USING HYBRID PARALLEL PROGRAMMING TECHNIQUES FOR THE COMPUTATION, ASSEMBLY AND SOLUTION STAGES IN FINITE ELEMENT CODES

R.R. PAZ†, M.A. STORTI†, H.G. CASTRO‡ and L.D. DALCÍN†

† Centro Internacional de Métodos Computacionales en Ingeniería (CIMEC), Instituto de Desarrollo Tecnológico para la Industria Química (INTEC), CONICET, Universidad Nacional del Litoral (UNL). Santa Fe, Argentina.
‡ Grupo de Investigación en Mecánica de Fluidos, Universidad Tecnológica Nacional, Facultad Regional Resistencia, Chaco, Argentina. castrohgui@gmail.com

Abstract— The so called “hybrid parallelism paradigm”, that combines programming techniques for architectures with distributed and shared memories using MPI (Message Passing Interface) and OpenMP (Open Multi-Processing) standards, is currently adopted to exploit the growing use of multi-core computers, thus improving the efficiency of codes in such architectures (several multi-core nodes or clustered symmetric multi-processors (SMP) connected by a fast net to do exhaustive computations).

In this paper a parallel hybrid finite element code is developed and its performance is evaluated, using MPI for communication between cluster nodes and OpenMP for parallelism within the SMP nodes. An efficient thread-safe matrix library for computing element residuals (or matrices) in FEM-like codes is introduced and fully described. The cluster in which the code was tested is the CIMEC’s ‘Coyote’ cluster, which consists of eight-core computing nodes connected through Gigabit Ethernet.

Keywords— Finite Elements, MPI, OpenMP, PETSc, hybrid programming, Matrix Library.

I. INTRODUCTION

A variety of engineering applications and scientific problems related to Computational Mechanics (CM) area, and particularly in the Computational Fluid Dynamics (CFD) field, demand high computational resources (Sonzogni et al., 2002). A great effort has been made over the years in order to obtain high quality solutions (Paz et al., 2006) for large-scale problems in realistic time (Behara and Mittal., 2009) using many different computing architectures (e.g., vector processors, distributed and shared memories, graphic process units or GPGPU’s).

Symmetric multi-processors (SMP) involve a hardware architecture with two or more identical processors connected to a single shared main memory. Other recent computing systems might use Non-Uniform Memory Access (NUMA). NUMA dedicates different memory banks to different processors; processors may access local memory quickly. Despite the differences with NUMA architectures, this work will use the term SMP in a broader sense to refer to a general many-processor many-core single-memory computing machine.

Since SMP have spread out widely in conjunction with high-speed network hardware, using SMP clusters have become attractive for high-performance computing. To exploit such computing systems the tendency is to use the so called hybrid parallelism paradigm that combines programming techniques for architectures with distributed and shared memory, often using MPI and OpenMP standards (Jost and Jin., 2003). The hybrid MPI/OpenMP programming technique is based on using message passing for coarse-grained parallelism and multi-threading for fine-grained parallelism.

The MPI programming paradigm defines a high-level abstraction for fast and portable inter-process communication and assumes a local/private address space for each process. Applications can run in clusters of (possibly heterogeneous) workstations or dedicated nodes, SMP machines, or even a mixture of both. MPI hides all the low-level details, like networking or shared memory management; simplifying development and maintaining portability, without sacrificing performance (see Section V.B).

Although message passing is the way to communicate between nodes, it could not be an efficient resource within an SMP node. In shared memory architectures, parallelization strategies that use OpenMP standard could provide better performances and efficiency in parallel applications. A combination of both paradigms within an application that runs on hybrid clusters may provide a more efficient parallelization strategy than those applications that exploit the features of pure MPI.

This paper is focused on a mixed MPI and OpenMP implementation of a finite element code for scalar PDE’s and discusses the benefits of developing mixed mode MPI/OpenMP codes running on Beowulf clusters of SMP’s. To address these objectives, the remainder of this paper is organized as follows. Section II provides a short description and comparison of different characteristics of OpenMP and MPI paradigms. Section III introduces and describes an efficient thread-safe matrix library called Fast-Mat for computing element residuals and Jacobians in the context of multi-threaded finite element codes. Section IV discusses the implementation of mixed (hybrid) mode application and describes a number of situations where mixed mode programming is potentially beneficial. Section V presents the implementation of an hybrid application such as advective-