8th. World Congress on Computational Mechanics (WCCM8)
5th European Congress on Computational Methods in Applied Sciences and Engineering (ECCOMAS 2008)
June 30 –July 5, 2008
Venice, Italy

TOWARD HIGHER PERFORMANCE FEM IMPLEMENTATIONS USING LAZY EVALUATION AND SYMBOLIC PROGRAMMING.

Hugo Leclerc[1]

[1] LMT-Cachan
(ENS Cachan/CNRS UMR8535/Universite Paris 6/PRES UniverSud Paris)
61, av. du President Wilson, F-94230 Cachan, France
leclerc@lmt.ens-cachan.fr

Key Words: *Finite Element Method, Symbolic programming, Code generation, Active Libraries, Parallel programming.*

ABSTRACT

Efficient use of processor abilities, memory hierarchies and distributed ressources to solve even the most basic equations has always been a difficult task. It has led to the idea that high-performance computing means simplifications, specialization and hand-coding. But this idea is challenged by recent work in the domain of "**Active Libraries**" [1] where implicit or explicit code generation allows the developers to concentrate on mathematical properties and clever optimizations, not on the repetitive but time consuming tasks.

This paper is about consequences on using both lazy evaluation and symbolic programming as a **computing model, to improve the performance of Finite Element Method implementations**.

Lazy evaluation basically consists in postponing as long as possible the necessary execution of the developer's instructions in order to collect in global graph form the dependencies and memory requirements of all the sub-steps – including the pre and post-processing ones. At first sight, It gives a convenient way to benefit from researches on static or pseudo-dynamic **scheduling and memory management** [2], allowing efficient out of core algorithms, management of CPU and memory heterogeneities...

But this graph can also be considered as a **symbolic representation** of a program. It allows automatic integration, differentiation or substitution, giving ways to automate usual transformations for methods like Newton-Raphson, Galerkin, Newmark, ... in a similar way as what is done in [3].

In the first part, this paper shows examples of some local optimizations that can be achieved using both lazy evaluation and symbolic programming, especially for memory and time consuming sub-steps. When possible, timings are compared to those of hand-coded versions, and state of the art libraries. As example, it is shown how **cache management** and automatic parallelization can benefit from symbolic information on fields and values.

Overall performance is finally discussed starting from some propositions to improve the basic methods, based each time on the automatic symbolic transformations which have been made accessible with this paradigm.

REFERENCES

[1] Todd L. Veldhuizen and Dennis Gannon. “Active Libraries: Rethinking the roles of compilers and libraries”. *Proceedings of the SIAM Workshop on Object Oriented Methods for Inter-operable Scientific and Engineering Computing (OO’98)*, Vol. **10**, `http://ubiety.uwaterloo.ca/~tveldhui/papers/oo98.ps`, 1998.

[2] Oliver Sinnen. “Task Scheduling for Parallel Systems”. *Wiley-Interscience*, 2007.

[3] Anders Logg. “Automating the Finite Element Method”. *Arch. Comput. Methods Eng*, Vol. **14**, Nb. 2, P. 93–138, 2007.