High performance computing using AUTODYN-3D

M. S. Cowler¹, O. La’adan², T. Ohta³
¹ Century Dynamics Incorporated, USA.
² Hebrew University of Jerusalem, Israel.
³ CRC Research Institute Incorporated, Japan.

Abstract

Computer simulations of dynamic events such as impact and penetration, blast wave loading of structures and underwater explosions have become increasingly important tools for understanding such phenomena. AUTODYN-3D is a general-purpose computer code that has been developed specifically for analyzing dynamic events of this kind. The program offers users a variety of numerical techniques with which to solve their problems. These include Lagrange, Shell, Euler, ALE (Arbitrary Lagrange Euler) and SPH (Smooth Particle Hydrodynamics) processors. As reliance on simulation becomes accepted, the complexity of the problems to be solved increases in size and resolution. However, the practical computation of these very large simulations has been restrained by the lack of performance of available computers. Problems requiring millions of elements and run-times that can run many weeks are not uncommon. Even the fastest single CPUs cannot easily cope with these larger problems. One approach to overcoming these limitations is to utilize parallel systems. To that end, parallel algorithms have been implemented for an arbitrary number of processors in AUTODYN-3D.

This paper describes the parallel algorithms that have been implemented in AUTODYN-3D, and presents benchmark calculations whose timing and scalability results shown clearly that large three-dimensional dynamic simulations can now be performed with an increase in throughput of orders of magnitude relative to classic single CPU configurations.
1 Introduction

AUTODYN 2D/3D are computer programs used to compute non-linear dynamic events [1], [2]. They have the ability to model different regimes of a physical event using different numerical techniques, each of which have some specific advantages. Thus, instead of applying the same general processor (solver) to all domains of a problem, a processor optimized for a particular regime of material behavior may be used. In a single analysis, one type of processor may be used for fluid behavior while another type may be used for solid/structural response. The various domains in the problem are then coupled together in space and time to provide an efficient and accurate solution.

Current processors available in AUTODYN-3D include Lagrange, Shell, Arbitrary Lagrange Euler (ALE), Smooth Particle Hydrodynamics (SPH) and Euler. In three dimensions, calculations performed with these processors can be extremely computationally intensive if large numbers of elements are employed and/or extensive surface areas exist, that require testing for contact. To allow such calculations to be performed in more practical time frames, work is in progress to parallelize each of these processors, so that calculations can be computed in a fraction of scalar processing times on systems such as Massively Parallel Processors (MPP) and Scalable Computing Clusters (SCC).

This paper describes the work completed at present, which includes parallelization of the Lagrange processor, the Shell processor, the ALE processor, and contact interactions between these processors. Timing and scalability results are presented from benchmark calculations performed on the MOSIX Scalable Computing Cluster at the Hebrew University of Jerusalem.

2 The Lagrange Processor

The approach that has been adopted for parallelization of the grid computation performed by the Lagrange processor is domain decomposition. Each Lagrange grid is defined in a three-dimensional index space (I,J,K). For parallel calculations, each grid is divided along index planes in the I, J and K directions to form smaller grids called sub-domains. These sub-domains are distributed amongst the CPUs of the parallel machine using an algorithm that attempts to minimize inter-CPU communications and balance the computational load on each machine. Each sub-domain is processed in parallel as if it were a standard Lagrange grid in serial processing. This importantly allows most of the source code for serial processing to be used without modification.

In AUTODYN-3D, the solution to a problem is advanced in time by repeatedly integrating the equations of motion over a number of small time steps using explicit time integration (a single time step is called a computational “cycle”). In an explicit time integration scheme, each element of a grid interacts only with its adjacent elements. Consequently, only a small number of point-to-point or element-to-element inter-CPU communications are required along sub-domain boundaries. Specifically, for the Lagrange processor, only the three components of nodal forces need to be exchanged at these boundaries each cycle.
This exchange takes only a small fraction of the time required to perform a Lagrange computational cycle, so the efficiency of such calculations is very good.

Because the grid structure does not normally change during the simulations (except when Lagrange elements are eroded and removed, or when modified by the user), a static decomposition of the entire index space is sufficient to achieve good performance.

3 The Shell Processor

The same domain decomposition approach has been adopted for the shell processor. Each Shell grid is defined in a two-dimensional index space (J,K). For parallel calculations, each grid is divided along index planes in the J and K directions to form smaller sub-domains. These sub-domains are also distributed amongst the CPUs of the parallel machine using an algorithm that attempts to minimize inter-CPU communications and balance the computational load on each machine. Since the time required to compute a shell element for one cycle is significantly less than the time to compute a Lagrange element, a weighting factor is used in the load balancing calculation to account for this. As with the Lagrange processor, only the three components of nodal forces need to be exchanged at sub-domain boundaries, so the efficiency of Shell calculations is very good.

4 The Arbitrary Lagrange Euler (ALE) Processor

AUTODYN-3D’s Arbitrary Lagrange Euler (ALE) processor, allows the user to specify exactly how each node of an ALE grid will move. Commonly used options include having some nodes move with the material (Lagrange), others remain fixed in space (Euler) and still more move in such a way that interior nodes of a deforming grid remain well (e.g. equipotentially) spaced. To achieve this capability, the computational cycle for the ALE processor performs the same calculations as the Lagrange processor, but adds a further advective phase.

The standard domain decomposition approach is once again used, but the calculation of the advective phase requires significantly more data to be exchanged at sub-domain boundaries.

Firstly, equipotential zoning (one of the most frequently exercised options), used to maintain good spacing of internal nodes of a grid, requires an iteration through the grid where each node is recursively placed at the equipotential center of its surrounding nodes. This procedure requires the three coordinates of nodes on sub-domain boundaries to be exchanged on each iteration. Since this data exchange might become significant if a large number of iteration steps were required for convergence, we allow a limit to be set on the maximum number of iterations that can be performed on one cycle. We have obtained good results in a number of problems with this limit set to one.

Secondly, the possible advection of material across sub-domain boundaries requires the exchange of element variable data (such as mass, momentum,
energy, etc.) along sub-domain boundaries twice per cycle. For each element, the first exchange requires the transfer of forty-three variables while the second exchange requires the transfer of seven variables. This is a significantly larger data exchange compared to Lagrange exchanges and could potentially create efficiency problems for slow networks. It is important to note, that even more data exchanges are required for the parallelization of the Euler processor (currently being implemented).

5 Contact Interactions

Contact logic is used when one surface element of attempts to penetrate another surface element. This requires a global search of Cartesian space to find possible contacts. Using the same sub-domain decomposition used for the grid calculation (Lagrange, Shell or ALE) is not an efficient way to parallelize contact interactions. This is because the contact surfaces that need to be processed come from elements that lie on the surface of the grid and thus comprise only a subset of the total elements in the grid. It is easy to see that some sub-domains might contain many surface elements, while others none at all. Moreover, if a calculation allows the erosion or removal of elements as penetration occurs, the actual surface elements of a grid will change as the calculation proceeds. We are therefore forced to use a second, dynamic domain decomposition of Cartesian space for the contact calculations.

Generally, any two surface elements anywhere in the simulation can come in contact with each other during some time step, even those that belong to the same object (self-interaction). Checking for all such contacts requires a global search in Cartesian space that in practice can take up to 50% of the overall CPU time. For efficiency, the contact nodes and faces are spatially sorted to speed this computation and to avoid unnecessary tests of distant elements. Thus, the contact algorithm used in AUTODYN-3D can be considered in two parts. Firstly, a calculation is performed to identify neighboring nodes/faces that require to be checked for interaction. Secondly, a detailed interaction calculation is performed for all these identified nodes/faces.

Determining which nodes/faces require to be checked for interactions is achieved with a bucket-sort. A grid of virtual work units is defined in Cartesian space. Each work unit is a cube, with sides twice the smallest face dimension of all interacting faces. In tests, this cube size was found to not only yield the most efficient computing times (due to the fine sort), but also to generate sufficient work units to allow efficient load balancing for the parallelization. These work units are virtual because storage for a particular work unit is only allocated when it is determined that the work unit contains nodes/faces that are to be tested for interaction.

The bucket-sort loops over all the surface nodes and faces of a problem, and constructs a list of the actual work units required for a particular analysis. The sort is performed in two passes, in which all the nodes are sorted, and then the faces are sorted.
First, each node is added to the work unit, which contains it. A hash table is used to achieve fast access time to the virtual work units, effectively yielding a complexity of $O(n)$, which significantly speeds up the entire sort. If the work unit does not exist, it is created at that stage.

Next, looping over all surface faces of the problem, each face is added to all work units, which contain nodes that might interact with the face. This is determined by checking each node of the face to see if it is contained within, or is in close proximity to, a work unit’s domain. At this stage, only work units that already contain surface nodes are considered. The proximity test is based on the size of the contact detection zone used for the interaction logic and the amount of “slack” allowed to enable the calculations described here to be performed less frequently than every cycle.

Finally, the node and face tables built for each work unit in the list are examined to determine the total number of node/face interactions that will be required to be computed for the work unit (this number is used to facilitate load-balancing in the parallel version). In general, this will equal the total number of nodes in the work unit times the total number of faces. However, this can be reduced if, for example, self-interaction of a subgrid with itself is not permitted, or two subgrids have been specified not to interact with each other. If the total number of interactions required to be computed for a work unit is found to be zero, then the work unit is removed from the list.

At the end of this procedure a compact group of work units has been generated, each containing a list of surface nodes and faces that require testing for interaction. Each node has been uniquely assigned to a particular work unit. Faces have been assigned to multiple work units, as required. These lists may be valid for a number of computational cycles, depending on the proximity test used to determine potential node-face interactions and on changes in surface definitions (if an element is eroded or removed, surfaces need to be redefined).

The detailed interaction calculation that is performed between the nodes and faces in each work unit list is very robust in that every impact is detected and dealt with correctly regardless of the deformation and relative movement of bodies or changes in surface definitions. The method is based on the work published by Thoma and Vinckier [3] and is described in more detail in an earlier paper [4].

Parallelization of the contact algorithm described above is fairly straightforward. Once the work units containing the node and face lists to be tested for impact have been generated, a load-balancing algorithm efficiently distributes them amongst the available CPUs, assuming each CPU has either the same speed or a pre-determined relative speed provided by the user. The data decomposition used for the contact interaction calculation is different from the one used for the grid calculation, so the load-balancing algorithm attempts to minimize the inter-CPU communications required between these two decompositions. Although a static decomposition is used for the Lagrange calculation, a dynamic decomposition has to be used for the contact calculation. Consequently, load balancing of the newly formed work units is performed for each cycle on which a sort is carried out. This allows contact calculations to
remain well load-balanced even when surfaces are reconfigured as the simulation progresses, or during the erosion (removal) of elements.

As our results show, the contact algorithm generates sufficient work units during the sort phase to allow efficient load balancing for parallel processing. Furthermore, the scheme uses simpler communication patterns than those that rely on recursive coordinate bisection (RCB) to assign equal amounts of nodes to all CPUs, and adapts well to heterogeneous systems where CPUs may have different speeds and workloads that may vary with time.

6 Benchmark Calculations

The results of three benchmark calculations have been used to demonstrate the efficiency and scalability of the parallel processing algorithms described in this paper. The benchmarks were performed on the MOSIX Scalable Computing Cluster at the Hebrew University of Jerusalem. This cluster included 25 high-end Pentium processors, with 128-512MB of main memory each. They were connected by a switch based fast-ethernet LAN, except for a few slow ethernet links. When the slow links participated in the calculations, the performance was reduced.

6.1 Lagrange Impact and Penetration Benchmark.

The first benchmark calculation is a steel projectile (cylinder) impacting an aluminum target plate at an angle of 60 degrees to normal. Figure 1 shows the initial configuration and the final configuration after the projectile penetrated the target. The projectile and target were modeled using the Lagrange processor with interaction calculations and erosion. 175,000 elements were used.

Figure 1: Initial and final configurations for the Lagrange benchmark calculation.

The calculation was executed for 1500 cycles on the CPU systems shown in table 1. From the table it can be seen that the performance remains very good up to 8 CPUs, achieving an order of magnitude improvement in the cycle time. When the number CPUs further increases, there is a degradation in performance. This can be partially attributed to the slow network links that were used, considering the communications patterns in this benchmark. Also small
variations in the computational times for each element may account for some of the loss in efficiency. A previous benchmark calculation reported in an earlier paper\[4\], using a simpler grid and communications structure, gave close to 100% efficiencies on up to 16 CPUs. The current benchmark has more practical value as it is typical of the type of 3D impact analyses AUTODYN is used for.

Table 1: Results of the Lagrange benchmark calculation

<table>
<thead>
<tr>
<th>No. of CPUs</th>
<th>Lagrange Benchmark</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Sec/Cycle</td>
</tr>
<tr>
<td>1</td>
<td>37.2</td>
</tr>
<tr>
<td>2</td>
<td>19.8</td>
</tr>
<tr>
<td>4</td>
<td>10.6</td>
</tr>
<tr>
<td>8</td>
<td>5.8</td>
</tr>
<tr>
<td>16</td>
<td>3.7</td>
</tr>
<tr>
<td>24</td>
<td>3.1</td>
</tr>
</tbody>
</table>

These tests are being repeated on a faster Myrinet network and the results will be presented at the conference.

6.2 Shell Crushing Benchmark.

The second benchmark calculation is the axial crushing of a cylinder by converging pistons moving at 20 m/s. Figure 2 shows the initial grid and two later configurations during the crushing process. The cylinder was modeled using the shell processor with self-interaction calculations. 250,000 elements were used.

Figure 2: Three configurations during the shell benchmark calculation

The calculation was run on the CPU systems shown in table 2. The results of this benchmark show excellent performance and scalability. In this case, we can expect linear speedup even beyond 32 processors. Note, in particular, the super-linear speed up for the cases of 2 and 4 CPUs. This can be attributed to a combination of cache effect, and the fact that all tests were performed on the
primary CPU. In scalar mode, the load fluctuations on that CPU, which can take up to 5% of the CPU power, affected the calculation time. In the parallel mode it only bothered the master, which mainly monitors the calculation.

Table 2: Results of the shell benchmark calculation

<table>
<thead>
<tr>
<th>No. of CPUs</th>
<th>Shell Benchmark</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Sec/Cycle</td>
<td>Speed-up</td>
<td>Efficiency</td>
</tr>
<tr>
<td>1</td>
<td>58.4</td>
<td>1</td>
<td>100%</td>
</tr>
<tr>
<td>2</td>
<td>28.2</td>
<td>2.07</td>
<td>103%</td>
</tr>
<tr>
<td>4</td>
<td>14.3</td>
<td>4.08</td>
<td>102%</td>
</tr>
<tr>
<td>8</td>
<td>7.4</td>
<td>7.89</td>
<td>98%</td>
</tr>
<tr>
<td>16</td>
<td>3.7</td>
<td>15.78</td>
<td>98%</td>
</tr>
<tr>
<td>24</td>
<td>2.5</td>
<td>23.36</td>
<td>97%</td>
</tr>
</tbody>
</table>

6.3 ALE Underwater Explosion Benchmark.

The third benchmark is the loading of a submerged steel plate by an underwater explosion. Figure 3 shows the initial and final grids for this calculation. The configuration was modeled using the ALE processor. 125,000 elements were used.

![Figure 3: Initial and final grids for the ALE benchmark calculation](image)

The calculation was run for 1000 cycles on the limited CPU systems shown in table 3. The results of this partial benchmark show that the communication overhead of the parallel ALE solver is larger than the other solvers. This is in accordance to our expectation, since more variables are required to be transferred. As with the Lagrange calculation, small variations in computation times for each element might also be a contributing factor in reducing efficiency. This benchmark is currently being run on a larger number of CPUs. The results of these calculations will be presented at the conference.
Table 3: Results of the ALE benchmark calculation

<table>
<thead>
<tr>
<th>No. of CPUs</th>
<th>ALE Benchmark</th>
<th>Speed-up</th>
<th>Efficiency</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Sec/Cycle</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>21.2</td>
<td>1</td>
<td>100%</td>
</tr>
<tr>
<td>2</td>
<td>12.5</td>
<td>1.7</td>
<td>85%</td>
</tr>
<tr>
<td>4</td>
<td>6.9</td>
<td>3.07</td>
<td>77%</td>
</tr>
</tbody>
</table>

7. Conclusions

Efficient and scalable parallel algorithms for computing the deformation and interaction of structures using Lagrange, Shell and ALE grids have been developed. The algorithms use a static decomposition for processing the grid calculation and a dynamic decomposition for computing contact surfaces.

The algorithms were tested on a Pentium based computing cluster with a moderate-speed LAN. The results show good to excellent speedup on up to 24 CPUs. This accounts for over an order of magnitude gain in the calculation speed. Reasonable speedup has been obtained in the case of ALE calculations, in which significantly more data is required to be exchanged between CPUs.

We intend to continue executing extensive benchmarks on similar systems with faster networks and on multi-CPU shared memory processors (SMPs). Some of the results of these benchmarks will be presented orally. We are also continuing to optimize the communication layer of AUTODYN-3D, which is based on PVM (Parallel Virtual Machine), and improving the load-balancing algorithms to account for variations in the computation times of elements (which are much more prevalent using the Euler processor).

References