ARTICLE IN PRESS

Computers and Mathematics with Applications ▮ (▮▮▮▮) ▮▮▮–▮▮▮

Contents lists available at ScienceDirect

Computers and Mathematics with Applications

journal homepage: www.elsevier.com/locate/camwa

Boundary element quadrature schemes for multi- and many-core architectures

Jan Zapletal [a,b,*], Michal Merta [a], Lukáš Malý [a,b]

[a] IT4Innovations, VŠB – Technical University of Ostrava, 17. listopadu 2172/15, 708 33 Ostrava-Poruba, Czech Republic

[b] Department of Applied Mathematics, VŠB – Technical University of Ostrava, 17. listopadu 2172/15, 708 33 Ostrava-Poruba, Czech Republic

ARTICLE INFO

Article history:
Available online xxxx

Keywords:
Boundary element method
Quadrature
SIMD
Vectorization
Intel Xeon Phi
Many-core architecture

ABSTRACT

In the paper we study the performance of the regularized boundary element quadrature routines implemented in the BEM4I library developed by the authors. Apart from the results obtained on the classical multi-core architecture represented by the Intel Xeon processors we concentrate on the portability of the code to the many-core family Intel Xeon Phi. Contrary to the GP-GPU programming accelerating many scientific codes, the standard x86 architecture of the Xeon Phi processors allows to reuse the already existing multi-core implementation. Although in many cases a simple recompilation would lead to an inefficient utilization of the Xeon Phi, the effort invested in the optimization usually leads to a better performance on the multi-core Xeon processors as well. This makes the Xeon Phi an interesting platform for scientists developing a software library aimed at both modern portable PCs and high performance computing environments. Here we focus at the manually vectorized assembly of the local element contributions and the parallel assembly of the global matrices on shared memory systems. Due to the quadratic complexity of the standard assembly we also present an assembly sparsified by the adaptive cross approximation based on the same acceleration techniques. The numerical results performed on the Xeon multi-core processor and two generations of the Xeon Phi many-core platform validate the proposed implementation and highlight the importance of vectorization necessary to exploit the features of modern hardware.

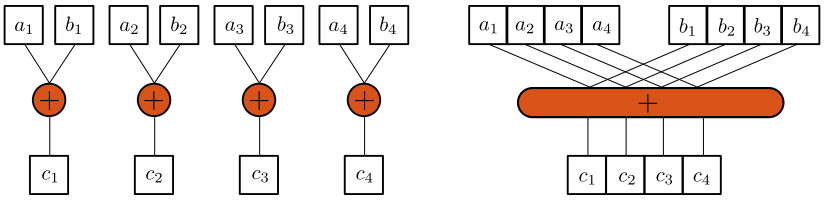
1. Introduction

An efficient implementation of the numerical quadrature routines is an inseparable part of any boundary element software. Contrary to the finite element method, where the integrands are often well-behaved polynomial functions and the integrals can sometimes be evaluated analytically, in the boundary element method (BEM) one has to deal with weakly singular integrals. Another price to pay for the dimension reduction is the quadratic complexity of the standard BEM both in terms of the computational time and memory requirements restricting its applicability to moderate problem dimensions. To overcome this issue, several fast BEM methods can be employed to lower the complexity to almost linear. This includes the fast multipole method [1–3] based on the approximation of the kernel by a truncated series, or the adaptive cross approximation (ACA) [4,5] building low-rank blocks based on an algebraic point of view. Although these sparsification

* Corresponding author at: IT4Innovations, VŠB – Technical University of Ostrava, 17. listopadu 2172/15, 708 33 Ostrava-Poruba, Czech Republic.
E-mail address: jan.zapletal@vsb.cz (J. Zapletal).

http://dx.doi.org/10.1016/j.camwa.2017.01.018
0898-1221/

Please cite this article in press as: J. Zapletal, et al., Boundary element quadrature schemes for multi- and many-core architectures, Computers and Mathematics with Applications (2017), http://dx.doi.org/10.1016/j.camwa.2017.01.018



(a) Scalar addition. (b) SIMD addition.

Fig. 1.1. Scalar and vectorized addition of two vectors $\boldsymbol{c} := \boldsymbol{a} + \boldsymbol{b}$.

methods are inevitable for large-scale engineering problems, it is still crucial to efficiently assemble the so-called non-admissible full blocks. Moreover, in the case of ACA, the low-rank approximation requires the evaluation of several rows and columns of every admissible block, which relies on the standard full assembly. The above mentioned approaches can be combined with a domain decomposition method, such as the boundary element tearing and interconnecting method [6–8]. Although these techniques allow for a massively parallel implementation in distributed memory systems, they still rely on an efficient intra-node implementation with an extra layer of inter-node parallelization.

In the paper we concentrate on the acceleration of the standard BEM assembly in shared memory and its acceleration by ACA. While the parallelization of similar scientific codes at the level of CPU cores has become standard, the feature that is still often overlooked is the in-core vectorization supporting the Single Instruction Multiple Data (SIMD) concept, see Fig. 1.1. Since the clock frequency of modern CPUs has not been growing as rapidly as in the previous decades, the increasing advertised theoretical performance limits can only be reached by utilizing all available parallelization techniques.

In 1999 Intel issued the SSE (Streaming SIMD Extensions) instruction set capable of concurrent processing of four 32-bit single-precision floating-point numbers. This feature has since been extended by the sets SSE2-4.2 adding more instructions and the ability to operate on two 64-bit double-precision operands. The AVX and AVX2 (Advanced Vector Extensions) instruction sets increase the register size from 128 bits to 256 bits. While the first commercially available Intel Many Integrated Core (MIC) architecture Knights Corner (KNC) supports the MIC specific Initial Many Core Instruction (IMCI) set, the plan for future is to (at least partially) unify the instructions across both multi- and many-core platforms—the upcoming Skylake and Knights Landing architectures are to include the new AVX-512 instruction set with 512-bit registers able to accommodate eight 64-bit double-precision operands.

The vectorization can be achieved by various techniques. The direct use of vector instructions can be utilized using the inline assembler, i.e., the assembly code directly inserted into a high-level programming language. Such code is, however, hard to understand and not portable between multiple architectures. A similar possibility is the use of intrinsic functions provided by the vendor of the compiler. These functions produce the corresponding assembly code, leading again to a non-portable code. Another option is to use an external library providing a high-level wrapper of the supported intrinsic functions. This technique leads to a portable code, since the wrapper functions are compiled to the supported vector instruction set. In [9] we describe this approach both for the semi-analytic and fully numerical integration schemes [4] using the Vc library [10]. A more user-friendly way is to use the auto-vectorization capabilities of modern compilers. To help the compiler identify the parts of the code suitable for vectorization the original code usually has to be refactored using techniques such as loop unrolling, tiling, reordering, or collapsing. This has been used by the authors in [11] with additional OpenMP 4.0 [12] pragmas and the accelerators used in the offload mode. Differently from [11], in this paper we concentrate on the native deployment of the code which allows for direct comparison of the capabilities of the available multi- and many-core platforms. Readers interested in multi- and many-core programming are referred to the monographs [13–15] and a special Knights Landing edition [16] with additional tips on programming on this architecture.

In the context of BEM, the vectorization paradigm has been leveraged in [17], where the authors rely on the automatic vectorization by the Fortran compiler. Although a reasonable speedup is reached, after the loop rearrangement the code is much less readable than the original. In [18] the author explicitly uses the intrinsic functions provided by the compiler to accelerate the generation of BEM matrices. Unfortunately, the treatment of singularities, which represents one of the crucial tasks in BEM and complicates the vectorization, is not discussed. The acceleration of the evaluation of the representation formula on the Intel Xeon Phi coprocessor is described in [19]. For a similar discussion regarding first-order finite element discretization on multi- and many-core architectures including the Xeon Phi coprocessor we also refer to [20].

There are several approaches for integrating the weakly singular kernel functions appearing in the Galerkin boundary element methods. The first approach is based on integral substitutions rendering the integrand analytical via the multiplication with the corresponding Jacobian allowing us to use the standard tensor product Gaussian quadrature schemes. The advantage of this approach lies in its versatility—the same procedure can be used for a wide class of kernels, including the kernels of the Laplace, Lamé, Stokes, Helmholtz, or Maxwell operators. This method has been studied in, e.g., [21,22].

The second, a seemingly preferable way, is to compute the inner surface integral analytically, which allows us to treat the limit singular entries. For the outer integral we can use a standard Gaussian quadrature scheme. This approach has been described for the Laplace, Lamé, and Helmholtz equations in [4,23,24]. The serious drawback of this method is that tedious

Please cite this article in press as: J. Zapletal, et al., Boundary element quadrature schemes for multi- and many-core architectures, Computers and Mathematics with Applications (2017), http://dx.doi.org/10.1016/j.camwa.2017.01.018