



Numerical Solution of PDEs Using the Finite Element Method

May 15 – 19 2017

Martin Kronbichler

(kronbichler@lnm.mw.tum.de)

Luca Heltai (luca.heltai@sissa.it)



Goals

- How to use deal.II for Finite Element computations
- Refresh numerical PDE knowledge
- Also:
 - Software best practices
 - C++, Debugging, IDEs, Visualization
 - Parallel computations with MPI

Schedule

| | Monday | Tuesday | Wednesday | Thursday | Friday |
|-------------|-----------------|------------------------|-----------------|----------|--------|
| 09:30-10:45 | Intro | Dimension independence | | | |
| 11:15-12:30 | First Steps | | | | |
| | | | | | END |
| 14:00-15:15 | Seminar (14:30) | | Hands-on | | |
| 15:45-17:00 | Hands-on | | Seminar (16:00) | | |

- Today:
 - What is deal.II?
 - Compiling, using an IDE
 - Overview about FEM
 - Basic tutorials (create mesh, solve Poisson's equation, visualization)
- Tuesday:
 - Finite Element Analysis (refinement, computing errors)
- Wednesday/Thursday/Friday:
 - Advanced topics
 - Time for projects

The plan

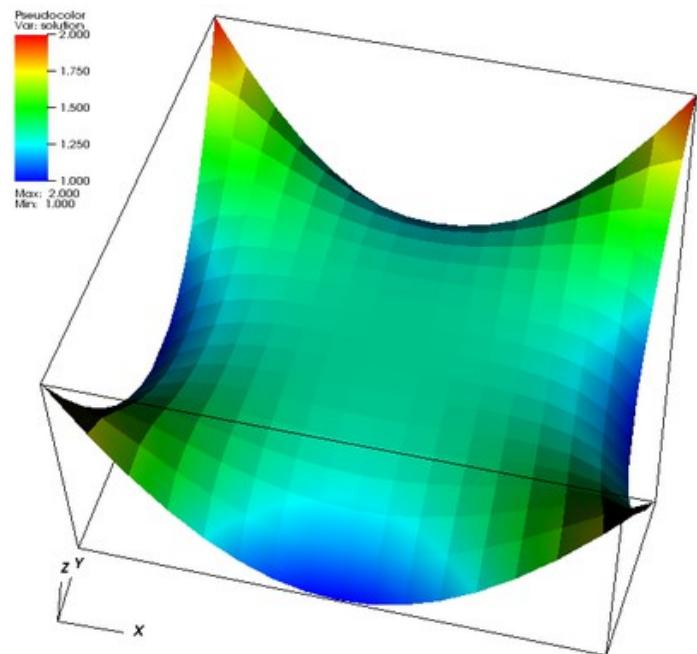
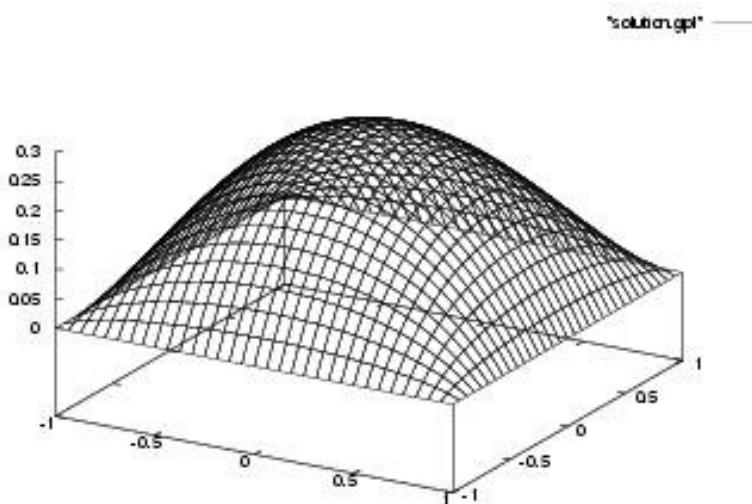
- Slides, some lectures on blackboard
- Many live demonstrations
- Exercises:
 - Work in groups of two!
 - Ask questions!
- Projects:
 - Required for MHPC students
 - Groups of two

Resources

- <http://indico.ictp.it/event/7751/overview>
 - Schedule, Rooms, etc.
- <http://dealii.org>
 - Manual
 - Tutorial steps
 - Tutorial videos
- On your machine: folder /scratch/smr1909/
 - Slides
 - Example programs
 - Exercises
 - Other files

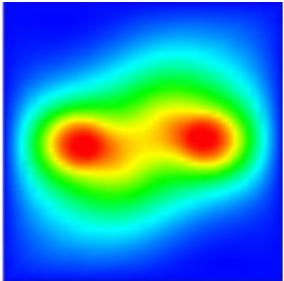
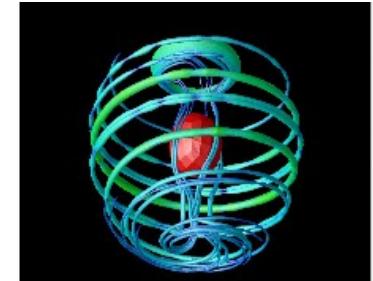
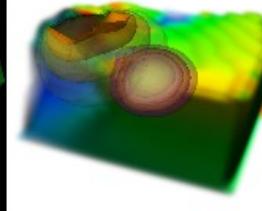
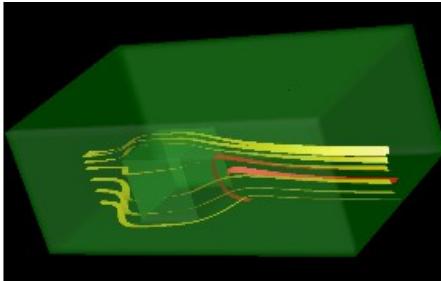
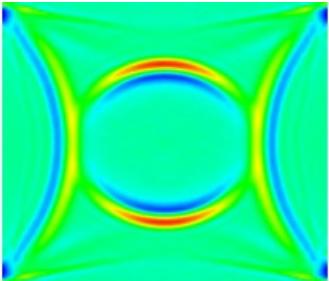
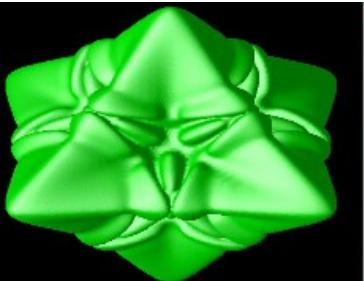
Finite Element Method

- Solve partial differential equations numerically
- Example:
$$\begin{aligned} -\Delta u &= f && \text{in } \Omega, \\ u &= 0 && \text{on } \partial\Omega. \end{aligned}$$



deal.II

- “A Finite Element Differential Equations Analysis Library”
- Open source, c++ library
- I am one of the four maintainers
- One of the most widely used libraries:
 - 900+ papers using and citing deal.II
 - ~600 downloads/month
 - 100+ people have contributed in the past 15 years
 - ~500,000 lines of code
 - 10,000+ pages of documentation
- Website: www.dealii.org



Features

- 1d, 2d, 3d computations, adaptive mesh refinement (on quads/hexes only)
- Finite element types:
 - Continuous and DG Lagrangian elements
 - Higher order elements, hp adaptivity
 - Raviart-Thomas, Nedelec, ...
 - And arbitrary combinations
- PDEs on surfaces embedded in higher dimensions

Features, part II

- Linear Algebra
 - Own sparse and dense library
 - Interfaces to PETSc, Trilinos, UMFPACK, BLAS, ..
- Parallelization
 - Laptop to supercomputers
 - Multi-threading on multi-core machines
 - MPI: 64,000+ processors
- Output in many visualization file formats

Development of deal.II

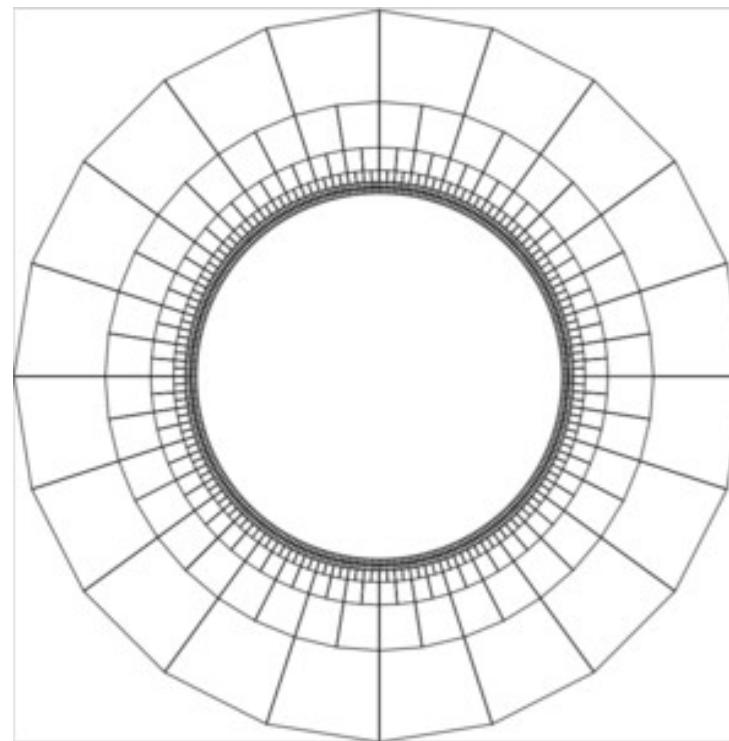
- Professional-level development style
- Development in the open, repository on github.com
- Mailing lists for users and developers
- Test suite with 8,700+ tests after every change
- Platform support:
 - Linux/Unix
 - Mac
 - Windows
- Hope to see you on github.com or the mailing list!

Lab Setup

- deal.II and all required dependencies are already installed at
/scratch/smr2909/
- (Demo, show lab01.pdf and run included step 1)

Lab 1 (step-1)

- See lab01.pdf
- Topic: creating meshes



Running examples

- In short:

```
cd examples/step-1  
cmake .  
make run
```

- cmake:

- Detect configuration, only needs to be run once!
- Input: CMakeLists.txt
- Output: Makefile, (other files like CMakeCache.txt)

- make:

- Code compilation
- Tool to execute commands in Makefile, do every time you change your code
- Input: step-1.cc, Makefile
- Output: step-1 (the binary executable file)

- Run your program with

```
./step-1
```

- Or (compile and run):

```
make run
```

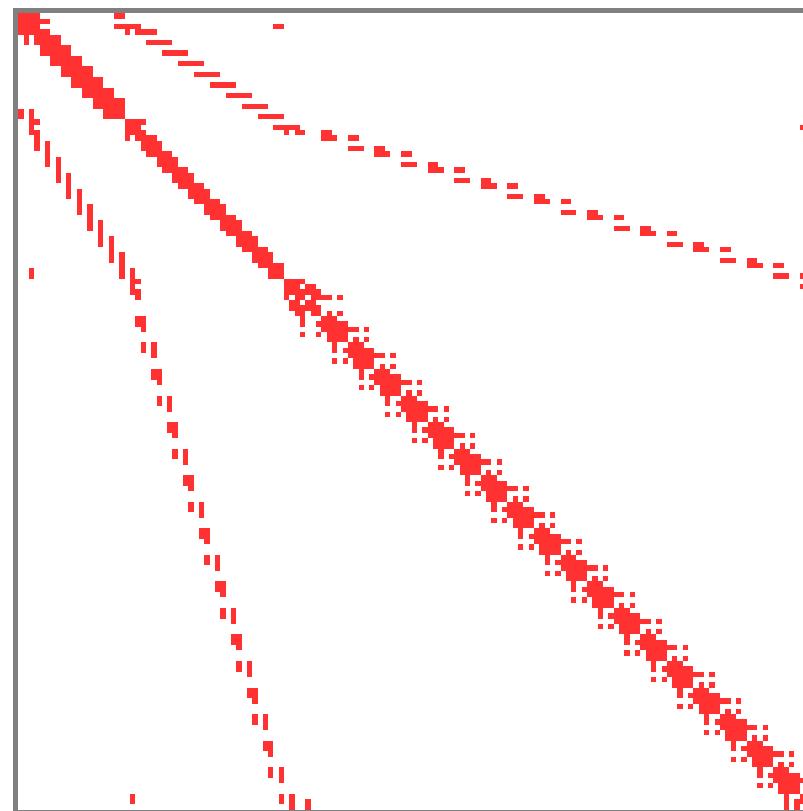
- Open in qtcreator IDE:

```
qtcreator .
```

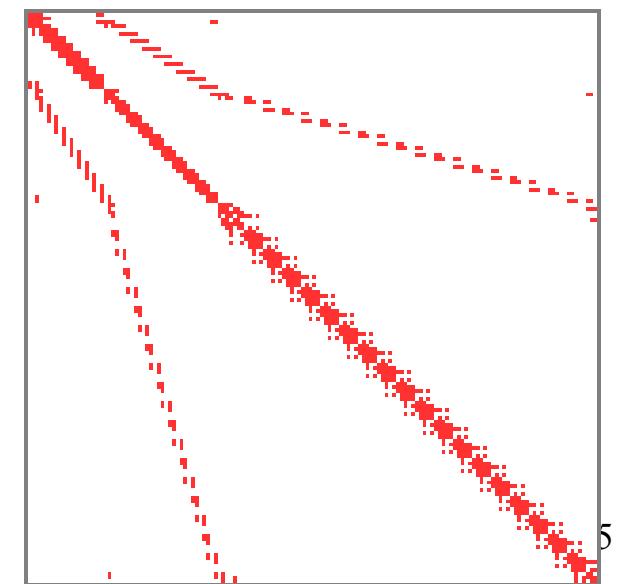
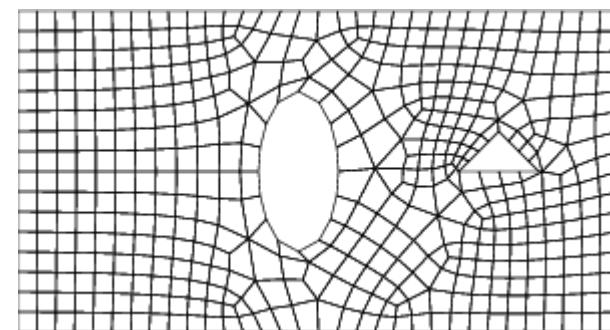
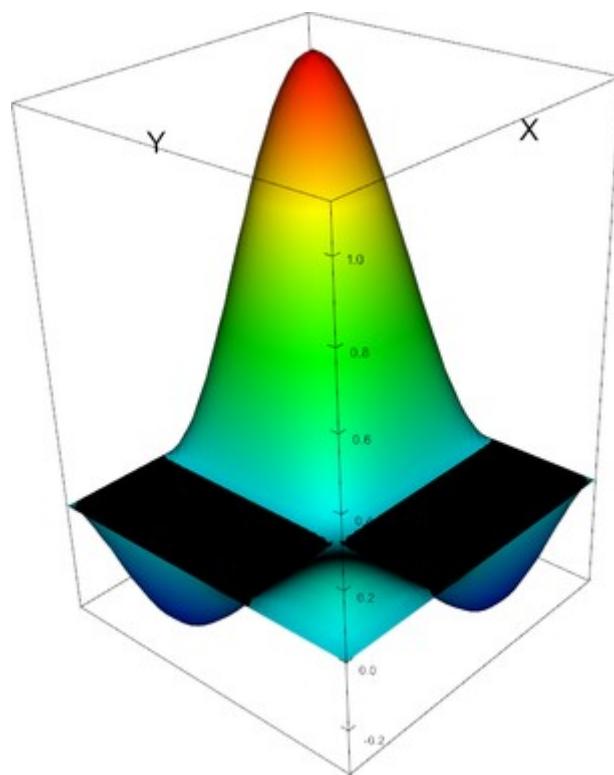
- Learn to use an IDE! (not vim, emacs, ...)
- Copy into home directory required!
- [Demo the above and open project in qtcreator]

Lab 2 (step-2)

- See lab02.pdf
- Topic: sparsity patterns of matrices



Finite Element Assembly



$$A_{ij} = (\nabla \phi_i, \nabla \phi_j)$$

$$A_{ij} \approx \sum_K \sum_q J_K^{-1}(x_q) \nabla \phi_i(x_q) \cdot J_K^{-1}(x_q) \nabla \phi_j(x_q) \cdot |\det J(x_q)| w_q$$

in pseudo-code:

```
for i=0,...,N-1:
    for j=0,...,N-1:
        for all K:
            A_ij += \sum_q grad_phi(i,q) grad_phi(j,q) JxW(q)
```

But most of these contribution are zero. So we switch the order of the loops to get

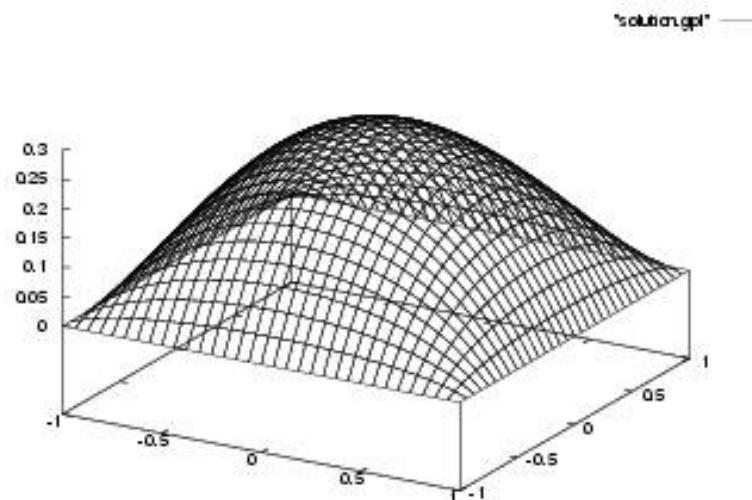
```
for all K:
    for i = 0,...,N-1:
        for j = 0,...,N-1:
            A_ij += \sum_q grad_phi(i,q) grad_phi(j,q) JxW(q)
```

which I can simplify to only look at non-zero basis functions:

```
for all K:
    a = 0
    for alpha = 0,...,n_local_dofs:
        for beta = 0,...,n_local_dofs:
            for q:
                a_{alpha,beta} += grad_phi(alpha,q) grad_phi(beta,q) JxW(q)
    A_ij += a
```

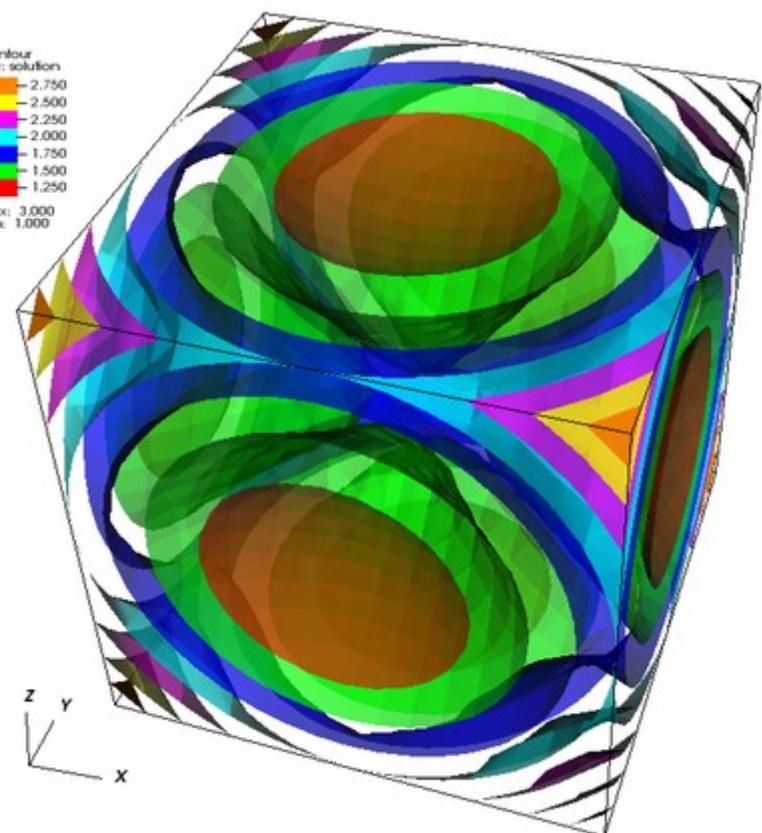
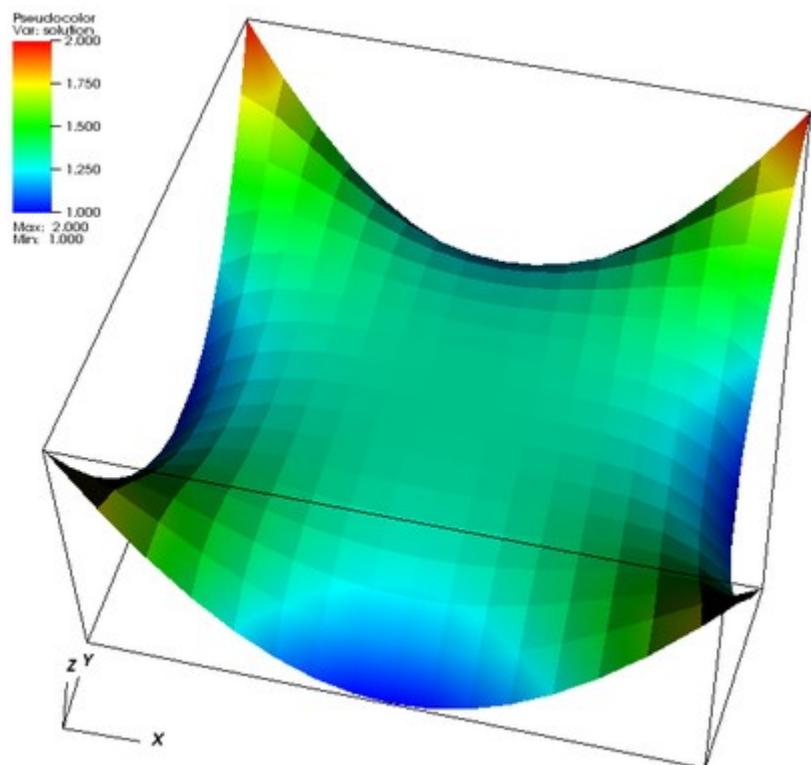
Lab 3 (step-3)

- See lab03.pdf
- Topic: solving Poisson's equation



Towards Lab 4 (step-4)

- Goals:
 - Dimension independent programming
 - Need: C++ templates



Templates in C++

- “blueprints” to generate functions and/or classes
- Template arguments are either numbers or types
- No performance penalty!
- Very powerful feature of C++: difficult syntax, ugly error messages, slow compilation
- More info: <http://www.cplusplus.com/doc/tutorial/templates/>
<http://www.math.tamu.edu/~bangerth/videos.676.12.html>
- Demos in /scratch/smr2909/lab-04/

Why used in deal.II?

- Write your program once and run in 1d, 2d, 3d:

```
DoFHandler<dim>::active_cell_iterator  
    cell = dof_handler.begin_active(),  
endc = dof_handler.end();  
  
for ( ; cell!=endc; ++cell)  
{ ...  
  
    cell_matrix(i,j) += fe_values.shape_grad (i, q_point)  
                      * fe_values.shape_grad (j, q_point)  
                      * fe_values.JxW (q_point));
```

- `cell->face ()` is a quad in 3d but a line in 2d
- Also: large parts of the library independent of dimension:
 - `hyper_cube` (square vs box), etc.

Class Templates for Functions

- Blueprint for a function
- One type called “number”
- You can use
“typename” or “class”
- Sometimes you need to state which function
you want to call:

```
template <typename number>
number square (const number x)
{ return x*x; };
```

```
int x = 3;
int y = square<int>(x);
```

```
template <typename T>
void yell ()
{ T test; test.shout("HI!"); };
```

```
// cat is a class that has shout()
yell<cat>();
```

Value Templates

- Template arguments can also be values (like int) instead of types:

```
template <int dim>
void make_grid (Triangulation<dim> &triangulation)
{ ...}
```

```
Triangulation<2> tria;
make_grid<2>(tria);
```

- Of course this would have worked here too:

```
template <typename T>
void make_grid (T &triangulation)
{ ...// now we can not access "dim" though
```

Class templates

- Whole classes instead of functions built from a blueprint
- Same idea:

```
template <int dim>
class Point
{
    double elements[dim];
    // ...
}

Point<2> a_point;
Point<5> different_point;
```

```
namespace std
{
    template <typename number>
    class vector;
}

std::vector<int> list_of_ints;
std::vector<cat> cats;
```

Example

```
template <unsigned int N>
double norm (const Point<N> &p)
{
    double tmp = 0;
    for (unsigned int i=0; i<N; ++i)
        tmp += square(p.elements[i]);
    return sqrt(tmp);
}
```

- Value of N known at compile time, never stored!
- Compiler can optimize (unroll loop)
- Fixed size arrays faster than dynamic
(`dealii::Point<dim>` vs `dealii::Vector<double>`)

Examples in deal.II

- Step-4:

```
template <int dim>
void make_grid (Triangulation<dim> &triangulation) {...}
```

- So that we can use Vector<double> and Vector<float>:

```
template<typename number>
class Vector< number > { number [] elements; ...};
```

- Default values (embed dim-dimensional object in spacedim):

```
template<int dim, int spacedim=dim>
class Triangulation< dim, spacedim > { ... };
```

- Already familiar:

```
template<int dim, int spacedim>
void GridGenerator::hyper_cube (Triangulation< dim, spacedim > & tria,
const double left, const double right) {...}
```

Explicit Specialization

- different blueprint for a specific type T or value

```
// store some information
// about a Triangulation:
.

template <int dim>
struct NumberCache
{};

template <>
struct NumberCache<1>
{
    unsigned int n_levels;
    unsigned int n_lines;
};

template <>
struct NumberCache<2>
{
    unsigned int n_levels;
    unsigned int n_lines;
    unsigned int n_quads;
}

// more clever:
template <>
struct NumberCache<2>:
public NumberCache<1>
{
    unsigned int n_quads;
}
```

Lab 4 (step-4)

- Dimension independent Laplace problem
- Triangulation<2>, DoFHandler<2>, ...
replaced by

Triangulation<dim>, DoFHandler<dim>, ...

- Template class:

```
template <int dim>
```

```
class Step4 { ... };
```

Lab 5

- Modified step-4 to check correctness
- Using the method of manufactured solutions
- Computing L2 and H1 errors and check orders

Computing Errors

- Important for verification!
- See step-7 for an example
- Set up problem with analytical solution and implement it as a Function<dim>
- Quantities of interest:

$$e = u - u_h$$

$$\|e\|_0 = \|e\|_{L_2} = \left(\sum_K \|e\|_{0,K}^2 \right)^{1/2} \quad \|e\|_{0,K} = \left(\int_K |e|^2 \right)^{1/2}$$

$$|e|_1 = |e|_{H^1} = \|\nabla e\|_0 = \left(\sum_K \|\nabla e\|_{0,K}^2 \right)^{1/2}$$

$$\|e\|_1 = \|e\|_{H^1} = \left(|e|_1^2 + \|e\|_0^2 \right)^{1/2} = \left(\sum_K \|e\|_{1,K}^2 \right)^{1/2}$$

- Break it down as one operation per cell and the “summation” (local and global error)
- Need quadrature to compute integrals

Computing Errors

- Example:

```
Vector<float> difference_per_cell (triangulation.n_active_cells());  
  
VectorTools::integrate_difference (dof_handler,  
                                  solution, // solution vector  
                                  Solution<dim>(), // reference solution  
                                  difference_per_cell,  
                                  QGauss<dim>(3), // quadrature  
                                  VectorTools::L2_norm); // local norm  
  
const double L2_error = difference_per_cell.l2_norm(); // global norm
```

- Local norms:

mean, L1_norm, L2_norm, Linfty_norm, H1_seminorm, H1_norm, ...

- Global norms are vector norms: l1_norm(), l2_norm(), linfty_norm(), ...

Lab 6

- Higher order mappings, see step-10/step-11
- Start with lab-6. Find a solution so that higher order mapping gives correct convergence order!

More features of deal.II

- Adaptive mesh refinement, including dual-weighted error estimators (step-6, step-7)
- Linear solvers: direct solvers, iterative solvers, preconditioners, multigrid, etc.
- Parallel computing: multithreading (step-9), MPI (step-40)
- Vector-valued problems: step-8, step-20, ...
- Other FE spaces: DG, RT, Nedelec, ...
-

More features of deal.II b)

- Non-homogeneous Neumann conditions, boundary integrals: step-7
- Systems of PDEs: step-8, step-20
- Nonlinear problems: step-15
- Time dependent problems: step-18, ...
- Fluid flow (Stokes, Navier-Stokes, ...): step-22, step-32,
...
- Complicated meshes: step-49
- Matrix-free computations: step-37, step-48

Resources

- Tutorials (by topic, graph, etc.):
<https://www.dealii.org/developer/doxygen/deal.II/Tutorial.html>
- Video lectures:
<http://www.math.tamu.edu/~bangerth/videos.html>
- Manual: <https://www.dealii.org/developer/doxygen/deal.II/index.html>
- Mailing list: <https://groups.google.com/forum/#!forum/dealii>
-

Fixing lab 6

- Yesterday we saw optimal 3rd order convergence of the L2 norm on the circle even with a linear mapping
- It turns out we applied the correct boundary conditions for Ω_h and only computed the error on Ω_h . In essence, we were solving on a piece-wise linear Ω on each refinement level.
- Take a look at lab06.pdf...

Lab 7: Adaptive Mesh Refinement

- Typical loop:
 - Solve
 - Estimate
 - Mark
 - Refine/coarsen
- Estimate is problem dependent:
 - Approximate gradient jumps: `KellyErrorEstimator` class
 - Approximate local norm of gradient: `DerivativeApproximation` class
 - Or something else
- Mark:

```
GridRefinement::refine_and_coarsen_fixed_number(...)  
GridRefinement::refine_and_coarsen_fixed_fraction(...)
```
- Refine/coarsen:
 - `triangulation.execute_coarsening_and_refinement ()`
 - Transferring the solution: `SolutionTransfer` class (maybe discussed later)

Constraints

- Used for hanging nodes (and other things!)
- Have the form:

$$x_i = \sum_j \alpha_{ij} x_j + c_j$$

- Represented by class ConstraintMatrix
- Created using `DoFTools::make_hanging_node_constraints()`
- Will also use for boundary values from now on:

```
VectorTools::interpolate_boundary_values(...,  
constraints);
```

- Need different SparsityPattern (see step-6):

```
DoFTools::make_sparsity_pattern (... , constraints, ...)
```

Constraints II

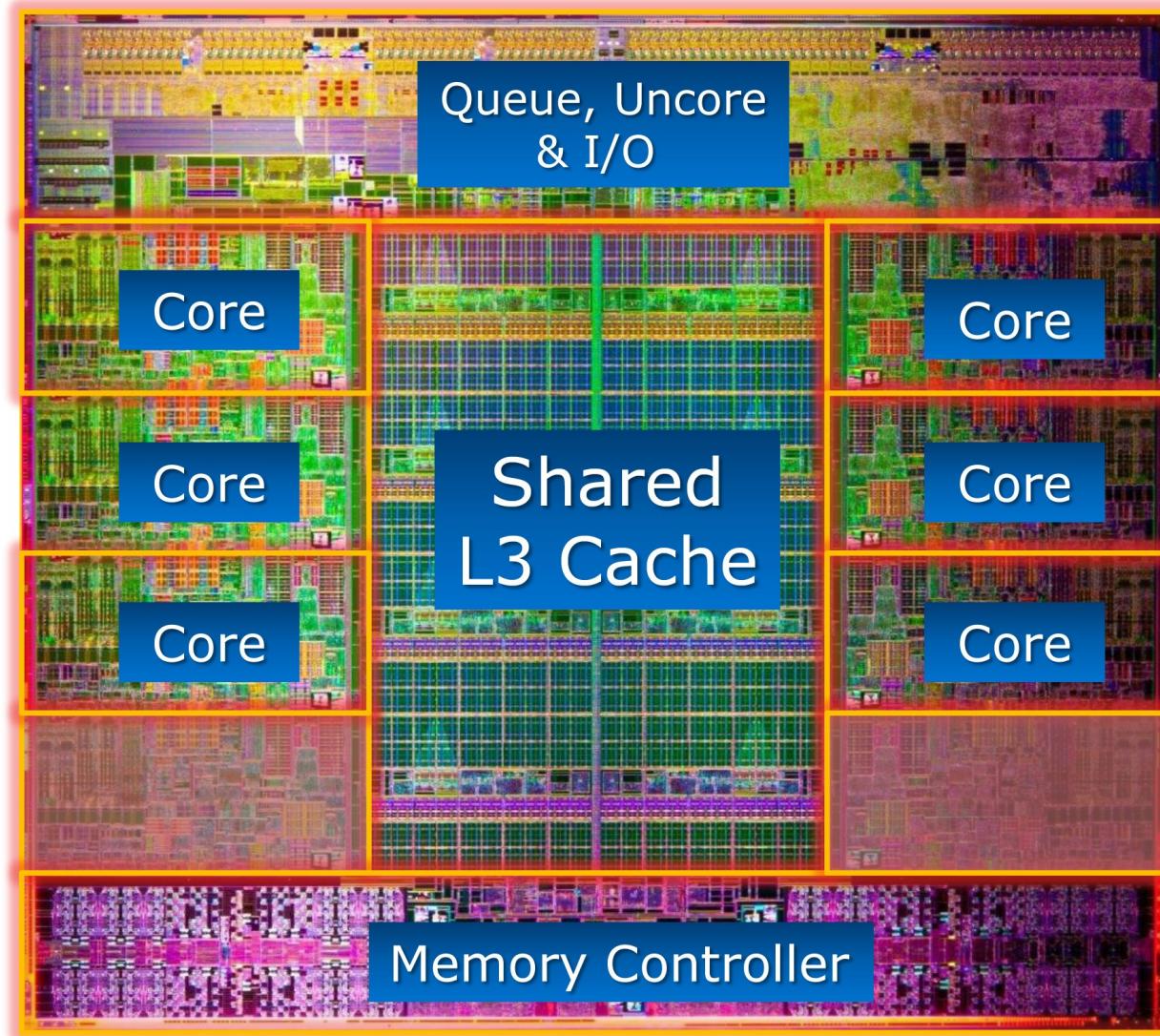
- Old approach (explained in video):
 - Assemble global matrix
 - Then eliminate rows/columns: `ConstraintMatrix::condense(...)`
(similar to `MatrixTools::apply_boundary_values()` in step-3)
 - Solve and then set all constraint values correctly: `ConstraintMatrix::distribute(...)`
- New approach (step-6):
 - Assemble local matrix as normal
 - Eliminate while transferring to global matrix:

```
constraints.distribute_local_to_global (cell_matrix, cell_rhs,
                                         local_dof_indices,
                                         system_matrix, system_rhs);
```
 - Solve and then set all constraint values correctly: `ConstraintMatrix::distribute(...)`

Vector Valued Problems

- (video 19&20)
- FESystem: list of FEs (can be nested!)
- Will give one FE with N shape functions
- Use FEValuesExtractors to do
`fe_values[velocities].divergence (i, q), ...`
- Ordering of DoFs in system matrix is independent
- See module “handling vector valued problems”
- Non-primitive elements (see `fe.is_primitive()`):
shape functions have more than one non-zero component, example:
RT

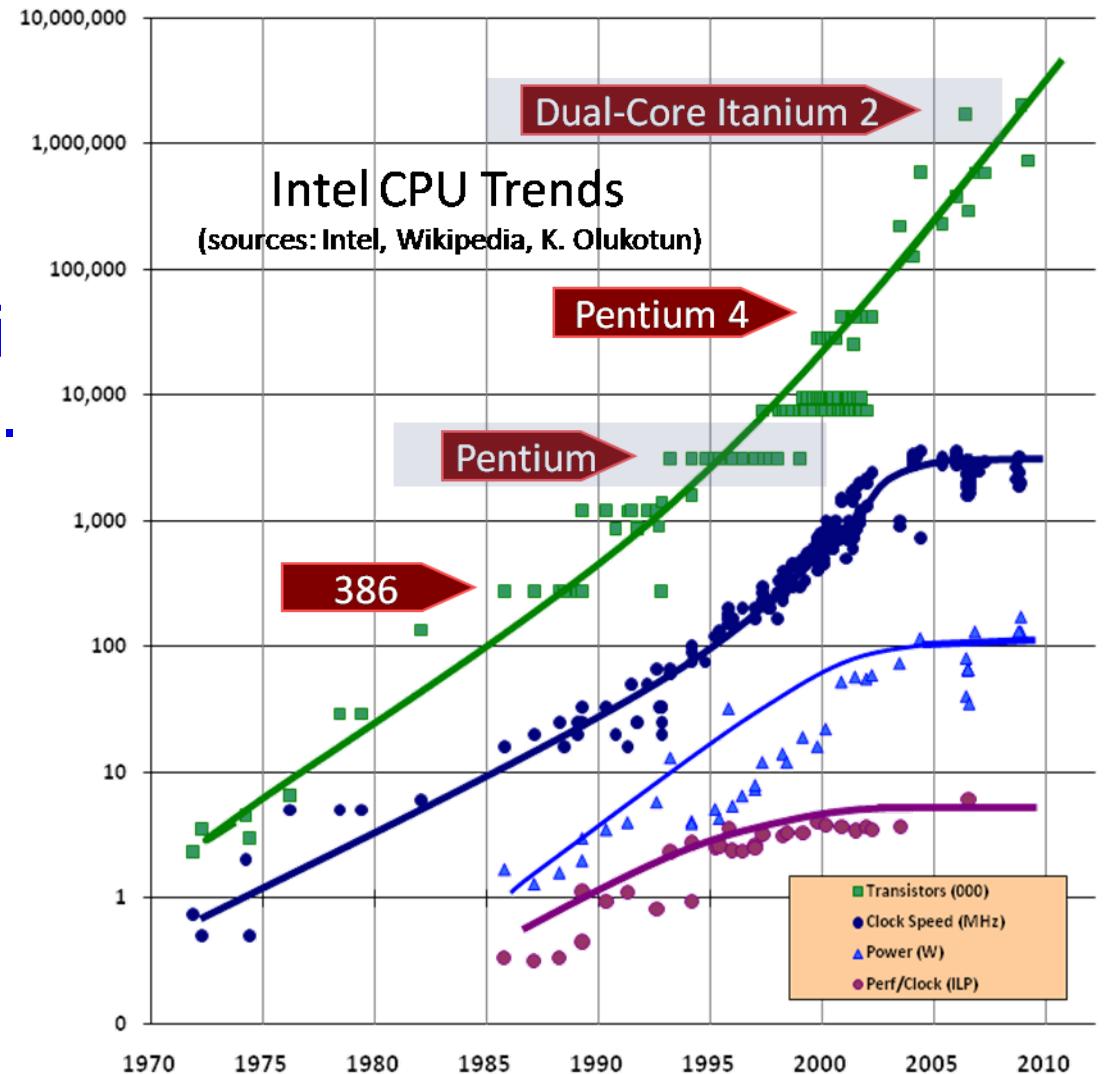
Parallel Computing: Introduction



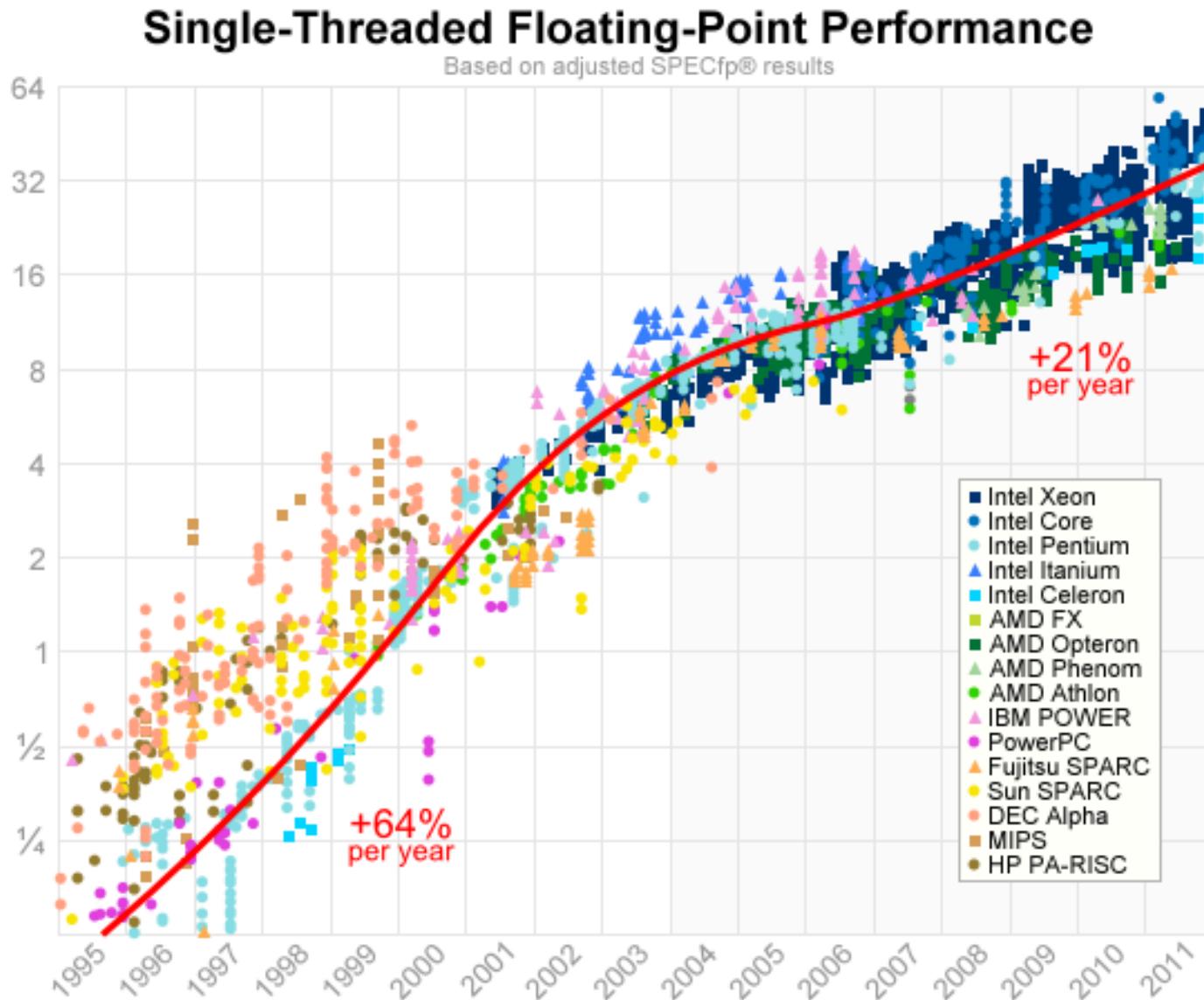
A modern CPU: Intel Core i7

Basics

- Single cores are not getting (much) faster
- “the free lunch is over”: <http://www.gotw.ca/publications/concurrency-ddj.htm>
- Concurrency is only option:
 - SIMD/vector instructions
 - Several cores
 - Several chips in one node
 - Combine nodes into supercomputer

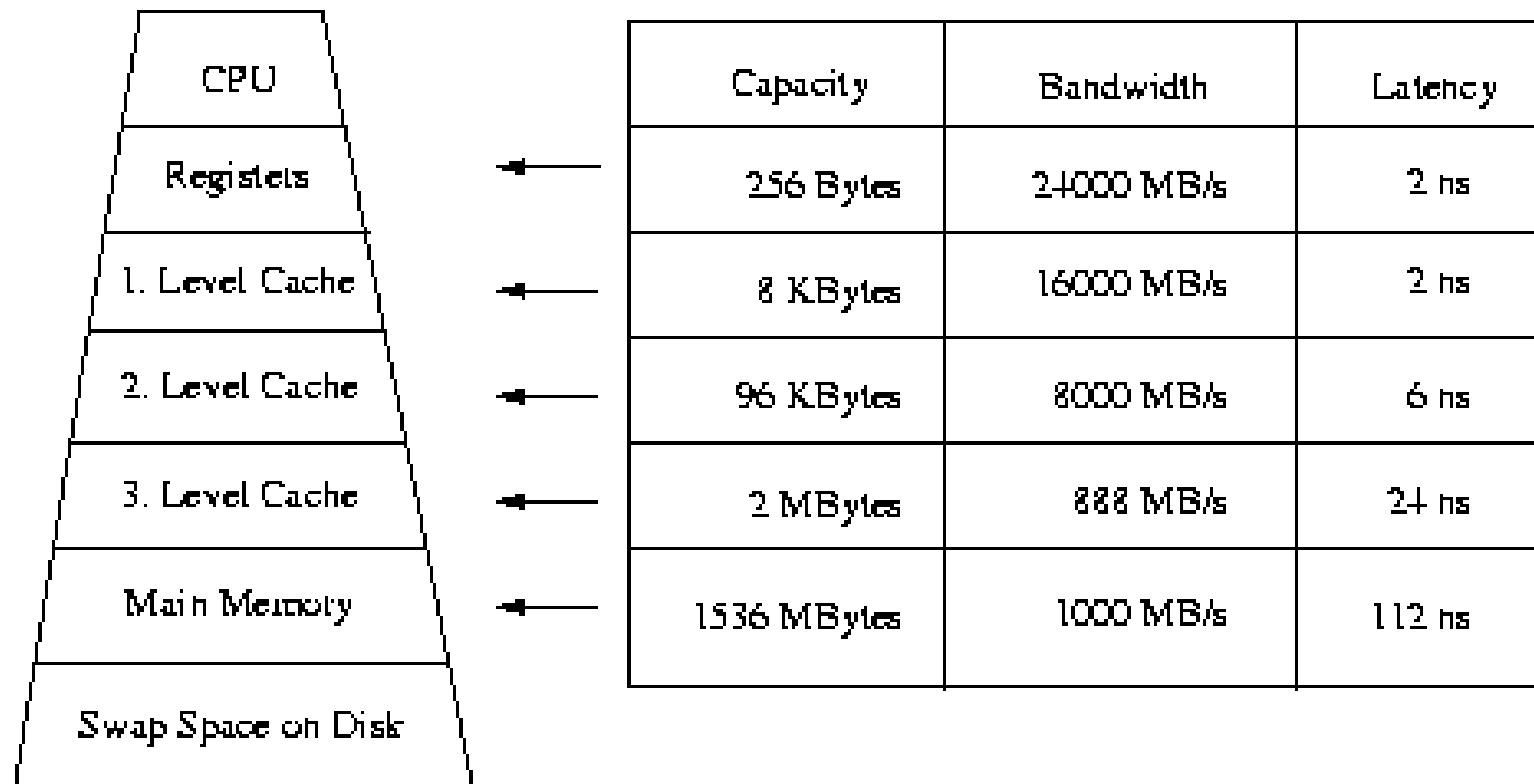


Single-thread performance increasing slowly

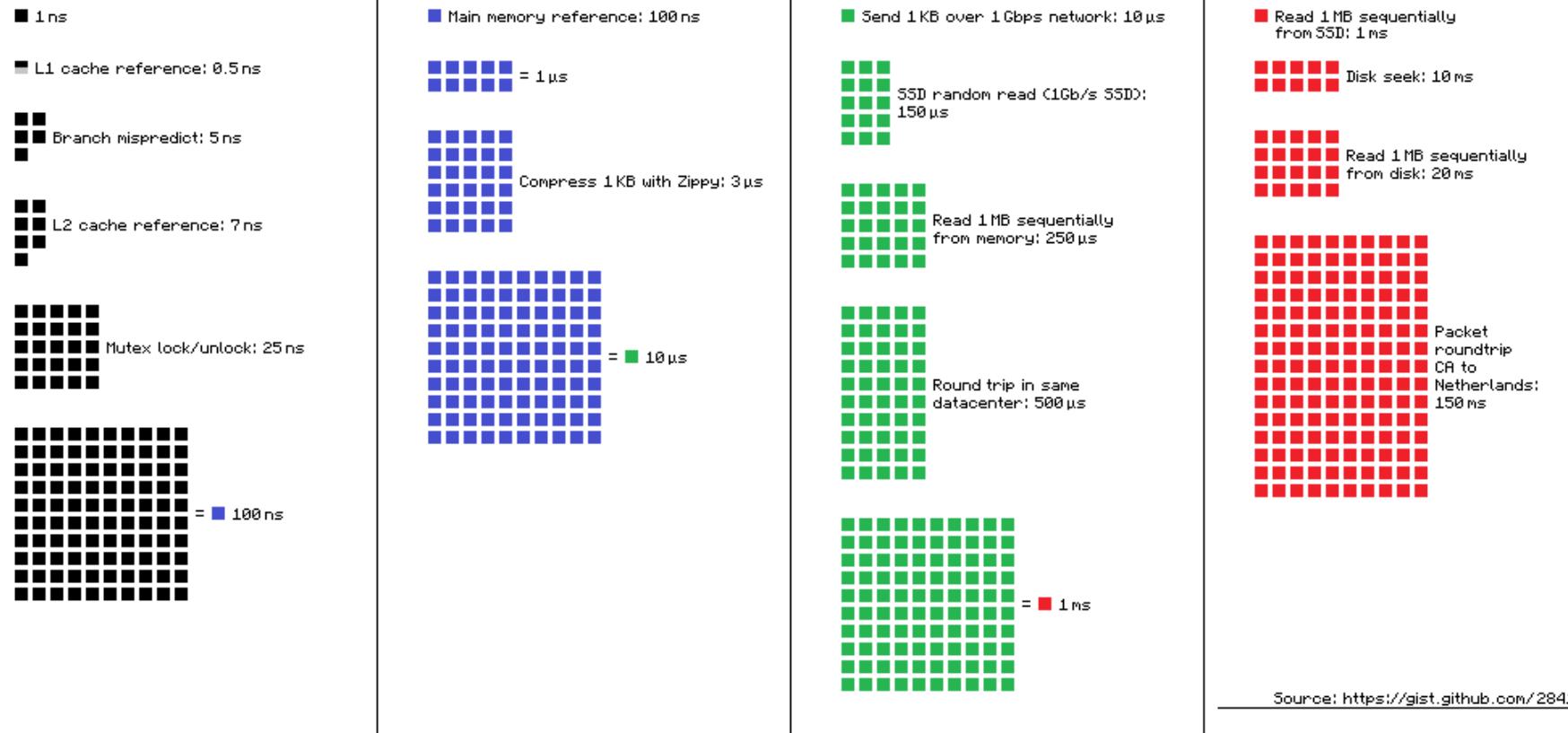


Hierarchy of memory

- Latency: time CPU gets data after requesting
- Bandwidth: how much data per second?
- prefetching of data, “cache misses” are expensive
- automatically managed by processor



Latency Numbers Every Programmer Should Know



Source: <https://gist.github.com/2841832>

<https://gist.github.com/hellerbarde/2843375>

Amdahl's Law

- Task: serial fraction s , parallel fraction $p=1-s$
- N workers (whatever that means)
- Runtime: $T(N) = (1-s)T(1)/N + sT(1)$
- Speedup $T(1)/T(N)$, N to infinity:
 $\text{max_speedup} = 1/s$
- http://en.wikipedia.org/wiki/Amdahl%27s_law
- Reality: $T(N) = (1-s)T(1)/N + sT(1) + aN + bN^2$

Summary

- Computing much faster than memory access
- Parallel computing required: no free lunch!
- Communication is serial fraction (or worse when increasing with $N!$)
- Communication in Amdahl's law is main challenge in parallel computing