medical imaging systems collect a massive amount of data in a short time and require fast computers for both image reconstruction and image analysis. Parallel processors in medical imaging are useful for real-time or near-real-time processing. Real-time imaging provides the physician or technician the ability to see the results of an imaging scan while the patient is still present. If the scan is corrupted due to patient motion or some other reason, it can be reacquired immediately without the

[1] E-mail: gary-christensen@uiowa.edu.

[2] This work was supported in part by the NIH grant NS35368 and a grant from The Whitaker Foundation.

0167-8191/98/$ – see front matter
PII: S0167-8191(98)00062-3

need for scheduling an additional scan afterwards. If the scan shows an abnormality it would also be possible to acquire another scan at a higher resolution or over a larger field of view. In the case of functional scans such as positron emission tomography (PET), single photon emission computed tomography (SPECT), and functional magnetic resonance (fMRI), immediate feed back can be used to modify the current task under examination. In each of these cases, the instant or near-instant feedback provides a tremendous cost savings in scanner time. This reduction can be used to offset the possible increased cost of a high-performance parallel computer.

Investigators [3] at Carnegie Mellon University, University of Pittsburgh Medical Center and the Pittsburgh Supercomputing Center recently demonstrated near-real-time processing of fMRI data using a Cray T3E massively parallel computer and a high-speed computer network. A subject was scanned inside an MRI scanner at the University of Pittsburgh Medical Center while doing an experimental mental task. The fMRI data was transmitted via high-speed network to the Pittsburgh Supercomputing Center, was processed, and transmitted back to the operating room. The Cray T3E at the Pittsburgh Supercomputing Center converted the raw fMRI data into 3D images, compensated for head movement and identified active areas of the brain. The fMRI reconstruction computed on the Cray T3E took about 10 s and was accomplished by distributing the computation over 512 application processors.

2. 3D image registration

Image registration is an important area of modern medical image analysis. Historically, the most important application of image registration has been for studying PET brain activation data [24–26]. Due to the poor image quality of a PET image, a PET experiment must be performed on multiple individuals and the results averaged. The data from each individual is registered to a common coordinate system such as the Talairach atlas [22] to remove individual anatomical shape difference and averaged to generate the final PET image. More recently, image registration has been used to generate individualized atlases of the head [18,8,2,16,10], for studying morphological changes in the brain due to disease [13,12], for studying craniofacial deformities [3–5], and radiation dose tracking in radiotherapy of cervical cancer [9]. Fig. 1 shows an example of how a normal anatomy as defined by a CT image volume can be used to measure the location and magnitude of a craniofacial deformity. The template shown in the top row was deformed in 3D – using a linear elastic deformation model – into the shape of the dysmorphic anatomy shown in the bottom row. The deviation from the normal shape of the template is encoded in the transformation function and can be used to classify the degree of deformity, aid in corrective surgical planning, and evaluate the effectiveness of the surgery.

[3] Nigel Goddard of PSC, Jonathan Cohen of UPMC and CMU, William Eddy of CMU, Doug Noll of UPMC, and Greg Hood of PSC.

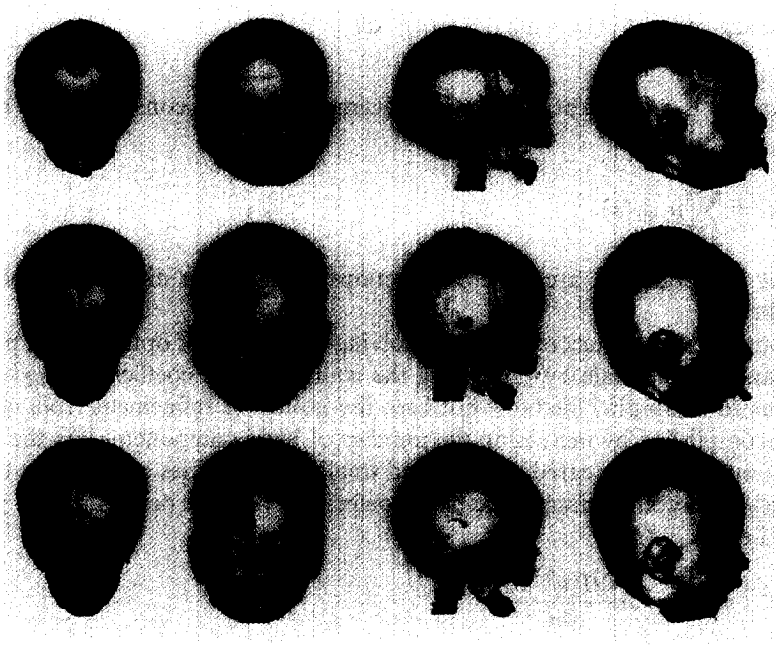

Fig. 1. 3D image registration can be used to measure craniofacial deformities imaged via CT. The top row shows 3D skin and bone surfaces of the template (a scan from a normal 3 month old infant), the middle row shows the 3D linear elastic transformation of the template into the shape of the target, and the bottom row shows the target data set (a scan from 3 month old infant with unilateral coronal craniosynostosis).

2.1. The linear elastic transformation model

The 3D linear elastic transformation model [18,7,23] is used to generate global non-rigid deformations of the template image volume. The goal is to find the mapping h from the coordinates $x \in \Omega$ of a template image $T(x)$ to a target image $S(X)$, i.e., the mapping $h: \Omega \rightarrow \Omega$ is defined as $h(x) = x - u(x)$ where $u(x)$ is called the displacement field.

A distance measure $D(u)$ is defined to measure the disparity between the template T and target S images with respect to the applied displacement u. The optimization problem used to estimate the best transformation is defined as

$$u = \arg \min_{u} \int \|Lu\|^2 + D(u). \tag{1}$$

Examples of valid distance functions include the Gaussian squared error distance $\int |T(h(x)) - S(x)|^2 dx$, the correlation distance [1,11], the minimum intensity variance distance [20,26] and the Mutual Information distance [15,21,14].

For the remainder of the paper, the minimization problem is assumed to be of the form

$$\hat{u} = \arg \min_{u} \gamma \int_{\Omega} |T(x-u(x)) - S(x)|^2 \, dx + \int_{\Omega} |Lu|^2, \tag{2}$$

where *gamma* is a constant and the displacement field u is constrained to have the form

$$u(x) = \sum_{k=0}^{d} \mu_k \phi_k(x). \tag{3}$$

The basis functions $\{\phi\}$ are the eigenfunctions of the linear differential operator L defined below.

The optimization is accomplished by solving a sequence of optimization problems from coarse to fine scale by estimating the complex basis coefficients $\{\mu_k\}$. This is analogous to multi-grid methods but here the notion of refinement from coarse to fine is accomplished by increasing the number of basis components. As the number of basis functions is increased, smaller and smaller variabilities between the template and target are accommodated. The basis coefficients $\{\mu_k\}$ are determined by gradient descent, i.e.,

$$\mu_k^{(n+1)} = \mu_k^{(n)} - \Delta \frac{\partial H(u^{(n)}|S)}{\partial \mu_k}, \tag{4}$$

where

$$\frac{\partial H(u^{(n)})}{\partial \mu_k} = -\gamma \int_{\Omega} (T(x-u^{(n)}(x)) - S(x)) \nabla T(x-u^{(n)}(x)) \cdot \phi_k(x) \, dx + \lambda_k^2 \mu_k^{(n)}, \tag{5}$$

$$u^{(n)}(x) = \sum_{k=0}^{d} \mu_k^{(n)} \phi_k(x). \tag{6}$$

Δ is a fixed step size and λ_k are the eigenvalues of the eigenvectors ϕ_k.

We assume that the linear differential operator L is of the form $(-a\nabla^2 - b\nabla\nabla + c)^p$, $p \geqslant 1$ and the eigenfunctions and eigenvalues of L satisfy $L\phi_k(x) = \lambda_k \phi_k(x)$. Assuming cyclic boundary conditions on the unit cube, the eigenfunctions and eigenvalues have the form given in [17]

$$\begin{aligned}
\phi_k^{(1)}(x) &= \alpha_k^{(1)} [\omega_{k_1}, \omega_{k_2}, \omega_{k_3}]^{\mathrm{T}} e^{-j\langle \omega_k, x \rangle}, \qquad \lambda_k^{(1)} = (a+b)(\omega_{k_1}^2 + \omega_{k_2}^2 + \omega_{k_3}^2) + c, \\
\phi_k^{(2)}(x) &= \alpha_k^{(2)} [-\omega_{k_2}, \omega_{k_1}, 0]^{\mathrm{T}} e^{-j\langle \omega_k, x \rangle}, \qquad \lambda_k^{(2)} = \lambda_k^{(3)} = a(\omega_{k_1}^2 + \omega_{k_2}^2 + \omega_{k_3}^2) + c, \\
\phi_k^{(3)}(x) &= \alpha_k^{(3)} [\omega_{k_1}\omega_{k_3}, \omega_{k_2}\omega_{k_3}, -(\omega_{k_1}^2 + \omega_{k_2}^2)]^{\mathrm{T}} e^{-j\langle \omega_k, x \rangle}
\end{aligned} \tag{7}$$

with $\phi_k^{(1)}(x) = [1,0,0]^{\mathrm{T}}$, $\phi_k^{(2)}(x) = [0,1,0]^{\mathrm{T}}$, and $\phi_k^{(3)}(x) = [0,0,1]^{\mathrm{T}}$ for $k = [0,0,0]^{\mathrm{T}}$. We define $\omega_k = [2\pi|k_1|/N, 2\pi|k_2|/N, 2\pi|k_3|/N]^{\mathrm{T}}$ for $k_i \in \{-N/2, \ldots, -1, 0, 1, \ldots, N/2\}$ and $\alpha_k^{(1)} = 1/\sqrt{\omega_{k_1}^2 + \omega_{k_2}^2 + \omega_{k_3}^2}$, $\alpha_k^{(2)} = 1/\sqrt{\omega_{k_1}^2 + \omega_{k_2}^2}$, and $\alpha_k^{(3)} = \alpha_k^{(1)}\alpha_k^{(2)}$.

3. Parallel implementation

A massively parallel computer is defined to be a computer with more than 1000 processor elements (PEs) and is classified as either a single-instruction stream, multiple-data stream (SIMD) or a multiple-instruction stream, multiple-data stream (MIMD) parallel architecture. SIMD systems are used for problems that require a large number of computations in which all the PEs perform the same operation in parallel. On each clock cycle, each PE performs the same operation on its private data. MIMD systems are used for algorithms that can be broken up into separate, independent parts to solve. Each part is assigned to a separate processor and all the parts are solved simultaneously.

In general, the PEs used in SIMD machines require less computer chip area and are less complex than those used in MIMD machines due to simpler instruction sets, narrower words, and smaller caches. As a result, SIMD computers generally have more processors than MIMD computers for the same price. Due to the simpler PE design, operations require more clock cycles to compute on a SIMD computer than a MIMD computer. However, the speed advantage of the SIMD computer comes from a large number of processors computing in parallel.

The MIMD and SIMD computers used in this work had similar peak computing performance. The MIMD computer used was the 16 processor Silicon Graphics (SGI) Challenge computer and was provided by the Advanced Research Computing Services at The University of Iowa. It consists of ten 90 MHz and six 75 MHz R8000 processors and has a peak performance of 5.4 Gflop (ten 360 Mflop and six 300 Mflop processors). The SIMD computer used was a 128×128 mesh connected MasPar computer with MP2 processors and was provided by Michael I. Miller, Department of Electrical Engineering, Washington University in St. Louis. The peak performance of this 128×128 massively parallel computer is quoted as 6.2 Gflops.

The increase in speed due to parallelizing an algorithm with N processors is governed by Amdahl's Law which is given by

$$X_A = \frac{s+p}{s+p/N} = \frac{1}{s+p/N} < \frac{1}{s}. \tag{8}$$

The factor X_A is the maximum sustainable speedup that can be achieved given that p is the parallelizable portion and $s = 1 - p$ is the sequential or non-parallelizable portion of the program. This means that the ideal or maximum achievable speedup of parallelizing an algorithm with N processors is an N times speedup. It also states that the maximum speedup is always less than the reciprocal of the sequential portion of the algorithm.

SIMD machines generally suffer a greater time penalty for serial portions of an algorithm than MIMD machines. This is because the serial portions of the algorithm must be executed on a single simple processor. It is conceivable that serial sections may be speed up by executing them on the host computer microprocessor instead of a single PE. However, this does not help in practice because the time to transfer data from the PE array to the host computer normally exceeds the processing time.

3.1. MIMD parallel implementation issues

The SGI Challenge computer is a symmetric multiprocessing (SMP), shared memory computer (see Fig. 2). SMP implies that the operating system treats all of the processors as equal and the shared memory is used for processor communication and synchronization. An important difference between shared memory and distributed memory systems is how the algorithm is parallelized. In a shared memory system the algorithm is partitioned into sub-algorithmic procedures that are distributed across the processors. Data movement is implicit in a shared memory system, so each processor has access to the whole memory as it executes. The drawback of bus-based SMP systems is that they do not scale up to large number of processors because the shared bus becomes a performance bottleneck. On the other hand, the MasPar is a distributed memory system. In this model the programmer needs to decide how the data is partitioned across the processors and how the data is moved from processor to processor.

The key to implementing a parallel algorithm on the multiprocessor SGI challenge is to maximize the ratio of computation to the parallelization overhead. Parallel overhead is the processing time spent in creating slave processes, starting/ending parallel regions, and executing the extra code added for parallelization. The parallelization overhead can be reduced by parallelizing the largest possible loops, i.e., the outermost loops. The parallelization overhead is generally recovered if the loop contains more than 100 floating point operations. In addition, one should not use more CPU's than are necessary (see Fig. 6).

It is important to distribute the work load equally across all of the available processors. This process is called load balancing and is typically trivial for image based data. Efficient parallelization also requires that both communication between processors and synchronization among processors be minimized.

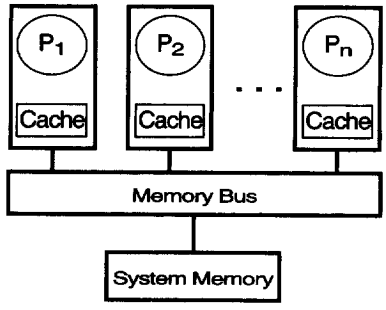

Fig. 2. The SGI Challenge computer is a shared memory architecture. Each processor has its own 64Kb cache which is used to store local data fetched from the main memory. Built in memory contention hardware insures that the data in each individual cache and the main memory remains consistent.

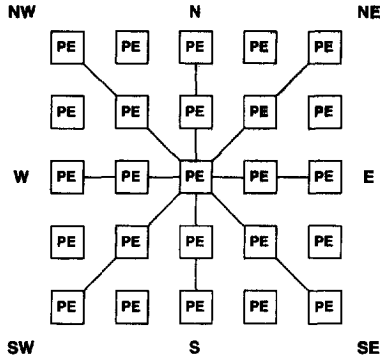

Fig. 3. The MasPar is well suited for solving partial differential equations because each processor can send/receive data to/from its eight nearest neighbors in parallel.

3.2. SIMD massively parallel implementation issues

The PEs in the MasPar [19] are connected in a 128×128 toroidal mesh (see Fig. 3) for efficient local communication and by a global router for nonlocal communication. The mesh connection of the PEs is called the xnet and allows each PE to send/receive data (communicate) to/from its 8 nearest neighboring PEs. This feature makes the MasPar good for solving problems that require local PE communication such as solving partial differential equations [6]. The global router allows PEs to send/receive data to/from other PEs that are separated by long distances. It also allows the PEs to communicate in non-regular patterns in parallel which is important for computing the deformed template.

3.3. Transforming the template

Each iteration of the registration algorithm requires computing the deformation of the template and the deformation of the x, y, and z-derivatives of the template (see Eq. (5)). Computing the deformation requires accessing the data from memory in a non-regular pattern due to the nonlinear nature of the transformation. The MIMD architecture has an advantage over the SIMD architecture for random memory access because each MIMD processor has random access to the whole memory. Mesh-connected SIMD computers suffer a loss in efficiency for randomly accessing memory because the non-regular memory access pattern of the distributed memory reduces data transfer parallelism.

Normally, a non-regular pattern of communication (random access) is very inefficient on mesh connected computers because it is not a parallel operation using nearest neighbor data transfer. However, the MasPar has two features that reduce this problem: the global router (see Fig. 4) and local PE indirect memory addressing (see Fig. 5). The global router and the local PE indirect memory addressing are used together to provide random access to the data required to deform the atlas. The

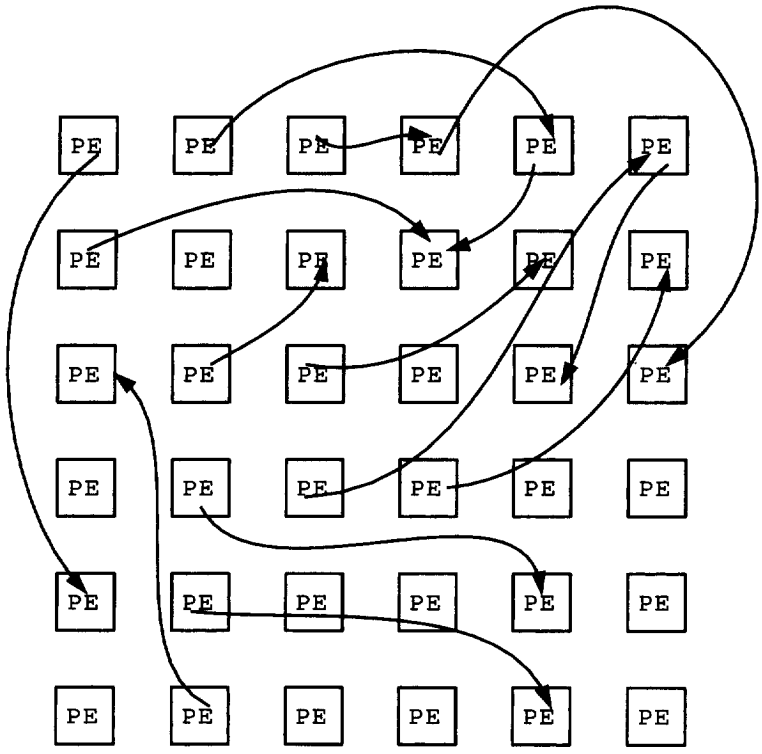

Fig. 4. An example of a non-regular pattern of PE communication required to compute the transformation of the template. The MasPar's global router is used to transfer data from source PEs to a destination PEs as indicated by the arrows in parallel.

template is transformed using these features in the following way. First, each PE calculates the memory address of the data that it needs to compute its value of the deformed template. This address is then converted into an address of a target PE and a local memory address for the target PE. Using the target PE addresses, the global router sends the local memory addresses from the requesting PEs to the target PEs in parallel. Next, all of the target PEs use indirect addressing to fetch the requested data from their local memory specified by the memory address in parallel. Finally, the target PEs sends the data back to the requesting PEs in parallel using the global router. Without the global router and the PE indirect addressing, the PEs would be very inefficient performing the random access required to transform the template.

The PEs on the MasPar are divided into 4×4 subgroupings called clusters. Only one PE per cluster can send data and one PE can receive data during a router data transfer. Therefore address contention occurs when two or more PEs request to send from or receive from the same cluster. The hardware of the router handles this address contention using a sequence of router data transfers until all of the data is transfered. This reduces the parallelism of data transfer but is still faster than

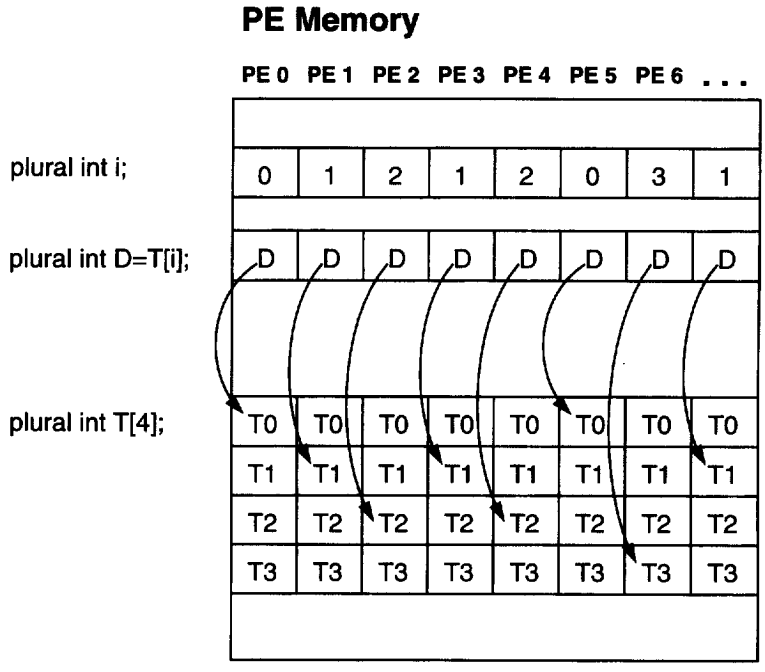

Fig. 5. Example of parallel indirect addressing. The value of D at each PE is set equal to the value of T offset by the index i in parallel.

nearest neighbor PE communication for large distance, random address data transfers.

4. Results

4.1. Performance improvement vs. the number of MIMD processors

A substantial performance improvement (see Figs. 6 and 7) was found with an increase in the number of processors used in the MIMD implementation of the 3D linear elastic image registration algorithm. As Amdahl's Law suggests, the best one can hope to achieve by increasing the number of processors is a linear increase in computation speed. Data communication between processors and memory contention are just two of the factors that prevent a multiple processor implementation from achieving the ideal speedup. The theoretical upper bound on performance improvement is shown by the straight line with slope one. The remaining three curves show the performance of the SIMD implementation on a 16 processor R80000 SGI Challenge for data volume sizes of $32 \times 32 \times 25$, $64 \times 64 \times 50$, and $128 \times 128 \times 100$ voxels, respectively. The $128 \times 128 \times 100$ curve shows nearly a linear speedup in computation time as the number of processors is increased. On the

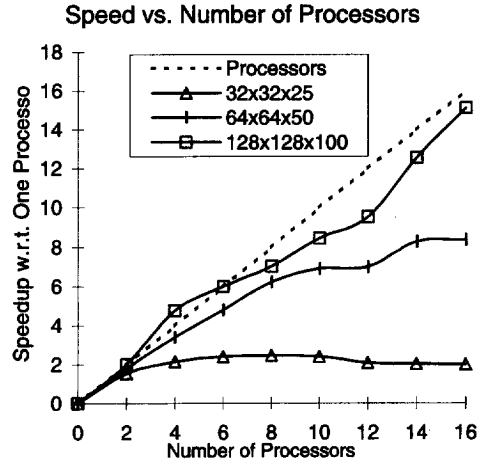

Fig. 6. Graph of the increase in performance of the 3D linear elastic image registration algorithm as a function of increasing the number of R8000 processors for data volumes sizes of $32 \times 32 \times 25$, $64 \times 64 \times 50$, and $128 \times 128 \times 100$ voxels. The straight line with slope one shows an estimate of the theoretical upper bound on the performance. See text for additional details.

other hand, the other two curves start out with a linear speed up, but then level off at a factor of 8 and 2 times speed up, respectively. The reason for the decrease in performance is because the time associated with the multiple processor implementation overhead is no longer negligible compared to the algorithmic computation time. In fact, the curve for the smallest data set actually decreases slightly as the number of processors is increased.

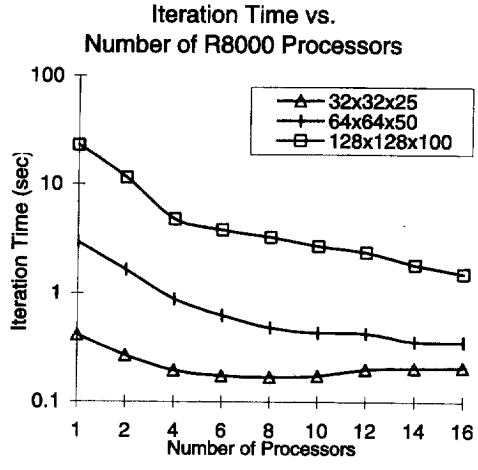

Fig. 7. Graph of the 3D linear elastic deformation algorithm as a function of the number R8000 processors for three different image volume sizes.

Notice that the $128 \times 128 \times 100$ curve surpasses the theoretical maximum for the four processors case. The reason for this apparent anomaly is because the reference single processor computation time had overhead associated with the multiple processor implementation. The calculation of performance increase was computed by dividing the average single 90 MHz processor iteration time by the average multiple processor iteration time. The speedup of 4.78 for the four processor case is due to multiple processor overhead contained in the single processor iteration time even though only one processor was used.

Another note about the graph in Fig. 6 is that the SGI Challenge computer used in this analysis had a mixture of ten 90 MHz and six 75 MHz R8000 processors. The multiple processor average iteration times were generated by an unknown combination of fast and slow processors. Thus, the increase in performance was under estimated because it was computed with respect to the iteration time of a 90 MHz R8000 processor. The mixture of processors also changes the shape of the theoretical upper bound. If we assume that the fastest processors are used before the slower processors, the upper bound curve would be a straight line of slope one for up to ten processors and the slope would decrease to 0.83 (75/90) for 11–16 processors.

The MIMD deformation code was also timed on a 10 processor, 195 MHz R10000 SGI Challenge computer at the Institute for Biomedical Computing, Washington University, St. Louis, Missouri. The algorithm ran approximately 10% faster when using 195 MHz R10000 processors instead of the 90 MHz R8000 processors. This result was expected because the peak performance of a 195 MHz R10000 processor is 390 MFlops while the peak performance of a 90 MHz R8000 processor is 360 MFlops.

4.2. Comparison of the MasPar and SGI challenge implementations

The 3D linear elastic deformation algorithm was implemented originally on the MasPar in 1993 [7]. Although the MasPar implementation was optimized for the SIMD parallel architecture, it used brute-force to compute the deformation. As a result, the time per iteration increased as the number of basis functions increased. The SGI Challenge implementation has been optimized so that the time per iteration is constant independent of the number of basis functions. A performance comparison between the MasPar and Challenge implementations is shown in Figs. 8 and 9. This graph assumes that the number of basis harmonics are increased by one in all three coordinate directions after every 40 iterations. Fig. 8 shows that the 16 processor MIMD implementation is 20 times faster than the brute-force SIMD 128×128 processor implementation for $128 \times 128 \times 100$ voxel volume deformations and 300 iterations. This graph also shows that the speedup for the $64 \times 64 \times 50$ voxel deformation is on the order of 12 times at 100 iterations when comparing a 64×64 processor MasPar to the 16 processor Challenge. The number of MasPar processors was reduced to 64×64 so that the mesh size would match the first two dimensions for this data set. Therefore, the performance of the MasPar could be improved by a maximum of four times if all 128×128 processors were used in this case.

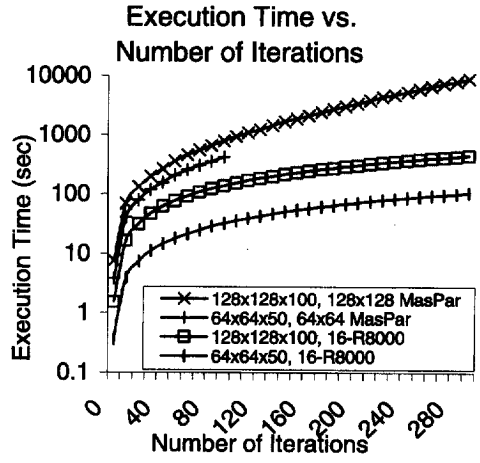

Fig. 8. Execution times for the 3D linear elastic transformation implemented on a 128 × 128 MasPar SIMD computer and a 16 processor SGI Challenge MIMD computer as a function of two different image volume sizes. Note that the MasPar was restricted to a 64 × 64 mesh for the 64 × 64 × 50 voxel image deformation.

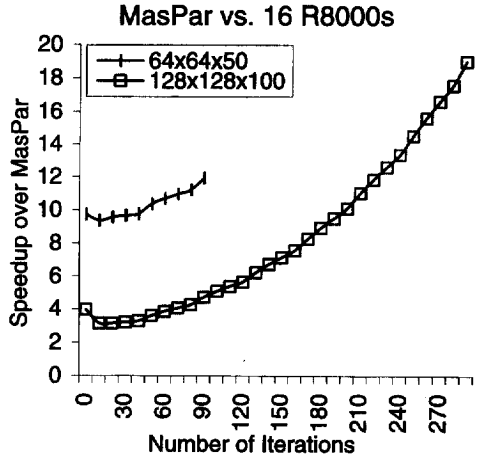

Fig. 9. Relative speedup of the optimized 3D linear elasticity algorithm implemented on the 16 processor SGI Challenge over the 128 × 128 MasPar SIMD computer as a function of a 64 × 64 × 50 and 128 × 128 × 100 volume sizes. Note that the MasPar was restricted to a 64 × 64 mesh for the 64 × 64 × 50 voxel image deformation.

Finally, we address the question of how much faster the 16 processor SGI Challenge is than the 128 × 128 MasPar for the optimized 3D linear elastic deformation algorithm. To answer this question, we need to estimate how fast the optimized deformation algorithm would be on the MasPar. It will be assumed that the fastest that an optimized iteration can be completed on the MasPar is equal to the

fastest iteration of the brute-force implementation. Thus, an optimized iteration for $128 \times 128 \times 100$ voxel volume would take 6.6 s on a 128×128 MasPar (see Fig. 8). Dividing 6.6 s by 1.52 s (the average time for the Challenge implementation) implies that the 16 processor SGI Challenge is a minimum of 4.3 times faster than the 128×128 MasPar for computing the 3D linear elastic deformation algorithm.

5. Conclusion

This paper discussed the issues involved with implementing a computationally intensive 3D medical image registration algorithm on a 16 processor SGI Challenge computer (SIMD architecture) and on a 128×128 processor MasPar computer (MIMD architecture). For large data sets, the MIMD implementation was shown to have nearly linear performance improvement as the number of processors were increased. Finally, the optimized MIMD implementation was shown to be roughly 20 times faster than the brute-force SIMD implementation. Applying similar optimizations to the MasPar implementation would make the 16 processor SGI Challenge over 4 times faster than the 128×128 MasPar for processing large data sets.

Acknowledgements

I would like to express my appreciation to Michael W. Vannier of the Department of Radiology, The University of Iowa Medical School for his help and support during the preparation of this paper. I would like to thank Issac Evans, Tom Casavant, and John Kuhl of the Department of Electrical and Computer Engineering, The University of Iowa, for their helpful comments regarding the SGI multiprocessor compiler. I would like to thank Michael I. Miller of the Department of Electrical Engineering, Washington University, for his helpful advise over the last eight years and for providing access to the MasPar computer. I would like to thank Fred Rosenberger of the Institute for Biomedical Computing, Washington University, for his help and insightful comments concerning computer architectures and compilers and for providing access to the 10 processor R10000 SGI Challenge computer. I would like to thank Jun Ni of the Advanced Research Computing Information Technology Services, The University of Iowa, for his help in providing access to the 16 processor R8000 SGI Challenge computer. I would also like to thank my student Hans Johnson for his help in preparing the graphs for this manuscript.

References

[1] R. Bajcsy, S. Kovacic, Multiresolution elastic matching, Technical Report MS-CIS-87-94, GRASP Lab 123, Department of Computer and Information Science, School of Engineering and Applied Science, University of Pennsylvania, Philadelphia, PA 19104-6389, 1987.

[2] G.E. Christensen, S.C. Joshi, M.I. Miller, Volume geometric transformations for mapping anatomy, IEEE Trans. on Med. Imaging 16 (6) (1997) 864-877.
[3] G.E. Christensen, A.A. Kane, J.L. Marsh, M.W. Vannier, A 3D deformable infant ct atlas, in: H.U. Lemke, M.W. Vannier, K. Inamura, A.G. Farman (Eds.), CAR '96: Computer Assisted Radiology, Elsevier, New York, 1996, pp. 847–852.
[4] G.E. Christensen, A.A. Kane, J.L. Marsh, M.W. Vannier, Synthesis of an individualized cranial atlas with dysmorphic shape, IEEE Proceedings of Mathematical Methods in Biomedical Image Analysis, 1996, pp. 309–318.
[5] G.E. Christensen, J.L. Marsh, M.W. Vannier, Computer simulation of normalcy in craniosynostosis, in: H.U. Lemke, K. Inamura, C.C. Jaffe, M.W. Vannier (Eds.), Computer Assisted Radiology, Springer, Berlin, 1997.
[6] G.E. Christensen, M.I. Miller, U. Grenander, M.W. Vannier, Individualizing neuroanatomical atlases using a massively parallel computer, IEEE Computer (1996) 32–38.
[7] G.E. Christensen, R.D. Rabbitt, M.I. Miller, 3D brain mapping using a deformable neuroanatomy, Physics in Medicine and Biology 39 (1994) 609–618.
[8] G.E. Christensen, R.D. Rabbitt, M.I. Miller, Deformable templates using large deformation kinematics, IEEE Transactions on Image Processing 5 (10) (1996) 1435–1447.
[9] G.E. Christensen, J.F. Williamson, K.S.C. Chao, M.I. Miller, F.B. So, M.W. Vannier, Deformable anatomical templates for brachytherapy treatment planning in radiotherapy of cervical cancer, in: R.A. Melter, A.Y. Wu, L.J. Latecki (Eds.), Vision Geometry VI, Proceedings of SPIE, vol. 3168, 1997, pp. 147–154.
[10] C.A. Davatzikos, J.L. Prince, R.N. Bryan, Image registration based on boundary mapping, IEEE Trans. on Medical Imaging 15 (1) (1996) 112–115.
[11] J.C. Gee, L. LeBriquer, C. Barillot, D.R. Haynor, R. Bajcsy, Bayesian approach to the brain image matching problem, in: M.H. Loew (Ed.), Image Processing, Proc. SPIE, vol. 2434, 1995, pp. 145–156.
[12] J.W. Haller, A. Banerjee, G.E. Christensen, M. Gado, S.C. Joshi, M.I. Miller, Y. Sheline, M.W. Vannier, J.G. Csernansky, 3D hippocampal morphometry by high dimensional transformation of a neuroanatomical atlas, Radiology 202 (2) (1997) 504–510.
[13] J.W. Haller, G.E. Christensen, S.C. Joshi, J.W. Newcomer, M.I. Miller, J.G. Csernansky, M.W. Vannier, Hippocampal MR morphometry by pattern matching, Radiology 199 (1996) 787–791.
[14] W.M. Wells III, P. Viola, H. Atsumi, S. Nakajima, R. Kikinis, Multi-modal volume registration by maximization of mutual information, Medical Image Analysis 1 (1) (1996) 35–51.
[15] F. Maes, A. Collignon, D. Vandermeulen, G. Marchal, P. Suetens, Multimodality image registration by maximization of mutual information, IEEE Transactions on Medical Imaging 16 (2) (1997) 187–198.
[16] J.C. Mazziotta, A.W. Toga, A. Evans, P. Fox, J. Lancaster, A probabilistic atlas of the human brain: theory and rationale for its development, Neuroimage 2 (1995) 89–101.
[17] M.I. Miller, A. Banerjee, G.E. Christensen, S.C. Joshi, N. Khaneja, U. Grenander, L. Matejic, Statistical methods in computational anatomy, Statistical Methods in Medical Research 6 (1997) 267–299.
[18] M.I. Miller, G.E. Christensen, Y. Amit, U. Grenander, Mathematical textbook of deformable neuroanatomies, Proceedings of the National Academy of Sciences 90 (24) (1993) 11944–11948.
[19] J.L. Potter (Ed.), The Massively Parallel Processor, Scientific Computation Ser, MIT Press, Cambridge, MA, 1985.
[20] C. Studholme, D.L.G. Hill, D.J. Hawkes, Multiresolution voxel similarity measures for MR-PET registration, in: Y. Bizais, C. Braillot, R. Di Paola (Eds.), Information Processing in Medical Imaging, vol. 3, Kluwer Academic Publishers, Boston, 1995, pp. 14–22.
[21] C. Studholme, D.L.G. Hill, D.J. Hawkes, Incorporating connected region labelling into automated image registration using mutual information, IEEE Proceedings of Mathematical Methods in Biomedical Image Analysis, 1996, pp. 23–30.
[22] J. Talairach, P. Tournoux, Co-Planar Stereotactic Atlas of the Human Brain, Beorg Thieme Verlag, Stuttgart, 1988.
[23] S. Timoshenko, Theory of Elasticity, McGraw-Hill, New York, 1934.

[24] A.W. Toga, P.K. Banerjee, B.A. Payne, Brain warping and averaging, J. Cereb. Blood Flow Metab. 11 (1991) S560.
[25] R.P. Woods, S.R. Cherry, J.C. Mazziotta, Rapid automated algorithm for aligning and reslicing PET images, Journal of Computer Assisted Tomography 16 (4) (1992) 620–633.
[26] R.P. Woods, J.C. Mazziotta, S.R. Cherry, MRI–PET registration with automated algorithm, Journal of Computer Assisted Tomography 17 (4) (1993) 536–546.