A method for OSEM PET Reconstruction on parallel architectures using STIR

Tobias Beisel, Stefan Lietsch and Kris Thielemans

Abstract—To accelerate image reconstruction of positron emission tomography (PET) data, we introduced an approach for parallel architectures by applying the message passing paradigm to an existing implementation of the ordered-subsets expectation-maximization (OSEM) algorithm for two- or three-dimensional (2D/3D) PET. To reduce the amount of time needed to complete a reconstruction, a cluster was used as well as different multi-core systems. A main focus was on the multi-core processors, as these systems are increasingly common and easy to use in medical environments.

The Open Source implementation ‘Software for Tomographic Image Reconstruction’ (STIR) was used as underlying reconstruction software, as it provides a wide range of usable scanner geometries and different algorithms. In particular STIR also offers an enhanced implementation of the OSEM algorithm. To allow for the use of arbitrary parallel systems, the standard message passing interface (MPI) was used to implement the parallel OSEM algorithm. Additionally, a special work package allocation algorithm was designed, that incorporates load balancing and improves the utilization of cached data. The resulting runtimes on up to 20 nodes and up to 8 cores were compared to the sequential runtime to determine the speedup and the parallel efficiency. The results using a limited set of test data achieved speedups of up to 8x, depending on the amount of data, the algorithm and the underlying architecture. We expect the algorithms to perform better for larger and more realistic data sets since the parallel overhead for distributing raw data and collecting the sub-results decreases in opposition to the actual computing time. The different parallel architectures were compared to determine the optimal system for PET Reconstruction. The cluster system achieved the best speedups using the available test data.

I. INTRODUCTION

POSITRON Emission Tomography (PET) is extensively used in modern internal medicine diagnosis. The images delivered by a scan are the basis for accurate diagnosis on tumors and cancer.

In recent years iterative image reconstruction techniques have gained importance and are the de facto standard to receive high quality image results. Modern PET scanners have a high number of detectors and are constantly improving their resolution, which leads to large amounts of measured data. This fact is transferable to the emerging system matrix, which gains size as well. Solving these non-linear systems of equations is complex and expensive on computation time. It takes up to hours to reconstruct the given data and provide a reliable diagnosis. Thus, speeding up this process is vital to provide quality results in an acceptable time for both the patient and the doctor.

A. Related Work

Several approaches have been made to improve the runtime of algorithms and the quality of their results by optimizing the EM-ML [1] or the OSEM [2] algorithm itself (e.g. [3] [4]). This often is a trade-off between runtime improvements and the loss of quality, or quality improvements by accepting increased runtime. We see great potential in speeding up existing algorithms by parallel implementation without decreasing the quality of the results.

Some successful approaches have been made to parallelize algorithms for PET image reconstruction by optimizing the underlying network structures [5] [6]. We concentrated instead on the idea of parallelizing an existing OSEM implementation on standard multi-core architectures, as these systems are of increasing interest and have a very good cost/performance ratio. The discontinued PARAPET [7] project dealt with ways to parallelize different algorithms (see [8] for the parallel implementation of OSEM). The (non-parallel) code of the PARAPET project was the predecessor of the Software for Tomographic Image Reconstruction (STIR) [9] which we used in our approach, too. STIR has been vastly changed and improved since then, which leads to the need of an adapted concept. The former parallel implementation is not transferable to the current version, as STIR was reimplemented in a more flexible and object oriented way. This complicated the message passing to a large extend. Still, the general idea of a data based distribution utilizing viewgrams (explained in Section II.) could be adapted. [10] describes a master-slave distribution, which is quite similar to the PARAPET approach. The EM-ML parallelization scheme described in their work can be transferred to most of the iterative approaches. The projection based distribution using a replicated local image copy turned out to be the most promising approach for a parallelization of STIR as well. The works of [11] and [12] partially describe a parallel realization of the OSEM algorithm using MPI and OpenMP. Their implementation was only tested on a cluster system with dual-core nodes. Recently, a lot of parallel implementations on Graphic Processing Units (GPU) were published (e.g. [13] [14]), but none of them is based on a powerful open source implementation like STIR. Our approach however is based on an open source implementation, which makes it more accessible and interesting for the PET community. It therefore needed a parallelization concept adapted to the software structure. In addition we support modern multi-core architectures and provide the corresponding results. This work is based on [15].

T. Beisel and S. Lietsch are with the Paderborn Center for Parallel Computing, University of Paderborn, Germany, e-mail: {beisel, lietsch}@uni-paderborn.de.

K. Thielemans is with Hammersmith Imanet Ltd, part of GE Healthcare, also maintainer of the STIR project (http://stir.sourceforge.net), email: kris.thielemans@ge.com.
II. IMAGE RECONSTRUCTION

In PET, a ring of detector units receive gamma rays produced by annihilation events in the examined object. The detected data consists of a number of coincidences between two detectors and is generally noisy. Therefore, it is advantageous to reconstruct the intensity images based on statistical methods.

Statistical algorithms are computationally intensive. This paper works towards improving this issue for the Ordered Subset Expectation Maximization (OSEM) algorithm [2], an iterative algebraic reconstruction method based on the EM – ML algorithm, additionally introducing the use of data-subsets.

Let

\[ W = \begin{pmatrix} W_1 \\ \vdots \\ W_n \end{pmatrix} \quad \text{and} \quad p = \begin{pmatrix} p_1 \\ \vdots \\ p_n \end{pmatrix} \]

be the system matrix \( W \) with \( n \) the number of subsets used and \( p \) be the measured data. The element \( w_{ij} \) of the system matrix \( W \) contains the information about the cutting length of the LOR \( i \) with the voxel \( j \) of the reconstructed image. That way it contains probabilities for pixels being the source of a measured event. With \( k \) numbering the sub-iteration (calculation of one subset) and \( f \) being the calculated intensities of the reconstructed pixels, the OSEM algorithm can be described as follows:

\[
\begin{align*}
\bar{f}_{k,0} &= \bar{f} \\
\bar{f}_{k,m} &= \bar{f}_{k,m-1} \frac{1}{W_{m}^T W_{m}^{-1}} W_{m}^T \bar{p}_{m} \quad ; \quad m = 1, \ldots, n \\
\bar{f}_{k+1} &= \bar{f}_{k,n}
\end{align*}
\]

This algorithm includes the forwardprojection \( W_{m} \bar{f}_{k,m-1} \) of the consecutive calculated image estimates (described by the intensities \( f \)) and the subsequent backprojection of the adjusted measurement values \( p \):

\[
W_{m}^T \frac{\bar{p}_{m}}{W_{m}^T \bar{f}_{k,m-1}}
\]

Multiplied with the normalization factor \( \frac{1}{W_{m}^T} \), the update image is applied to the current image estimate, resulting in the image estimates for the following sub-iteration \( k + 1 \).

A. Software for Tomographic Image Reconstruction (STIR)

This work uses STIR as a basis for parallelization. STIR focuses on being very modular and flexible. It is designed general enough to allow the extension towards other modalities like CT or MRI. It has been chosen as an underlying implementation, as it is offers a set of algorithms, supports several scanner geometries and is provided as C++ open source library. To our best knowledge it is the only open source solution providing such functionality. In addition it is widely known and used as a research tool within the PET community.

In this work we focus on the OSMAP-OSL algorithms of STIR, which provides an OSEM implementation extended by the MAP OSL algorithm by Green [16]. The OSEM implementation in STIR can be abstracted as shown in Algorithm 1.

Algorithm 1 OSEM.reconstruct(Proj_data)

1: for all sub-iterations do
2: for all segments do
3: for all views of one subset do
4: forward_project(viewgrams, image_estimate) 
5: update_measured_viewgrams(viewgrams) 
6: back_project(update_image, meas_viewgrams) 
7: end for 
8: end for
9: image_estimate* = update_image() 
10: optionally filter image_estimate 
11: end for

The algorithm performs the combination of a forward- and a backprojection on the viewgrams of each view, while only one subset is taken into account in each sub-iteration. The subsets are processed sequentially, using a different one in each sub-iteration. “Viewgrams” are a collection of all the PET data at one particular “view” (or azimuthal angle) and “segment number” (or ring difference) and varying axial positions \( z \) and tangential positions \( l \). Segments additionally vary the axial angle and thus construct a 3D set of data. Reconstructing only one subset for each segment reduces the computation time vastly with an acceptable loss of image quality. One subset represents the remaining data, as the subsets are assumed to fulfill the balanced subset condition [17]. The actual backprojection, and the forwardprojection respectively, breaks down the data to the level of single bins to be projected. STIR offers several different actual projectors, which are not discussed here, as they do not affect the general parallelization approach. Nevertheless they would affect the run-time and hence speed-up factors.

III. PARALLEL APPROACH

To further improve the performance of the already quite efficient sequential implementation, a parallel version seemed very promising. The goal was to develop a parallel implementation specialized on multi-core architectures, but possibly also including support for cluster systems. Shared memory architectures have the general advantage of avoiding network

![Fig. 1. X-Y-cutting plane illustration of parallel projections during the measurement. The data is saved in viewgrams, which save a 2-dimensional subset of the measurements in the field-of-view.](image-url)
A. Theoretical Parallelization Considerations

In medical image reconstruction algorithms the projections are well known as the parts of the algorithms demanding most computation time and thus are the natural starting-point for an efficient parallel implementation. This was verified by runtime analysis of the STIR algorithms using Valgrind. In fact, the projections make up at least 80 percent of the total runtime (see Section IV for details on the used test data). These measurements assumed a standard amount of 36 sub-iterations, which on average deliver a minimum-quality for reconstructed images, although for quantitative measurements, a higher number of sub-iterations would be necessary. 36 OSEM sub-iterations approximately correspond 36 full EM-ML iterations when it comes to the image quality, but only need the time of $36/n$ full EM-ML iterations in the non-parallelized implementation.

The remaining runtime is mainly consumed by the initial calculation of a sensitivity image, which includes the backprojection of the complete measured data.

A parallelization can only be done within each sub-iteration of the OSEM algorithm since the updated intensities $f^{k+1}$ rely on the calculated intensities $f^k$ of the previous sub-iteration (as shown in equation (2)) and thus prevent the parallel treatment. Because of that only the forward and backward projections within each sub-iteration and, according to that, within the calculation of one subset could be parallelized. This demands a parallelization of equation (3) only, as the normalization factor seen in equation (2) is pre-calculated at the beginning of the reconstruction. However, the operations in a sub-iteration can be parallelized in “sub”-subsets of the data. In this work, we have used viewgrams as the set of data to be distributed among the processing nodes. Distributing on a lower level would lead to high overhead, especially on distributed machines, as the amount of data given by a single view is already rather small.

The shown parallel algorithm (Algorithm 2) considers these theoretical restrictions and uses a master-slave architecture: Within each sub-iteration, the current estimate of the intensity image has to be broadcasted. The intensity estimates of each subsets’ viewgrams can be calculated in parallel, as they are independent from each other. After each sub-iteration, the image estimate has to be reduced at the master node. A reduction in terms of parallel computing is a summation of results at one node, in our case the summing up of the local image estimates to one combined image estimate at the master’s node. This simple scheme can be used in message passing as well as in thread based parallelization.

Algorithm 2

```
1: broadcast image_estimate
2: for all views and segments in the subset do
3: distribute viewgrams
4: calculate multiplicative update factor on each slave
5: use calculated factor on local image_estimates
6: end for
7: reduce reconstructed images at master node
```

B. Parallel STIR Implementation

Considering this works’ target to support distributed memory machines also, we focus on a MPI-based parallelization for better portability. This decision leads to a certain communication overhead, which had to be proven to be worth it. Some initial information has to be made available to all slaves participating in the reconstruction. This includes information about the projectors, the initial image estimate, a local target image and the objective function to be used for reconstruction. In addition the projection data needed for the actual reconstruction had to be made available. This is the largest overhead emerging compared to the sequential version. To keep this overhead small a special distribution scheme described in the next section has been developed.

The implementation of the parallel algorithm described in Algorithm 3 was done using a workpool based master-slave approach. In this approach it is the slaves duty to ask the master for work. As these requests cannot be handled at the same time, the algorithm was designed to process one request at a time, queuing the remaining request to be served one after another. This way each slave is provided with the next work package as soon as possible. As the inner loop of Algorithm 1 traverses the views of the current subset, it is guaranteed that in every loop exactly one view is distributed to one requesting slave.

This approach stands in opposition to a master-driven distribution, which only supports an a priori distribution of the viewgrams among the slaves or a master polling for available slaves. Doing that would avoid the overhead of slaves asking for work, but also prevents dynamic load-balancing, as some slaves might need longer for the same amount of work. Thus, the slowest slave would determine the overall computation time. Because of that, the workpool approach was chosen to be realized in the implementation.

1) Support for cached reconstruction data: A special distribution algorithm has been designed to support caching of reconstruction data at the working nodes. The distribution scheme aims at sending the same work packages to the working nodes in each iteration. As the measured data does not change over the iterations, a significant amount of redundantly sent data can be avoided. Load-balancing can be easily incorporated through the workpool approach, as working nodes have to ask for new data anyway and can be supplied on request.

This distribution scheme is realized on the master node by saving the combination of the slave-id and view-segment
number (vs_num) of the initial distribution (first iteration) and redistributing the work packages the same way in following iterations. This way, the measured data has to be sent only once and the slaves are reconstructing the same data sets each iteration. Because of that, the system matrix has to be calculated only once at the slaves and STIR-internal caching of the matrix is more effective. Variations can be easily absorbed, if workloads lead to different speeds.

2) Master node implementation: The following pseudo-code depicts the master’s distribution algorithm as described and implemented:

Algorithm 3 compute_update_img(Image curr_img_est)
1: ...
2: mark all views as not yet processed
3: for view_count do
4: wait for request from slave
5: if not_empty(former processed views of slave) then
6: send vs_num of former processed view to slave
7: mark view as processed
8: else
9: if first iteration then
10: get any unprocessed view
11: else
12: get unprocessed view of busiest slave
13: end if
14: send viewgrams(view) to be processed on slave
15: mark view as processed
16: add view to list of former processed views of slave
17: end if
18: end for
19: ...

Optimal load-balancing can theoretically be achieved by dynamically shifting workload from the node having the most remaining work packages to the one which is asking for additional work after being done with its initial work. Saving the view-to-node assignment leads to efficient redistribution of work packages in case of nondeterministic network load.

3) Slave node implementation: The slave has to implement an infinite loop to receive work packages to reconstruct. In fact there have to be two loops: one outer loop describing the loop over the OSEM sub-iterations and the inner loop representing the views of that specific sub-iteration. In case of a completed sub-iteration the local results have to be reduced at the master node, thus a reduction over the local image estimates has to be undertaken. Algorithm 4 specifies how the slaves were implemented in this work.

In every pass of the inner loop, one of three possible message-tags is received:
- NEW_VIEWGRAM_TAG is used to notify that new viewgrams, hence a view not already saved at the specific slave, has to be received.
- REUSE_VIEWGRAM_TAG is used in case of an already saved view resulting in reuse of available data.
- END_ITERATION_TAG is used to announce the end of an iteration and thus the need to reduce the calculated results.

Algorithm 4 Slave algorithm pseudo-code
1: Receive(initialization_data)
2: initialize_reconstruction() {sub-iteration loop}
3: while true do
4: Receive(Image img_est) {view-reconstruction-loop}
5: while true do
6: Receive(Tag t, Identifier i)
7: if t=REUSE_VIEWGRAM_TAG then
8: Viewgrams = load_viewgram(i)
9: else if t=NEW_VIEWGRAM_TAG then
10: Receive(Viewgrams v)
11: else if t=END_ITERATION_TAG then
12: break
13: end if
14: img_est += reconstruct_viewgram(v)
15: Send(Request req) {request new work}
16: end while
17: Reduce(img_est)
18: end while

This implementation is rather simple, but still uses the available computation power of the slaves in the best possible way.

4) distributed package: Fig. 2 shows the main parts of the parallel implementation. Besides the algorithms shown above, the distributed package represents the most important parallel features. As a collection of distributed communication functions it encapsulates the whole functionality implemented in MPI. That way, sending, receiving or reducing values can easily be done by calling a method from the package and simply handing in the input and output parameters. This allows for the transparent use of MPI functions. Additional functions could be added by reproducing the simple structure of the given functions.

While DistributedWorker implements the slaves as explained above, the master implementation was integrated into the existing STIR classes. The distributable class implements the loop over the views of the current subset and thus was altered to use the slaves objective function rather than the local reconstruction. Thus it distributes the views to the requesting slaves. The cache supporting algorithm described above was implemented in an extra function distributableCacheEnabled, which uses the master’s special distribution scheme implemented in DistributedCachingInformation. The latter also saves the distribution information as described before.

5) Parallelizing the sensitivity calculation: The sensitivity calculation in STIR makes use of an object hierarchy for computing the normalization and attenuation correction factors. To calculate the factors this object would have to be available to all slaves. As there are a few different types of normalization objects, this is a non-trivial task. As an alternative, we could calculate the factors at the master and send only those that each slave needs. In this work we have not parallelized the sensitivity calculation, and concentrated on the work during the iterations. As our timings include the sensitivity calculation, this has an impact on the theoretically
obtainable speedup of our parallelization scheme, especially when using only a few (full) iterations.

C. Theoretical Evaluation

A combination of the workpool approach with the described caching algorithm has several advantages:

- optimal load-balancing
- reduced communication overhead by saving projection data and avoiding a polling master
- effective reuse of cached system matrix
- minimized idle times for the slaves

The current evaluation described in Section V has some drawbacks concerning the used test data however, as only limited data was available for test runs. The transmission time of the identifiers is 110 up to 460 times faster than the time needed for the actual data. However, the effective time saved is too small (≈ 0.1 sec. for the complete reconstruction) to have deep impact on the total runtime. The computation time of the reconstruction overcomes the savings by orders of magnitude. In addition the cache-miss rates regarding the system matrix were measured as already minimal (< 0.02%) without using the described distribution scheme. This was because all slaves were identical in our system and the tests were run without additional system and network load. Thus, the assignment of work packages was nearly optimal without using the enhanced distribution scheme. It was observable, that cache-misses merely occurred in the first subiteration of the reconstruction. The rate was reduced to < 0.001% in the parallel version, but did not affect the total runtime essentially. However, we expect the idea to significantly affect the total runtime if it comes to bigger amounts of measured data. This is conceivable, especially as the scanners improve in resolution and thus deliver larger data files. Load scenarios additionally increase the effectiveness of the algorithm.

IV. MATERIALS

A. Data used for performance tests

The implementation was tested with a 2D and a 3D data set measured on an ECAT EXACT HR+ tomograph. Each segment has 144 views, while the 3D data consists of 5 segments. In STIR, projection matrix elements are only computed for 36 out of 144 views and 3 out of 5 segments, as the remaining views and segments can be derived by symmetries. The remaining views to be computed are called basic-views, basic-segments exist accordingly. The size of each work package, the viewgrams of one view respectively, is calculated by the number of axial positions and tangential positions and the number of viewgrams related to each other (mostly 8 in our case). The used scanner delivers 63 axial positions and 288 tangential positions. According to that, each work package has the amount of 63*288*8 float values. This sums up to a size of 567kB of overall sample data in each communicated work package. The resulting images have dimensions of (265; 265; 63) with voxel size (2, 25; 2, 25; 2, 425). Thus the images have a size of 265*256*63 pixels. Using a 32 bit float value for each pixel in STIR, resulting images have a size of 15.75MB and have to be broadcasted and reduced in each iteration.

Fig. 3. Intel Xeon Clovertown architecture: Each of its 8 cores has its own 8kb L1-Cache and every 2 cores share 4MB of L2-cache.
B. Parallel Architectures

An Intel Xeon Clovertown System with 8 cores was used as a test architecture as well as a cluster system described subsequently. GCC 4.1.0 was used in the tests.

1) Intel Xeon Clovertown: The Intel Xeon Clovertown system (Fig. 3) is a 2-way system, a combination of two Intel Clovertown processors, each consisting of 2 Intel Xeon Woodcrest dies with 2 cores. The cache is big enough to handle either the measured data distributed by the master-node or part of a local image estimate, but not both. In addition, the 2 internal cores will compete for the L2 cache. Furthermore, the L2 cache would occasionally be used for the master-slave communication. Therefore, we expect that access to the main memory via the Front Side Bus (FSB) will increase and might emerge as a bottleneck once more than 3 or 4 slaves are used.

2) ARMINIUS Cluster: Besides the Intel Clovertown system, the Xeon-DP based ‘ARMINIUS’ Cluster\(^1\) of the University of Paderborn was used, which provides up to 400 Ethernet- or Infiniband-connected Dual Intel Xeon 3.2 GHZ EM64T processors with 2MB L2-Cache and 4 GByte main memory each. Measurements of this work were limited to ScaMPI compiler, achieving the best performance among MPI compilers, as known from experience. GCC 4.1.0 was used in combination with those compilers.

\(^1\)see http://www.uni-paderborn.de/pc2

V. RESULTS

Running parallel STIR on an Intel Xeon Clovertown system leads to the runtimes and speedups shown in Fig. 4 to 7 using different numbers of subsets. The number of subsets plays a major role in the benchmark. The higher the number of subsets, the less computations have to be calculated in each sub-iteration for constant communication overhead (image broadcast and reduction) and hence the less efficient the parallelization can be.

The results shown in Fig. 4 and 5 were obtained with 2D data, which provides only a rather small amount of viewgrams and hence a relatively large overhead in communication. The master-slave architecture allows a maximum speedup of \(n - 1\), as \(n\) defines the number of nodes and the master-node does not contribute to the actual reconstruction.

The possible number of nodes is limited by the amount of data and the amount of basic-views and -segments to be reconstructed in STIR. With \(n\) being the number of subsets, \(v\) and \(u\) the number of basic-views and -segments respectively

\[
t \leq \left( \frac{v - 1}{n} \right) \cdot u
\]

shows the maximum number \(t\) of processing nodes. Using the HR+ scanner, we have generally 4 views related by symmetry, 0 and 45 degrees have only 2. That means: \(v = 144/4 + 1, u = 1\) using 2D- and \(u = 3\) using 3D-data. This leads to upper-bounds on utilizable processor-cores to be used for the parallel reconstruction on specific data sets. Regarding the
test-data used in this work, Table I shows the restrictions to be considered when interpreting the results. As equation (4) calculates the slaves only, 1 node is added as a master node in the table. Using the 2D dataset, only 5 nodes can be used efficiently for a high number of subsets. Thus, the speedup decreases from that point.

On Intel’s Clovertown, we were able to identify the FSB as the bottleneck for the parallel reconstruction. As local images were stored for each slave and these images have a size of roughly 16 MB (using float values) which have to be read and written during the reconstruction, the amount of data transferred to and from the main memory exceeds the available bandwidth of the system bus. Because of that, the total runtimes stagnate and the speedup decreases using more than 5 nodes. Overcoming this problem is a trade-off between storing less image data and more measured data at the slaves. Using a more adopted approach concerning the memory architecture might lead to better results on different shared memory machines.

<table>
<thead>
<tr>
<th>subsets</th>
<th>max. nodes</th>
<th>max. nodes</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>2D data</td>
<td>3D data</td>
</tr>
<tr>
<td>2</td>
<td>19</td>
<td>55</td>
</tr>
<tr>
<td>3</td>
<td>13</td>
<td>37</td>
</tr>
<tr>
<td>4</td>
<td>10</td>
<td>27</td>
</tr>
<tr>
<td>6</td>
<td>7</td>
<td>18</td>
</tr>
<tr>
<td>9</td>
<td>5</td>
<td>13</td>
</tr>
</tbody>
</table>

The concept of the OSEM algorithm itself and its implementation in STIR conflicts the parallelization efficiency, as less data is computed in total. On the other hand lower number of subsets led to higher quality of the reconstructed image, as more data is taken into account. Thus the parallel STIR implementation can provide better results in the same amount of time or results of the same quality in less time. Using the much larger 3-dimensional data, the upper bound on nodes does not affect the results on Intels’ Clovertown system, as the number of available processing nodes and the FSB bandwidth become the restricting factors.

This is why the cluster system was analyzed as well. With up to 400 available nodes, each having its own main memory, the only boundaries are set by the amount of data. The tested amount of 3D data was large enough to be computed on at least 13 nodes using up to 9 subsets.

It is observable in Fig. 8 to 11, that the total runtime can be reduced to a fraction of the sequential runtime. A few more facts have to be considered evaluating these results. The speedup of the cluster system has to be qualified concerning the total runtime. In fact, comparing the sequential runtime of the Intel Xeon Clovertown with one Dual Intel Xeon of the cluster system, the Intel Clovertown system outperforms the cluster clearly. The sequential runtime on the cluster is slower by an approximate factor of 3. In addition we have to consider, that using a distributed memory system always leads to certain communication overheads, which affect the total runtime as well.

Apart from the results shown in Fig. 4 to 11, the caching
of measured data at the slaves was tested as well and showed the tendency to small speedups on larger amounts of data, in particular on 3D data sets used on the cluster. As the communication costs on the cluster are higher than on multi-core systems, using the cached data lead to better speedup on distributed memory cluster systems. This trend could not yet be quantified with the available 3D data sets. In addition, high network load was not yet tested.

VI. CONCLUSION

The measured results indicate that the Intel Clovertown system is the better choice for the parallel STIR implementation concerning the clinical use. The speedups are inferior to the cluster results, but the overall runtime is crucial for the practical use. Concerning 2D data this system could reduce the total time from 9 to 3.5 minutes. Using 3D data the time was reduced from 29 to 9 minutes in the best case. This fact, and the ease of use within a medical environment compared to a cluster system, makes it a promising architecture for the parallel computation of PET algorithms on site. Current and next generation accelerators like Graphic Processing Units or Clearspeed accelerators provide a much higher number of cores and thus further compensate the drawback of the restricted number of processing nodes on the Intel Clovertown system. The shared memory architectures are most likely the superior architecture to be implemented in the medical environment, supported by the facts that they are smaller, easier to handle and much better in cost-effectiveness. Nevertheless, the limited system bus bandwidth has to be addressed in future implementations, as more data could even increase the impact of this bottleneck. Thus, adapting the implementation to specific memory architectures is crucial for achieving better speedups.

STIR might not be the best choice concerning the performance, but it definitely is a widely accepted research tool, which provides substantial functionality on open source basis. This extent had some drawbacks concerning a parallelization, as complex existent software prevents an optimized bottom up approach. However, its easy access to broad functionality makes it a well approved application and thus an interesting tool concerning a parallel acceleration.

A. Future work

The achieved results provide a satisfying basis for further improvements. Enhancements on the parallel software might be achieved by improving the parallelization of the algorithm’s initialization, in particular the computation of the sensitivity image. This work also enables parallelization of other algorithms, such as the OSSPS algorithm [18]. We expect the speedup factor to be much larger for data acquired by modern PET scanners with larger number of detectors. In addition, it is possible to switch off some symmetries used in STIR, which promises higher speedups using the parallel version on a large number of nodes. This still has to be tested. The decision to prefer shared memory systems rather than distributed memory systems in terms of a better cost-benefit ratio allows an adapted implementation incorporating the use of shared memory space on specific architectures. Further optimizations on used data structures could improve the caching of projection data implemented in this work and address the FSB bottleneck shown on the Intel Clovertown system.

ACKNOWLEDGMENT

The authors would like to thank Dr. Harald Fricke of the ‘Herz- und Diabeteszentrum Nordrhein-Westfalen’ for the consistent support and the provision of the used test data.

REFERENCES