Parallel MLEM on Multicore Architectures

Tilman Küstner[1], Josef Weidendorfer[1], Jasmine Schirmer[2], Tobias Klug[1], Carsten Trinitis[1], and Sybille Ziegler[2]

[1] Department of Computer Science, Technische Universität München
{tilman.kuestner,josef.weidendorfer,tobias.klug, carsten.trinitis}@cs.tum.edu

[2] Department of Nuclear Medicine, Technische Universität München
jasmine.schirmer@tum.de, s.ziegler@lrz.tu-muenchen.de

Abstract. The efficient use of multicore architectures for sparse matrix-vector multiplication (SpMV) is currently an open challenge. One algorithm which makes use of SpMV is the maximum likelihood expectation maximization (MLEM) algorithm. When using MLEM for positron emission tomography (PET) image reconstruction, one requires a particularly large matrix. We present a new storage scheme for this type of matrix which cuts the memory requirements by half, compared to the widely-used compressed sparse row format. For parallelization we combine the two partitioning techniques recursive bisection and striping. Our results show good load balancing and cache behavior. We also give speedup measurements on various modern multicore systems.

1 Introduction

In contrast to computer tomography (CT), which aims at structural imaging, positron emission tomography (PET) visualizes functional processes, by measuring the distribution of a tracer consisting of radioisotopes injected into a patients body. Clinical PET scanners for example assist in tumor diagnosis. PET research currently focuses on improving spatial resolution and sensitivity of the technique.

A PET scanner consists of fixed detectors, usually arranged in a ring around the subject to be analyzed. A positron-emitting radioisotope can be detected indirectly, as positrons annihilate with electrons, creating two 511 keV gamma photons traveling in opposite directions. When two detectors each record a photon within a certain time window, an annihilation event is assumed somewhere along the line connecting the detectors. This line is called the line of response (LOR). The number of detected events influences the quality of the measurement, while the coverage of three-dimensional space of interest (field of view, FOV) by LORs affects the achievable resolution. The resolution is usually better at the center than at the edges of the field of view. The FOV is commonly divided into a three-dimensional grid, where each grid cell is called a voxel.

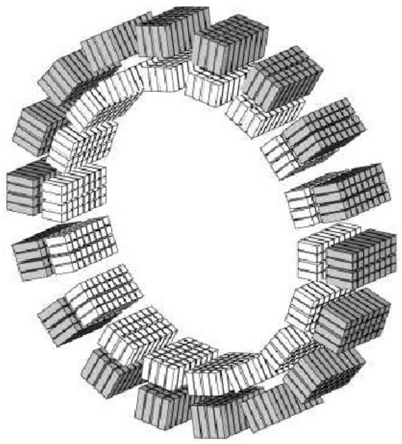


Fig. 1. Geometry of MADPET-II

The experimental small-animal scanner MADPET-II [8] was developed at the Department of Nuclear Medicine. This scanner is able to resolve the issue of poor spatial resolution by adding a second ring of detectors (see Fig. 1). This leads to a quadratic increase of measurement data, and consequently, a significant increase in computational demand for the post processing step, the 3D image reconstruction.

Several algorithms can be used for reconstructing PET images. One is filtered back-projection (FBP), which is based on an analytic solution of the Radon transform. Other algorithms are based on iterative reconstruction such as the maximum likelihood maximization (MLEM) algorithm. This algorithm usually outperforms FBP in terms of image quality, but is more computationally intensive.

The huge memory requirements of MLEM come from fixed input data, describing the geometrical and physical properties of the scanner. This data is arranged in a matrix which gives the probability of an event occurring in one voxel being recorded by a given pair of detectors. This so-called system matrix is sparse, since for voxels outside a given LOR, the detection probability is almost certainly zero. The system matrix can be measured in a physical experiment, or it can be computed from analytic models or by Monte Carlo simulation. The simulation takes a number of physical effects into account, which influence the trajectory and detection of the photons.

The MLEM algorithm is iterative meaning that it starts with an estimate of the solution, which is then corrected in every iteration step. In each step, two vector scaling operations and two sparse matrix-vector multiplications (SpMV or sometimes SpMxV) are carried out. The latter operations are known for a number of performance problems [3]. Amongst them are indirect referencing and irregular access of the source vector as well as high load on the memory subsystem created by the traversal of the matrix. We parallelize the MLEM algorithm by splitting the SpMV operations into smaller ones. As target systems, we have computer clusters using multicore processors in mind.

2 Related Work

A comprehensive overview of iterative algorithms for image reconstruction in general is given in [7]. Parallel algorithms for reconstructing both CT and PET images are described in [5]. The article also provides an overview of feasible acceleration techniques, and covers a broad range of parallel environments, from networks of workstations, to peer-to-peer and grid computing.

During the 1990s, various approaches to parallelizing MLEM have been proposed [1,2,6]. They exploit the symmetries in the scanner geometry and partition the FOV and thus the image vector in a compatible way. All proposals are limited by the computational power available in the respective years. Thus, in all the articles cited, image, measurement and system data is much smaller than in our work.

Given future systems with accelerators, another method of dealing with the huge memory requirements is computing the system matrix on the fly from a simplified analytical model. In a previous work [4] we made use of this technique, implementing the MLEM algorithm on the IBM Cell BE and using the fast accelerator cores of the Cell processor. Currently, more work is done here to test more accurate and faster analytical models [10].

3 The MLEM Algorithm

The MLEM algorithm was first proposed by Shepp and Vardi [11]. It can be viewed as an implementation of the more general expectation-maximization (EM) algorithm, applied to the problem of image reconstruction. In the following, we will give a short formal explanation of the algorithm.

The system matrix $A = (a_{ij})$ gives the probability of a photon emitted from voxel j being recorded by detector pair i. Its number of columns m equals the number of voxels, its n rows correspond to the n detector pairs or LORs. Let f denote the image vector and g the measuring vector. Disregarding all stochastic effects, we can state that the MLEM algorithm tries to approximate a solution of the set of linear equations $Af = g$.

An MLEM iteration step is given by

$$f_j^{(q+1)} = \frac{f_j^{(q)}}{\sum_{l=1}^{n} a_{lj}} \sum_{i=1}^{n} a_{ij} \frac{g_i}{\sum_{k=1}^{m} a_{ik} f_k^{(q)}} \quad . \tag{1}$$

We use the superscript q to denote the iteration number. One step can be divided into several parts. First, the forward projection (FP) is calculated:

$$h_i^{(q)} = \sum_{k=1}^{m} a_{ik} f_k^{(q)} \tag{2}$$

This equation describes a multiplication of the sparse matrix A with the image vector $f^{(q)}$. The resulting vector $h^{(q)}$ shows what the measurement would look

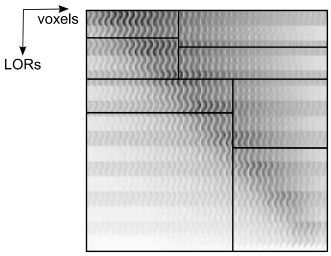


Fig. 2. Density plot and possible partitioning of matrix II

like for the approximate image vector $f^{(q)}$. Subsequently, the forward projection is compared to the actual measurement and a correction factor is derived:

$$c_j^{(q)} = \sum_{i=1}^{n} a_{ij} \frac{g_i}{h_i^{(q)}} \tag{3}$$

This step is called back projection (BP) and matches a sparse matrix-vector multiplication, this time using the transposed matrix. Finally, the vector of correction factors and an additional scaling factor are applied to the image vector (see Eq. 1). The scaling factor can of course be calculated before the iteration starts.

4 Implementation

To come up with an efficient implementation of the MLEM algorithm, a fitting storage format for the system matrix has to be created. Our two test matrices, labeled I and II, describe the sensitivity of the small animal pet scanner MADPET-II [8], which has 1152 detectors arranged in two rings (see Fig 1). This results in $n = 662976$ lines of response or matrix rows. The field of view is divided into $m = 784000$ voxels arranged in a $140 \times 140 \times 40$ grid.

The test matrices were both generated by Monte Carlo simulation, but with different parameters. Two million annihilation events per voxel were simulated. Besides the information of the LOR, the simulation also returns the absorption energy of a detection. Whether a given detection is counted for the resulting matrix depends on a chosen energy threshold. The energy threshold for matrix I was set to 400 keV, resulting in a 2.6 GB matrix, whereas the energy threshold for matrix II was set to 200 keV, resulting in a 17.6 GB matrix. Matrix I has 0.12% non-zero entries, whereas the sparsity of matrix II is 0.84%.

To provide an impression of the matrix structure we include a grayscale plot of matrix II in Fig. 2.

4.1 Matrix Storage Format

As the same number of events is simulated in every voxel, only the number of successfully detected events needs to be stored in the matrix. Even in the longest running Monte Carlo simulation, the largest number of successfull recordings was below 100, because a huge amount of generated photon pairs completely miss any detector.

We use a modified version of the compressed sparse row format (CSR) to store the matrix. Because of the low number of possible values of a matrix element, an array to explicitly store the values is not needed. Let $\tilde{a}_i$ denote the maximal entry of each matrix row. In detail, the three arrays of our matrix format are the following:

- *column* – An array whose length equals the total number of matrix entries. It contains the column numbers of the entries. The elements are sorted into sections according to their respective row number. Within the sections, the elements are sorted in ascending order of their value.
- *value* – An array of length $\sum \tilde{a}_i + 1$. It contains indices of the array *column*. The element $value[j]$ points to the column number of a matrix element in row i with a value of $j - row[i] + 1$.
- *row* – An array of length $n + 1$, containing indices of the array *value*. The element $row[i]$ points to the first non-zero element of row i.

As an example, we show how the following matrix is stored in our format.

$$\begin{pmatrix} 1 & 1 & 1 & 1 & 0 \\ 0 & 1 & 2 & 1 & 0 \\ 0 & 3 & 2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 \end{pmatrix} \rightarrow$$

row	0 1 3 6 6 7
value	0 4 6 7 7 8 9 11
column	0 1 2 3 1 3 2 2 1 2 3

By using this format, we can store a system matrix in about half the amount of memory that would be required for the CSR format, due to the short length of the array *value*.

It should be noted that the multiplication with the transposed matrix in the second stage of the MLEM algorithm does not raise any additional issues. The same matrix format can be used; only the type of access to the source and destination vectors is interchanged. During BP the source vector is accessed sequentially, whereas the destination vector is accessed according to the sparsity pattern of the matrix.

4.2 Parallelization

Having cluster environments in mind, we use the message passing interface (MPI) as a basis of our parallelization. On multicore architectures, MPI uses shared memory for communication. Future work will evaluate the performance of a hybrid parallelization, i.e. OpenMP within the nodes and MPI as an inter-node communication model on computer clusters.

We use recursive bisection to parallelize the SpMV operations. Blocks of equal numbers of non-zero matrix elements are created. We then assign each block to one MPI process and each process is mapped to one processor core. Using this "owner computes" approach, each core calculates a small SpMV operation. The partial results are gathered in an all-reduce operation. The blocks resulting from recursively dividing matrix II three times are displayed in Fig. 2.

If the number of columns of one block is large, it can be further subdivided into vertical stripes. In terms of strategies for cache optimization this is known as one-dimensional blocking. The stripes are processed in sequential order by the core processing the block. Creating stripes turns out to be especially beneficial on machines with small last-level CPU cache, as will be shown in section 5.1. Striping creates sections in the vector that is affected by the problem of irregular access. The sections fit into the last-level cache. This applies to the source vector of the multiplication during forward projection and the destination vector during back projection.

Given this parallelization idea, a process only needs to load its assigned block of the system matrix. Thus, the huge memory consumption of a large matrix can be distributed to the nodes of a computer cluster or the nodes of a machine with non-uniform memory access.

5 Experiments and Results

In this section we present results gained from tests on five machines with both uniform (UMA) and non-uniform memory access (NUMA). We use Intel Compiler 10.1 with the option -O3 for compiling and OpenMPI 1.3 as MPI runtime environment. In order to obtain reliable results we pin the MPI processes to CPU cores using the tool `taskset`[1]. The importance of pinning will be shown in section 5.3.

The first system consists of two Intel Xeon 5335 (Clovertown) processors at 2.6 GHz and 8 GB main memory. The Clovertown processor has four cores, with two cores sharing a 4 MB L2 cache each. See Fig. 3 (a) for the design of this UMA system that will be referred to as Clovertown.

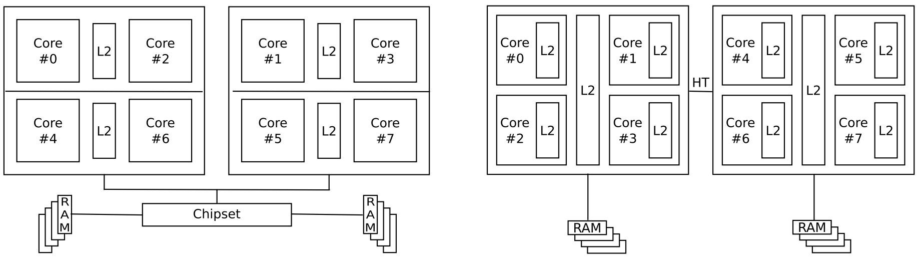
The second system is equipped with two AMD Opteron 2352 (Barcelona) processors at 2.1 GHz and 16 GB main memory. Each CPU has four cores, each of which has a private L2 cache. All cores on a chip share a 2 MB L3 cache. This NUMA machine will be referred to as Barcelona and is displayed in Fig. 3 (b).

The third system is a Sun Fire X4600 M2. It comprises 8 AMD Opteron 8218 (Santa Rosa) dual-core processors with 1 MB L2 cache per core. Each CPU has access to 8 GB local memory and three HyperTransport interfaces. We will refer to this system as X4600. The precise system architecture can be found in [12].

The fourth system is made of four Intel Xeon X7460 and will be referred to as Dunnington. The Dunnington is a hexa-core processor with a 16 MB L3 cache shared by all cores. Furthermore, the cores are grouped into pairs with an L2 cache of 3 MB per pair. The system is equipped with 32 GB of main memory.

The last system is an Intel Nehalem pre-production system. It consists of two Intel Xeon X5570 running at 2,9 GHz and 12 GB DDR3 memory. The CPU has

[1] http://userweb.kernel.org/∼kzak/util-linux-ng



(a) Intel Clovertown system (b) AMD Barcelona system

Fig. 3. System architectures of test machines

four cores (HyperThreading disabled), an 8 MB L3 cache and two QPI interfaces. This NUMA machine will be referred to as Nehalem.

Matrix I was used on Clovertown, Barcelona and Nehalem, whereas matrix II was used on X4600 and Dunnington. This applies to all results presented below. Also we do not report timings for the vector scaling operations in MLEM as these operations need less time than inter-process communication.

5.1 Striping

Subdiving matrix blocks into vertical stripes proved beneficial on every system. Most systems showed best performance with stripes of 200 000 columns in width. This corresponds to sections of approximately 0.8 MB in the input vector (forward projection) and output vector (back projection) of the SpMV operation.

The X4600 with its small L2 cache of 1 MB showed exceptional good speedup with even smaller stripes. For example, the time per iteration step on a single core dropped form 185 s to 52 s when using 16 vertical stripes. This corresponds to a vector section length of 200 KB, which easily fits into the L2 cache.

5.2 Load Balancing

The bisection approach generally gives good load balancing, as can be seen in Fig. 4. The chart displays the relative load imbalance, which we define as

$$L_{\mathrm{rel}} = \frac{\max\limits_i |t_i - t_{\mathrm{avg}}|}{t_{\mathrm{avg}}} \ .$$

The load imbalance was measured in runs with eight processes and averaged over five iteration steps. Fig. 4 also shows that striping improves load balancing, especially on the X4600 system. We attribute the remaining load imbalance to the fact that partitioning is based on the number of matrix elements per block. A more sound fundament of block size would be the number of cache misses in the source and destination vectors. But this approach would require dynamic load balancing by repartitioning the large matrix after the first iteration steps, which is inefficient.

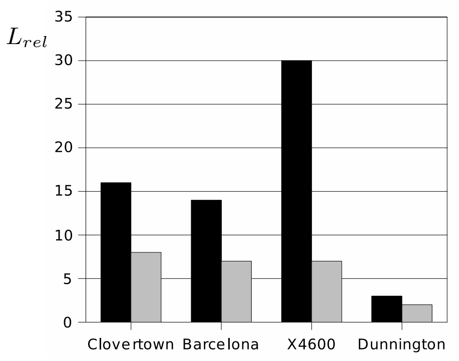


Fig. 4. Relative load imbalance with striping switched off (black) and on (gray)

5.3 Core Pinning

The selection of CPU cores is essential for good speedup. The core numbers in Fig. 5 correspond to those in Fig. 3. FP and BP denote forward and back projection, respectively. Total time also encompasses the time needed for communicating the partial results. Taking the Clovertown system as an example it can be seen, that pinning two processes to cores 0 and 2, which share a common L2 cache, returns a speedup of only 1.1, whereas using cores 0 and 1, results in a speedup of 1.8. On NUMA machines it is advantageous to use cores with separate memory links (see Fig. 5 (b)). In test runs with two processes this improved the speedup by about 25% on the Barcelona system.

N	Cores	FP [s]	BP [s]	Total [s]	Speed-up
1	0	3.75	3.89	7.64	
2	0,2	3.12	3.58	6.76	1.1
2	0,1	2.11	2.13	4.30	1.8
4	0,2,4,6	1.28	1.31	2.99	2.6
4	0,1,4,5	0.83	0.85	1.95	3.9

(a) Clovertown

N	Cores	FP [s]	BP [s]	Total [s]	Speed-up
1	0	6.43	5.79	12.23	
2	0,4	3.81	3.77	7.65	1.6
2	0,1	3.23	2.90	6.17	2.0
4	0,1,4,5	1.78	1.76	4.07	3.0
4	0,1,2,3	1.50	1.33	3.16	3.9

(b) Barcelona

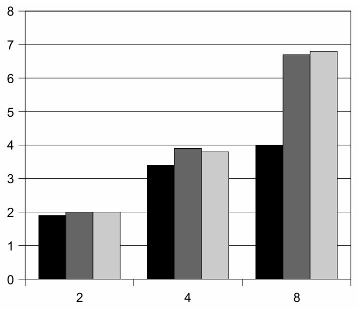
Fig. 5. Different alternatives for process to core pinning

5.4 Speedup

Finally, we present speedup data for all five test machines. We used the best pinning and optimal striping for these runs. Fig. 6 compares the speedup on Clovertown, Barcelona and Nehalem, using the smaller matrix I. The column labeled "Comm." gives the communication time for forward (FP) and back projection (BP), respectively. It can be seen, that Clovertown's system architecture only scales well up to two processes. With more processes running, the memory bandwidth of the front-side bus is saturated. The final speedup of 4.0

N	FP [s]	Comm. [s]	BP [s]	Comm. [s]	Total [s]	Speed-up
1	2.68	0.0	2.61	0.0	5.29	
2	1.37	0.01	1.35	0.02	2.75	1.9
4	0.73	0.05	0.71	0.04	1.54	3.4
8	0.57	0.11	0.58	0.08	1.34	4.0

(a) Timings on Clovertown

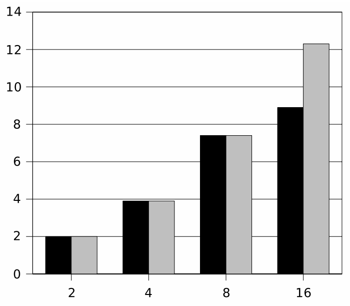


(b) Speedup on Clovertown (black), Barcelona (darkgray) and Nehalem (lightgray)

Fig. 6. Results for Clovertown, Nehalem and Barcelona, using matrix I

N	FP [s]	Comm. [s]	BP [s]	Comm. [s]	Total [s]	Speed-up
1	26.67	0.0	25.66	0.0	52.33	
2	13.28	0.13	12.81	0.12	26.34	2.0
4	6.66	0.22	6.43	0.23	13.54	3.9
8	3.38	0.22	3.29	0.21	7.10	7.4
16	2.34	0.63	2.33	0.61	5.91	8.9

(a) Timings on X4600



(b) Speedup on X4600 (black) and Dunnington (gray)

Fig. 7. Results for X4600 and Dunnington, using matrix II

when using eight cores can easily be outperformed by Barcelona with a speedup of 6.8.

The X4600 system scales reasonably well up to 8 cores, i.e. as long as we only use one core per chip (Fig. 7 (a)). The final speedup is 8.9, whereas Dunnington reaches a speedup of 12.3 on 16 cores.

6 Conclusion and Outlook

In this paper we applied two partitioning techniques, recursive bisection and striping, to the SpMV operations in MLEM. The techniques provided good load balancing and cache behavior. We also showed the importance of pinning MPI processes to cores on multicore architectures. There is ongoing work to automate this process [9]. Our approach reduces memory requirements of large matrices by using data parallelism and distributing matrix blocks to processor cores. A considerable amount of memory is also saved by our new matrix format. Future work will focus on improving cache usage, by reordering rows and columns of the system matrix.

References

1. Chen, C.M., Lee, S.-Y., Cho, Z.H.: Parallelization of the EM algorithm for 3-D PET Image Reconstruction. IEEE Transactions on Medical Imaging 10(4), 513–522 (1991)
2. Desjardins, B., Lecomte, R.: Parallel Approach to Iterative Tomographic Reconstruction for High Resolution PET Imaging. In: IEEE Nuclear Science Symposium, vol. 2, pp. 1551–1555 (1997)
3. Goumas, G., Kourtis, K., Anastopoulos, N., Karakasis, V., Koziris, N.: Understanding the Performance of Sparse Matrix-Vector Multiplication. In: 16th Euromicro Conference on Parallel, Distributed and Network-Based Processing, pp. 283–292 (2008)
4. Minde, J., Weidendorfer, J., Klug, T., Trinitis, C.: PET-Bildrekonstruktion auf der Cell-BE. In: Tagungsband Kommunikation in Clusterrrechnern und Clusterverbundsystemen, Tagung, RWTH Aachen, Aachen, Germany, vol. 3 (2007)
5. Ni, J., Li, X., Wang, G.: Review of Parallel Computing Techniques for Computed Tomography Image Reconstruction. Current Medical Imaging Reviews 2(4), 405–414 (2006)
6. Picard, Y., Selivanov, V., Verreault, M., Lecomte, R.: Optimizing Communications for Parallel ML-EM Image Reconstruction on Large Clusters of Processors. In: Conference Record of the IEEE Nuclear Science Symposium, vol. 3, pp. 1574–1580 (1998)
7. Qi, J., Leahy, R.M.: Iterative reconstruction techniques in emission computed tomography. Physics in Medicine and Biology 51, 541–578 (2006)
8. McElroy, D.P., Hoose, M., Pimpl, W., Spanoudaki, V., Schüler, T., Ziegler, S.I.: A true singles list-mode data acquisition system for a small animal PET scanner with independent crystal readout. Physics in Medicine and Biology 50, 3323–3335 (2005)
9. Ott, M., Klug, T., Weidendorfer, J., Trinitis, C.: Autopin - Automated Optimization of Thread-to-Core Pinning on Multicore Systems. In: First Workshop on Programmability Issues for Multi-Core Computers, Gothenburg, Sweden (2008)
10. Schirmer, J., Torres-Espallardo, I., Minde, J., Spanoudaki, V., Trager, R., Ziegler, S.: Analytic and Monte Carlo Derived System Matrices for Small Animal PET Imaging. In: Annual Congress of the European Association of Nuclear Medicine, Munich, Germany (2008)
11. Shepp, L.A., Vardi, Y.: Maximum likelihood reconstruction for emission tomography. IEEE Transactions on Medical Imaging MI-1(2), 113–122 (1982)
12. Sun Microsystems: Sun Fire X4600 M2 Server Architecture, White Paper, `http://www.sun.com/servers/x64/x4600/arch-wp.pdf`