Abstract

Durham University
Department of Computer Science

Peano is a framework for large-scale simulations using dynamically adaptive Cartesian grids. It is used today for Earthquake and Black Hole simulations, for example. The fourth generation of the software is currently under development.
Peano's development as well as the push behind ExaHyPE, a solver engine built upon Peano, always has been shaped by the ambition to implement state-of-the-art numerics. In our field, this implies multiscale algorithms where others work with "flat" data structures, dynamically changing data structures where others rely on something static, writing multi-numerics/multi-physics codes where others focus on one thing, supporting hybrid architectures where others commit either to GPGPU- or CPU-only, and so forth.
In this talk I briefly categorise the software and present application areas. After that, I focus on the software's genesis. Peano has started off as a collection of codes for solving incompressible fluids, yet spread out into many application areas, it has been shaped (and misshaped) by dozens of core developers, and it has grown repeatedly into a state that made it hard to maintain and extend further. Therefore, each generation has become a complete rewrite—also as we tried to bring in new, fancy numerics every time.
I will explain which software design patterns we use today in our framework in an attempt to deliver software that is fast, maintainable and usable for all the different communities involved. With our complex agenda, it is basically impossible to find developers among PhDs, academics or RSEs that master all areas of relevance. So we need a strict separation of concerns (and flaws) which materialises in our code as the Hollywood Principle: Don't call us, we call you. In short, we take a lot of freedom away from developers how they can realise things. Instead, we force them to focus on what they want to do.

Durham University
Department of Computer Science

My project expired and my team left, so let's rewrite all the software from scratch

SORSE

T. Weinzierl

January 2021

Vision: Allow groups with decent computational background to write an exascale solver for

$$\boldsymbol{M}\frac{\partial}{\partial t}Q + \boldsymbol{\nabla}\cdot\mathbf{F}(Q) + \sum_i \boldsymbol{\mathcal{B}}_i \frac{\partial Q}{\partial x_i} = \mathbf{S} + \sum \delta$$

within a year.

ExaHyPE's software roadmap:

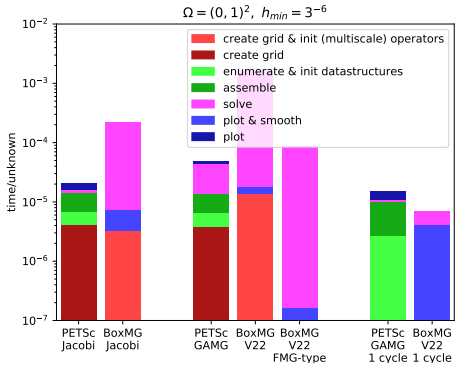
- Users decide *what* to solve (PDE terms)
- Engine decides *how* to solve it (algorithmic building blocks)
- Runtime decides *where* and *when* (scheduling)

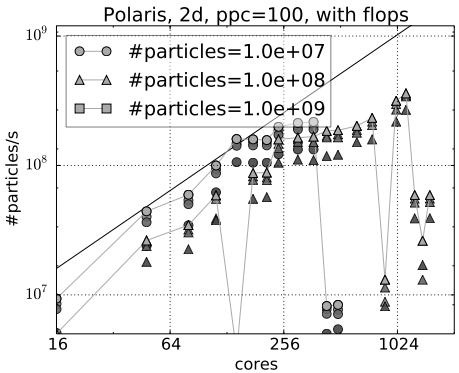
$\Rightarrow$ It is an engine that you tailor towards your application*

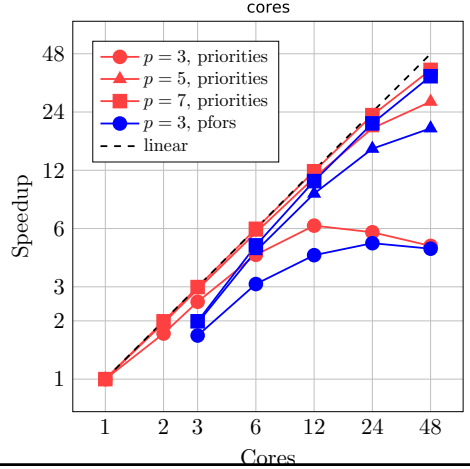
* as long as it is a first-order hyperbolic system of PDEs.

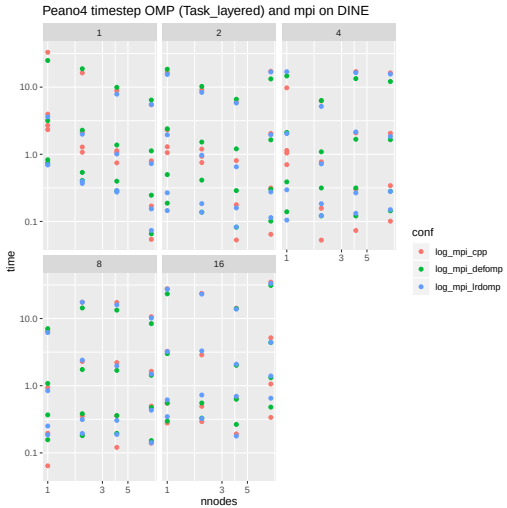
The look-n-feel metaphor for plots

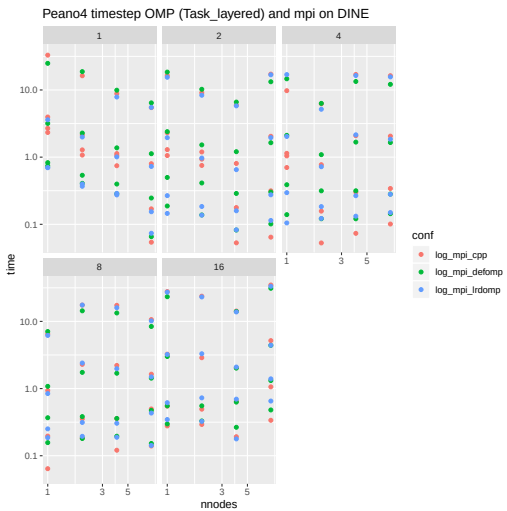
Durham University
Department of Computer Science

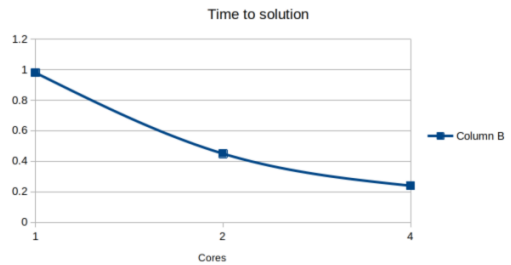


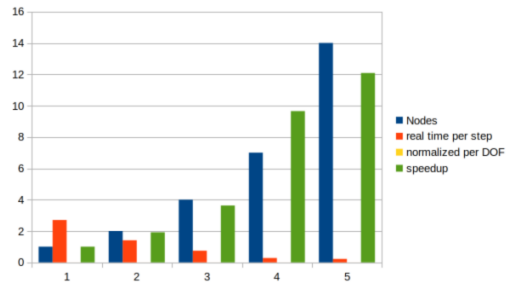






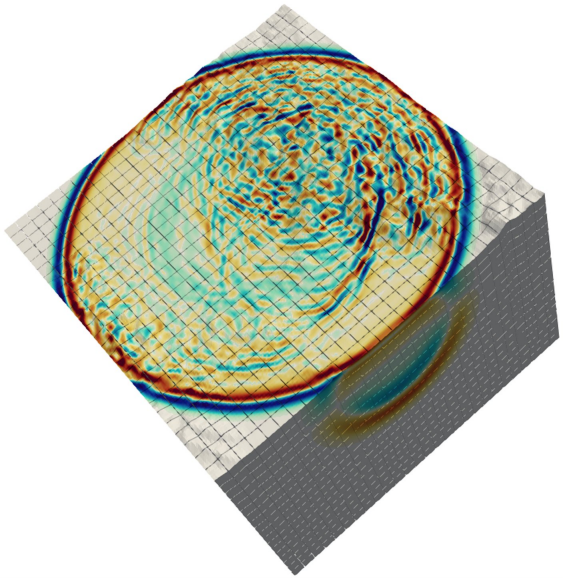


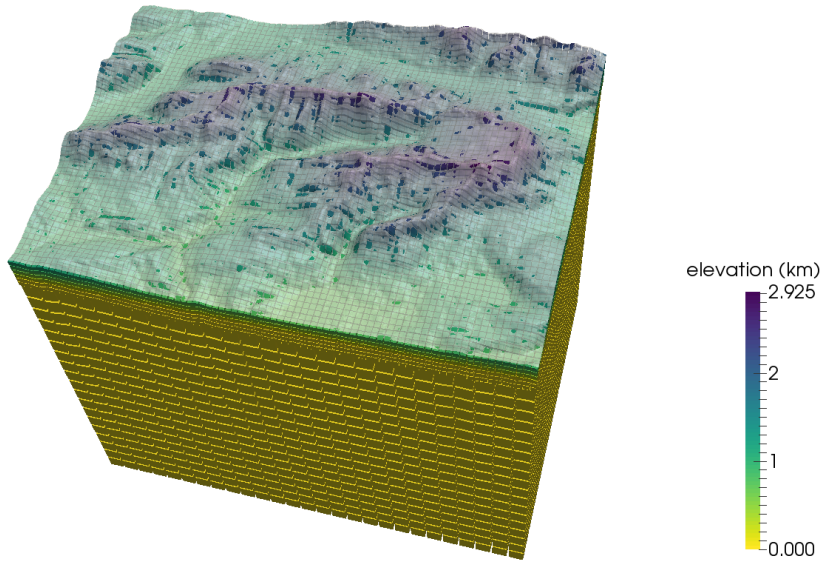




Look-n-feel for a compute engine

Durham University
Department of Computer Science





It is not about the visuals (Paraview/TecPlot), it is about

- ADER-DG plus Finite Volumes
- Spacetrees with three-partitioning
- Intel-specific kernels tailored towards seismic applications (and some astro, too)

A PDE solver engine alike a game engine

Durham University
Department of Computer Science

Generic problem formulation:

$$\boldsymbol{M}\frac{\partial}{\partial t}Q + \nabla \cdot \mathbf{F}(Q) + \sum_i \boldsymbol{\mathcal{B}}_i \frac{\partial Q}{\partial x_i} = \mathbf{S} + \sum \delta$$

Usage:

1. Write a Python specification script
   - Order $p \in \{2, 3, \ldots, 9\}$ in space and time or patch-sizes
   - Flavour of solver
   - Plotting intervals and domain sizes
2. Generator yields stand-alone C++ simulation code
3. Implement physics, not numerics (generated C++ classes/SymPy/pre-manufactured)
4. Type in `make`

Implications:

$\Rightarrow$ User focuses on PDE terms, boundary conditions, adaptivity control, ...

$\Rightarrow$ Engine fixes and provides the compute-n-feel (CS+Math)

```
git clone -b p4
libtoolize; aclocal; autoconf; autoheader; cp src/config.h.in .; automake --add-missing
./configure --enable-exahype --enable-loadbalancing-toolbox
make
cd examples/exahype2/euler
jupyter-notebook Euler.ipynb
```

The Euler equations

Durham University
Department of Computer Science

```
project = exahype2.Project( ["examples", "exahype2", "euler"], "finitevolumes", "." )
project.add_solver( exahype2.solvers.fv.GenericRusanovFixedTimeStepSize(
 "Euler", patch_size, unknowns, aux_values, min_h, max_h, time_step_size ))
build_mode = peano4.output.CompileMode.Release
project.set_global_simulation_parameters(
 dimensions, [0.0,0.0,0.0], [1.0,1.0,1.0],
 end_time, first_snapshot, plot_interval )
project.set_load_balancing( "toolbox::loadbalancing::RecursiveSubdivision" )
project.set_Peano4_installation("../../..", build_mode)
peano4_project = project.generate_Peano4_project()
peano4_project.output.makefile.parse_configure_script_outcome( "../../.." )
peano4_project.generate()
```

```
double examples::exahype2::euler::Euler::maxEigenvalue( ... ) {
  constexpr double gamma = 1.4;
  const double irho = 1./Q[0];
  #if Dimensions==3
  const double p = (gamma-1) * (Q[4] - 0.5*irho*(Q[1]*Q[1]+Q[2]*Q[2]+Q[3]*Q[3]));
  #else
  const double p = (gamma-1) * (Q[4] - 0.5*irho*(Q[1]*Q[1]+Q[2]*Q[2]));
  #endif
  const double u_n = Q[normal + 1] * irho;
  const double c   = std::sqrt(gamma * p * irho);
  return std::max( std::abs(u_n-c), std::abs(u_n+c) );
}
```

```
void examples::exahype2::euler::Euler::flux( ... ) {
 ...
}
```

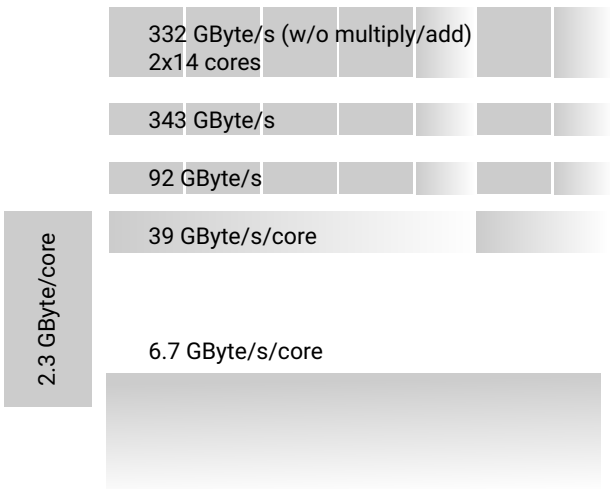
This talk's subject: How did we get there?

(and does this idea with the engine work out)

Outline

Cache-aware algorithms

Durham University
Department of Computer Science



SuperMUC Phase 2

Memory accesses become performance-critical

- widening compute-memory gap
- energy requirements

$\Rightarrow$ caches ride to rescue

Cache optimisation techniques

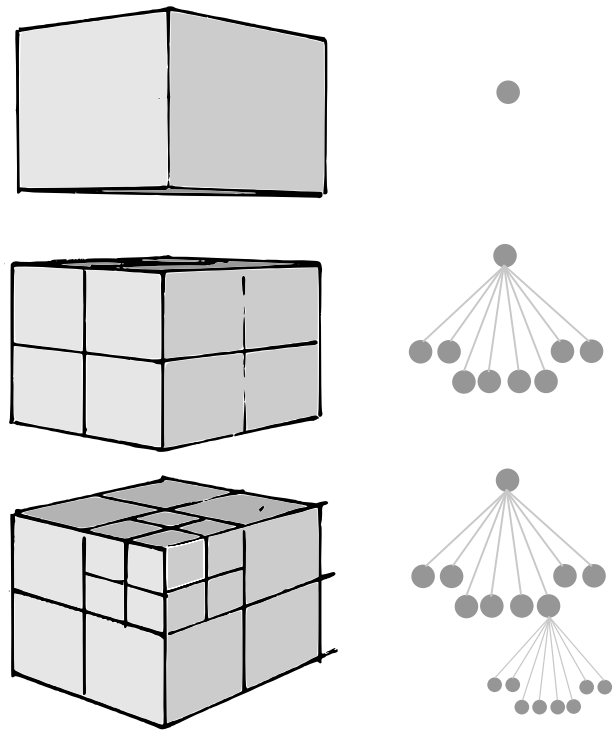
- Spatial data reuse
  (stencil fusion)
- Temporal data reuse
  (multiple iterations)
- Blocking

$\Rightarrow$ difficult for changing, irregular data (AMR)

Bio: Pre-2005; worked on this as student research assistant to support PhD student in RSE role

Spacetree grids

Durham University
Department of Computer Science



Team:

- Work driven by Christoph Zenger
- MSc@TUM
- Three PhD students and 9+ MSc theses

Classics:

- Generalisation of quadtrees/octrees
- Facilitate dynamic AMR
- Span multiscale mesh
- Tensor-product style

New idea:

- Linearise via SFCs
- Store data on stacks
  more on this in a second

$\Rightarrow$ Intrinsic cache-oblivious

(*) M. Mehl, T. Weinzierl, Ch. Zenger: *A cache-oblivious self-adaptive full multigrid method*. Numerical Linear Algebra with Applications, 13(2–3), pp. 275–291 (2006)

There's no first generation of Peano

Durham University
Department of Computer Science

Achievements #1

- Numerous MSc dissertations
  parallelisation, lb, CFD, multigrid, $d \geq 4$, ...
- Three PhD theses (2d, 3d, adaptivity criteria)
- One PostDoc/habilitation

Achievements #2

- A lot of publications
- Two DFG (German EPSRC) projects
- 2.5 PhD positions

Achievements #3

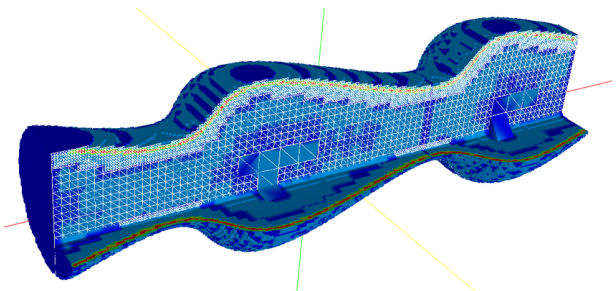
- Numerous code branches using different code bases

$\Rightarrow$ rather a code family

Outline

Algorithms that solve a relevant problem

Durham University
Department of Computer Science



Drift Ratchet

Driving force: CFD research projects

- Drift-ratchet
  massive AMR with FSI
- DFG benchmark
  dynamic AMR

Code base

- MSc students (almost) all left
- No PhD pursued academic career
- Forks' code bases incompatible

Way out: PhD which is open-ended

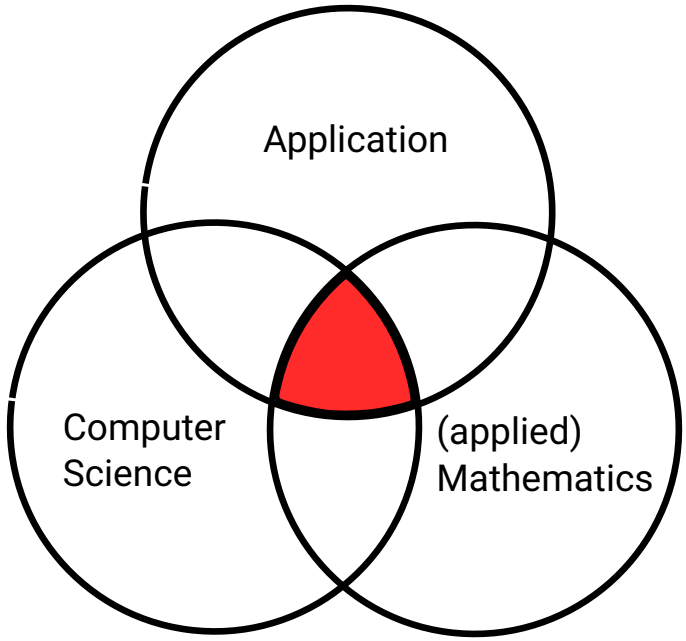
- "just combine these features"
  classic RSE task as a starter (*)
- rewrite from scratch
  computer scientists like this anyway
- try not to make same error again
  no code divergence

Bio: 2005–2009; worked as PhD student and had combined RSE+fundamental research role

(*) please see next slides

Why were branches incompatible?

Durham University
Department of Computer Science



- Three different disciplines “overlap”
- Data organisation, movements, mathematical schemes, physical models, . . . all overlap, too $\Rightarrow$ one big code mess (“the monster”)

```
#pragma omp parallel                          // how to schedule
for (int i=0; ...)  {                         // how is data organised
  tarch::la::Vector<14,double> myX =          // what is computed
}
```

Why was rewrite challenging?

Durham University
Department of Computer Science

- Code is dense (single sign change has long-term implications)
- Code is floating-point heavy
- Code is running on parallel computer (determinism)
- Code is tricky (core algorithm's description approx. 40+ pages)
  (and key graduate had left for high-profile industry job)

$\Rightarrow$ core algorithm+infrastructure took me 2+ years

The Hollywood principle

Durham University
Department of Computer Science

> Hollywood principle: Don't call us, we call you.

Fancy slogan for [GHJV94]'s

- Composite pattern plus
- Visitor pattern plus
- Parallel tree traversals

```
for (auto& cell: cellsInTree) {
 for (auto& faces: cell.facesUsedFirstTime()) touchFacesFirstTime(p);
 touchCellFirstTime(p);
 touchCellLastTime(p);
 for (auto& faces: cell.facesUsedLastTime()) touchFacesLastTime(p);
}
```

Todos for domain expert (in classic frameworks):

- Translate into a BFS or DFS (and switch)
- Combine two traversals
- Run parts in parallel

Lessons learned

Durham University
Department of Computer Science

Achievements

- Delivered upon projects
- Graduated
- Some key papers formalise key concepts as pseudo code
  (after three years)

$\Rightarrow$ will turn out as game changer

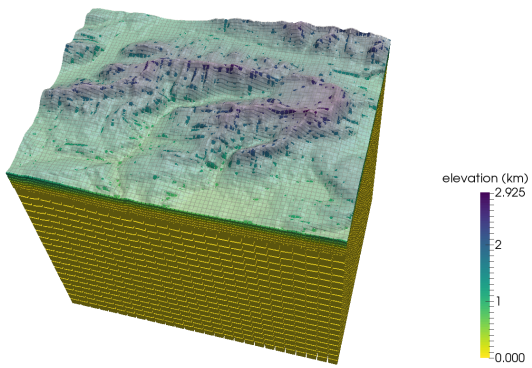
Issues

- Re-education does not go down well
  “I want to solve it that way”
- People work around concept
- And the code quality wasn’t that great either . . .

Outline

ExaHyPE & multicores

Durham University
Department of Computer Science



Earthquake in the alps

Driving forces behind evolution

- Emancipate from application domain
- Separate maths and application
- Support manycores (KNL)

Code base

- Over-generalised callback interfaces
- Tons of fancy C++ code
  (pattern pollution)

Agenda

- Rewrite key algorithms (clean-up)
  (cmp. Fowler's refactoring)

$\Rightarrow$ translate into manycore era

- Apply call-back idea once more (layers)

$\Rightarrow$ separate maths–application

- Introduce heavy code generation

$\Rightarrow$ free developers from glue code burden

Bio: 2010–2019; PostDoc at TUM, Assistant Prof in Durham

A successful rewrite

Durham University
Department of Computer Science

Algorithm 2 Particle-in-dual-tree algorithm (continued in Algorithm 3).

```
1: function touchVertexFirstTime(v)
2:     if v refined then                                  ▷ i.e. all surrounding cells are refined
3:         for all m ∈ M associated to v do               ▷ preamble
4:             moved(m) ← ⊥
5:         end for
6:     end if
7:     invoke application-specific operations
8: end function
9: function enterCell(cell c ∈ T)
10:    if c refined then                                  ▷ preamble
11:        for all 2^d adjacent vertices v do
12:            for all m ∈ M associated to v do
13:                if moved(m) = ⊥ and x(m) contained in c then
14:                    identify child vertex v' whose dual cell holds x(m)
15:                    assign m to v'                     ▷ drop
16:                end if
17:            end for
18:        end for
19:    end if
20:    invoke application-specific operations
21:    if c unrefined then
22:        for all 2^d adjacent vertices v do
23:            for all m ∈ M associated to v do
24:                if moved(m) = ⊥ ∧ x(m) contained in c then
25:                    update position x of m             ▷ move
26:                    moved(m) ← ⊤
27:                end if
28:            end for
29:        end for
30:    end if
31:    invoke application-specific operations
32: end function
```

- Software reuse did **not** work, as signatures changed
- Unit tests partially reused

A not so successful code generation

Durham University
Department of Computer Science

- Create new DSL to what algorithm steps do exist

$\Rightarrow$ generate all glue code for generic interfaces

- Create new DSL to specify data dependencies

$\Rightarrow$ extract code dependencies (concurrency)

Lessons learned

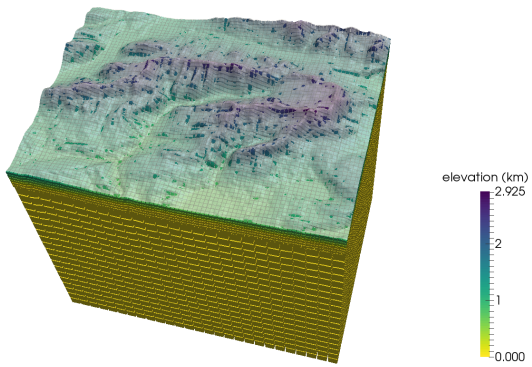
Durham University
Department of Computer Science

- Users don’t like signature changes
- Users don’t like a code generator that overwrites their manual “fixes” either
- Users still don’t like re-education
- Users tend to “fix” your code generator
- Users don’t know C++
- Users don’t like DSLs either

Outline

ExaClaw & GPUs

Durham University
Department of Computer Science



Earthquake in the alps

ExCALIBUR

- Connect to third-party libs (Fortran)
- Port parts of code to GPUs
- Improve all aspects of scalability

Code base

- Lead developer behind ExaHyPE left
- Exascale paradigms had changed
  (KNLs vs GPGPUs)

Strategies

- Rewrite key algorithms (once more)

$\Rightarrow$ translate into accelerator era

- Rewrite code generators in Python

$\Rightarrow$ problem-specific call-backs

- Document key design decisions formally

$\Rightarrow$ see TOMS and SISC tracks

Kick-off: 2019/2020 in Durham

Did it work?

Durham University
Department of Computer Science

Vision: Allow groups with decent computational background to write an exascale solver for

$$\boldsymbol{M}\frac{\partial}{\partial t}Q + \boldsymbol{\nabla}\cdot\mathbf{F}(Q) + \sum_i \boldsymbol{\mathcal{B}}_i \frac{\partial Q}{\partial x_i} = \mathbf{S} + \sum \delta$$

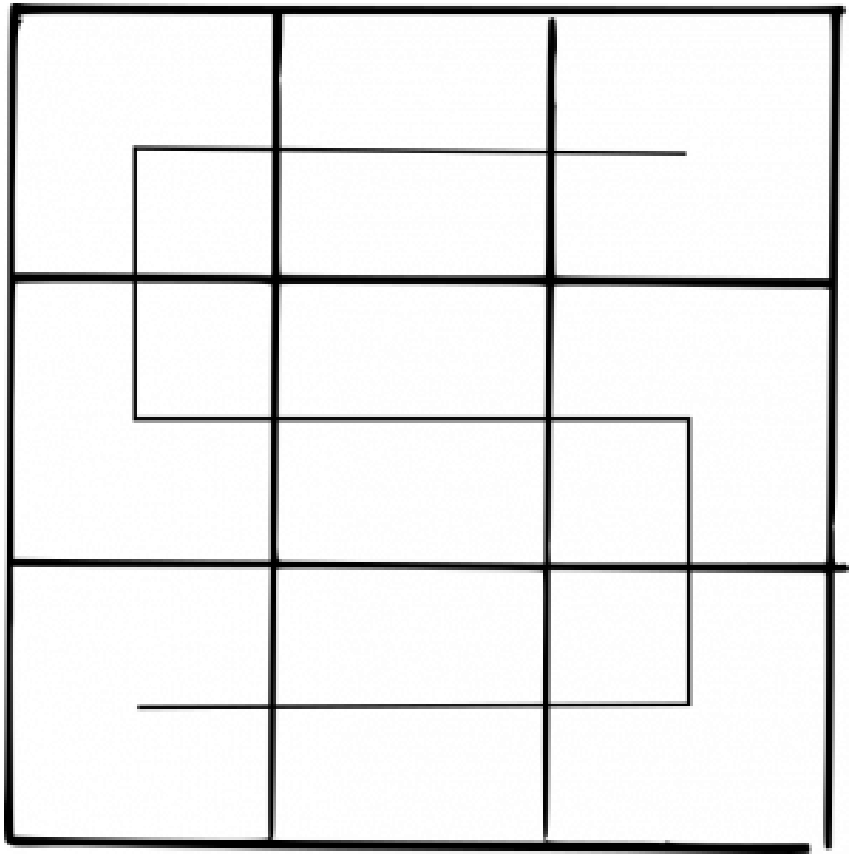
within a year.

- I have to leave the assessment to you (please try it out)
- Benchmark (at the moment) is purely papers and grants and ...
- This question/vision driven by applications is irrelevant to this talk

Outline

Software status-quo

Durham University
Department of Computer Science



- Still called Peano, as it is still relying on Peano's SFC for spacetrees
- For users, it now has a Python-esque compute-n-feel (Jupyter notebooks)
- Under the hood, C++ and top-notch algorithms
  don't compete with NationalLabs in terms of maturity or diversity

Lessons learned/to be discussed

Durham University
Department of Computer Science

- Classic software engineering of limited help
  - Pair programming
  - Documentation (prose)
  - Code reviews
  - $\Rightarrow$ lack of manpower (team overlap) plus qualification in all three areas
  - $\Rightarrow$ for PI, enabling features have higher priority
- Don't branch
  - Colleagues leave
  - Work towards paper deadlines (quick n dirty fixes)
  - $\Rightarrow$ leads to "how to realise feature" schism
  - $\Rightarrow$ biased interpretation of agile methods and sprints
- Don't use OO or Design Patterns
  - Programming skills underdeveloped
  - Pattern pollution due to fascination
  - Interface overspecification
  - Too slow, i.e. code rewrites might become mandatory
  - $\Rightarrow$ document patterns explicitly and minimise their use
  - $\Rightarrow$ Python-based code generation maybe better (DSLs)
  - $\Rightarrow$ please challenge mainstream notions of code quality
- Formalise
  - Algorithm papers help more than open source
  - $\Rightarrow$ Let people reimplement rather than reuse
  - Formalisation of constraints, data structures, assertions, preconditions, ... helps
  - $\Rightarrow$ At least as important as unit tests, e.g.
- Re-educate