Parallel Cell-Based Finite element test with Dealii

Stephen O'Sullivan

MIT Kavli Institute for Astrophysics and Space Research

December 10 2012

Outline

Punchline

Want to discretize some continuous differential operator
Typical Finite element method

- Construct sparse matrix
- Solve using linear algebra engines

Finite element method

Parameterize geometry

- Divide structure in simple (unstructured) cells
- Enables us to perform integrations and define basis functions

Build basis functions (polynomials on the cells)

- Continuous
- Simple derivatives
- Easy to integrate over cells

Solve $Lu = b$ for some sparse matrix L

Matrix-free vs sparse matrix

IDEA: Main memory is bottle neck for scientific computing (big grids which do not fit in cache) - Substitute looking up matrix elements in memory by re-computing them. Access the matrix by evaluating matrix-vector products.
Don't assemble a global sparse matrix Instead only store

- the unit cell shape function information
- the enumeration of the dofs (vertices of mesh)
- transformation from unit cell to real cell

Arrange operations such that successive Matrix-Vector operations are performed $\rightarrow$ less memory transfer during computations

Matrix-free class in Dealii

MPI parallelization on clusters of distributed nodes

- Domain decomposed into subdomains of equal size and assigned to individual processors.
- Each processor holds a coarse mesh on the entire domain
- Data associated to dofs on the subdomain boundaries are available to all processors involved.

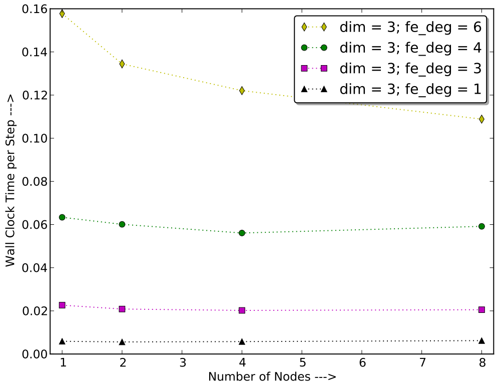
Thread parallelization scheduled by Threading Building Blocks library

- Subdivide cells into subdomains with precomputed neighbor relations
- Use Intel Threading Building Blocks to schedule tasks dynamically in such a way that no neighboring cells are worked on simultaneously. (Color graph problem: No two adjacent cells with the same color).
- Algorithm:
    1. Partition domain such that cells belonging to partition k at most overlap with partitions k-1,k,k+1 so that all even/odd partitions are independent
    2. Within each partition subdivide the cells following the same strategy as in part (1)

Vectorization by clustering of two or more cells into a SIMD data type for operator application

- Custom vectorization within the CPU (SIMD)
- Operations are typically the same on all cells

Scaling with number of nodes


dim = 3; fe_deg = 6
dim = 3; fe_deg = 4
dim = 3; fe_deg = 3
dim = 3; fe_deg = 1
Wall Clock Time per Step --->
Number of Nodes --->