Journal of Computational Physics 324 (2016) 246–257

Contents lists available at ScienceDirect

Journal of Computational Physics

www.elsevier.com/locate/jcp

GPU accelerated spectral finite elements on all-hex meshes

J.-F. Remacle [a,b,*], R. Gandham [a], T. Warburton [a]

[a] Department of Computational and Applied Mathematics, Rice University, United States
[b] Université catholique de Louvain, Institute of Mechanics, Materials and Civil Engineering (iMMC), Bâtiment Euler, Avenue Georges Lemaître 4, 1348 Louvain-la-Neuve, Belgium

ARTICLE INFO

Article history:
Received 8 July 2015
Received in revised form 2 August 2016
Accepted 4 August 2016
Available online 9 August 2016

Keywords:
Spectral finite elements
GPU computing
Hexahedral meshes

ABSTRACT

This paper presents a spectral element finite element scheme that efficiently solves elliptic problems on unstructured hexahedral meshes. The discrete equations are solved using a matrix-free preconditioned conjugate gradient algorithm. An additive Schwartz two-scale preconditioner is employed that allows h-independence convergence. An extensible multi-threading programming API is used as a common kernel language that allows runtime selection of different computing devices (GPU and CPU) and different threading interfaces (CUDA, OpenCL and OpenMP). Performance tests demonstrate that problems with over 50 million degrees of freedom can be solved in a few seconds on an off-the-shelf GPU.

1. Introduction

Recent research efforts [1,23] have led to the development of 3D hex-dominant mesh generation systems that are fast and reliable. It is now possible (e.g. with Gmsh [10]) to create meshes of general 3D domains that contain over 80% of hexahedra in volume in a fully automatic manner.

We foresee that fully automatic hex-meshing procedures will be available in the next decade. This perspective allows finite element researchers to reconsider some commonly held beliefs, namely that tet-meshing may not remain the only solution for automatic mesh generation.

Quadrilateral meshes in 2D and hexahedral meshes in 3D are usually considered to be superior to triangular/tetrahedral meshes. There are numerous modeling reasons to prefer hexes: boundary layers in CFD [22], inaccuracy or locking problems in solid mechanics [2].

From a high order spectral finite element perspective, hex meshes provide considerable advantages. First, although this is not specific to spectral finite elements, a hex mesh contains about seven times fewer elements than a tet mesh with the same number of vertices. Fewer elements mean less data storage and a faster assembly procedure. Taking advantage of the inherent tensor-product structure of hexahedral basis functions one can dramatically reduce the number of floating point operations for computing finite element operators. The local cartesian structure of the mesh provides natural overlapping patches of elements that enables the construction of efficient local preconditioners. Finally, spectral hex-meshes can achieve relatively high throughput on GPUs following the approaches described below.

* Corresponding author.
E-mail addresses: jean-francois.remacle@uclouvain.be (J.-F. Remacle), rajeshgandham@gmail.com (R. Gandham), tim.warburton@gmail.com (T. Warburton).

http://dx.doi.org/10.1016/j.jcp.2016.08.005
0021-9991/

The use of GPUs for accelerating finite element solvers for elliptic problems is of course not new. In early work Göddeke et al. [13] investigated scalability of finite element solvers on GPU clusters. Later Göduke described multigrid methods for finite element methods on GPU clusters [12]. Cecka et al. [3] and Markall et al. [16] discussed algorithms for efficient stiffness matrix assembly on GPUs. Knepley et al. [15] described algorithms for efficient evaluation of finite element integrals on GPUs. Gaveled et al. [11] introduced a finite element toolkit that integrates geometric multigrid techniques with sparse approximate inverse algorithms on GPUs. Furthermore, pushing the envelope of GPU based finite element software design Fu et al. [7] describe a systematic approach to pipelining finite element methods. Largely these prior approaches have focused on optimizing the process of stiffness matrix assembly. The current work differs by first using a high-order finite element approach and secondly adopting a matrix-free approach that in its leanest form only requires storage for mesh vertex coordinates, residual vector, solution vector, load vector, and indexing arrays.

In this paper, we propose a numerical scheme that allows us to solve Poisson-like problems on unstructured all-hex meshes using the massive multi-threading capacities of modern computer hardware. An extensible multi-threading programming API is used as a common kernel language [17] to try our numerical scheme on different devices (GPU and CPU) and using different thread programming interfaces (CUDA, OpenCL, and OpenMP).

This paper is structured as follows. In §2, standard properties of spectral finite elements are presented in brief. The numerical method is presented in §3 and §4: preconditioned conjugate gradients are used for solving linear systems. A two-scale additive Schwartz preconditioner is used for accelerating the convergence. Details of implementation are presented in §5 and results are presented in §6.

2. Spectral finite elements on hexahedral meshes

Consider a domain $\Omega \in R^3$ with boundary $\Gamma = \Gamma_D \cup \Gamma_N$ and the following model problem: find $u(x, y, z)$ that satisfies

$$cu - \nabla \cdot (\kappa \nabla u) = s \quad \text{on} \quad \Omega, \tag{1}$$

$$u = u_0 \quad \text{on} \quad \Gamma_D \tag{2}$$

$$\frac{\partial u}{\partial n} = g \quad \text{on} \quad \Gamma_N \tag{3}$$

where $c(x, y, z) > 0$, $\kappa(x, y, z) > 0$ and $s(x, y, z)$ is a given source term. We further suppose that s, u_0 and g satisfy the standard regularity assumptions and, without loss of generality, that $u_0 = 0$. A weak formulation of (3) is: find $u \in H_0^1(\Omega)$ that satisfies

$$\int_\Omega [\kappa \nabla u \cdot \nabla w + cu\ w]\ dxdydz = \int_\Omega r\ w\ dxdydz\ \ \forall w \in H_0^1(\Omega) \tag{4}$$

where $H_0^1(\Omega) = \{u \in H^1(\Omega),\ u|_\Gamma = 0\}$.

2.1. Interpolation

Consider now a mesh constructed of unstructured hexahedra. On each hexahedron e, the finite element interpolation basis is a tensor products of one dimensional basis of P_n that are the set of Lagrangian interpolants $\phi_j(t)$, $j = 0, \ldots, n$ on the Gauss–Lobatto Legendre (GLL) quadrature points in the reference domain: $t_i \in [-1, +1]$, $i = 0, ..., n$, $\phi_j(t_i) = \delta_{ij}$ [4].

In the reference hexahedron $\xi, \eta, \zeta \in [-1, +1]$ of element e, fields are interpolated as

$$u^e(\xi, \eta, \zeta) = \sum_{i=0}^{n} \sum_{j=0}^{n} \sum_{k=0}^{n} u_{ijk;e} \phi_i(\xi) \phi_j(\eta) \phi_k(\zeta) \tag{5}$$

where $u_{ijk;e}$ are the values of u at the $(n+1)^3$ nodes of element e. We define the derivation matrix D following [4] as

$$D_{ij} = \left. \frac{d\phi_i}{dt} \right|_{t=t_j}. \tag{6}$$

2.2. Local and global vectors

Consider a mesh made of N_E unstructured hexahedra with a total of N GLL nodes and a scalar field u interpolated on the mesh. In the following, two representations of u will be used, one that is defined locally to one element and a second that is defined globally on the mesh. The local version of u is denoted by

$$u_{ijk;e},\ \ 0 \le i, j, k \le n,\ \ 1 \le e \le N_E.$$

A global indexing of the GLL nodes is defined that associates a unique number to every GLL node $\mathcal{N}$ of the mesh. The global version of u is noted