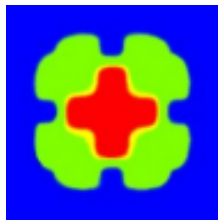
deal.II and hybrid programming

Guido Kanschat

IWR, Universität Heidelberg

Department of Mathematics, Texas A&M University



deal.II Workshop 2013, College Station, August 19–22, 2013

Challenges of new architectures

- Multicore (more than 8!)
- Heterogeneous resources
  - CPU(s)
  - GPU(s)
- Hierarchy of parallelism
  - shared memory
  - fast communication
  - slow communication

IWR

Challenges of new architectures

- Multicore (more than 8!)
- Heterogeneous resources
  - CPU(s)
  - GPU(s)
- Hierarchy of parallelism
  - shared memory
  - fast communication
  - slow communication

Most of these already apply to desktop computers!

IWR

Example: Conjugate gradient for high order DG

- 2048×2048 cells, Q_7 elements
- 64 dofs per cell, 256M dofs total
- DG coupling between cells
- parallel for, local multiplication as template
- 100 conjugate gradient steps

Threads	Wall time	Efficiency
1	18033	1
5	3795	0.95
10	2041	0.88
20	1096	0.82
40	661	0.68
80	427	0.53
160	396	0.28

IWR

Assumptions on hardware development I

- Processing frequency will remain constant or drop
  - Power consumption is superlinear in frequency
- Computing power of units will increase
  - more cores
  - more threads

Assumptions on hardware development II

- Memory access will remain “slow” and “expensive”
  - Wiring for high frequency signals
  - Physical barriers due to moved charges
- Hierarchical memory access
  - Cache hierarchies
  - On chip, on board
  - Exclusive access, shared access

IWR

Assumptions on hardware development III

- Vectorization?
  - Intel Xeon Phi
    - Vector instructions
    - Issued by compiler
  - nVidia Kepler
    - Vectorization by hand
    - Libraries

IWR

1 Introduction

2 Assumptions or Predictions

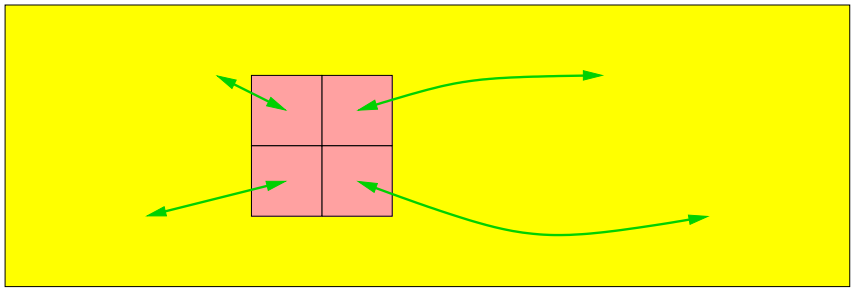
3 **Programming models**
- Multicore
- Dispatch
- Message passing
- Hybrid programming

4 Structure of FEM Software

5 Desirable data structures

6 Wishes for deal.II

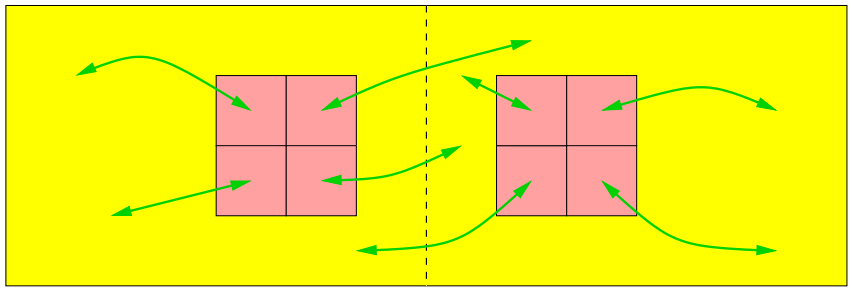
Multicore programming



Examples: parallel for, pipeline

- Slice problem into small units
- Assign units to task/thread
- Task pulls data from shared memory as needed
- Memory access speed uniform

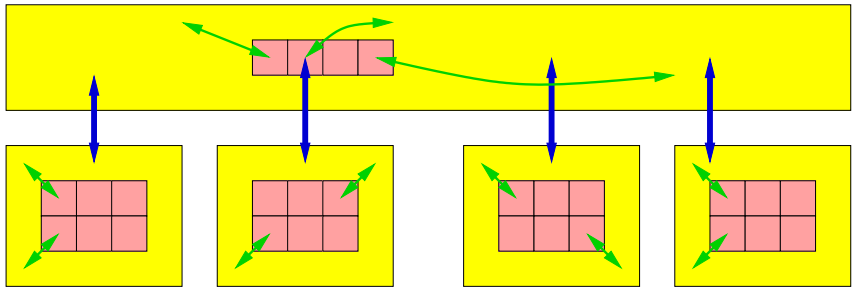
Multicore programming



Examples: parallel for, pipeline

- Slice problem into small units
- Assign units to task/thread
- Task pulls data from shared memory as needed
- Memory access speed uniform
  - But: caches and multiple multicore processors

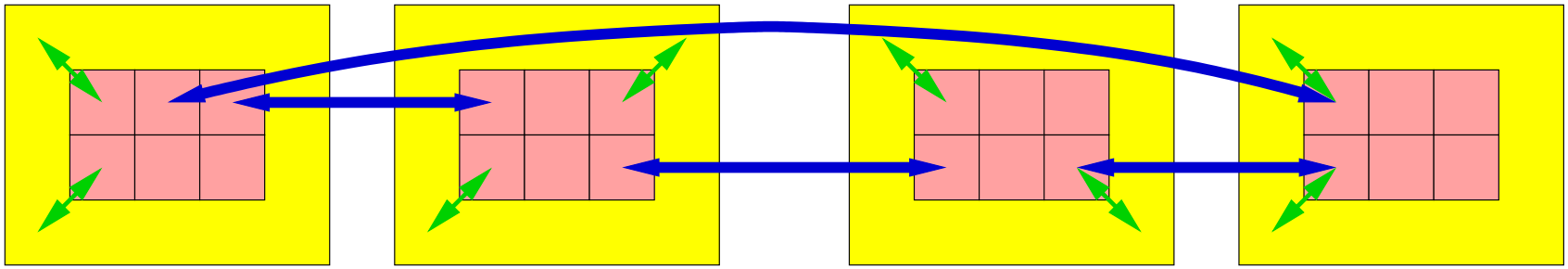
Dispatch



Example: Kepler, Xeon Phi

- Master process slices problem into small units
- Data is copied to memory on computing hardware
- Units are dispatched to computing hardware
- Localization of memory access crucial

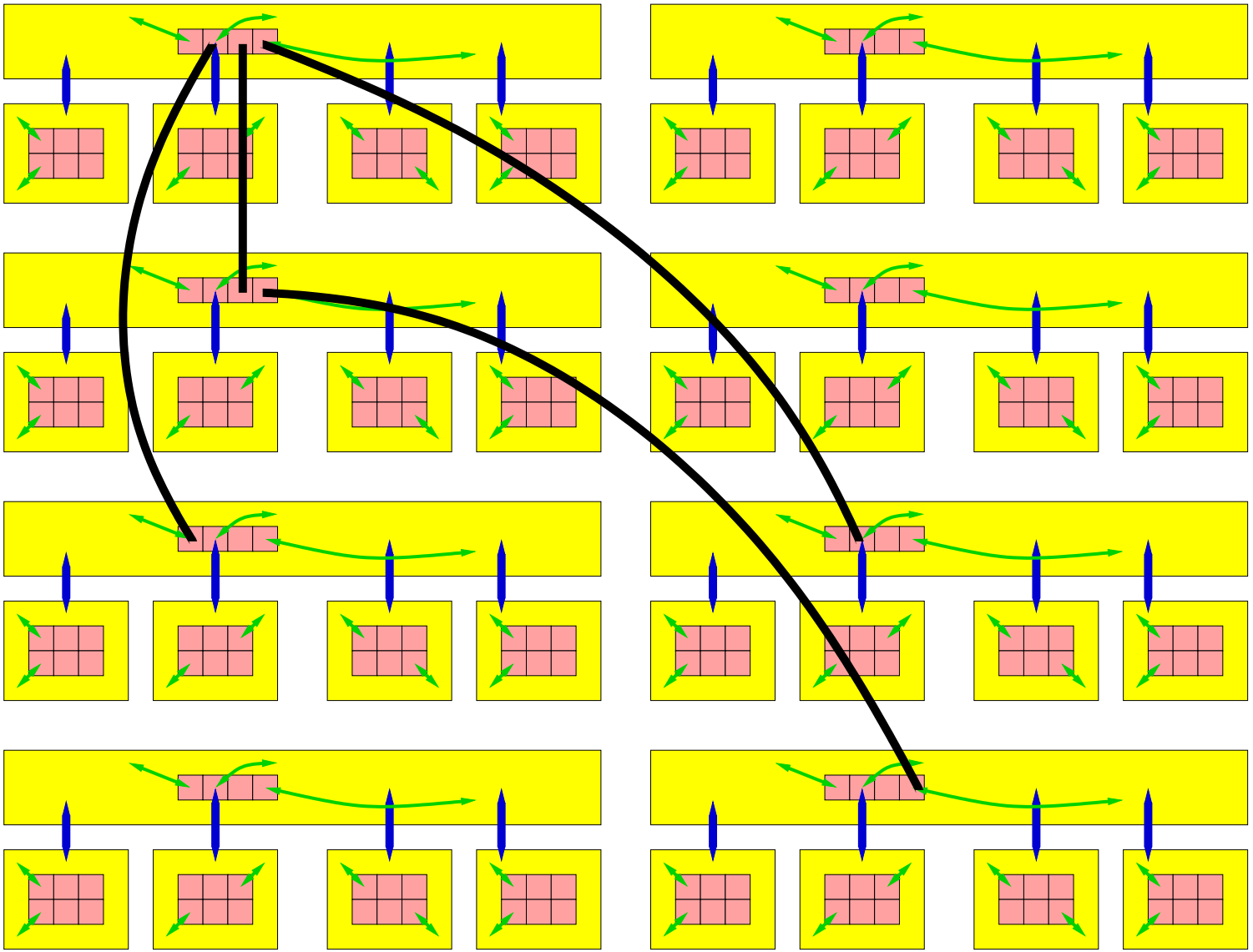
Message passing



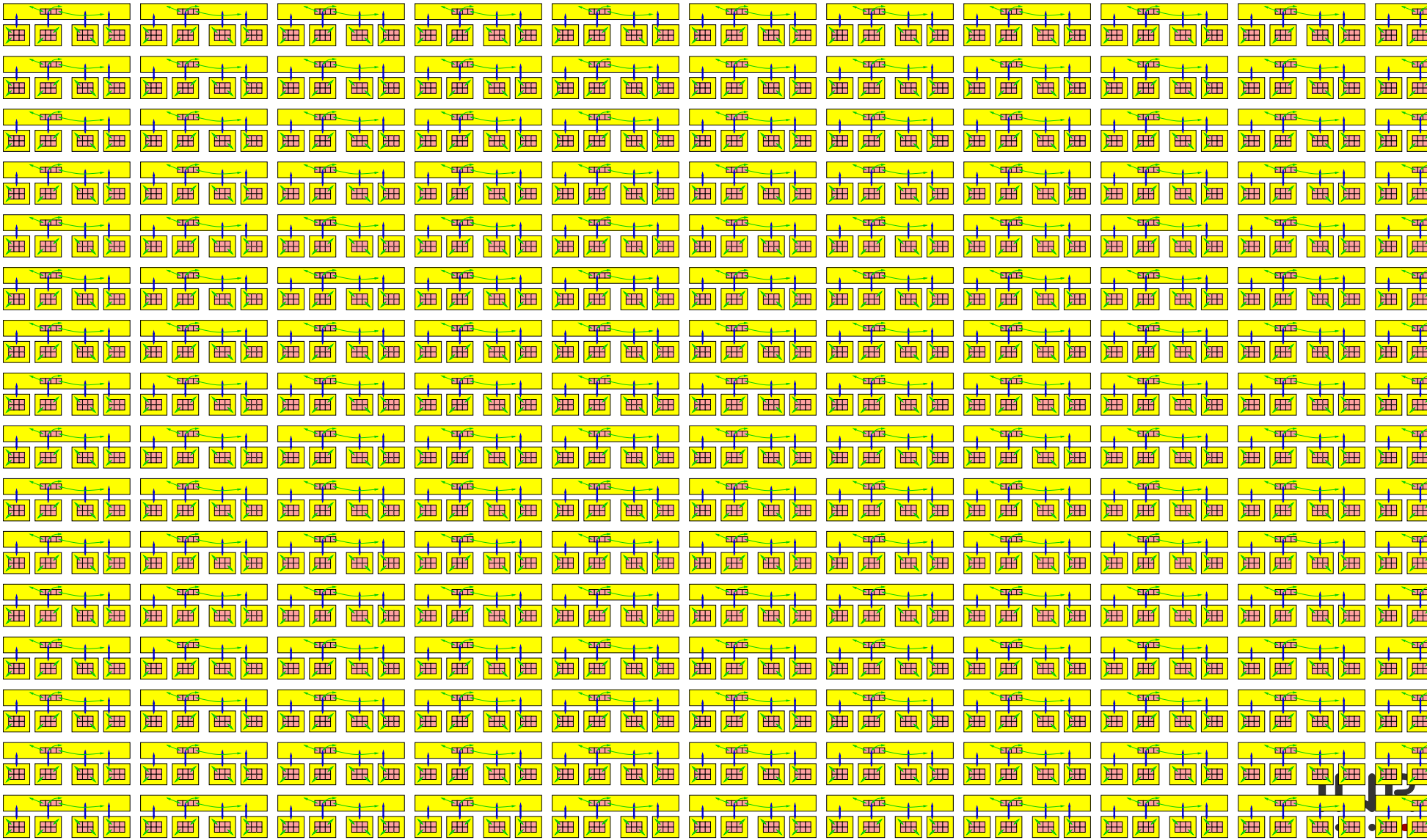
Example: clusters, MPI, grid computing

- Each process only has a small unit of the problem
- Process knows its relationship to others
- All data is local from the start

Hybrid architecture



Hybrid architecture



Hybrid programming or “all of the above”

- Even laptops come with powerful GPUs
- Desktops will soon have several highly parallel computing units
- Nodes in clusters will be heterogeneous

Are we expecting a general purpose solution?

- Cuda-aware MPI?
- Hierarchical TBB?

What if ...?

Source: Mark Pellegrini, Wiki Commons, © 2007

Integration

- Computation of right hand sides
- Matrix-vector multiplication
- Error estimates
- Level transfers in geometric multigrid
- Schwarz-type smoothers

Almost every action in a finite element program can be considered integration

Integration

- Computation of right hand sides
- Matrix-vector multiplication
- Error estimates
- Level transfers in geometric multigrid
- Schwarz-type smoothers

Almost every action in a finite element program can be considered integration

- Integration on cells (hierarchy, level, leaves)
- Integration on faces (boundary, regular and refined interior)
- Integration over cell patches
- Transfer between mesh levels
- Transfer and coupling between meshes

IWR

Types of global loops

- Loop over all active cells
- Level loops
- Cells only or **cells and faces**
- Without or **with hanging nodes**
- Loops for transfer between mesh levels
- Loops interfacing between two meshes

MeshWorker abstraction

Currently

Loop over cells, loop over cells and neighbors

Future

Loop over pairs of cells

- Parallelization by leading cell or pair
- Extension to MeshWorker or MatrixFree

Input data

- Amount of input data a priori unknown
  - previous Newton iterate
  - previous time step(s)
  - state and control variables
- Implementation in MeshWorker
  - Named data vectors
  - automatic evaluation in quadrature points

IWR

Message passing

Well established tools

- Generate distributed mesh
  - E. g. p4est
- Parallel linear algebra
  - E. g. Trilinos, Petsc
  - Caveat: sparse matrices!
- Distribute hierarchy levels for multigrid

IWR

Loop in multicore/dispatch environment

- Natural parallelization by cell
  - Perfect scaling for DG and explicit timestepping (Warburton)
  - Race condition for continuous elements
  - Possibly too coarse grained for GPUs
- Parallelization by test function
  - No race condition
  - Possibly too fine grained for multicore
  - Integrals need cells or faces
- Parallelization by cell/face/edge/vertex
  - No race condition
  - Still issue about missing information

  $\Rightarrow$ Stencils
- Rearranging cells (Martin)

IWR

Loop and dispatch

- Kernel runs on different architecture
  - Separation of loop and kernel, “mesh and cell”
    - E. g. Fenics: Dolphin and FFC
    - Meshworker in deal.II?
    - Loop and kernel in different compilation units?
- Memory for kernel limited
- Data locality through Peano curves
- View the loop as data stream to the kernels

Streaming model

Critical operation: feeding data to the processing unit for the kernels

Streaming model

Critical operation: feeding data to the processing unit for the kernels

Benchmark

- Solution vector: 8 bytes per degree of freedom
- Few data vectors (Newton residual, previous time steps)

Data structures: mesh

Separate topology and geometry

- Topology: adaptive, hierarchical structure
  - local refinement
  - geometric multigrid
- p4est
  - only few bytes per cell for DG
  - some more bytes for lower dimensional objects
  - load balancing included
- Anisotropic refinement?

Data structures: mesh

Separate topology and geometry

- Topology: adaptive, hierarchical structure
  - local refinement
  - geometric multigrid
- p4est
  - only few bytes per cell for DG
  - some more bytes for lower dimensional objects
  - load balancing included
- Anisotropic refinement?

Implement geometry algorithmically

Data structures: degrees of freedom

Focus on higher order elements

- Full hp flexibility
  - One integer indicating the first degree of freedom on each cell, face, edge, vertex
- Compare to $p^3 > 100$ vector entries per cell
- But: no renumbering of dofs possible

Data structures: matrices

No!

But, quadrature of matrices is order p^{2d}

- Use full quadrature only for Newton residuals
- Use local matrix approximations for linearization
  - Affine instead of nonlinear cell mapping
  - Constant or low order coefficients or nonlinearity
- Use even coarser approximation for preconditioning

Data structures: FEValues

- Introduced long ago to enable optimization
- Never heavily exploited
- Precompute as much as possible on reference cell
- Provide all required values as DATA
- Tends to become very large for higher order
- Uses up valuable cache space

$\Rightarrow$ not maintainable $\rightarrow$ FEEvaluation?

Wishes for the future

- Generalization and unification of MeshWorker and MatrixFree
- Separation of “kernel” code
- Reimplementation of quadrature and elements as templates

ANS

Archive of Numerical Software

www.archnumsoft.org