Parallelisierung elementarer Algorithmen expliziter und impliziter Codes

Uli Göhner

DYNAmore GmbH, uli.goehner@dynamore.de, Stuttgart, Germany

Abstract:

The usage of parallel hardware platforms has become a standard in the high performance computing community. The problem of scalability for high numbers of prcessors used in parallel are faced within each of the different software vendors. In this paper we describe some basic algorithms, which are used in the different FEM software programs. Different parallelization concepts are shortly described and the resulting scalability problems are discussed.

Keywords:
Parallelization, algorithmical aspects, implicit and explicit time stepping

1 Introduction

In the recent years, the usage of massively parallel computing has increased considerably in both research and industrial applications of high performance computing applications. The distributed memory hardware architecture has become popular because of the flexibility of the underlying processor technology and the possibility to use standard components both for the processor and the communication technology [3]. In the very recent past even heterogenous hardware architectures have been used for high performance computing applications. Grid technologies open more and more possibilities to integrate unused computing resources into the computing centre.

2 Different strategies for parallelization

There are different levels of parallelization:
- parallelization on bit-level
- parallelization on instruction level
- parallelization on operating system level
- parallelization on programming level

In the case of parallelization on bit-level, e.g. several phases of an processor operation can be done in parallel using the pipelining principle. On instruction level e.g. a complicated expression with different independent operations can be parallelized. Parallelization on operating system level can be done by all operating systems, which support parallel hardware architectures. If it is not included in the standard system, additional components for resource (i.e. processor+memory) allocation and resource management can be done by separate software components. For all of the above parallelization techniques the software development and the applied algorithms are fairly straight forward and well known since a couple of years. The most challenging part of parallelization comes into the game for the software developer, if one wants to have a parallelization on programming level with the following requirements:
- scaling possbilities
- usage of standard components, because of price performance reasons

In this case, the software developer needs to identify and coordinate parallel data fluxes. Appropriate algorithms must be chosen and implemented. In most cases a total rewrite and definiton of a new restructured software and data architecture of the complete software package needs to be done.

2.1 Hardware Architecture

The algorithmical choice depends on the type of the underlying hardware architecture. In the classification due to Flynn the following categories exist:
- SISD – Single Instruction Single Data
- SIMD – Single Instruction Multiple Data
- MISD – Multiple Instruction Single Data
- MIMD – Multiple Instruction Multiple Data

All modern parallel computers belong to the latter category Multiple Instruction Multiple Data, which means, that a number of different subprograms are started on different processors with different input data. Within the MIMD category we have two different types of parallel arhitecture, the SMP (Shared Memory) and the MPP (Message Passing Protocol) architecture. In the SMP-variant all processors have one common memory adress block. In the MPP-variant each processor has its own memory. If one of the processors needs information from another, the required data must be transferred from one local memory to the other local memory via appropriate Message Passing Protocols.

2.2 Different Software Aspects

For the software developer it is a major difference, wether the software should be ported to a SMP-parallel or a MPP-parallel system. In the case of SMP-parallel porting it is sufficient to prevent those memory blocks, which are used by different processors, from parallel read-write operations by different processors. Many parallel development systems provide a couple of tools, which make the porting of a serial software to a SMP-platform relatively easy. If the software is to be ported to a MPP environment, all variables, which need to be shared among the different processors, need to be exchanged via standard messages. Different ways for the message passing can be used, e.g. point to point or collective communication strategies. Commonly used software components for the message passing are PVM (Parallel Virtual Machine) or MPI (Message Passing Interface). To get reasonable performance, a decomposition of the problem must be undertaken, which tries to keep the necessary communication load as small as possible.

3 Algorithmical aspects

3.1 Basic linear Algebra

Matrix-Vector computations can be parallelized without much communication costs by assigning appropriate matrix blocks to the different processors. The result vector is divided in an analogous way. In a similar way also vector- or matrix-products can be calculated in parallel. It is much more difficult, to calculate the inverse of a matrix, which is necessary in all implicit-type based FEM-codes.
Given a set of linear equations

$$Ax = b \tag{1}$$

where the matrix A is decomposed into lower and upper triangular matrices L and U and the diagonal terms, which are denoted by D:

$$A = D - L - U \tag{2}$$

Besides direct solver strategies, iterative solver have become more popular in the last years, because the number of floating point operations needed for the system solution can be an order of magnitude lower in the case of iterative solvers. There are a number of direct solvers available, which have been sucessfully applied on parallel hardware architectures, e.g. Cholesky factorization technique [11] or multifrontal solver techniques [2]. Going to iterative solver techniques makes parallelization easier, as in this case only matrix-vector-computations need to be done. As an example consider one of the basic iterative solvers, the Jacoby iteration technique, which is given by:

$$x_{n+1} = x_n - D^{-1}(Ax_n - b) \tag{3}$$

The big disadvantage of all elementary iterative solvers is the slow convergence rate, which must be improved by appropriate preconditioning techniques.

3.2 Application to implicit Codes

In the case of implict time-stepping, large linear equation systems need to be solved at each time step. In nonlinear or special linear cases with complex geometry, or in the presence of convection-dominated terms in the underlying differential equations, the linear system becomes ill-conditioned. In these cases a number of problem-specific preconditioning techniques have been proposed.[5] Preconditioning techniques can be simply described in the following way: Instead of solving equation (1), the equation is multiplied by a preconditioning matrix P:

$$PAx = Pb \tag{4}$$

If P=A^{-1} Equation (4) is trivial. Thus all iterative solvers converge within one iteration. A good preconditioner calculates P in a very fast and efficient way and accelerates the convergence behavior of the iterative solver considerably also for ill-conditioned matrices, which arise in all complex nonlinear applications. Different approaches based on ideas like hierarchical shape functions, multigrid approaches are common practice [10]. A non-trivial preconditioning technique takes more time than the iterative solver itself. So the construction of the preconditioner must be done in parallel, too. It turns out, that for the preconditioning it is better to distribute the different matrix blocks according to geometric aspects as solely algebraic criteria. [4]

3.3 Application to explicit Codes

In explicit-type codes most of the work done can be viewed as matrix-vector-type operations. The distribution of the matrix blocks to the different processors can be done due to algebraic (e.g. SMP-version) or geometric criteria (domain decomposition). For the element processing phase, there is almost no communication needed and an effectivity of 99.7 % can be achieved [1]. The nodal update involves element contributions from different domains for all nodes on the interface between separate domains. To keep the communication cost small, one tries to divide the mesh in such a way, that the number of nodes on the interface is as small as possible. There are a couple of strategies for the domain decomposition, based on geometrical properties, graph-theory or a multi-level-approach [6], [7], [8].

3.4 Contact

Unless there is no mesh adaption for both implicit and explicit codes, the domain decomposition can be done once in advance of the whole computation. The treatment of the contact behavior can change the situation completely. As soon as elements from different domains come into contact, communication between these domains must occur. Also for contact detection a global search routine is needed. Dynamic domain decomposition needs to be applied, to keep communication down and also to keep the load balance between the different processors [9].

4 Summary

Some basic numerical algorithms for parallel hardware architectures have been shown. Their application within explicit or implicit codes have been described. With the evolving need for standards within the high performance computing community it will be interesting to see, wether the usage of standard parallel libraries for these basic algorithms will become a common practice in future.

5 Literatur

[1] Hallquist, J.O. LS-DYNA Theoretical Manual.

[2] Ashkraft, C. Grimes, R., Maker, B: „Experiences with LS-DYNA Implicit MPP“, LS-DYNA Intern. Conf., 2004.

[3] Wainscott, B. Wang, J.: „ Message Passing and Advanced Computer Architectures”, LS-DYNA Intern. Conf., 2000.

[4] Oden, J.T., Patra, A. Feng, Y.: „Domain Decomposition for adaptive hp finite element methods“, Proc. Sev. Intl. Conf. Domain Decomposition Meth., 1993.

[5] Saad, Y.: „Iterative Methods for Sparse Linear Systems“ PWS publ., N.Y., 1996.

[6] Farhat, C.: “A Simple and efficient automatic FEM domain decomposer”, Computers and Structures, 1988.

[7] Barnard, S.T., Simon, H.D.: “Fast multilevel implementation of recursive spectral bisection for partitioning unstructured problems”, Concurrency: Practice and Experience, 1994.

[8] Karypis, G.: „Multi-Constraint Mesh Partitioning for Contact/Impact Computations”, 2003.

[9] Hendrickson, B. Devine, K.: „Dynamic Load Balancing in Computational Mechanics”, Comp. Meth. Appl. Mech. & Eng., 2000.

[10] Vuik, C. Kan, J.J.I.M., Wesseling, P.: „A Black Box Multigrid Preconditioner For Second Order Elliptic Partial Differential Equations“ Proc. ECCOMAS, 2000.

[11] J. Choi, J., Dongarra, J., Pozo, R., Walker, D.W.: “ScaLAPACK : a scalable linear algebra library for distributed memory concurrent computers”, Proc. 4th Symp.Frontiers of Massively Parallel Computers, IEEE, Comput. Soc. Press, 1992.