What is Hardware-Oriented Numerics?

Matthias Möller

Numerical Analysis

DIAM lunch colloquium, November 16, 2016

Overview

1. From Numerical Analysis to Hardware-Oriented Numerics
2. HWON example: mixed-precision methods
3. HWON application: simulation of flow problems

Numerical Analysis: Past, Present, and Future(?)

Given a *problem* $p \in \mathcal{P}$:

1. Find a *method* $m \in \mathcal{M}$ that solves problem p

2. Find an *algorithm* $a \in \mathcal{A}$ that realizes method m

QoI: errors, rate of convergence, FLOP, stability, monotonicity, ...

Numerical Analysis: Past, Present, and Future(?)

Given a *problem* $p \in \mathcal{P}$:

1. Find a *method* $m \in \mathcal{M}$ that solves problem p
2. Find an *algorithm* $a \in \mathcal{A}$ that realizes method m

QoI: errors, rate of convergence, FLOP, stability, monotonicity, ...

Given a *hardware* $h \in \mathcal{H}$:

3. Find an *implementation* $i \in \mathcal{I}$ that realizes algorithm a

QoI: FLOPS, memory bandwidth, parallel speed-up, ...

Numerical Analysis: Past, Present, and Future(?)

Given a *problem* $p \in \mathcal{P}$: (I)BVP

1. Find a *method* $m \in \mathcal{M}$ that solves problem p
   continuous Galerkin P_1-FEM
2. Find an *algorithm* $a \in \mathcal{A}$ that realizes method m
   matrix-free Krylov solver with element-wise Gaussian quadrature

QoI: errors, rate of convergence, FLOP, stability, monotonicity, ...

Given a *hardware* $h \in \mathcal{H}$:

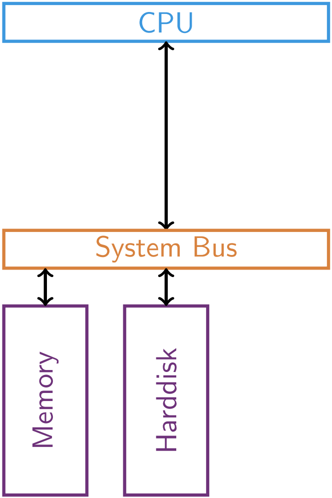
3. Find an *implementation* $i \in \mathcal{I}$ that realizes algorithm a
   OpenMP parallelized SHMEM C++ code using Eigen library

QoI: FLOP<u>S</u>, memory bandwidth, parallel speed-up, ...

Proposition 1

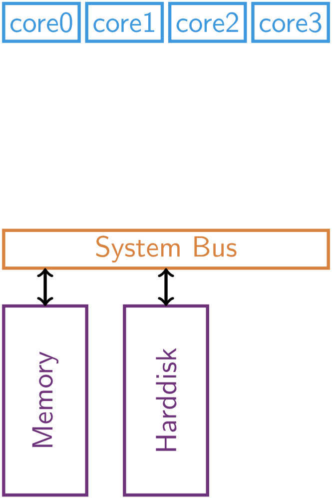
The only quality measure of a numerical algorithm and its implementation that matters in practical applications is the **wall-clock time** (and possibly the amount of memory) required to solve a problem $p \in \mathcal{P}$ to a prescribed accuracy **on a concrete hardware**.

Hardware in practice: your laptop/desktop computer



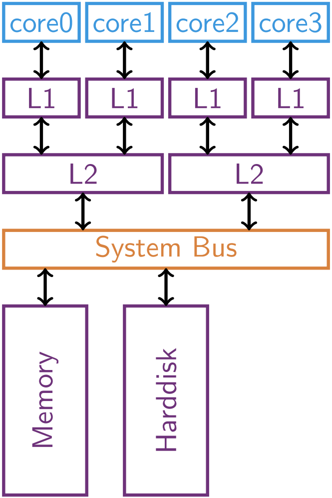
Hardware in practice: your laptop/desktop computer

- multi-core CPU
  - parallel algorithms
  - vectorized algorithms



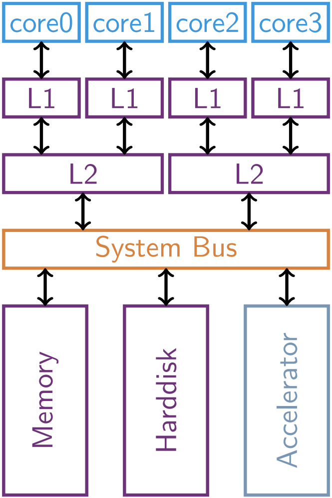
Hardware in practice: your laptop/desktop computer

- multi-core CPU
  - parallel algorithms
  - vectorized algorithms
- memory hierarchy
  - cache-oblivious algorithms
  - latency hiding algorithms



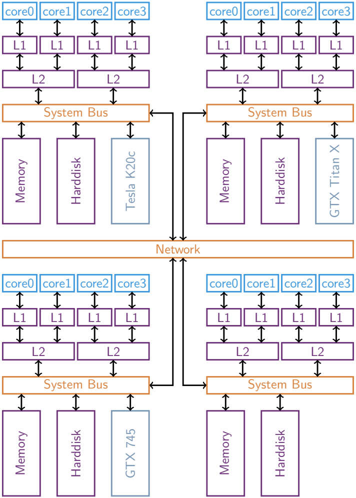
Hardware in practice: your laptop/desktop computer

- multi-core CPU
  - parallel algorithms
  - vectorized algorithms
- memory hierarchy
  - cache-oblivious algorithms
  - latency hiding algorithms
- many-core accelerator (GPU)
  - algorithms for heterogeneous architectures (off-loading)



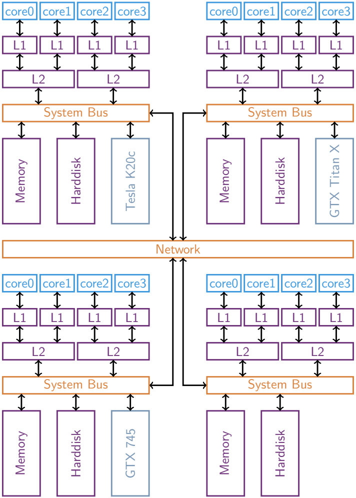
Hardware in practice: DIAM cluster

- multi-core CPU
  - parallel algorithms
  - vectorized algorithms
- memory hierarchy
  - cache-oblivious algorithms
  - latency hiding algorithms
- many-core accelerators
  - algorithms for heterogeneous architectures (off-loading)
- network-connected devices
  - distributed algorithms for even more heterogeneous systems
  - asynchronous algorithms
  - fault-tolerant algorithms



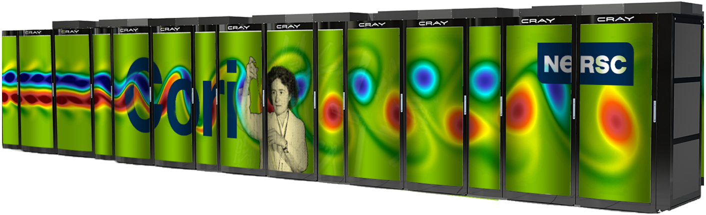
Hardware in practice: DIAM cluster

- multi-core CPU
  - parallel algorithms
  - vectorized algorithms
- memory hierarchy
  - cache-oblivious algorithms
  - latency hiding algorithms
- many-core accelerators
  - algorithms for heterogeneous architectures (off-loading)
- network-connected devices
  - distributed algorithms for even more heterogeneous systems
  - asynchronous algorithms
  - fault-tolerant algorithms



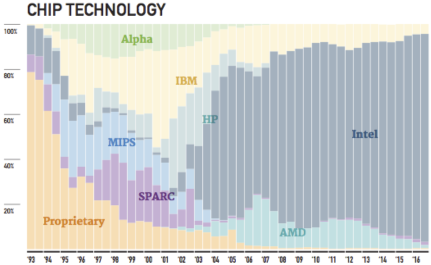
Hardware in practice: Top500 from November 2016

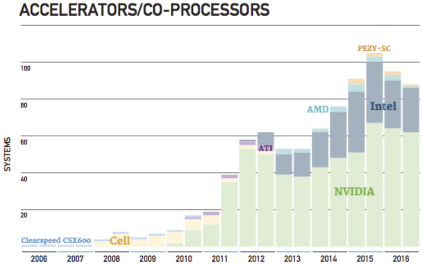
	Name	Specs	Cores
1	Sunway TL	Shenwei 260C 1.45 GHz	10,649,600
2	Tianhe-2	Intel 12C 2.2GHz + Xeon Phi 1.1 GHz	3,120,000
3	Titan	Opteron 16C 2.2GHz + NVIDIA GPU	560,640
4	Sequoia	IBM BlueGene/Q Power 16C 1.6GHz	1,572,864
5	Cori	Intel 16C 2.3GHz + Xeon Phi 1.4 GHz	622,336



Hardware in practice: Top500 from November 2016

	Name	Specs	Cores
1	Sunway TL	Shenwei 260C 1.45 GHz	10,649,600
2	Tianhe-2	Intel 12C 2.2GHz + Xeon Phi 1.1 GHz	3,120,000
3	Titan	Opteron 16C 2.2GHz + NVIDIA GPU	560,640
4	Sequoia	IBM BlueGene/Q Power 16C 1.6GHz	1,572,864
5	Cori	Intel 16C 2.3GHz + Xeon Phi 1.4 GHz	622,336



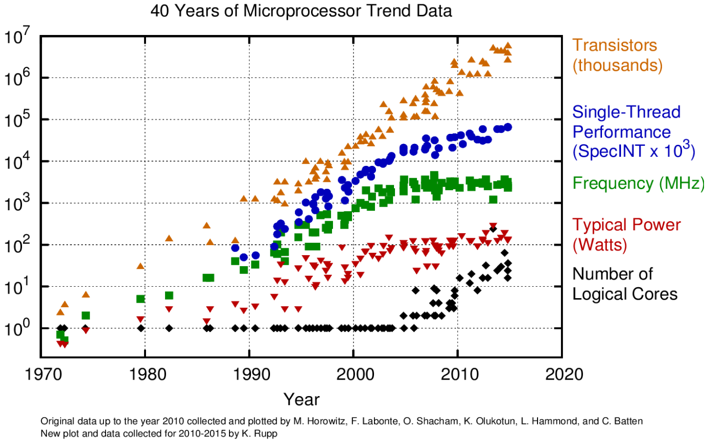


Strategies to deal with ongoing hardware trend

- **Just ignore it; it will pass!**

Strategies to deal with ongoing hardware trend

- **Just ignore it; it will pass!** No, it will not because it's physics that keeps us from simply increasing single core performance.



Strategies to deal with ongoing hardware trend

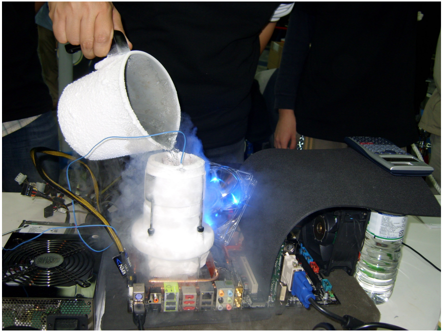
- **Just ignore it; it will pass!** No, it will not because it's physics that keeps us from simply increasing single core performance.

- **Then let's cheat physics!**

Strategies to deal with ongoing hardware trend

- **Just ignore it; it will pass!** No, it will not because it's physics that keeps us from simply increasing single core performance.

- **Then let's cheat physics!** Better not, the costs are prohibitive and there will be an end to this strategy, again set by physics.



Strategies to deal with ongoing hardware trend

- **Just ignore it; it will pass!** No, it will not because it's physics that keeps us from simply increasing single core performance.
- **Then let's cheat physics!** Better not, the costs are prohibitive and there will be an end to this strategy, again set by physics.
- **Trust in the power of compilers/tools to auto-magically parallelize/vectorize/distribute/make it fault-tolerant/... your algorithm!**

Strategies to deal with ongoing hardware trend

- **Just ignore it; it will pass!** No, it will not because it's physics that keeps us from simply increasing single core performance.
- **Then let's cheat physics!** Better not, the costs are prohibitive and there will be an end to this strategy, again set by physics.
- **Trust in the power of compilers/tools to auto-magically parallelize/vectorize/distribute/make it fault-tolerant/... your algorithm!** Good luck, and thanks for the fish.

Proposition 2

It's time (since 2005) for a **radical paradigm shift**: Hardware trends must be incorporated into the design and analysis of numerical methods and algorithms, and their implementations.

Hardware-Oriented Numerics

State of the art

Given a problem $p \in \mathcal{P}$ and a target hardware $h \in \mathcal{H}$:

1. Find *best combination* $(m, a, i)_{p,h} \in \mathcal{M} \times \mathcal{A} \times \mathcal{I}$ that solves problem p on hardware h in shortest time with prescribed accuracy

Hardware-Oriented Numerics

State of the art

Given a problem $p \in \mathcal{P}$ and a set of target hardware $\{h_1, h_2, \dots\} \subset \mathcal{H}$:

1. Find *best combinations* $(m, a, i)_{p,h_k} \in \mathcal{M} \times \mathcal{A} \times \mathcal{I}$ that solve problem p on hardware h_k in shortest time with prescribed accuracy

Hardware-Oriented Numerics

State of the art

Given a problem $p \in \mathcal{P}$ and a set of target hardware $\{h_1, h_2, \dots\} \subset \mathcal{H}$:

1. Find *best combinations* $(m, a, i)_{p,h_k} \in \mathcal{M} \times \mathcal{A} \times \mathcal{I}$ that solve problem p on hardware h_k in shortest time with prescribed accuracy

Next step

2. Develop a strategy that automatically inspects the available hardware and chooses the best combinations $(m, a, i)_{p,h_k}$

Hardware-Oriented Numerics

State of the art

Given a problem $p \in \mathcal{P}$ and a set of target hardware $\{h_1, h_2, \dots\} \subset \mathcal{H}$:

1. Find *best combinations* $(m, a, i)_{p,h_k} \in \mathcal{M} \times \mathcal{A} \times \mathcal{I}$ that solve problem p on hardware h_k in shortest time with prescribed accuracy

Next step

2. Develop a strategy that automatically inspects the available hardware and chooses the best combinations $(m, a, i)_{p,h_k}$

Future vision

3. Automatically determine and schedule best combinations $(m, a, i)_{p_j,h_k} \in \mathcal{M} \times \mathcal{A} \times \mathcal{I}$ for multi-physics problems $\{p_1, p_2, \dots\} \subset \mathcal{P}$ and target hardware $\{h_1, h_2, \dots\} \subset \mathcal{H}$

HWON, is it really that new?

Iterative refinement

For $m = 1, \ldots$ repeat

1. Compute residual

$$r_m = b - Ax_m$$

2. Solve system

$$Ad_m = r_m$$

3. Add correction

$$x_{m+1} = x_m + d_m$$

until convergence

- Wilkinson 1948: code for the Automatic Computing Engine to solve linear system $Ax = b$

HWON, is it really that new?

Iterative refinement

For $m = 1, \ldots$ repeat

1. Compute high-prec residual

$$r_m^{hp} = b^{hp} - A^{hp} x_m^{hp}$$

2. Solve low-prec system

$$A^{lp} d_m^{lp} = \mathrm{LP}(r_m^{hp})$$

3. Add high-prec correction

$$x_{m+1}^{hp} = x_m^{hp} + \mathrm{HP}(d_m^{lp})$$

until convergence

- Wilkinson 1948: code for the Automatic Computing Engine to solve linear system $Ax = b$

Mixed-precision variant

- Wilkinson 1963/Moler 1967: error + convergence analysis

HWON, is it really that new?

Iterative refinement

For $m = 1, \ldots$ repeat

1. Compute high-prec residual

$$r_m^{hp} = b^{hp} - A^{hp} x_m^{hp}$$

2. Solve low-prec system

$$A^{lp} d_m^{lp} = \mathrm{LP}(r_m^{hp})$$

3. Add high-prec correction

$$x_{m+1}^{hp} = x_m^{hp} + \mathrm{HP}(d_m^{lp})$$

until convergence

- Wilkinson 1948: code for the Automatic Computing Engine to solve linear system $Ax = b$

Mixed-precision variant

- Wilkinson 1963/Moler 1967: error + convergence analysis
- Anderson *et al.* 1995: driver for the LAPACK benchmark

HWON, is it really that new?

Iterative refinement

For $m = 1, \ldots$ repeat

1. Compute high-prec residual

$$r_m^{hp} = b^{hp} - A^{hp} x_m^{hp}$$

2. Solve low-prec system

$$A^{lp} d_m^{lp} = \mathrm{LP}(r_m^{hp})$$

3. Add high-prec correction

$$x_{m+1}^{hp} = x_m^{hp} + \mathrm{HP}(d_m^{lp})$$

until convergence

- Wilkinson 1948: code for the Automatic Computing Engine to solve linear system $Ax = b$

Mixed-precision variant

- Wilkinson 1963/Moler 1967: error + convergence analysis
- Anderson *et al.* 1995: driver for the LAPACK benchmark
- Göddeke *et al.* 2007: speed-up double-precision on GPUs

HWON, is it really that new?

Iterative refinement

For $m = 1, \ldots$ repeat

1. Compute high-prec residual

$$r_m^{hp} = b^{hp} - A^{hp} x_m^{hp}$$

2. Solve low-prec system

$$A^{lp} d_m^{lp} = \mathrm{LP}(r_m^{hp})$$

3. Add high-prec correction

$$x_{m+1}^{hp} = x_m^{hp} + \mathrm{HP}(d_m^{lp})$$

until convergence

- Wilkinson 1948: code for the Automatic Computing Engine to solve linear system $Ax = b$

Mixed-precision variant

- Wilkinson 1963/Moler 1967: error + convergence analysis
- Anderson *et al.* 1995: driver for the LAPACK benchmark
- Göddeke *et al.* 2007: speed-up double-precision on GPUs
- NVIDIA SC15: Mixed-precision arithmetic on Pascal GPUs

Mixed-precision methods

Iterative refinement

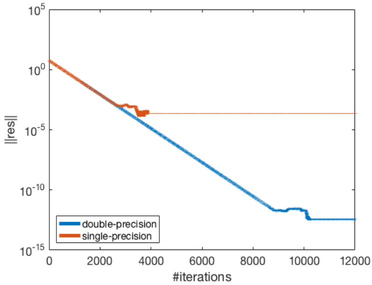
For $m = 1, \ldots$ repeat

1. Compute residual
$$r_m = b - Ax_m$$
2. Solve system
$$Ad_m = r_m$$
3. Add correction
$$x_{m+1} = x_m + d_m$$

until convergence

1d Poisson problem with 40 unknowns and Jacobi 'solver'

$$d_m = (\mathrm{diag} A)^{-1} r_m$$



Mixed-precision methods

Iterative refinement

For $m = 1, \ldots$ repeat

1. Compute residual

$$r_m^{dp} = b^{dp} - A^{dp} x_m^{dp}$$

2. Solve system

$$A^{sp} d_m^{sp} = \mathrm{SP}(r_m^{dp})$$

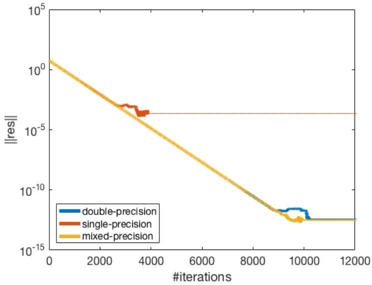
3. Add correction

$$x_{m+1}^{dp} = x_m^{dp} + \mathrm{DP}(d_m^{sp})$$

until convergence

1d Poisson problem with 40 unknowns and Jacobi 'solver'

$$d_m^{sp} = (\mathrm{diag} A^{sp})^{-1} \mathrm{SP}(r_m^{dp})$$



Mixed-precision methods

Iterative refinement

For $m = 1, \ldots$ repeat

1. Compute residual

$$r_m^{dp} = b^{dp} - A^{dp} x_m^{dp}$$

2. Solve system

$$A^{sp} d_m^{sp} = \mathrm{SP}(r_m^{dp})$$

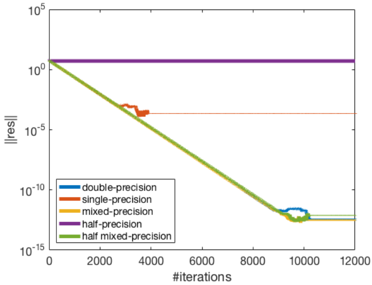
3. Add correction

$$x_{m+1}^{dp} = x_m^{dp} + \mathrm{DP}(d_m^{sp})$$

until convergence

1d Poisson problem with 40 unknowns and Jacobi 'solver'

$$d_m^{sp} = (\mathrm{diag} A^{sp})^{-1} \mathrm{SP}(r_m^{dp})$$



Mixed-precision methods in practice

Theory: The mixed-precision iterative refinement converges to high-precision accuracy if matrix A is *'not too ill-conditioned'*

$$\#\text{iter} \approx f(\log(\text{cond}_2(A)), \log(\epsilon_{\text{high}}/\epsilon_{\text{low}}))$$

Mixed-precision methods in practice

Theory: The mixed-precision iterative refinement converges to high-precision accuracy if matrix A is *'not too ill-conditioned'*

$$\#\text{iter} \approx f(\log(\text{cond}_2(A)), \log(\epsilon_{\text{high}}/\epsilon_{\text{low}}))$$

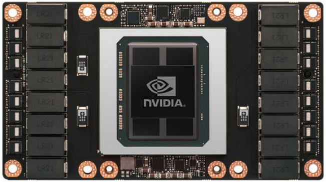
Application: preconditioned mixed-precision defect correction iteration

$$x_{m+1}^{dp} = x_m^{dp} + (C^{sp})^{-1}(b^{dp} - A^{dp}x_m^{dp})$$

with single-precision preconditioner C^{sp}. This strategy can be applied recursively, e.g., if the hardware supports multiple precisions efficiently.

Mixed-precision methods on GPUs

NVIDIA Tesla P100



Memory	12GB
DP perf.	5.3 TeraFLOPS
SP perf.	10.6 TeraFLOPS
HP perf.	21.2 TeraFLOPS

If you only store the preconditioner C as matrix and realize the multiplication with A as on-the-fly operation the maximum number of non-zero entries you can store is

- $\approx 2.1e^9$ in double precision
- $\approx 4.3e^9$ in single precision
- $\approx 8.9e^9$ in half precision

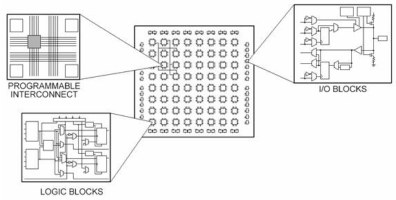
Solution/preconditioning step is

- $\approx 2\times$ faster in single precision
- $\approx 4\times$ faster in half precision

compared to double precision

Mixed-precision methods on FPGAs

Field Programmable Gate Array

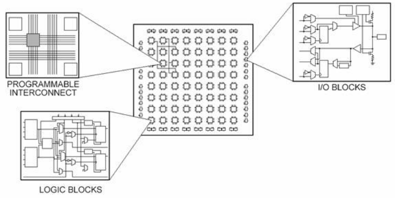


Within the limits of the hardware you can define your own (non-IEEE 754) representation of numbers

- Floating-point number $\pm 0.d_1 d_2 \dots d_n \cdot \beta^e$
- Fixed-point number $\mathrm{Q}m.n$
  $n + m + 1$, i.e. signed integer with n fractional bits

Mixed-precision methods on FPGAs

Field Programmable Gate Array



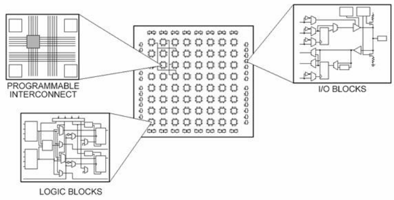
Within the limits of the hardware you can define your own (non-IEEE 754) representation of numbers

- Floating-point number $\pm 0.d_1 d_2 \ldots d_n \cdot \beta^e$
- Fixed-point number Q$m.n$ $n+m+1$, i.e. signed integer with n fractional bits

Ongoing Honours project by Dennis Pouw:
Smart software technologies for enabling next-generation HWON

Mixed-precision methods on FPGAs

Field Programmable Gate Array



Within the limits of the hardware you can define your own (non-IEEE 754) representation of numbers

- Floating-point number $\pm 0.d_1 d_2 \dots d_n \cdot \beta^e$
- Fixed-point number Q$m.n$ $n + m + 1$, i.e. signed integer with n fractional bits

Ongoing Honours project by Dennis Pouw:
Smart software technologies for enabling next-generation HWON

Topic for Bachelor project:
Mixed-precision iterative refinement on reconfigurable hardware

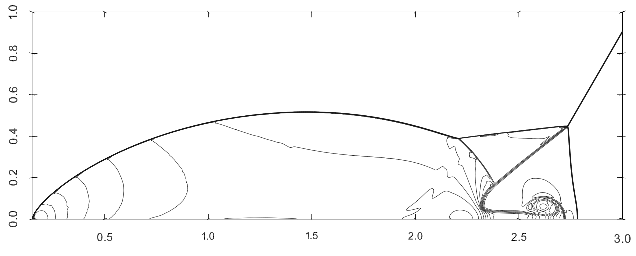
HWON: Not just for nerds anymore

```cpp
#include <vector>
#include <vexcl/vexcl.hpp>
vex::Context ctx( vex::Filter::DoublePrecision );

typedef double high;
typedef float  low;

// Double-precision matrix in CSR format and dense vectors
std::vector<int> row = { 0, 1, 4, 7, 10, 11 };
std::vector<int> col = { 0,
                         0, 1, 2,
                            1, 2, 3,
                               2, 3, 4,
                                     4 };
std::vector<high> ddata = {  1.0,
                            -1.0, 2.0, -1.0,
                                 -1.0,  2.0, -1.0,
                                       -1.0,  2.0, -1.0,
                                                    1.0 };
vex::sparse::csr<high> A(ctx, row.size(), col.size(), row, col, ddata);
vex::vector<high>      b(ctx, row.size()), x(ctx, row.size()); b = 1; x = 0;

// Single-precision preconditioner
std::vector<low> fdata = { 1.0, 2.0, 2.0, 2.0, 1.0 };
vex::vector<low> C(ctx, fdata);

// Mixed-precision iterative refinement
for (int iter=0; iter<10; iter++ )
    x += (b-A*x)/C;
```

My research interest

High-resolution methods for flow problems on HPC architectures

- Convection-diffusion problems
- Compressible flow problems



Variational formulation

Divergence form of a first-order problem

$$\partial_t u + \nabla \cdot \mathbf{f}(u) = 0$$

Variational formulation

Divergence form of a first-order problem

$$\partial_t u + \nabla \cdot \mathbf{f}(u) = 0$$

Galerkin ansatz ("find solution u s.t. for all w")

$$\int_\Omega w \partial_t u - \nabla w \cdot \mathbf{f}(u) \,\mathrm{d}\Omega + \int_\Gamma w \mathbf{n} \cdot \mathbf{f}^b(u) \,\mathrm{d}s = 0$$

with boundary fluxes $\mathbf{f}^b$. Here you can impose boundary conditions

Spatial discretization

Fletcher's **group formulation**[1]

$$u_h = \sum_A \varphi_A(\mathbf{x}) u_A(t), \quad \mathbf{f}_h = \sum_A \varphi_A(\mathbf{x}) \mathbf{f}_A(t), \quad \mathbf{f}_A = \mathbf{f}(u_A)$$

Semi-discrete problem

$$M\dot{u} + \mathbf{C}\mathbf{f} + \mathbf{S}\mathbf{f}^b = 0$$

with constant coefficient matrices

$$M = \left[\int_\Omega \varphi_A \varphi_B \, \mathrm{d}\Omega \right] \quad \mathbf{C} = \left[- \int_\Omega \nabla\varphi_A \varphi_B \, \mathrm{d}\Omega \right] \quad \mathbf{S} = \left[\int_\Gamma \varphi_A \varphi_B \mathbf{n} \, \mathrm{d}s \right]$$

They can be assembled and stored during pre-processing step

[1] C.A.J. Fletcher, CMAME 37 (1983) 225–244.

Spatial discretization

Fletcher's **group formulation**[1]

$$u_h = \sum_A \varphi_A(\mathbf{x}) u_A(t), \quad \mathbf{f}_h = \sum_A \varphi_A(\mathbf{x}) \mathbf{f}_A(t), \quad \mathbf{f}_A = \mathbf{f}(u_A)$$

Semi-discrete problem

$$M\dot{u} + \mathbf{C}\mathbf{f} + \mathbf{S}\mathbf{f}^b = 0$$

Read the above as sequence of SpMV-operations

$$\mathbf{C}\mathbf{f} = \sum_{d=1}^{\mathrm{dim}} C_d f_d, \quad \mathbf{S}\mathbf{f}^b = \sum_{d=1}^{\mathrm{dim}} S_d f_d^b$$

[1] C.A.J. Fletcher, CMAME 37 (1983) 225–244.

Fully discrete problem

Abstract formulation of semi-discrete problem

$$M\dot{u} + N(u) = 0$$

Fully discrete problem

Abstract formulation of semi-discrete problem

$$M\dot{u} + N(u) = 0$$

Discretization in time by **explicit SSP Runge-Kutta method**, e.g.

$$\begin{aligned} Mu^{(1)} &= Mu^n - \Delta t N(u^n) \\ Mu^{n+1} &= \frac{1}{2}Mu^n + \frac{1}{2}Mu^{(1)} - \frac{1}{2}\Delta t N(u^{(1)}) \end{aligned}$$

Fully discrete problem

Abstract formulation of semi-discrete problem

$$M\dot{u} + N(u) = 0$$

Discretization in time by **explicit SSP Runge-Kutta method**, e.g.

$$\begin{aligned} Mu^{(1)} &= Mu^n - \Delta t N(u^n) \\ Mu^{n+1} &= \frac{1}{2}Mu^n + \frac{1}{2}Mu^{(1)} - \frac{1}{2}\Delta t N(u^{(1)}) \end{aligned}$$

Finishing touches

- Stabilization of divergence term by algebraic flux correction
- Efficient implementation by smart-and-fast expression templates

Question 1

What is a good choice of basis functions in the spirit of HWON?

Question 1

What is a good choice of basis functions in the spirit of HWON?

- **Low-order FEM?** low FLOP/byte ratio (-)

Question 1

What is a good choice of basis functions in the spirit of HWON?

- **Low-order FEM?** low FLOP/byte ratio (-)
- **High-order FEM?** greater FLOP/byte ratio (+)

Question 1

What is a good choice of basis functions in the spirit of HWON?

- **Low-order FEM?** low FLOP/byte ratio (-)
- **High-order FEM?** greater FLOP/byte ratio (+)
- **Structured grids?** overset grids for complex geometries (?)

Question 1

What is a good choice of basis functions in the spirit of HWON?

- **Low-order FEM?** low FLOP/byte ratio (-)
- **High-order FEM?** greater FLOP/byte ratio (+)
- **Structured grids?** overset grids for complex geometries (?)
- **Unstructured grids?** flexible (+), high-order grid generation open problem (-), indirect addressing (-)

Question 1

What is a good choice of basis functions in the spirit of HWON?

- **Low-order FEM?** low FLOP/byte ratio (-)
- **High-order FEM?** greater FLOP/byte ratio (+)
- **Structured grids?** overset grids for complex geometries (?)
- **Unstructured grids?** flexible (+), high-order grid generation open problem (-), indirect addressing (-)
- **Discontinuous Galerkin?** well-established in HPC (+), unstructured grids (?), excessive duplication of DOFs (-)

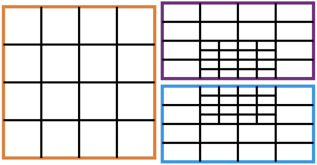
Question 1

What is a good choice of basis functions in the spirit of HWON?

- **Low-order FEM?** low FLOP/byte ratio (-)
- **High-order FEM?** greater FLOP/byte ratio (+)
- **Structured grids?** overset grids for complex geometries (?)
- **Unstructured grids?** flexible (+), high-order grid generation open problem (-), indirect addressing (-)
- **Discontinuous Galerkin?** well-established in HPC (+), unstructured grids (?), excessive duplication of DOFs (-)
- **Continuous Galerkin?** unconventional in hyperbolic flows (?), less DOFs (+), stabilization more problematic (-)

The big picture

- Combine **unstructured multi-block coarse grid** ('patches') with
  - topologically **structured fine grid** within each patch;
  - **locally refined fine grid** where required for accuracy
- Apply **Isogeometric Analysis** approach on each patch
- Couple multiple patches by DG- or Nitsche-type approach



The big picture

- Combine **unstructured multi-block coarse grid** ('patches') with
  - topologically **structured fine grid** within each patch;
  - **locally refined fine grid** where required for accuracy
- Apply **Isogeometric Analysis** approach on each patch
- Couple multiple patches by DG- or Nitsche-type approach

HWON considerations:

- associate patches with devices (DG to reduce communication)

The big picture

- Combine **unstructured multi-block coarse grid** ('patches') with
  - topologically **structured fine grid** within each patch;
  - **locally refined fine grid** where required for accuracy
- Apply **Isogeometric Analysis** approach on each patch
- Couple multiple patches by DG- or Nitsche-type approach

HWON considerations:

- associate patches with devices (DG to reduce communication)
- if a patch becomes computationally too expensive then split it up into multiple patches (intrinsically supported by IgA via successive continuity reduction) and reschedule new patches to (more) devices

Polynomial spaces

Definition

The space of polynomials of degree p over the interval $[a, b]$ is

$$\Pi^p([a,b]) := \left\{ q(x) \in \mathcal{C}^\infty([a,b]) \,:\, q(x) = \sum_{i=0}^{p} c_i x^i, c_i \in \mathbb{R} \right\}$$

Example: $\Pi^2([0,1])$

- Canonical basis

$$\mathcal{B} = \{1, x, x^2\}$$

- Polynomials

$$q(x) = c_0 + c_1 x + c_2 x^2$$

Spline space

Definition

Let $\mathcal{P} = \{a = x_1 < \cdots < x_{p+1} = b\}$ be a partition of the interval Ω_0 and $\mathcal{M} = \{1 \leq m_i \leq p+1\}$ a set of positive integers. The polynomial spline of degree p is defined as $s : \Omega_0 \mapsto \mathbb{R}$ if

$$s|_{[x_i, x_{i+1}]} \in \Pi^p([x_i, x_{i+1}]), \qquad i = 1, \ldots, k$$

$$\frac{d^j}{dx^j} s_{i-1}(x_i) = \frac{d^j}{dx^j} s_i(x_i), \qquad \begin{array}{l} i = 2, \ldots, k, \\ j = 0, \ldots, p - m_i \end{array}$$

Polynomial splines of degree p form the spline space $\mathcal{S}(\Omega_0, p, \mathcal{M}, \mathcal{P})$.

Knot vectors

Definition

A knot vector is a sequence of non-decreasing values $\xi_i \in [a, b] \subset \mathbb{R}$ in the parameter space $\Omega_0 = [a, b]$

$$\Xi = (\xi_1, \xi_2, \ldots, \xi_{n+p+1})$$

where

- p is the polynomial order of the B-splines
- n is the number of B-spline functions
- ξ_i is the i-th knot with knot index i

Knots ξ_i can have multiplicity $1 \leq m_i \leq p + 1$. The knot vector is called open if the first and last knot have multiplicity $p + 1$.

B-spline basis functions

Cox-de Boor recursion formula

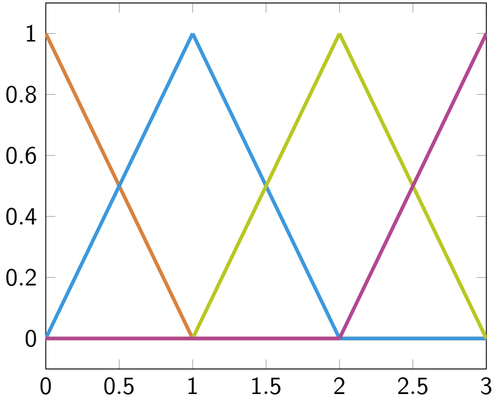
$p = 0$

$$N_{i,0}(\xi) = \begin{cases} 1 & \text{if } \xi_i \leq \xi < \xi_{i+1} \\ 0 & \text{otherwise} \end{cases}$$

$p > 0$

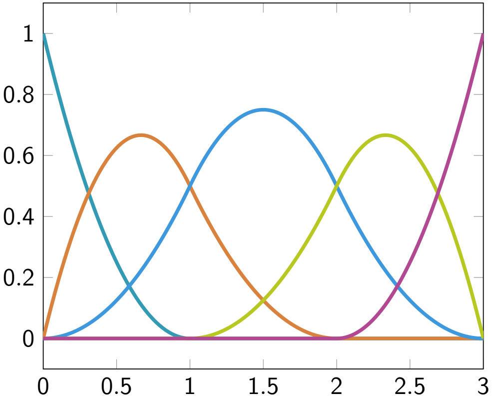
$$N_{i,p}(\xi) = \frac{\xi - \xi_i}{\xi_{i+p} - \xi_i} N_{i,p-1}(\xi) + \frac{\xi_{i+p+1} - \xi}{\xi_{i+p+1} - \xi_{i+1}} N_{i+1,p-1}(\xi)$$

B-spline basis functions



Linear basis functions corresponding to $\Xi = \{0, 0, 0, 1, 2, 3, 3, 3\}$

B-spline basis functions



Quadratic basis functions corresponding to $\Xi = \{0, 0, 0, 1, 2, 3, 3, 3\}$

Properties of B-spline basis functions

Compact support

$$\text{supp } N_{i,p}(\xi) = [\xi_i, \xi_{i+p+1}), \quad i = 1, \dots, n$$

Strict positiveness

$$N_{i,p}(\xi) > 0 \quad \text{for } \xi \in (\xi_i, \xi_{i+p+1}), \quad i = 1, \dots, n$$

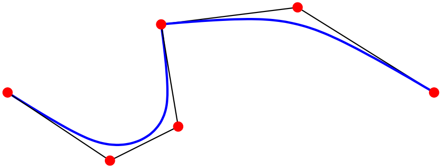
Partition of unity

$$\sum_{i=1}^{n} N_{i,p}(\xi) = 1 \quad \text{for all } \xi \in [a, b]$$

Spline curves

Geometric mapping $\mathbf{G} : \Omega_0 \mapsto \Omega_h \simeq \Omega$

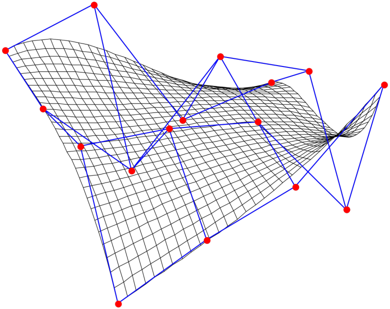
$$\mathbf{G}(\xi) = \sum_{i=1}^{n} N_{i,p}(\xi)\mathbf{B}_i \qquad \text{set of control points } \mathbf{B}_i \in \mathbb{R}^d, d \geq 1$$



Spline surfaces

Geometric mapping $\mathbf{G} : \Omega_0 \mapsto \Omega_h \simeq \Omega$

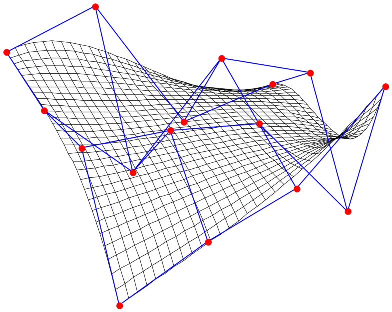
$$\mathbf{G}(\xi, \eta) = \sum_{i=1}^{n} \sum_{j=1}^{m} N_{i,p}(\xi) N_{j,q}(\eta) \mathbf{B}_{i,j} \qquad \mathbf{B}_{i,j} \in \mathbb{R}^d, d \geq 2$$



Spline surfaces

Geometric mapping $\mathbf{G} : \Omega_0 \mapsto \Omega_h \simeq \Omega$

$$\mathbf{G}(\boldsymbol{\xi}) = \sum_{\mathbf{A}} \hat{\varphi}_{\mathbf{A}}(\boldsymbol{\xi})\mathbf{B}_{\mathbf{A}} \qquad \mathbf{B}_{\mathbf{A}} \in \mathbb{R}^d,\, d \geq 2,\ \text{multi-index } \mathbf{A}$$



Marriage of geometry and discretization

Geometric mapping

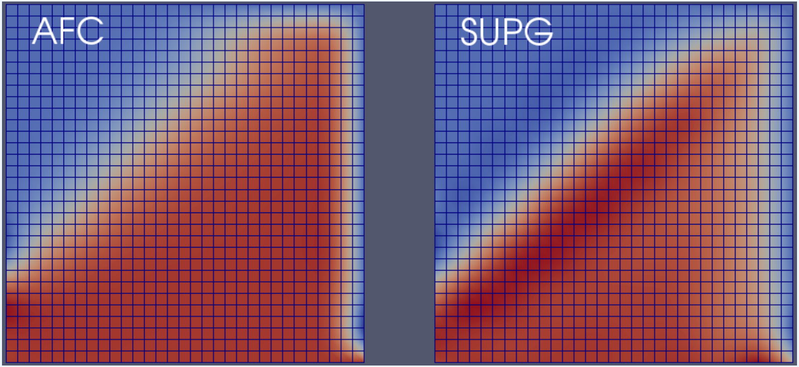
$$\mathbf{G}(\xi) = \sum_{\mathbf{A}} \hat{\varphi}_{\mathbf{A}}(\xi)\mathbf{B}_{\mathbf{A}} \qquad \text{'push-forward'} \;\; \mathbf{G} : \Omega_0 \mapsto \Omega_h$$

Ansatz space

$$V_h = \text{span}\{\varphi_{\mathbf{A}}(\mathbf{x}) = \hat{\varphi}_{\mathbf{A}} \circ \mathbf{G}^{-1}(\mathbf{x})\} \qquad \text{'pull-back'} \;\; \mathbf{G}^{-1} : \Omega_h \mapsto \Omega_0$$

Application: Convection-diffusion equation

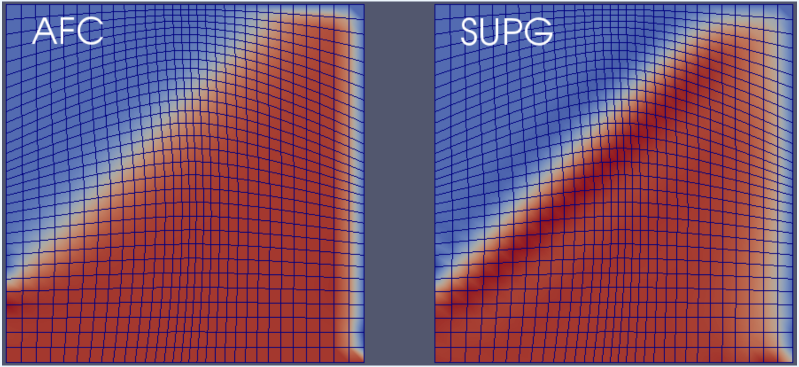
Convection skew to the mesh



Quadratic bi-variate B-spline basis functions.

Application: Convection-diffusion equation

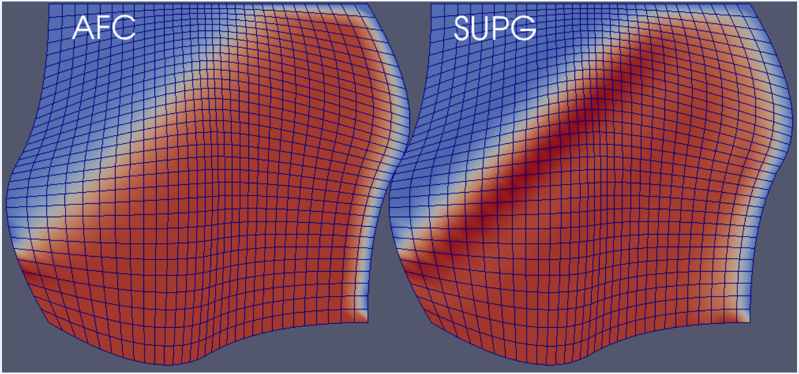
Convection skew to the mesh



Quadratic bi-variate B-spline basis functions.

Application: Convection-diffusion equation

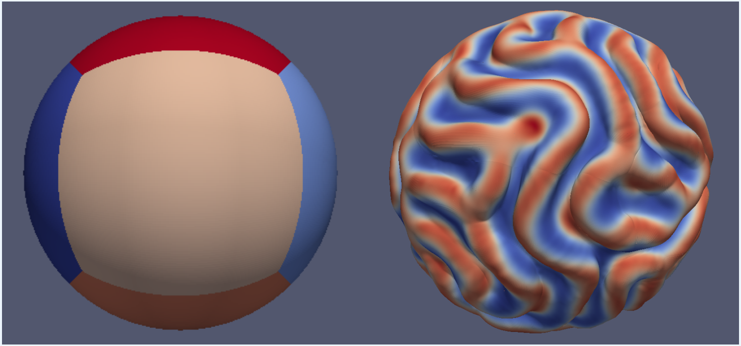
Convection skew to the mesh



Quadratic bi-variate B-spline basis functions.

Application: PDEs on evolving manifolds

Human brain development (MSc project by J. Hinz)



There is much more to investigate in a master project if you are interested.