Parallel Calculations in LS-DYNA

Oleg Vartanovich Mkrtichev¹, Andrey Aleksandrovich Reshetov¹∗

¹Moscow State University of Civil Engineering (National Research University), Moscow, Russia

Abstract. Nowadays, structural mechanics exhibits a trend towards numeric solutions being found for increasingly extensive and detailed tasks, which requires that capacities of computing systems be enhanced. Such enhancement can be achieved by different means. E.g., in case a computing system is represented by a workstation, its components can be replaced and/or extended (CPU, memory etc.). In essence, such modification eventually entails replacement of the entire workstation, i.e. replacement of certain components necessitates exchange of others (faster CPUs and memory devices require buses with higher throughput etc.). Special consideration must be given to the capabilities of modern video cards. They constitute powerful computing systems capable of running data processing in parallel. Interestingly, the tools originally designed to render high-performance graphics can be applied for solving problems not immediately related to graphics (CUDA, OpenCL, Shaders etc.). However, not all software suites utilize video cards’ capacities. Another way to increase capacity of a computing system is to implement a cluster architecture: to add cluster nodes (workstations) and to increase the network communication speed between the nodes. The advantage of this approach is extensive growth due to which a quite powerful system can be obtained by combining not particularly powerful nodes. Moreover, separate nodes may possess different capacities. This paper considers the use of a clustered computing system for solving problems of structural mechanics with LS-DYNA software. To establish a range of dependencies a mere 2-node cluster has proven sufficient.

1. Introduction

Parallel computing is a way of organizing computerized calculations when a software program is executed as a set of interacting parts running in parallel (simultaneously). As of today, several modes of running parallel computing are available. E.g., a software program may represent a single multi-stream process executed within an operating system (OS). Such structure of computing provides that the simultaneously executed streams are interacting via shared memory (figure 1a). At the same time, actual parallelism is supposed to be achieved with multi-core CPUs. This way of structuring parallel computing is targeted at computing systems belonging to SMP architecture (Symmetric Multiprocessing). Another way of organizing parallel computing implies that a program is executed as a total of interacting processes. A single machine (as OS processes) or multiple networked machines may execute these processes. If the processes are executed by different machines (i.e. by a shared memory system), such configuration is called distributed computing. Processes interact via the so-called MPI (Message Passing Interface). The processes running on different machines exchange messages via network communication (figure 1b). This architecture of a computing system is referred to as MPP (Massive Parallel Processing). Good scalability of the system by extension makes for an advantage of MPP over SMP.
It should be noted that the executable program, or the way its code is structured, determines the correlation between the number of processors in the system and the capacity. The capacity can be assessed using Amdahl’s law (figure 2):

\[ S_p = \frac{1}{\alpha + \frac{1 - \alpha}{p}} \tag{1} \]

where, \( \alpha \) – the portion of the program code executed consecutively, \( p \) – number of computing units (cores, processors, nodes etc.)

2. MPI applications in distributed computing

As mentioned previously, MPP architecture means that a software program is executed as a total of processes interacting via MPI interface. This in turn presumes that the software program in question was developed using MPI API (Application Programming Interface). MPI API provides software developers with a set of commands and features facilitating process synchronization, data transfer between processes via messages etc.

An MPI-program run on cluster suggests that the processes be distributed between the cores of the processors belonging to the cluster nodes. When synchronizing the data flows from different nodes the MPI interface transfers messages via network communication. Hence, for the system to operate effectively, a network with sufficient throughput is required. Besides, a key prerequisite for the efficiency of software is good balancing of load among the node processors, which is most evident in case of a cluster whose nodes have varying capacities.

Let us consider a generic configuration of MPI program’s launch and operation.

Firstly, an MPI interface must be installed on all nodes, and all the nodes must have an MPI interface service started (in the case of MS-MPI it is MS-MPI Launch Service). Secondly, all the nodes must feature identical credentials with identical passwords. The software can be started (for instance, with a name program1) from the master node with the command below:
where, hosts_list stands for the file with a list of nodes; the path to the working folder contains a network address of the storage accommodating the working folder. The file hosts_list contains data about the number of processes to be started on each of the nodes.

Program execution involving MPI is effected as follows:

1) The master node decomposes the task into subtasks and transfers them to the computing nodes based on the number of the processes, node capacity and other considerations (figure 3). The master node can at the same time act as a computing node and take over a subtask.

2) As the program is running, synchronization and file creation processes take place (e.g. creation of files with solutions) etc.

3) At the completion of the program all the created processes and the mpiexec program are terminated, the established connections are closed.

Figure 3. Task decomposition (LS-DYNA).

A more detailed account of parallel computing can be seen in [1-3].

3. Distributed computing in LS-DYNA

A cluster comprised of two nodes with varying capacities, HALL9000-5 and ANDREWR-X200LA, are going to be used for problem solving. Each node is equipped with a quad core processor (incl. hyperthreading) and a corresponding RAM: HALL9000-5 (Intel Core i5 2.9 GHz, 8 GB), ANDREWR-X200LA (Intel Core i3 1.7 GHz, 4 GB). The nodes are connected via a gigabit network. MPP single precision solver LS-DYNA R8.0.0 will be used. We will create one process per node for convenience of analyzing the dependencies. Figure 3 presents the problem to be solved. The structure consists of columns, rods and slabs. Columns and rods are modeled as beam finite elements, while slabs take the form of shells. The material is linear. External strains are set as inertia forces (own weight and earthquake accelerogram [4-6]). The explicit method will be used for solving. The solution will involve several patterns of task decomposition in order to determine the dependency between the utilization of processors, network traffic and the approach to decomposition.

The first approach to decomposition is as follows: we split the structure into two equal parts and assign them to the nodes for solution. The section will be effected along the rods and slabs (figure 4).
Figure 4. Task decomposition.

The outcome is presented in figure 5.

![Figure 5. a) Computing report, b) Network diagram.](image)

The outcome lets us conclude that the CPU utilization was balanced unevenly (figure 5a, CPU/Avg_CPU). The first node with higher capacity proved underutilized (0.70793) and the second overloaded (1.29207). The reason is that the task divided into two equal parts caused the first node to run calculations faster than the second one. At the same time, every iteration done by the first node for its portion of the task requires the other node to provide the boundary conditions obtained in the previous iteration. As the latter node solves its portion of the task slower, the former node idles, which slows down the performance of the entire clustered system.

Figure 6. Task decomposition.

Under the second mode of decomposition, we divide the structure into two uneven parts and assign the larger part to the more powerful node, while the less powerful one receives the smaller part (figure 6). The decomposition section was determined based on the weighting coefficients derived from the performance analysis of the cluster nodes.
The result is presented in figure 7.

![Figure 7. a) Computing report, b) Network diagram.](image)

The results give rise to the conclusion that the utilization was balanced in this case: HALL9000-5 (1.01935), ANDREWR-X200LA (0.98065).

Although the utilization of the CPUs proved balanced, this way of decomposition cannot be declared optimal. The fact is that the structure was sectioned along the beams and slabs. Once we draw the section along the columns, the number of boundary points between two subtasks will reduce and therefore the network diagram will shrink. Naturally, the network communication capability available for this task contains a large surplus, however, we will obtain a more robust task decomposition for research purposes. This task appeared to be incapable of being decomposed by the columns with CPU utilization remaining balanced (figure 8), since the resulting volumes of subtasks do not match the capacity of the nodes.

The optimal decomposition is achieved by combining the sections by columns, by beams and by slabs (figure 9). The outcome (figure 10) indicates that the CPU utilization is balanced and network traffic is less intense (figure 5b, 7b, 10b). This way of decomposition required the least computing time.

![Figure 8. a, c) Task decomposition; b, d) Computing report.](image)
4. Conclusions
Therefore, efficient use of LS-DYNA software suite for distributed computing using clusters comprised of nodes with varying performance characteristics necessitates optimized methods of task decomposition. For decomposition to be optimized, node CPUs must be balanced more evenly and the number of messages exchanged between nodes must be reduced.

5. References
[1] Krysl P and Bittnar Z 2001 Comput. Struct. 79(3) 345-60.
[2] París J, Colominas I, Navarrina F and Casteleiro M, 2013 Comput. Struct. 125 62-73.
[3] Jin H Q, Jespersen D, Mehrtra P, Biswas R, Huang L and Chapman B 2011 Parall. Comput. 37(9) 562-75.
[4] Mkrtchyan O V and Myasnikova E S 2009 Struct. Mech. Calculat. Struct. 1 38-40.
[5] Mkrtchyan O V, 2000 Reliability of multi-element rod systems of engineering structures, The dissertation author's abstract on competition of a scientific degree of the doctor of technical sciences, Moscow.
[6] Mkrtchyan O V, Dzhinchvelashvili G Q and Bunov A A 2014 Proc. Eng. 91 48-53.

Acknowledgements
This study was performed with the support of the RF Ministry of Education and Science, grant №7.1524.2017/Project Part.