A Finite-Element Method Optimized for Modern Processors

Karl Ljungkvist

Department of Information Technology, Uppsala University, Sweden
karl.ljungkvist@it.uu.se

UPPSALA UNIVERSITET

Conclusion

By replacing the sparse matrix-vector product with multiple dense tensor operations, the computational intensity (flop per byte) of a high-order FEM computation can be increased by an order of magnitude, which translates to a corresponding speedup on modern compute-intensive processors.

Introduction

The finite-element method is popular when numerically solving PDEs due to its flexibility with respect to geometry and adaptivity. It results in a large sparse linear system which is typically solved using iterative methods, where the majority of the computational work consists of matrix-vector products. However, with its low computational intensity, the sparse matrix-vector product is very poorly suited for execution on modern multi-core processors, where data transfer is expensive and computations are cheap (see **Background**). This issue grows with the number of unknowns and becomes especially severe when high-order basis functions in 3D are used.

A Matrix-Free Approach

Most of the time in the sparse matrix-vector product is spent reading the large system matrix A. In a FEM discretization, A is built up as a sum over all the elements in the mesh $\mathcal{K}$ of small contributions a_K,

$$A = \sum_{K \in \mathcal{K}} P_K a_K P_K^T ,$$

where the a_K are local matrices, and the P_K are tall and skinny permutation matrices selecting the unknowns living on element K. Exploiting this structure, we can merge the construction and multiplication steps and remove the need to explicitly form A. Thus,

$$v = Au = \left(\sum_{K \in \mathcal{K}} P_K a_K P_K^T \right) u = \sum_{K \in \mathcal{K}} P_K a_K \left(P_K^T u \right) .$$

Since P_K simply extracts a few unknowns from u, this is essentially a number of small dense matrix-vector multiplications.

Tensor Structure

For a mass matrix, the local multiplication has the form

$$v_i = \sum_j a_{ij} u_j = \sum_j \sum_q \varphi_i(\mathbf{x}_q) \varphi_j(\mathbf{x}_q) w_q |J(\mathbf{x}_q)| u_j , \qquad (1)$$

where i and j are local indices, φ_i are basis functions, J is the Jacobian, and $\mathbf{x}_q$ and w_q are quadrature points and weights respectively.
To further improve performance, we optimize this operations by using the tensor structure of basis functions and quadrature points on quadrilateral/hexahedral meshes,

$$\varphi_i(\mathbf{x}_q) = \underbrace{\psi_\mu^\alpha \psi_\nu^\beta \cdot \ldots \cdot \psi_\eta^\gamma}_{D \text{ factors}}, \text{ where } \psi_\mu^\alpha := \psi_\mu(x^\alpha)$$

where the $(\mu, \nu, \ldots, \eta)$ and $(\alpha, \beta, \ldots, \gamma)$ correspond to i and q respectively. Using this, and assuming a 3D case, (1) can be rewritten as

$$u^{\alpha\beta\gamma} = \sum_\mu \psi_\mu^\alpha \sum_\nu \psi_\nu^\beta \sum_\eta \psi_\eta^\gamma u_{\nu\mu\eta} ,$$

and

$$v_{\nu'\mu'\eta'} = \sum_\alpha \psi_{\mu'}^\alpha \sum_\beta \psi_{\nu'}^\beta \sum_\gamma \psi_{\eta'}^\gamma w^{\alpha\beta\gamma} |J^{\alpha\beta\gamma}| u^{\alpha\beta\gamma} .$$

This is a set of tensors contractions, i.e. essentially dense matrix-matrix multiplications, which have a high degree of computations per data access.

Results

To compare our approach (Matrix-Free) with the standard sparse-matrix based technique (SpMV), we compute the data usage and the computational intensity for problems with an equal number of unknowns with varying polynomial degree of basis functions. Figures 1 and 2 show the results for 2D and 3D respectively.

Figure 1. *2D*

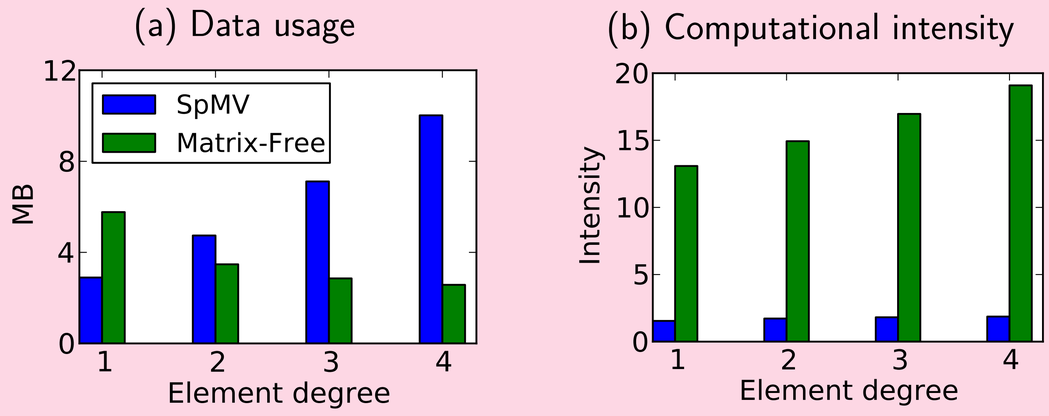
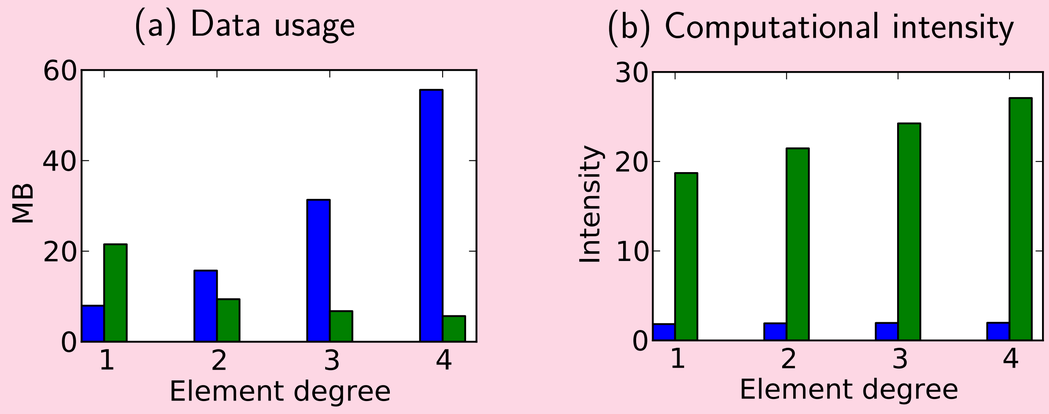


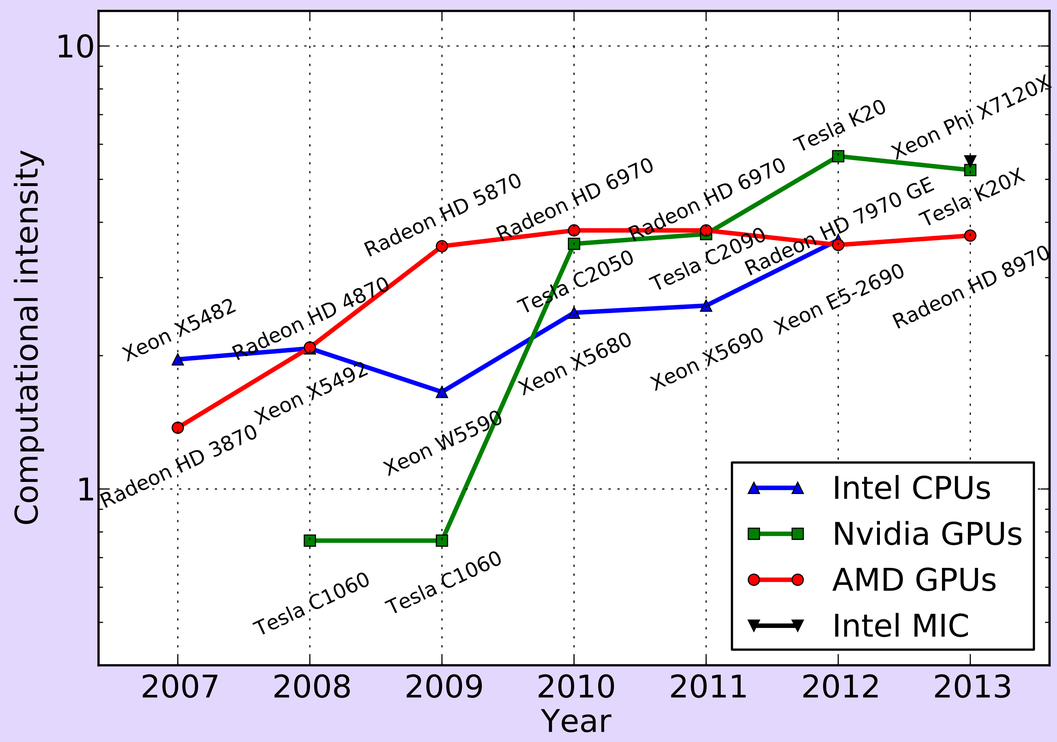
Figure 2. *3D*



Background: Hardware Trends

Recently, the the increase in compute power of processors has not been matched by the increase of memory bandwidth. To an increasing extent, processors need to perform multiple arithmetic instructions for each memory access in order to saturate the compute units. (See Fig. 3)

Figure 3. *Ratio of compute power to bandwidth (double prec.)*



With this increasing importance of bandwidth, the *computational complexity*, i.e. the number of floating-point operations, is becoming a less relevant characterization of operations. Metrics which are more relevant today include *data usage*; the amount of data accessed, and *computational intensity*; the amount of computations per data access,

$$CI = \frac{\#\text{ floating-point operations}}{\#\text{ bytes accessed}} .$$