Parallel Cell-Based Finite Element test with Deal.II

Stephen O'Sullivan
Department of Physics and MIT Kavli Institute,
Massachusetts Institute of Technology, Cambridge, MA 02139
(Dated: December 20, 2012)

This paper concerns the use of the Matrix-Free class in the finite element Deal.II library as applied to a time-dependent non-linear pde, especially its parallel processing functionality.

I. INTRODUCTION

Time-dependent non-linear pdes are generally hard if not impossible to solve analytically so we are restricted to applying numerical methods in such situations. Finite element methods were originally applied to self-adjoint operators (elliptic equation) but have eventually found use in non-self-adjoint operators like those involving hyperbolic equations. Typically hyperbolic pdes have been solved with finite difference methods, for example the Crank-Nicholson method, due to their simplicity and efficiency. However these methods are inflexible regarding mesh refinement, mesh transformation and variable coefficients. The extra overhead of using the finite element method can/should be aleviated by the use of libraries which take care of the generic parts of finite element engineering namely grid handling and refinement, handling degrees of freedom, input and output of results, etc. There are many such packages but here we concentrate on one such package, Deal.ii. This library provides classes which wrap around core linear algebra utilites such as PetSc or Trilinos which both support MPI and shared memory pthreads. This allows us to distribute matrices and vectors across several computers within an MPI network by simply using the releveant data structures and classes supplied by the library. It's also worth mentioning that the library can be linked to METIS which provides high quality partitions.

A typical FE implementation involves the following stages:

1. Preprocessing:
   (a) Discretize the geometry to generate a mesh
   (b) Figure out the transformation between physical and local element coordinates
2. Assembly: Start with the geometric structures formed from the mesh and from the stiffness matrix associated with the discretization to form a global linear system
3. Solution of the algebraic system

Physically relevant pdes are generally time-dependent and can also be non-linear. In such cases coefficient matrices will need to be assembled at each iteration during a simulation. To circumvent this Kronbichler and Kormann[2] provide an implementation which involves discarding the typical procedure of separating the finite assembly routines from the linear algebra which keeps operations focussed at the finite element cell level with the result that we are able to abandon unwieldly sparse matrices. The idea is that we can aleviate the memory bottle-neck associated with large grids by recomputing information from the cell and leveraging underused CPU power.

II. MATRIX-FREE FINITE ELEMENT METHOD

The finite element method is well known to be mathematically described by the problem of finding a finite dimensional function $u_h \in V_h$ that satisfies $a(u_h, \phi_h) = (f, \phi_h) \forall \phi_h \in V_h$. The choice of bases ϕ_i for this space V_h will be such that it is defined locally on the cells of the mesh. The small support of these basis functions will result in a sparse matrix $L_{ij} = a(\phi_i, \phi_j)$.

$$A(u) = f \tag{2.1}$$

for some differential operator A

The standard finite element method begins by taking a pde and then integrating it over a known set of trial functions v. To discretize this expression we can express the functions as having support on a finite dimensional basis. The problem then reduces to finding $u \in V$ such that $\forall v \in V$

$$\phi(u,v) = \int fv \tag{2.2}$$

where V is a finite dimensional space, more specifically a set of piecewise polynomials. This equation can be converted into matrix form by explicitly writing the functions over their basis functions

$$u = \sum_i u_i v_i \tag{2.3}$$

$$A(u) = \sum_{k=1}^{n \text{ cells}} C^T P_k^T A_k (P_k C u) \tag{2.4}$$

where P_k denotes the matrix which defines the location of the cell-related degrees of freedom. What we are doing here is simply constructing a matrix-vector product which represents information globally across the entire grid from information defined locally on each cell. C represents the information from having different levels of refinement over different cells leading to so-called hanging nodes being transfered to the global matrix.

If the cells are partitioned across different MPI processes then information about the degrees of freedom will need to be passed from one processor to another. The algorithm for evaluating $A(u)$ will then look like the following

1. Get the vector values from foreign MPI processes that are needed for compututation on the locally owned cells
2. looping over local cells calculate:
   (a) the local vector values $u_k = (P_k C)u$
   (b) $A_k u_k$ by quadrature
   (c) add local contribution to global result $v = v + (P_k C)^T v_k$
3. send info on dof to other nodes

The quadrature operation itself can be computed without the need for storing the local matrices A_k by evaluating the FE function, u_k, and its derivatives on all quadrature points over each of the trial basis functions. Now if one uses a Gauss-Lobatto quadrature, the integrals are approximated by the underlying Gauss-Lobatto quadrature formula at the same points. This reduces the complexity of the finite element operator evaluation since the value of the FE function at the quadrature points is simply given by 1-d coefficients.

III. SINE-GORDON SOLUTION

The Sine-Gordon equation is given by

$$u_{tt} - \Delta u = -\sin(u) \qquad \text{for } (x,t) \in \Omega \times (t_o, t_f) \tag{3.1}$$

$$n.\Delta u = 0 \qquad \text{for } (x,t) \in \partial\Omega \times (t_0, t_f) \tag{3.2}$$

$$u(x,t_0) = u_0(x) \tag{3.3}$$

The analytic solution is

$$u(x,t) = \prod_{i=1}^{d} -4 \arctan\left(\frac{m}{\sqrt{1-m^2}\frac{\sin\sqrt{1-m^2}t+c_2}{\cosh mx_i+c_1}} \right) \tag{3.4}$$

We use this to specify our initial value $u_0(x)$. Discretizing in time with a second order leap-frog gives us

$$\frac{\partial}{\partial t}\left[\frac{u^n - u^{n-1}}{\Delta t}\right] = \Delta u - \sin(u) \tag{3.5}$$

$$\frac{\frac{u^{n+1}-u^n}{\Delta t} - \frac{u^n - u^{n-1}}{\Delta t}}{\Delta t} = \Delta u - \sin(u) \tag{3.6}$$

$$\frac{u^{n+1} - 2u^n + u^{n-1}}{(\Delta t)^2} = \Delta u - \sin(u) \tag{3.7}$$

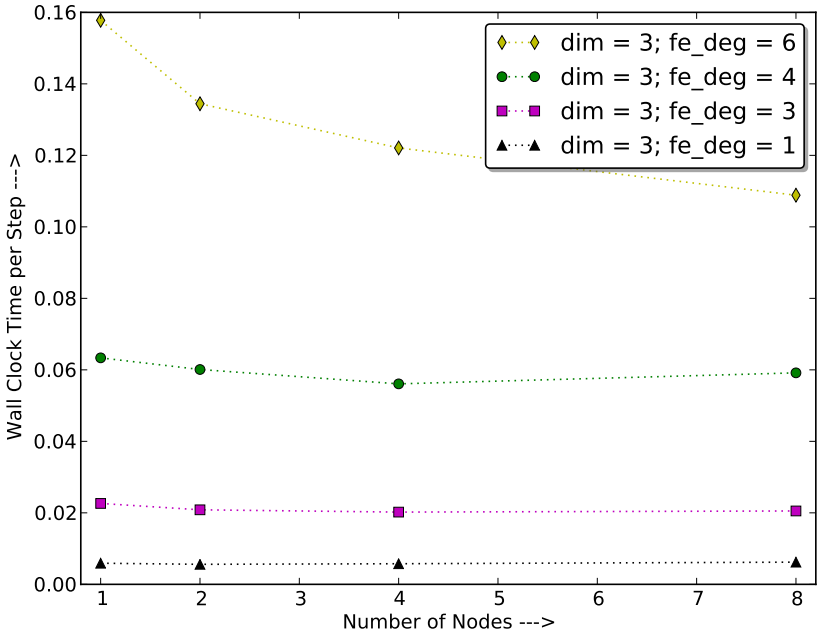

FIG. 1: Plot of wall time versus number of nodes

$$u^{n+1} = 2u^n - u^{n-1} + \Delta u(\Delta t)^2 - (\Delta t)^2 \sin(u) \tag{3.8}$$

Integrating this expression over a test function v gives

$$(v, u^{n+1}) = (v, 2u^n - u^{n-1} - (\Delta t)^2 \sin(u^n)) - (\Delta v, (\Delta t)^2 \Delta u^n) \tag{3.9}$$

We can represent this problem in matrix form as

$$MU^{n+1} = L(U^n, U^{n-1}) \tag{3.10}$$

Notice that because we are using Gauss-Lobatto based elements, the mass matrix M here is diagonal and easily invertible. In this implementation the matrix L is in matrix-vector form and acts on the cells defined by the grid.

IV. PARALLEL MATRIX-FREE IMPLEMENTATION

Parallelization via MPI is implemented with the p4est package which is accessible through the deal.II library. The domain is decomposed into subdomains and each processor handles a course mesh for the entire grid as well as a fine mesh on its local subdomain. The matrix-free class in deal.II has a vector type which passes the necessary information between processors. Now since each processor has several cells associated with it may be possible to use multi-threading to operate on more than one cell simultaneously so long as the cells do not share any common degrees of freedom. The matrix-free class includes a facility for sub-dividing the cells so as to avoid this interference. The solver uses a Kelly error estimator to refine the mesh where it is required. Once the mesh is refined Deal.II has a class which transfers the new mesh even if the mesh is in distributed memory.

V. RESULTS AND CONCLUSIONS

The plot shows the computation time versus the number of active processors for simulations of the three-dimensional Sine-gordon equation based on finite element degrees from one up to six. The degrees of freedom range from 60,000 to 32 million. There is clearly a scaling behaviour with the number of nodes, at least for the solution with finite degree equal to six. Parallelization seems to make more of a difference as the finite element degree is increased and

the problem becomes more complex. I am not really sure how to interpret the relative flatness of these results. Unfortunately I could not get the multi-threading functionality to work. This could be either a result of how Deal.II was compiled on the cluster or perhaps a bug in my code. In any case I was unable to diagnose the problem. It would have been nice to see how the MPI and multi-threading functionalities would have interplayed together.

---

[1] W. Bangerth, R. Hartmann, and G. Kanschat, ACM Trans. Math. Softw. **33** (2007), ISSN 0098-3500, URL `http://doi.acm.org/10.1145/1268776.1268779`.

[2] M. Kronbichler and K. Kormann, Computers & Fluids **63**, 135 (2012), ISSN 0045-7930, URL `http://www.sciencedirect.com/science/article/pii/S0045793012001429`.

[3] C. Akcadogan and H. Dag, in *Parallel and Distributed Computing, 2003. Proceedings. Second International Symposium on* (2003), pp. 1 – 8.