Ultrasound Computer Tomography, Distributed Volume Reconstruction and GRID Computing

Tim Oliver Müller, Nicole Valerie Ruiter, Rainer Stotzka,
Michael Beller, Wolfgang Eppler and Hartmut Gemmeke

Institute of Data Processing and Electronics, Forschungszentrum Karlsruhe,
Hermann-von-Helmholtz-Platz 1, 76344 Eggenstein-Leopoldshafen
Email: Tim.Mueller@ipe.fzk.de

Abstract. At Forschungszentrum Karlsruhe the demonstrator of a new medical imaging system based on 3D ultrasound computer tomography will be available at the beginning of 2005. This system requires not only the recording but also the processing of very large datasets of about 3.5 GBytes. On a single PC the reconstruction of a high resolution volume will take up to several weeks. However the reconstruction process can be parallelized. Our long term goal is to accelerate the reconstruction process significantly by the use of GRID technologies. In this paper we present our approach to parallelize the reconstruction by A–scan–decomposition using Java RMI and Matlab.

1 Introduction and Motivation

Improving medical imaging often leads to significantly increasing amounts of data. In particular this concerns 3D ultrasound computer tomography (USCT) for breast cancer detection developed at the Forschungszentrum Karlsruhe that requires the processing of approximately 3.5 GBytes (580 thousand A–scans) to reconstruct a volume from the dataset. At present the reconstruction of a volume of 256x256x256 voxels from a simulated dataset using a PC (1.6 GHz, Pentium 4, 1 GByte memory) takes approximately three weeks. Though the implementation of the algorithms is experimental, a significant acceleration of the reconstruction process is inevitable to achieve acceptable reconstruction times in the range of minutes and thus making 3D ultrasound computer tomography clinically relevant. The reconstruction can be accelerated either by using faster computing hardware or by parallelizing the algorithms. The use of fast supercomputers would be too expensive for a single institution like e.g. a hospital. Parallelized algorithms require massive computing power, but only for a very short time. Such erratic burst requests for computing power are optimally covered by a GRID that is a network of several thousand computing units. The computing units of a GRID are shared between many users, which integrate their own computing units into the GRID and in exchange consume computing power from other users just at the time they need it.

The processing of any application can be accelerated using a GRID if a) the application can be decomposed into many subprocesses of lower complexity and b) the results of mentioned subprocesses can be composed into the final result fast enough.

2 Methods

To transfer the USCT volume reconstruction to GRID, two objectives have to be covered. In this paper we show the decomposition of the reconstruction process and the distribution of the subprocesses to computing units using Java RMI. The derived methods will be integrated into GRID services in future work to achieve our long term goal.

2.1 3D Ultrasound Computer Tomography

So far a 2D demonstrator of ultrasound computer tomography has been successfully developed and reproducible images of high resolution [1] were obtained. Because of this promising results we build an experimental 3D hardware setup that will be finished in early 2005 [2]. The algorithms for 3D reconstruction and data handling in Matlab have been implemented and tested. The reconstruction of a volume of 64x64x56 voxels of simulated point-spread objects was successful and took approximately 8 hours on a 1.6 GHz Pentium 4.

For a methodical measurement the 2D algorithms and data have been chosen, because of the significant shorter runtime. However the reconstruction algorithms of 2D and 3D are identical. The reconstruction principle for 2D/3D ultrasound computer tomography is explained in [2, 3]. For this algorithms two different decomposition methods have been considered:

Pixel/Voxel–Decomposition: The image/volume to be reconstructed is divided into subimages/subvolumes. The smallest possible subprocess is the reconstruction of a single pixel/voxel. The size of the result is the number of pixels/voxels times their size in bytes. For each subprocess the complete dataset is required.

A–Scan–Decomposition: The reconstruction algorithms base on the principle of superposition. Basically this means the contribution of each A–scan to all pixels/voxels can be summed up. The smallest possible subprocess is the reconstruction of a full image/volume using only a single A–scan. The size of the result is the number of computing units times the size of the reconstructed image/volume. For each subprocess only a small part of the dataset is needed.

For now we decided to use the A–scan–decomposition, because the reconstruction of 2D images of 1024x1024 pixels leads to a transfer size of 3.5 GBytes (size of the whole dataset) and 8 MBytes for the result per computing unit, whereas the pixel/voxel–decomposition would need for e.g. 8 computing units 28 GBytes of network traffic.

2.2 GRID Technologies and Client–Server Models

The main idea behind GRID computing is to provide transparent services (GRID–services) for high–intensive computing applications using a computer network. GRID–services need an abstract coordinating layer called middleware. Examples of well known middleware are the GlobusToolkit based on OGSA (Open Grid Service Architecture) [4], AliEn or gLite [5]. An easy to use middleware for USCT is not yet available. Existing technologies i.e. development drafts or implementations need to be adapted and reimplemented. But until now no "winner"is in sight and the choice of a suitable middleware is delayed. Therefore for the test the distributed algorithms are provided on a conventional Client–Server model. The experience with this model will be integrated into the development of suitable GRID–services.

Well known conventional Client–Server models are Applets, Servlets, Common Gateway Interface (CGI), etc. Another widely used technology to control parallel processes is the Message Passing Interface (MPI), which is available for many different platforms. Each of these technologies has its advantages, but we prefer the Java RMI [6] as Client–Server model, because it fits best into our existing Java projects. It is easy to implement, virtually platform independent and provides several security mechanisms. Last but not least the development of graphical user interfaces is very easy in Java. But almost any other model would have been suitable as well.

2.3 Implementation of Framework

Our model of distributed processing is very similar to the organization of a bee hive. The bee queen (as client) has knowledge of a set of tasks that have to be executed. The bees of the hive (service providers, i.e. servers) register at the bee queen and wait for tasks to process. The bee queen deploys tasks to idle bees. The deployment is performed using a parameterized remote procedure call of a Java RMI interface. After a server has successfully finished its task, it notifies the client. Again a task is deployed until no more tasks are left. From time to time the bee queen checks for dead bees i.e. malfunctioning servers. If a server does not answer this request it is presumedly dead and deleted from the task deployment list. The task for this bee is revoked and deployed to another bee. Bees may enter or leave the hive dynamically.

The client possesses a graphical user interface for visual feedback of the status. The active bees are displayed and several levels of communication may be logged and visualized. The deployment of the tasks is triggered manually via the GUI or automatically at start–up. The Client–Server model is implemented as java library. Communication and data structures can be easily recycled for other applications, which are suited for distributed processing. For any other application, only two new interfaces need to be implemented: the interface *Task* that wraps the concrete decomposed task to be performed for this application, and the interface *TaskIterator*, which enumerates the decomposed tasks in a suitable order.

Table 1. Overview of computing units. All units contain Pentium 4 processors except for unit 3, which is a Pentium 3. The total column contains the sum of the rows.

CPU	1	2	3	4	5	6	Total
Clock GHz	1.6	3.0	1.0	3.2	2.2	3.0	14.0
Memory MBytes	512	2048	512	1024	1024	1024	6144
MFlops	227	429	193	453	313	429	2044

Table 2. Running times in hours:minutes for different resolutions and numbers of computing units. The units are combined in order of tab. 1.

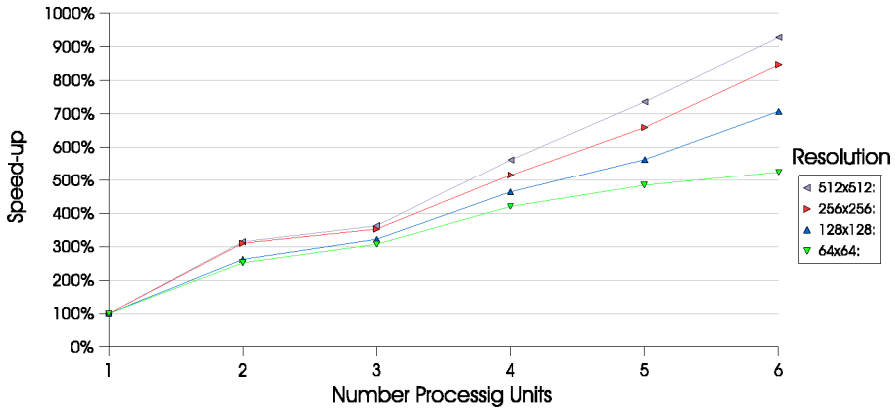
Resolution	1 CPU	2 CPUs	3 CPUs	4 CPUs	5 CPUs	6 CPUs
64x64	3:19	1:14	1:05	0:47	0:41	0:38
128x128	4:34	1:44	1:25	0:59	0:48	0:39
256x256	11:05	3:34	3:08	2:09	1:40	1:18
512x512	34:54	11:06	9:37	6:12	4:44	3:45

The dataset for 2D image reconstruction consists of 100 emitter positions each with 1456 receiver positions. A *Task* was defined as a the reconstruction of an image of size NxN from a single emitter position, thus leading to 100 different task, which the *TaskIterator* iterates from position 1 to 100. Each task executes a system command starting Matlab with a dynamically generated Matlab–Script. The required data is accessed via a network shared drive. After finishing the task the resulting image is summated to a previous image, if existing, and stored locally as previous image for the next task. After all tasks are finished, the images from the different computers are composed on the client's computer for the final result.

3 Results

The 2D reconstruction was measured for different resolutions and an increasing number of computing units. A network with very different computing units (see tab. 1) has been used. Tab. 2 shows the running time for different resolutions adding more and more computing units. Fig. 1 shows the speed–up in percent adding more and more computing units. Overall, the acceleration of the reconstruction follows almost linear the number of units/clock frequency for the six computing units used. For the resolutions 256x256 and 512x512 no limit caused by decomposition/composition was found yet. However the runtime using six computing units at resolution 128x128 and 64x64 is almost identical, which indicates that no further acceleration is possible for this resolutions. This limit is caused by accessing the dataset from a shared network directory. For reconstruction the complete dataset of 20 GBytes has to be transferred, which on a 100 MBit ethernet network takes approximately 27 minutes theoretically without any overhead considered.

Fig. 1. The figure shows the speed–up depending on the number off computing units. Except for the resolution of 64x64 pixels the reconstruction is accelerated virtually linearly. Using 6 CPUs the absolute runtime for 64x64 and 128x128 pixels is almost identical (see Tab. 2), indicating that no further acceleration is possible because of the network shared dataset.



4 Discussion and Future Work

The results show clearly the potential of distributed reconstruction for USCT. The performance gain was virtually linear for all resolutions limited only by accessing the dataset from a network shared directory. A solution may be to replicate locally the dataset. Though the linear performance gain was expected, the usability of the proposed decomposition/composition is still to be shown in a "real"GRID with e.g. 100 computing units. However, the required speed–up of several magnitudes can only be achieved if an alternative for distributing the dataset via a network shared drive is used.

The implementation of the distributed image/volume reconstruction as Java RMI is a first step to use GRID technologies and middleware, which are inevitable for our vision of realtime USCT imaging.

References

1. Stotzka R, Ruiter NV, Müller TO, et al. High resolution image reconstruction in ultrasound computer tomography using RF signal deconvolution. In: SPIE Medical Imaging; 2005.
2. Müller TO, Stotzka R, Ruiter NV, et al. 3D Ultrasound–Computertomography: Data Acquisition Hardware. In: MIC – IEEE Medical Imaging Conference; 2004.
3. Stotzka R, Würfel J, Müller TO, et al. Medical Imaging by Ultrasound–Computertomography. In: SPIE Medical Imaging. vol. 4687; 2002. p. 110–119.
4. Alliance Globus. Homepage. http://www.globus.org/; 2004.
5. Project AliEn. Homepage. http://alien.cern.ch/; 2004.
6. Wollrath A, Waldo J. SUN Java Homepage – The Java RMI Tutorial. http://java.sun.com/docs/books/tutorial/rmi/index.html; 2004.