Sparse Matrix Algorithms and Advanced Topics in FEM
MATH 9830

Timo Heister
(heister@clemson.edu)

http://www.math.clemson.edu/~heister/math9830-spring2015/

Tasks Lab 1

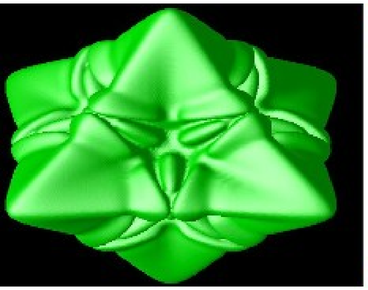
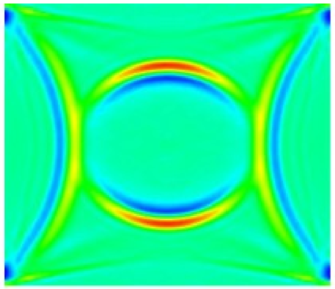
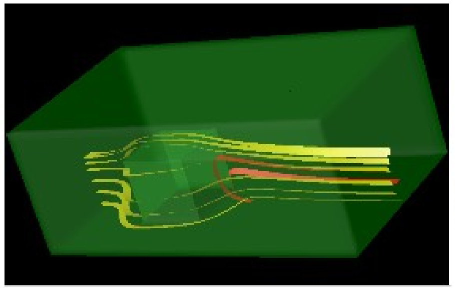
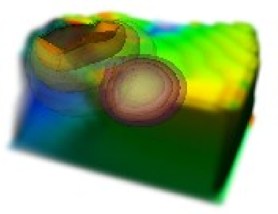
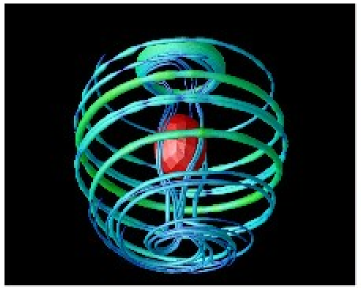
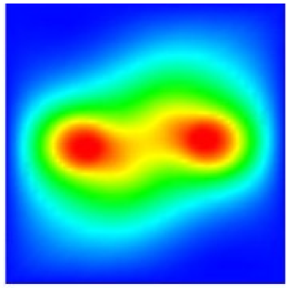
- Grab class repo
  - Git clone https://github.com/tjhei/spring15_9830
  - Compile example: g++ main.cc
  - Run: ./a.out
  - Do question 2 from homework 1
  - Work on the other tasks listed in the cc
- Install deal.II
- Run deal.II step 2
  - Print sparsity pattern, change refinement
  - Use cuthill_mckee, other ordering?
  - Change to dim=3, etc.

Tasks Lab 2

- Deal.II intro
- Work on hw2
- Bonus: step 2
  - Print sparsity pattern, change refinement
  - Use cuthill_mckee, other ordering?
  - Change to dim=3, etc.

deal.II

- “A Finite Element **D**ifferential **E**quations **A**nalysis **L**ibrary”
- Open source, c++ library
- I am one of the four maintainers
- One of the most widely used libraries:
  - ~600 papers using and citing deal.II
  - ~600 downloads/month
  - 100+ people have contributed in the past 10 years
  - ~600,000 lines of code
  - 10,000+ pages of documentation
- Website: www.dealii.org

Features

- 1d, 2d, 3d computations, adaptive mesh refinement (on quads/hexas only)
- Finite element types:
  - Continuous and DG Lagrangian elements
  - Higher order elements, hp adaptivity
  - Raviart-Thomas, Nedelec, …
  - And arbitrary combinations

Features, part II

- Linear Algebra
  - Own sparse and dense library
  - Interfaces to PETSc, Trilinos, UMFPACK, BLAS, ..
- Parallelization
  - Multi-threading on multi-core machines
  - MPI: 16,000+ processors
- Output in many visualization file formats

Development of deal.II

- Professional-level development style
- Development in the open, open repository
- Mailing lists for users and developers
- Test suite with 6,000+ tests after every change
- Platform support:
  - Linux/Unix
  - Mac
  - Work in progress: Windows

Installation

- How to install from source code, configure, compile, test, run “step-1”
- Ubuntu (or any other linux) or Mac OSX
- Steps:
  - Detect compilers/dependencies/etc. (cmake)
  - Compile & install deal.II (make)

Prerequisites on Linux

- Compiler: GNU g++
- Recommended:

```
$ sudo apt-get install subversion openmpi1.6-bin
openmpi1.6-common g++ gfortran libopenblas-dev
liblapack-dev zlib1g-dev git emacs gnuplot
```

- manually: cmake (in a minute)
- Later: eclipse, paraview
- Optional manually: visit, p4est, PETSc, Trilinos, hdf5

On Mac OS

- If OSX 10.9 follow instructions from wiki: https://github.com/dealii/dealii/wiki/MacOSX
- Later manually: eclipse, paraview
-

cmake

- Ubuntu 12.04 has a version that is too old
- If newer ubuntu do:

```
$ sudo apt-get install cmake
```

  … and you are done

- Otherwise: install cmake from source or download the 32bit binary

cmake from binary

- Do:

```
export CMAKEVER=2.8.12.1
wget http://www.cmake.org/files/v2.8/cmake-$CMAKEVER-Linux-i386.sh
chmod u+x cmake-$CMAKEVER-Linux-i386.sh
./cmake-$CMAKEVER-Linux-i386.sh
```

- Answer “q”, yes and yes
- Add the bin directory to your path (.bashrc)
- You might need

```
    sudo apt-get install ia32-libs
```

Cmake from source

```
wget ¬ http://www.cmake.org/files/v2.8/cmake-2.8.12.1.tar.gz

tar xf cmake-2.8.11.1.tar.gz

./configure

make install
```

Install deal.II

- http://www.dealii.org/8.1.0/readme.html
- Extract:

  `tar xf deal.II-8.1.0.tar.gz`
- Build directory:

  `cd deal.II; mkdir build; cd build`
- Configuration:

  `cmake -D CMAKE_INSTALL_PREFIX=/?/? ..`

  (where /?/? is your installation directory)
- Compile (5-60 minutes):

  `make -j X install`

  (where X is the number of cores you have)
- Test:

  `make test` (in build directory)

- Test part two:

  ```
  cd examples/step-1
  cmake -D DEAL_II_DIR=/?/? .
  make run
  ```
- Recommended layout:

  `deal.II/`

`build`	< build files
`installed`	< your inst. dir
`examples`	< all examples!
`include`	
`source`	
`...`	

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:
  - 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
```

How to create an eclipse project

- Run this once in your project:

  cmake -G "Eclipse CDT4 - Unix Makefiles" .

- Now create a new project in eclipse (“file->import->existing project” and select your folder for the project above)
- Eclipse intro:
  - http://www.math.tamu.edu/~bangerth/videos.676.7.html
  - http://www.math.tamu.edu/~bangerth/videos.676.8.html

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
-

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));
```

- Also: large parts of the library independent of dimension

Function Templates

- Blueprint for a function
- One type called “number”
- You can use “typename” or “class”

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

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

- Sometimes you need to state which function you want to call:

```
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 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(v.elements[i]);
 return sqrt(tmp);
};
```

- Value of N known at compile time
- 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;
}
```

step-4

- Dimension independent Laplace problem
- Triangulation<2>, DoFHandler<2>, …

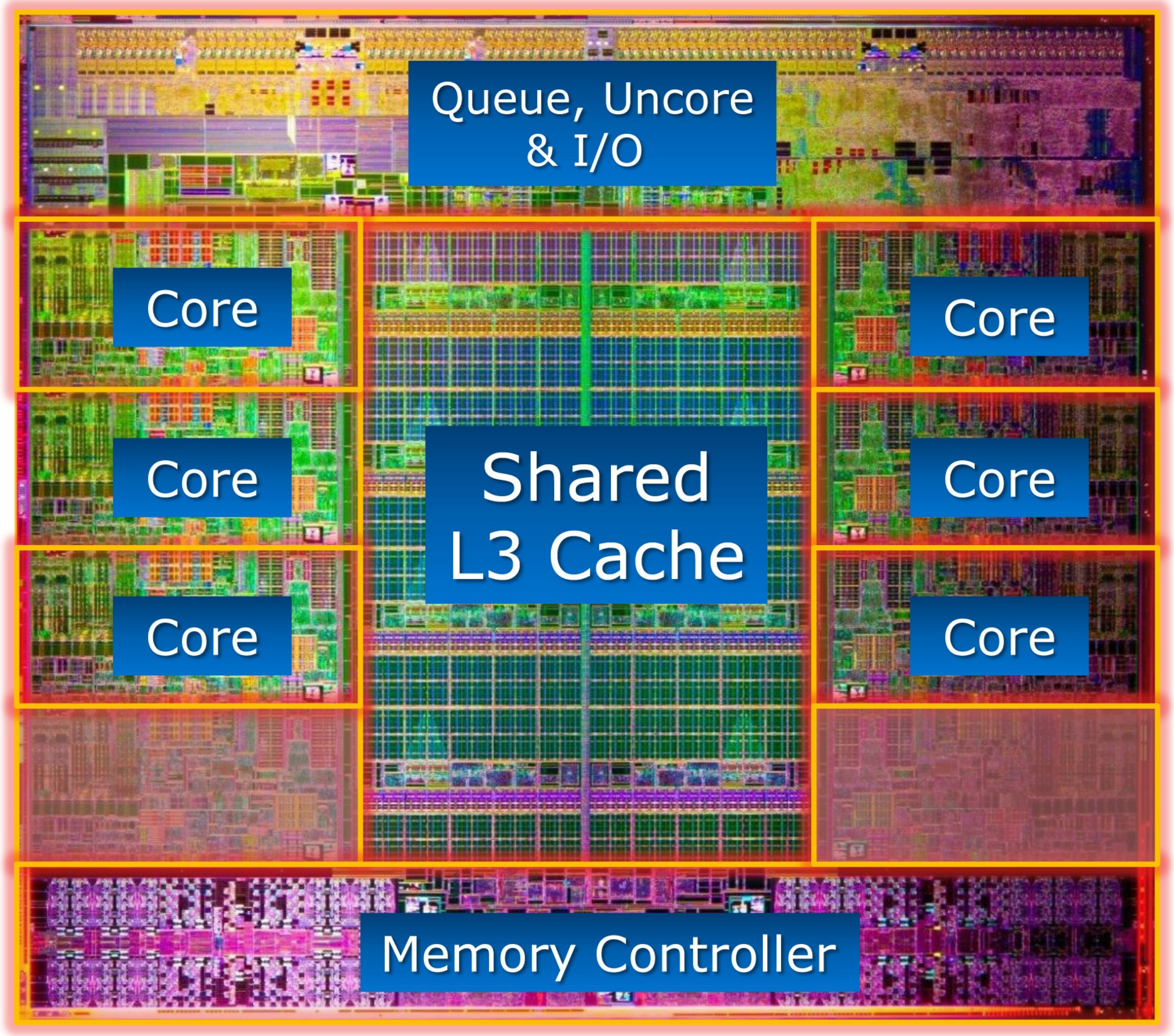
  replaced by

  Triangulation<dim>, DoFHandler<dim>, …
- Template class:

```
template <int dim>

class Step4 {};
```

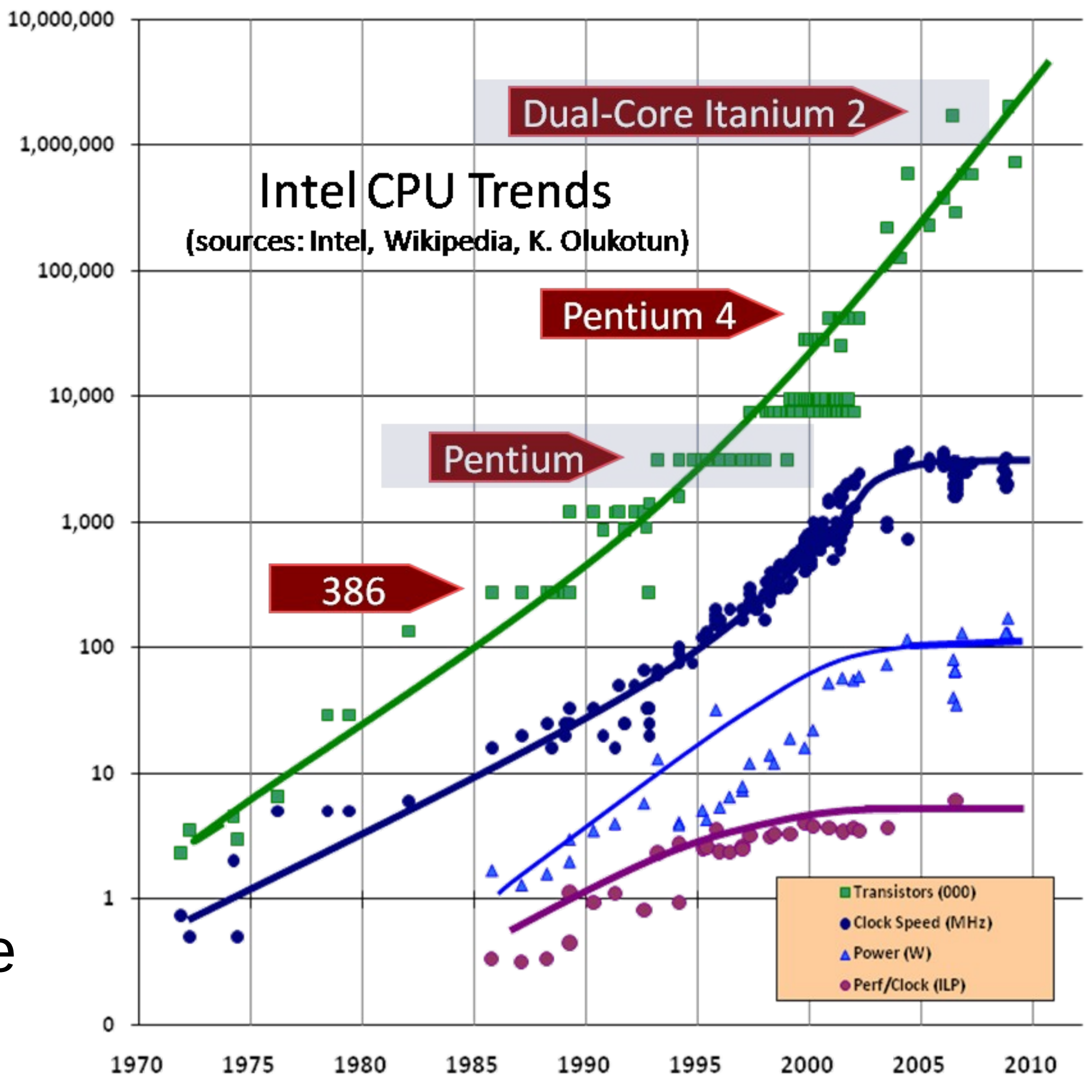
Parallel Computing: Introduction



A modern CPU: Intel Core i7

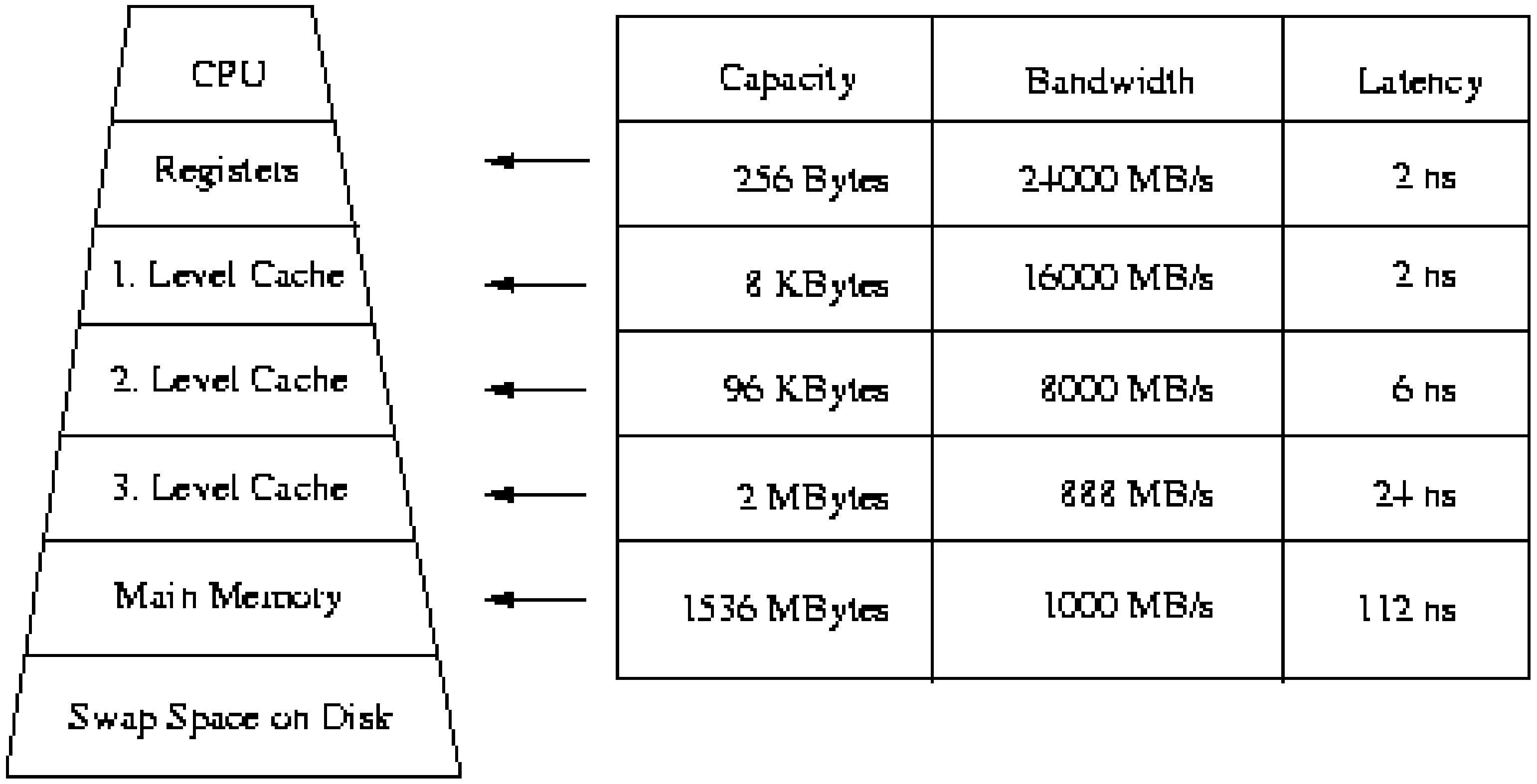
Basics

- 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

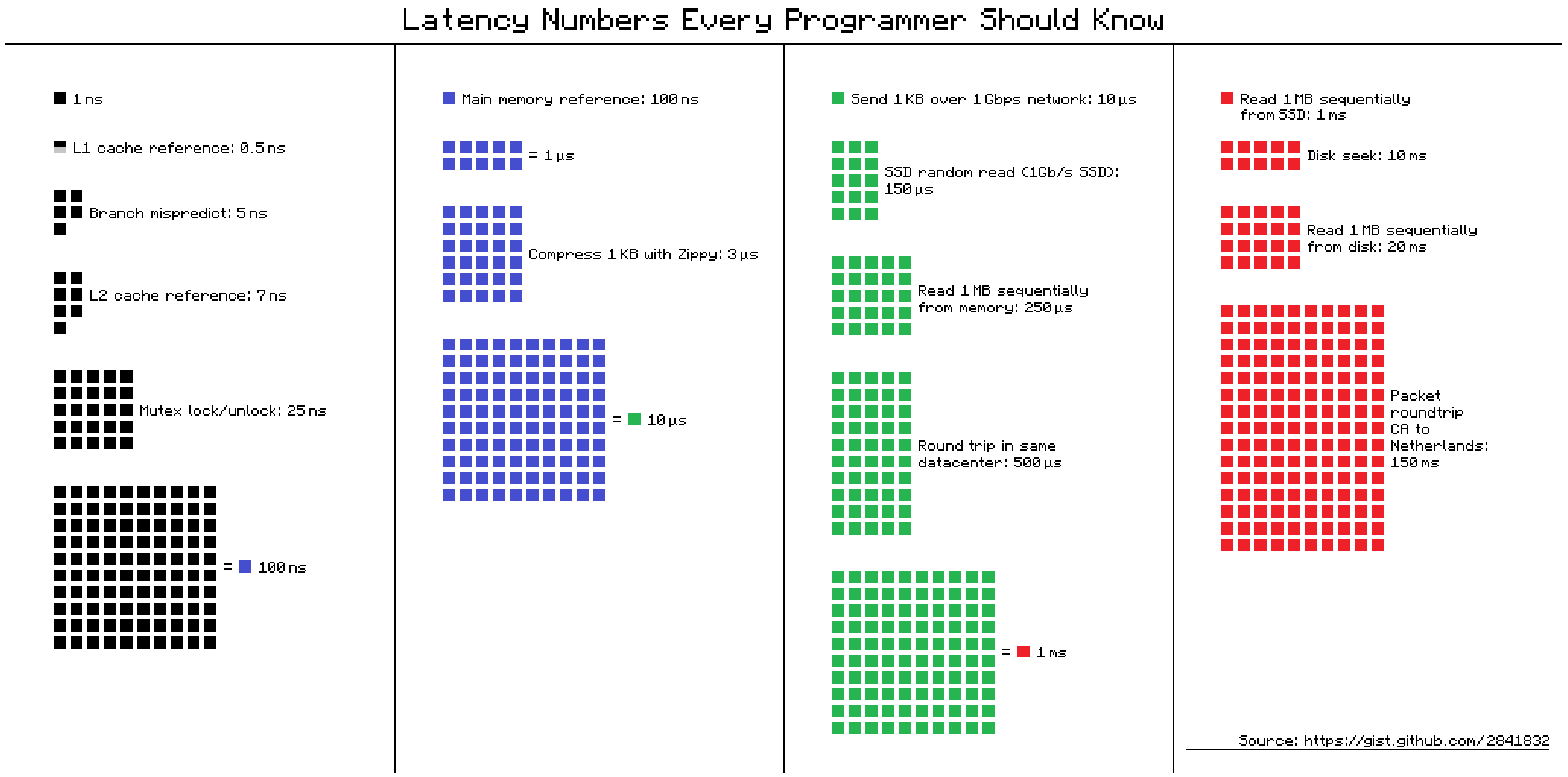


Hierarchy of memory

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



Capacity	Bandwidth	Latency
256 Bytes	24000 MB/s	2 ns
8 KBytes	16000 MB/s	2 ns
96 KBytes	8000 MB/s	6 ns
2 MBytes	888 MB/s	24 ns
1536 MBytes	1000 MB/s	112 ns



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:

  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

Multithreading

- Idea: call functions in separate background thread and continue immediately
- All threads can access the same memory (dangerous, but easy)
- Can spawn arbitrary many threads, scheduled by operating system (pthreads, CreateThread, …)
- Wrapper for c++: std::thread, boost::thread
  - See demos
- Higher level libraries (later):
  - OpenMP (parallelize loops, tasks, ...)
  - TBB = Intel Threading Building Blocks (tasks, algorithms)
  - deal.II wraps TBB in a very easy task based interface

Multithreading limits

- One machine only (no cluster) => MPI
- Wrong abstraction:
  - Creating threads is expensive
  - How many to create?
  - How to split work?
  - Reuse them?
  - Synchronization difficult
- Better: task based library based on threads (later)

MPI

- Need MPI library and compile with compiler wrappers (mpicxx instead of g++) or use a cmake script that does that
- Need to run with (for 4 processes):

```
mpirun -n 4 ./main
```

- Reference http://mpi.deino.net/mpi_functions/
- Most basic program:

```
#include <mpi.h>

int main(int argc, char **argv)
{
  MPI_Init(&argc, &argv);

  int rank, size;
  MPI_Comm_rank(MPI_COMM_WORLD, &rank);
  MPI_Comm_size(MPI_COMM_WORLD, &size);

  MPI_Finalize();
}
```

MPI_Send and MPI_Recv

- Send and receive a message from <source> to <dest> of <count> elements
- <comm> is always MPI_COMM_WORLD for us
- <tag> can be any integer buts needs to be the same
- <status> can be MPI_STATUS_IGNORE if you don't care
- Some examples for <datatype>: MPI_INT, MPI_DOUBLE
- <source> can be MPI_ANY_SOURCE if you want to receive any message

```
int MPI_Send(
  void *buf,
  int count,
  MPI_Datatype datatype,
  int dest,
  int tag,
  MPI_Comm comm);
```

```
int MPI_Recv(
  void *buf,
  int count,
  MPI_Datatype datatype,
  int source,
  int tag,
  MPI_Comm comm,
  MPI_Status *status);
```

MPI_Barrier

- Code:

```
MPI_Barrier(MPI_Comm comm)
```

- Waits until all ranks arrived at this line, then all continue

MPI_Probe

- Wait until a matching message arrives an return info about it in <status>

```
int MPI_Probe(
  int source,
  int tag,
  MPI_Comm comm,
  MPI_Status *status
);
```

- <source> source rank or MPI_ANY_SOURCE
- <tag> tag value or MPI_ANY_TAG
- <status> contains:
  - .MPI_SOURCE the source rank
  - .MPI_TAG the tag
- You can get information about the size of the message using MPI_Get_count()

MPI_Bcast

- Send the same data from <root> to all
- Result: `rank[j].buffer[i]=rank[root].buffer[i]`

```
int MPI_Bcast(
  void *buffer,
  int count,
  MPI_Datatype datatype,
  int root,
  MPI_Comm comm
)
```

MPI_Reduce

- Combine elements from <sendbuf> using <op> from all ranks and store in <recvbuf> on <root>
- MPI_OP examples: MPI_SUM, MPI_MIN, MPI_MAX, ...
- Result:

```
rank[root].recvbuf[i]=op(rank[0].sendbuf[i], …, rank[n-1].sendbuf[i])
```

```
int MPI_Reduce(
  void *sendbuf,
  void *recvbuf,
  int count,
  MPI_Datatype datatype,
  MPI_Op op,
  int root,
  MPI_Comm comm
);
```

MPI_AllReduce

- Combine elements from <sendbuf> using <op> from all ranks and store in <recvbuf> on all ranks
- Like reduce, but result is available everywhere:

  rank[j].recvbuf[i]=op(rank[0].sendbuf[i], …, rank[n-1].sendbuf[i])

```
int MPI_Allreduce(
  void *sendbuf,
  void *recvbuf,
  int count,
  MPI_Datatype datatype,
  MPI_Op op,
  MPI_Comm comm
);
```

MPI_Gather

- Collect data from all ranks at <root>

```
int MPI_Gather(
  void *sendbuf,
  int sendcnt,
  MPI_Datatype sendtype,
  void *recvbuf,
  int recvcnt,
  MPI_Datatype recvtype,
  int root,
  MPI_Comm comm
);
```

MPI_AllGather

- Collect data from all ranks at every rank
- (like Gather but copied to everyone)

```
int MPI_Allgather(
  void *sendbuf,
  int sendcount,
  MPI_Datatype sendtype,
  void *recvbuf,
  int recvcount,
  MPI_Datatype recvtype,
  MPI_Comm comm
);
```

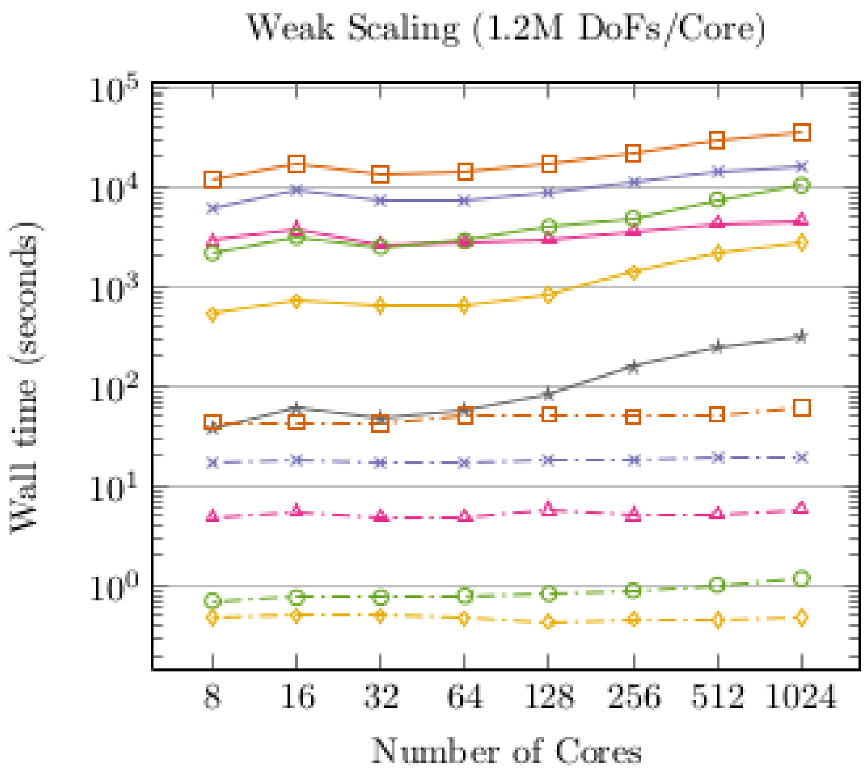
MPI_Scatter

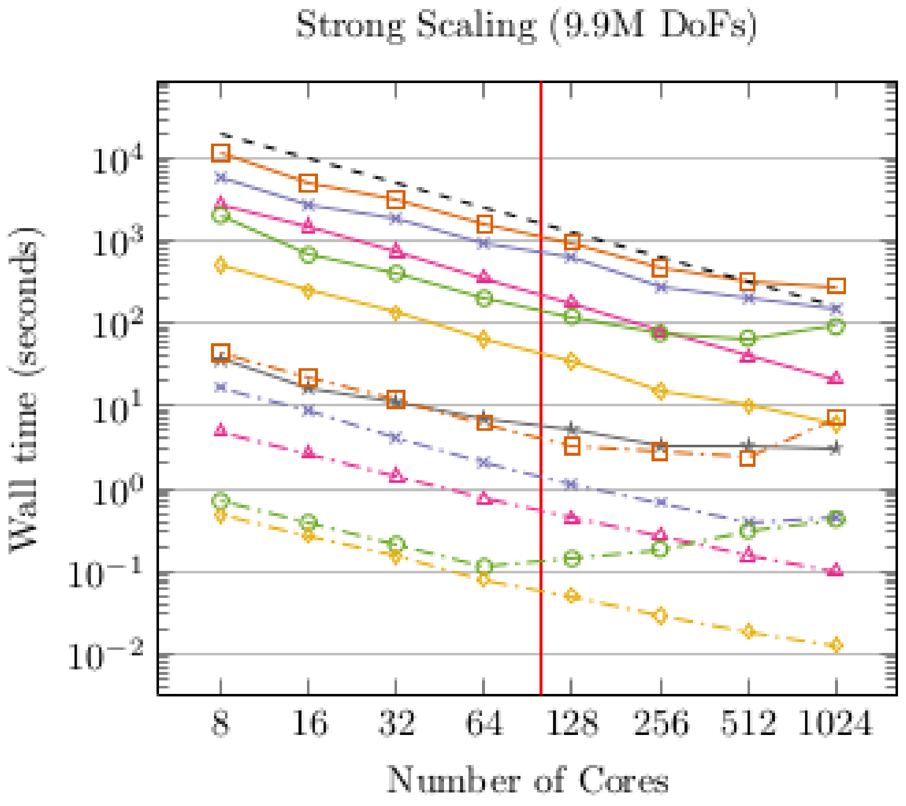
- Send different data from <root> to all
- <sendbuf> is only used on <root>

```
int MPI_Scatter(
  void *sendbuf,
  int sendcnt,
  MPI_Datatype sendtype,
  void *recvbuf,
  int recvcnt,
  MPI_Datatype recvtype,
  int root,
  MPI_Comm comm
);
```

Weak/Strong Scaling

- Weak:
  - Fixed problem size per CPU
  - Increase CPUs
  - Optimal: constant time
- Strong:
  - Fixed total problem size
  - Increase CPUs
  - Optimal: linear decrease





The following slides are for the future....

MPI vs. Multithreading

Boundary conditions

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(...)` or

  `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 Values 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

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 or interest:

$$e = u - u_h$$

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

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

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

- 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(), ...`

ParameterHandler

- Control program at runtime without recompilation
- You can put in:

  ints (e.g. number of refinements), doubles (e.g. coefficients, time step size), strings (e.g. choice for algorithm/mesh/problem/etc.), functions (e.g. right-hand side, reference solution)

- Stuff can be grouped in sections
- See class-repository: prm/

ParameterHandler

```
# order of the finite element to use.
set fe order   = 1

# Refinement method. Choice between 'global' and 'adaptive'.
set refinement = global

subsection equation
  # expression for the reference solution and boundary values. Function of x,y (and z)
  set reference = sin(pi*x)*cos(pi*y)

  # expression for the gradient of the reference solution. Function of x,y (and z)
  set gradient  = pi*cos(pi*x)*cos(pi*y); -pi*sin(pi*x)*sin(pi*y)

  # expression for the right-hand side. Function of x,y (and z)
  set rhs       = 2*pi*pi*sin(pi*x)*cos(pi*y) + sin(pi*x)*cos(pi*y)
end
```