Design of a FEM Computation Engine for Real-Time Laparoscopic Surgery Simulation

Alex Rhomberg, Rolf Enzler, Markus Thaler, Gerhard Tröster
Swiss Federal Institute of Technology
Electronics Laboratory
CH-8092 Zürich
e-mail: rhomberg@ife.ee.ethz.ch

Abstract

We present the design of a computation engine for a real-time laparoscopic surgery simulator. Since this simulator requires realistic real-time and real-world behaviour, a physically correct model of the biological system has to be used, which leads to FEM modeling of the organs. The requirements imposed by dynamic FEM calculations cannot be met with state of the art parallel computers, since they are not optimized for the special needs of the underlying algorithm. Based on these requirements, we are designing a parallel processing engine that optimally supports communicational and computational needs. To support concurrent development of soft- and hardware, we have developed a simulator for the target hardware which allows continuous monitoring of the overall system performance.

1 Introduction

Laparoscopy is a special surgical technique for operations in the lower abdomen which only requires a few small incisions to insert instruments and an endoscope. In old style laparotomy, the skin had to be cut to access the operation site. The basic idea behind laparoscopic surgery is to minimize damage to healthy tissue while reaching the actual surgical location. This results in a major gain in patient recovery after the operation. The price for this advantage is paid by the surgeon who loses direct contact with the operation site.

The operations are usually performed under monoscopic vision and under highly restricted manipulative freedom which requires very special skills of the surgeon. Up to now, no appropriate training devices are available which would allow to fully acquire these skills before actual intervention on patients.

In our project, we are building a Virtual Reality based Laparoscopic Surgery Simulator (LASSO) that helps surgeons to gain experience. The simulator consists of an Onyx2 parallel computer by Silicon Graphics Industries which is used for the computer graphics part, a force feedback device for tactile physical I/O and a parallel computer optimized to simulate the organs in the lower female abdomen based on a Finite Element Method (figure 1). In the following, we concentrate on this specialized parallel computer.

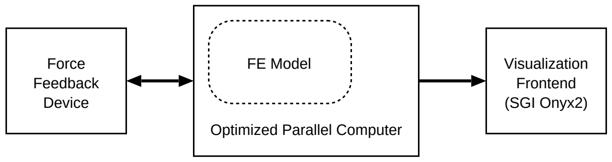
Force Feedback Device
FE Model
Optimized Parallel Computer
Visualization Frontend (SGI Onyx2)

Figure 1. System Overview

The difference between our and other approaches is the modeling of the human organs. Existing simulators are based on computer graphic approaches and use simplified methods like mass-spring models [3], which can be calculated on state of the art workstations. To get realistic results, physics based modeling must be used. Thus our approach starts from a mechanical representation rather than using only Computer Graphics methods. The search for physically correct methods leads to the Finite Element Method, which is very computationally intensive.

2 Finite Element Calculation

A Finite Element model is used to simulate the behaviour of the organs and filaments in the lower female abdomen.

To use the Finite Element Method (FEM) the organs are divided into lots of tiny cubes, the elements, which interact with each other [1]. With static FE calculation, the deformations and forces of a model, e.g. a bridge, are calculated for given boundary conditions. With dynamic FEM, different time-steps have to be calculated where the new positions of the elements depend not only on external forces, but also on old positions and movements.

For dynamic FE calculation the following differential equation has to be solved:

$$\mathcal{M}\frac{\partial^2 \vec{u}}{\partial t^2} + \mathcal{C}\frac{\partial \vec{u}}{\partial t} + \vec{f}_{int} = \vec{f}_{ext} \qquad (1)$$

Here $\vec{u}$ are the displacements, $\mathcal{M}$ is the mass matrix, $\mathcal{C}$ is the damping matrix and $\vec{f}_{int}$ and $\vec{f}_{ext}$ are the internal and external forces, respectively.

2.1 Implicit and Explicit Methods

There are two principles to solve this equation. One is to implicitly solve the set of nonlinear differential equation for $\partial^2\vec{u}_t/\partial t^2$, $\partial\vec{u}_t/\partial t$ and $\vec{u}_t$ given $\partial^2\vec{u}_{t-\Delta t}/\partial t^2$, $\partial\vec{u}_{t-\Delta t}/\partial t$ and $\vec{u}_{t-\Delta t}$. Implicit solvers are able to use rather large time-steps Δt which would be preferable for real time systems. The problem is that they cannot handle collision detection, which is required in our case where different organs can collide [8]. This makes implicit solvers unusable.

The other approach is to use the explicit method, where the new displacements are calculated out of the old positions:

$$\vec{u}_{t+\Delta t} = \mathbf{F}\left(\vec{u}_t,\, \vec{u}_{t-\Delta t},\, \Delta t\right) \qquad (2)$$

When an organ is touched by a surgical instrument or by an other organ, the forces propagate through it as a shock-wave at the speed of sound, which itself depends on the material parameters. The calculations must be fast enough to follow this shock-wave, which sets the upper limit for the time-step Δt:

$$\Delta t < \frac{\text{Smallest Dimension}}{\text{Speed of Sound}} \qquad (3)$$

Since the whole computation has to be done real-time, this time-step directly affects the necessary performance of the parallel computer.

By changing material parameters, the speed of sound can be decreased up to a certain point which also changes how an organ feels to the surgeon. The parameter changes can not go too far, otherwise the model would give an unrealistic impression like e.g. foam rubber. The other way to increase the minimum time-step is to maximize the smallest dimension. With this, there is a second advantage: If the elements become larger, there are also fewer of them that have to be calculated. The needed computational power relates to $O(n^{4/3})$ where n is the number of elements in the system, since the minimum dimensions will likely decrease when the element count increases. On the other hand, decreasing the element count is contrary to the goal of modeling organs in great detail.

2.2 Our System

Within our system the chosen time-step is rather a trade-off between the requirements of the physicists and those of the hardware designers. We chose 100μs as time-step, which allows to realize an appropriate hardware at moderate cost and yields results that feel good to the surgeon using the simulator. From the hardware designer's point of view, this short time-step still poses many challenges. Unlike most other FE problems, it is not the number of elements but the rate at which the new element positions have to be determined again and again. Compared to static FE problems, where often millions of elements are needed, we only aim at 2000 elements.

3 Performance Requirements

3.1 Computation

Analysis of optimized explicit Finite Element algorithms shows, that approximately 700 floating point operations per element are needed in each time-step. Additional computation time has to be reserved for collision detection and handling. This leads to the number of 10 MFLOPS per element for the chosen time-step of 100μs. For 2000 elements, a total of 20 GFLOPS sustained are needed.

This amount of computational power is much too high for a state of the art workstation. Implicit FE calculation can be implemented well on vector-supercomputers, since the main task is to solve huge sparse systems of linear equations [6], [4]. But the time-steps and the vectors of the explicit method are too short, therefore a high performance parallel machine is needed.

The latest processors are able to deliver $>$300 MFLOPS with optimized programming, so 64 of these will meet our demands. As Finite Element calculations have a tendency to become unstable, accurate computation using double precision floating point operations is necessary. Software overhead for communication has to be subtracted from the available computation time. As each processor has to send and receive five or more messages during one 100μs time-step, any protocol/network that needs processor time in the range of microseconds is not acceptable.

3.2 Communication

There are three different types of communication: exchange of forces, collision detection and handling as well as data transfer to the graphics engine. To calculate new positions, forces in the corners of the elements (nodes) have to be added to get the resulting force and thus the movement of the nodes. If these elements are distributed among different processors, forces have to be communicated. For a FE model with 2000 elements on a $4 \times 4 \times 4$ processor machine, 5600 force-vectors have to be exchanged each time-step. This results in a communication bandwidth of 1.35 GByte/s. This is a huge number, but the fact that data only has to be transferred between neighbours eases the requirements.

The second type of communication is used for collision detection. Only part of the surface nodes are potentially colliding with other parts of the FE model, thus about 300 vectors have to be exchanged for a 2000 element model (72 MByte/s).

Finally, the information for all the surface nodes must be sent to the graphics engine. This takes place once every 40ms, but to avoid adding too much latency to the system, data should be sent in a shorter time (20 time-steps) . With this, 18 MByte/s has to be sent to the graphics engine interface.

To exchange forces, high communication bandwidth is needed, but the pattern is well known and data travels only short distances. This fact can be exploited by using an appropriate communication network and lightweight protocols that have minimal software overhead. Communication of surface data is more difficult and therefore is better done on a different and more flexible network.

4 Parallel Processing Architecture

On a parallel computer, the computation has to be distributed among different processors. This problem can be solved rather intuitively. Division in time is impossible, because the position of an element at a certain time can only be determined when the previous positions are known. Thus we cannot distribute different time-steps among different processors like in image rendering. This leaves division in space. A part of the elements are assigned to every processing element (PE). Elements that touch each other interact, thus they have to exchange data. If elements are located on different PEs, interprocessor communication is necessary.

To keep requirements low, the structure of the communication network is adapted to the problem, which is of a three-dimensional nature. This leads to a three-dimensional

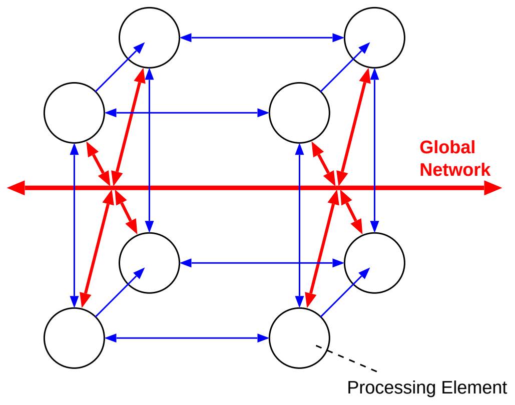

Figure 2. 3D Communication Architecture

mesh where every PE is connected to its six neighbours (figure 2). Every PE calculates a cube of the whole FE model, so it has to exchange data from the faces of its cube with PEs that share the same faces. In addition there are also edges and corners, where data has to be shared among processors that are not connected by a direct channel. Such data has to be routed through other PEs. This is actually no drawback, since in most cases the intermediate PEs have also data to transmit to the same target PE, so they just add their values before transmitting the received data block.

4.1 Local Communication

Data that has to be shared among neighbours must be available within the same time-step it is calculated and communicated. Thus it is necessary that the communication network provides enough bandwidth and very low latency.

To achieve the necessary low latency, we have to exploit the specialties of our communication pattern. If data has to be exchanged between processors, it must be communicated in every time-step. Fortunately it is known at program time which data values have to be sent in what order to which destination. Secondly, when the elements are assigned to the processors, only those processors that have elements which share common faces of the cube have to exchange data (collision detection is an exception to this). Having this in mind, the best way to provide sufficient communication bandwidth with low latency and minimum software overhead is to use point to point communication and to transmit pure data without additional address information. There is no need for routing information when point to point channels are used. Sending only data values minimizes software overhead on the sender side. But the data still has to be recognized by the receiver. The address information of the data is implicitly given by the order within the time-step it is sent. It is known at compile-time which values have to be exchanged, so the tool that partitions the elements also sets up the order in which data values are sent and received.

4.2 Global Communication

Communication takes mostly place between neighbours, but data for collision detection has to be exchanged between two processors that may be far away within the mesh. Part of these data values have to be broadcast to all other processors. Additionally, several processors can send and receive data at the same time.

Routing such data through the local channels is very inefficient. Bandwidth is used that is needed for local communication. The routing may consume processor time which can better be utilized for FE calculation and routing through several PEs increases latency to an unacceptable level. Therefore, such messages are better sent over a global network over which any PE can exchange data with all others. Data must be identified, thus packets of data must be sent along with information about the receiver as well as an identification.

5 Software Structure

Software and hardware are developed concurrently with the help of a library that simulates the hardware architecture with its communication channels. The simulator enables the programmer both to test the parallel program and to speed up computations of large test sets by using a SGI Onyx2 parallel computer.

5.1 Computation Cycle

Within every time-step, the same basic calculations have to be executed, using both local and global communication:

- The inner forces of each element are determined.
- The forces are accumulated in the element nodes.
- Collisions between different organs are detected.
- The resulting force in each element node is calculated.
- The new displacements are determined from the resulting forces and the last positions.

The software is structured to fit the hardware as well as the requirements imposed by the FE method. It must handle local and global communication and it should perform computations in parallel with communication. This means that

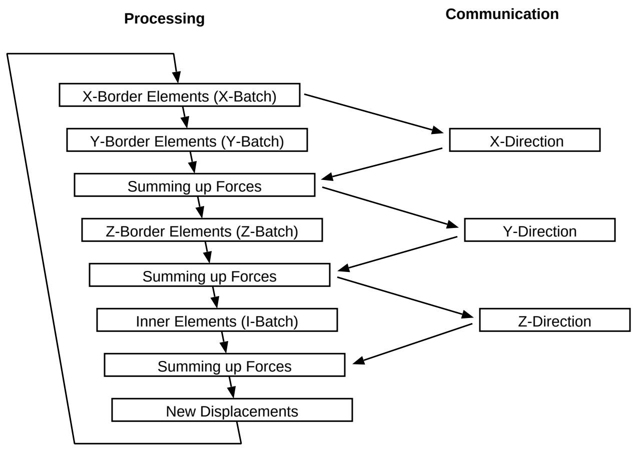

Figure 3. Program Flow

there must always be some part of the program that can be executed between sending and receiving data to and from a neighbour, respectively. This is illustrated in figure 3.

As seen above, inner forces must be computed, accumulated and new displacements have to be calculated. If several processors share a node at the boundaries of their respective cubes, each processor receives all the forces and uses them to determine the new displacements. With this, only one local communication step per neighbour is necessary. Accumulation of rounding errors can be eliminated by exchanging the positions of the nodes in an additional communication step every second (10’000 time-steps).

The elements which have to be handled by one processor are divided into four batches and local communication is done in three steps. First, all elements on the X-borders (X-batch) are calculated and the forces of the shared nodes are sent to the neighbours in X-direction. While these values are communicated, the elements on the Y-borders (Y-batch) are processed. At the same time, data from the X-neighbours is received, which can be added to the newly computed forces if necessary and is then sent on in Y-direction. The same holds for the Z-batch. While these forces are communicated, the elements without any shared nodes are processed (I-batch). This ensures that computation is continued during data exchange. After these steps the inner forces of all nodes are known (figure 4 shows a 2D example).

5.2 Collision Detection

The next part is collision detection, which uses the global network for data transfer. To save time on collision detection, dynamic behaviour of the model is taken into account. Elements do not move faster then 10cm/s, so if two

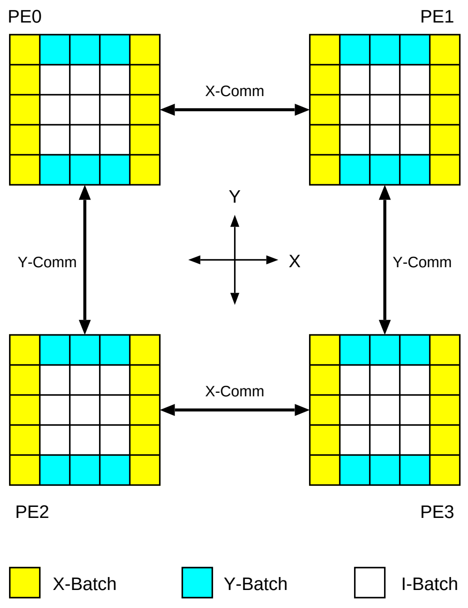

Figure 4. Processing and Communication Scheduling (2 Dimensional)

elements are 2cm apart, the earliest time they can touch is 100ms or 1000 time-steps away. A collision between these elements cannot take place within the next 1000 time-steps and therefore needs no attention. Secondly, the system is built in a hierarchical manner. Every PE first checks bounding boxes with every other PE, before checking for collisions on an element level.

Thus collision detection takes the following steps:

- Every 1000 time-steps all bounding boxes are compared.
- If close enough bounding boxes are detected, the according PEs establish a communication channel to exchange the displacements of nodes that may be involved in collisions.
- During the next 1000 time-steps, the displacements of these nodes are sent to and received from the other PEs. Collisions are detected and each PE determines the consequences for its nodes. Because these computations are symmetrical, all force vectors generated by collisions add up to zero.

Since the communication channels are redefined only all 1000 time-steps, software overhead is minimized.

6 Conclusions

A realistic simulation of human organs poses hardware requirements that cannot be met by commercial computers. Especially, the communication latency and software overhead has to be extremely low. Our system is designed to accommodate these requirements by exploiting all possibilities provided by the special structure of our problem. Two communication systems are employed providing high bandwidth for local communication and higher flexibility for global connections.

Simulators do not only help to develop software but they are also used to determine usage of communication channels and the impact of message overhead and latency on total computation time. Thus they are useful tools to find the interprocessor network that best meets the requirements of real-time Finite Element calculation in our application.

References

[1] K. Bathe. *Finite Element Procedures in Engineering Analysis*. Prentice Hall, 1996.

[2] A. Gunzinger, U. Müller, W. Scott, B. Bäumle, P. Kohler, H. Vonder Mühll, F. Müller-Plathe, W. van Gunsteren, and W. Guggenbühl. Achieving supercomputer performance with a DSP array processor. In *Supercomputing '92*, pages 543–550, Minneapolis, 1992. IEEE/ACM, IEEE Computer Society Press.

[3] U. Kühnapfel et al. Endosurgery with KISMET. In *Virtual Reality World*, pages 165–171, 1995.

[4] C. Pommerell. *Solution of large unsymmetric systems of linear equations*, volume 17 of *Series in microelectronics*. Hartung-Gorre Verlag, Konstanz, 1st edition, 1992.

[5] M. Sagar, D. Bullivant, G. Mallinson, and P. Hunter. A Virtual Environment and Model of the Eye for Surgical Simulation. In *Computer Graphics Proceedings*, Annual Conference Series, pages 205–212, 1994.

[6] V. Taylor. *Application-Specific Architectures for Large Finite-Element Applications*. PhD thesis, Dept. El. Eng. and Comp. Sci., Univ. ov California, Berkeley, California, 1991.

[7] H. Vonder Mühll. *Concept and Implementation of a Scalable Architecture for Data-Parallel Computing*, volume 59 of *Series in microelectronics*. Hartung-Gorre Verlag, Konstanz, 1st edition, 1996.

[8] Z.-H. Zhong. *Finite Element Procedures for Contact-Impact Problems*. Oxford Science Publications, 1993.