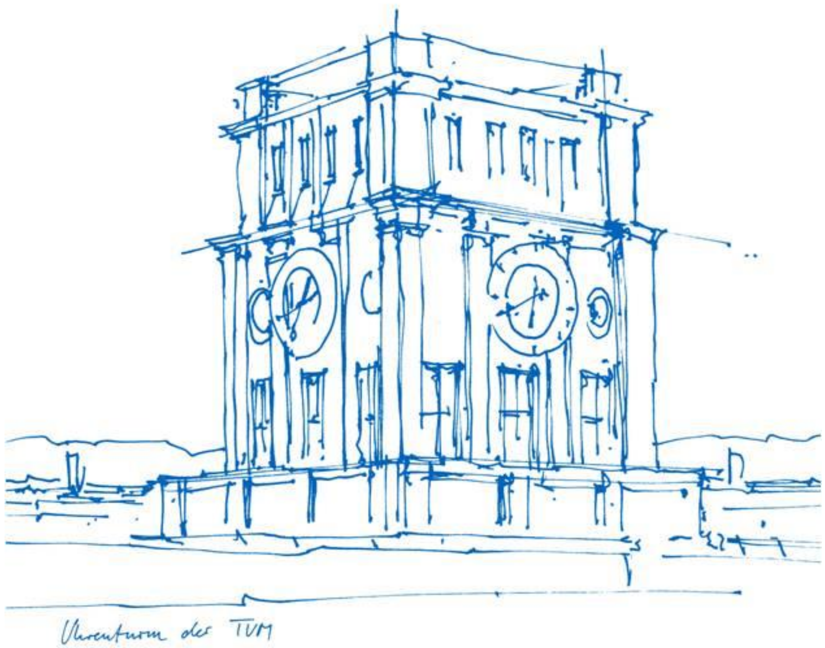
ExaHyPE, an Exascale Hyperbolic PDE Engine

Jean-Matthieu Gallard
Technical University of Munich (TUM)
Department of Informatics
Garching, 21. April 2017

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 671698.



The ExaHyPE Project

Horizon 2020 project in the FETHPC call

“Towards Exascale High Performance Computing”

(New mathematical and algorithmic approaches)

ExaHyPE participants:

- **TUM:** Michael Bader, Jean-Matthieu Gallard, Leonhard Rannabauer
- **Durham Univ.:** Tobias Weinzierl, Dominic Charrier, Benjamin Hazelwood
- **Univ. Trento:** Michael Dumbser, Francesco Fambri, Maurizio Tavelli
- **LMU Munich:** Alice Gabriel, Kenneth Duru
- **Frankfurt IAS:** Luciano Rezzolla, Sven Köppel, Alejandro Cruz Osorio
- **RSC:** Alexander Moskowsky & Co.
- **BayFOR:** Robert Iberl, Teresa Kindermann (project management)
- associated: Leibniz Supercomputing Centre and JSCC RAS

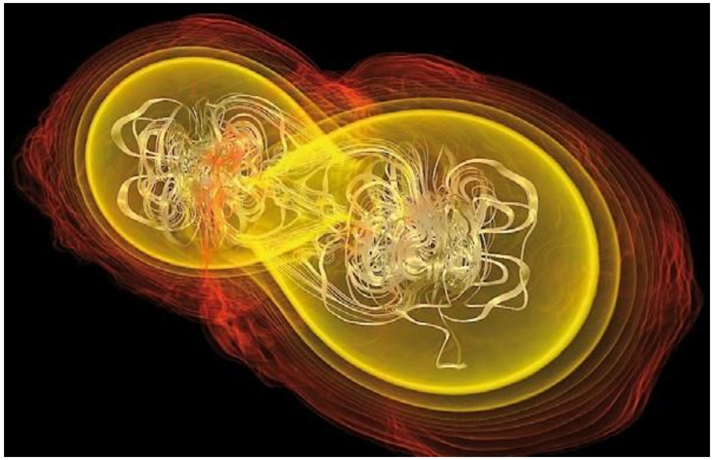
Outline

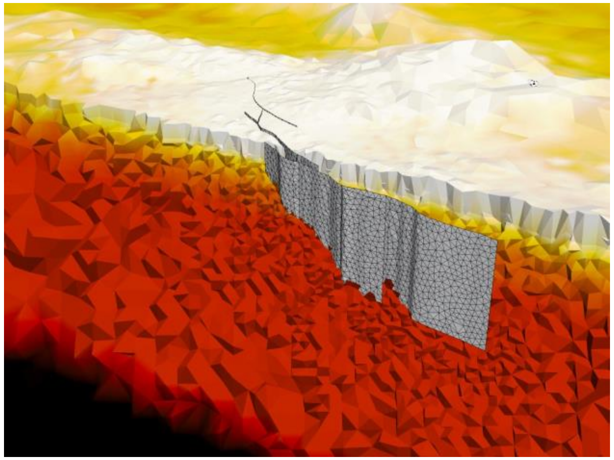
1. What is ExaHyPE
2. ExaHyPE’s component
3. Code Generation
4. Some benchmarks
5. Outlook

Towards an Exascale Hyperbolic PDE Engine

ExaHyPE Goal: a PDE “engine” (as in “game engine”)

- Enable medium-sized interdisciplinary research teams to realize extreme-scale simulations of grand challenges within one year
- Focus on hyperbolic conservation laws and specific numerics
- Concentrate on two specific grand challenges in the project:

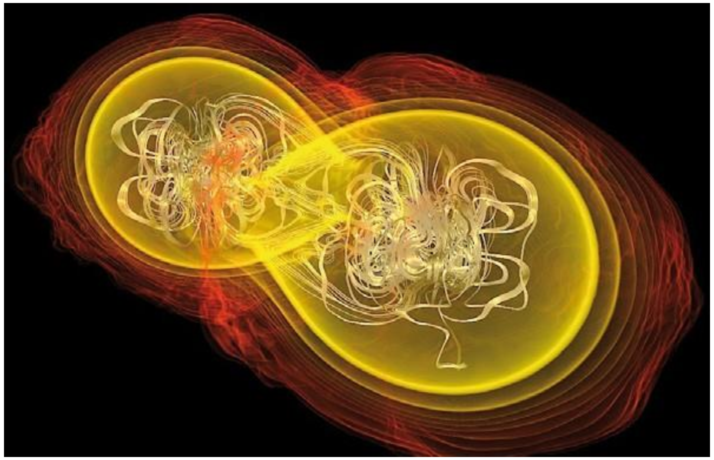




Towards an Exascale Hyperbolic PDE Engine

ExaHyPE Goal: a PDE “engine” (as in “game engine”)

- Enable medium-sized interdisciplinary research teams to realize extreme-scale simulations of grand challenges within one year
- Focus on hyperbolic conservation laws and specific numerics
- Concentrate on two specific grand challenges in the project:



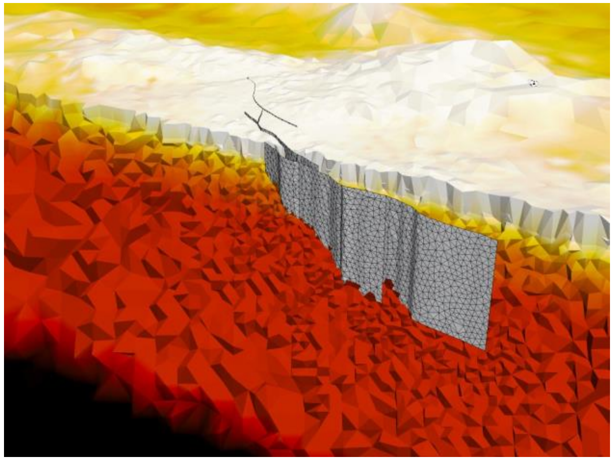
Astrophysics: merger of binary system of neutron stars (FIAS)

Towards an Exascale Hyperbolic PDE Engine

ExaHyPE Goal: a PDE "engine" (as in "game engine")

- Enable medium-sized interdisciplinary research teams to realize extreme-scale simulations of grand challenges within one year
- Focus on hyperbolic conservation laws and specific numerics
- Concentrate on two specific grand challenges in the project:

Seismology: regional earthquake simulation (LMU)



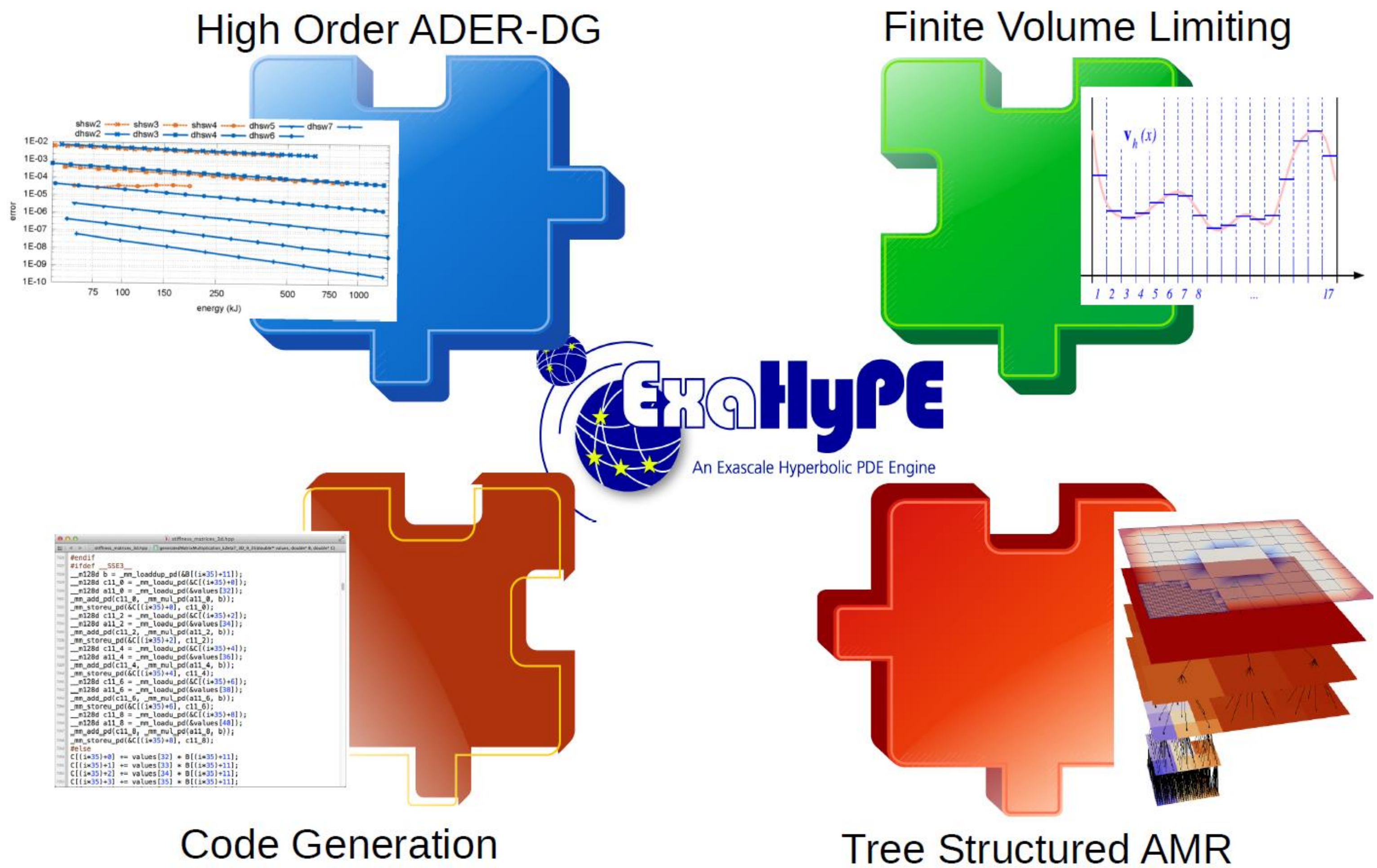
Towards an Exascale Hyperbolic PDE Engine

Requirements for Exascale Algorithms:

- Avoid data-transfer/communication and synchronisation
- Maximize arithmetic intensity and maximize “science per flop”/“science per Watt”
- Dynamic load balancing with lightweight adaptive response

➔ Focus on **High Order Discretisation** and **Adaptivity** in Space and Time

Components of the ExaHyPE Engine



Engine Architecture and Application Interface

Application Layer – user provides:

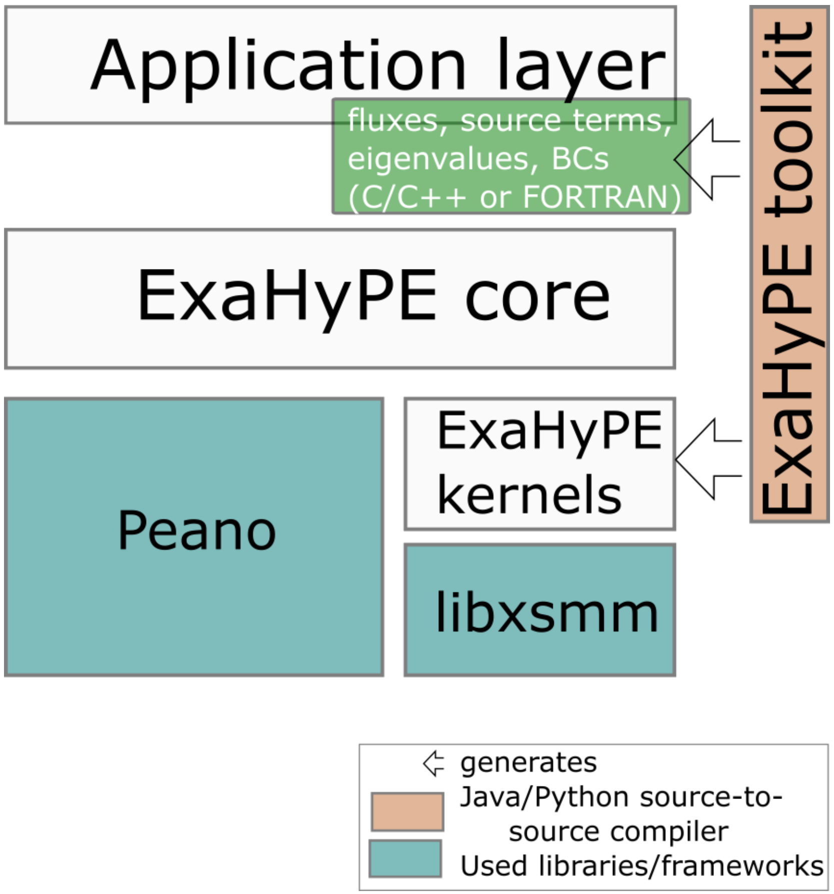
- C/Fortran code for PDEs

ExaHyPE toolkit generates:

- core routines, templates for application-specific functions
- kernels tailored to discretization order, number of quantities, etc.

Peano framework:

- hybrid MPI+Intel TBB parallelism
- data structures for parallel AMR



Code Generation: Optimizations and Constraints

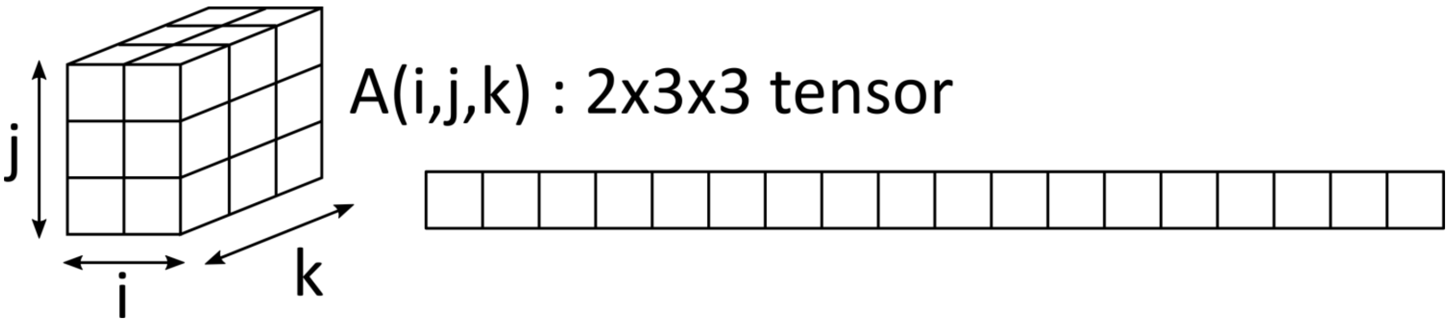
Desired Optimizations:

- Loop vectorization
- Efficient data access
- Matrix-tensor multiplications: Intel's Libxsmm

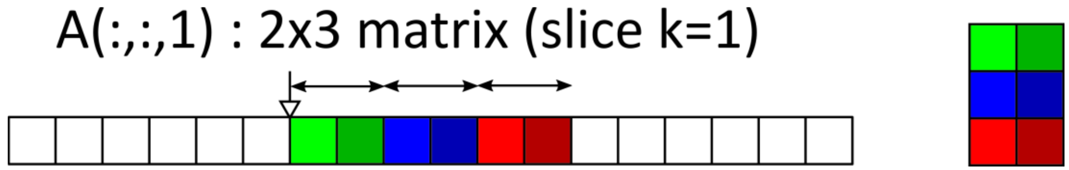
Constraints posed by:

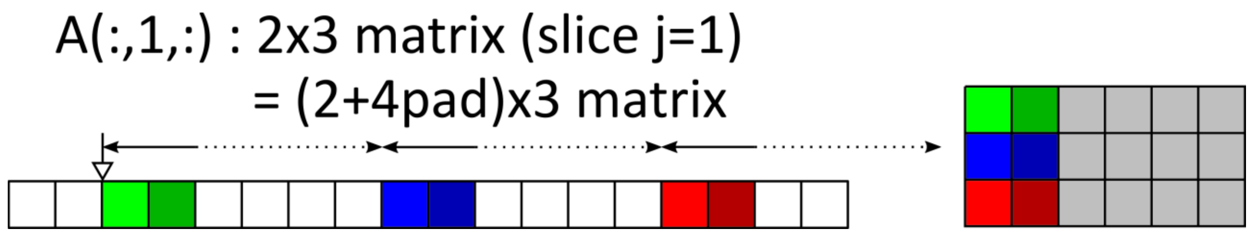
- Memory access: seamless integration with other user code fragments required
- Matrix multiplication: Libxsmm doesn't implement entire BLAS

Libxsmm: Accessing Matrices in Tensor



Fortran notations vs stride access for matrix multiplication:





Benchmark cases

Cases:

- EulerFlow 3D
  - No sources, no NCP
  - 5 quantities

- GRMHD
  - No sources, but NCP
  - 19 quantities

- Z4 Kerr-Schild
  - Sources + NCP
  - 54 quantities

Hardware:

- Intel Xeon E5-2697v3 (HSW)
- Single core measurements

Compiler:

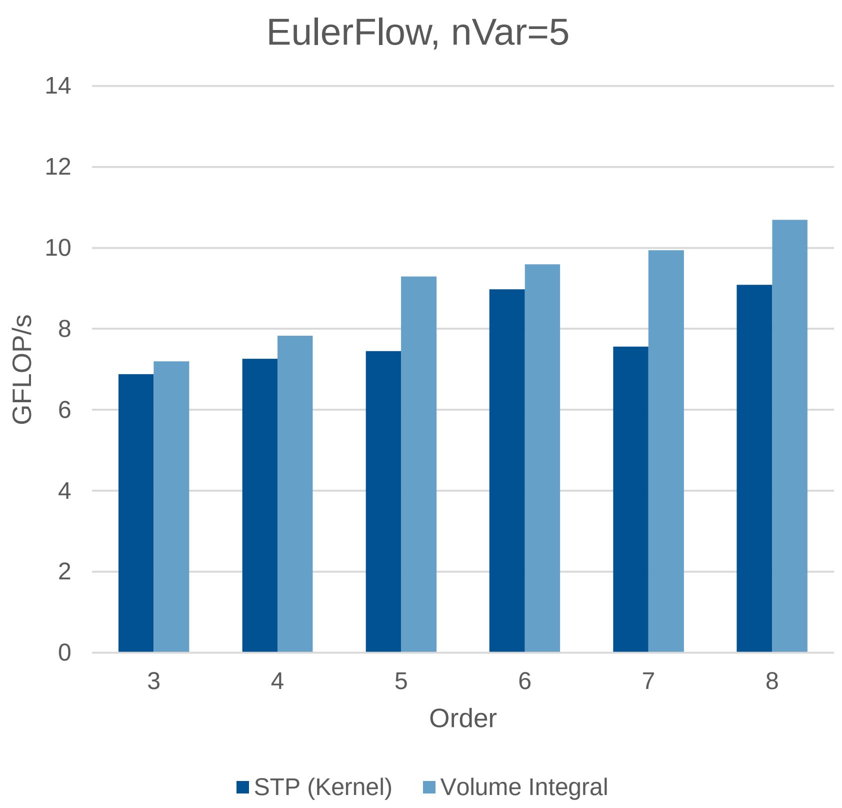
- Intel compiler 16.0.4 for C++ (icpc)
- gfortran v4.9
- Aggressive optimization options

  -fast -fstrict-aliasing -std=c++0x

  -restrict -no-ipo -ip -xCORE-AVX2 -fma

Gflops measurements, single core HSW

- Small number of quantities: scaling with order
- Big number of quantities: plateau instead of further scaling



Probable cause of the plateau in libxsmm

Libxsmm called on sliding slices of the tensor to perform Matrix-Matrix multiplications

*gemm(&A[i*X],&B[0],&C[i*Y]);*

Problem:

A and C slices not reused

↳ A and C not in L1 cache

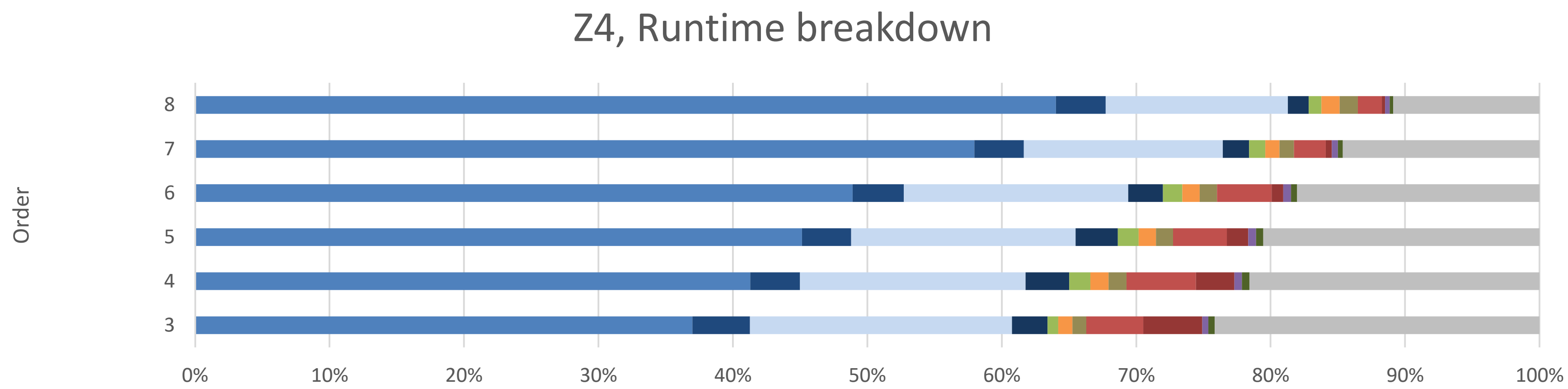
↳ need to be fetched from L2/L3/RAM

↳ bottleneck in performances

Solution:

Customize generated gemms to use L1 prefetching

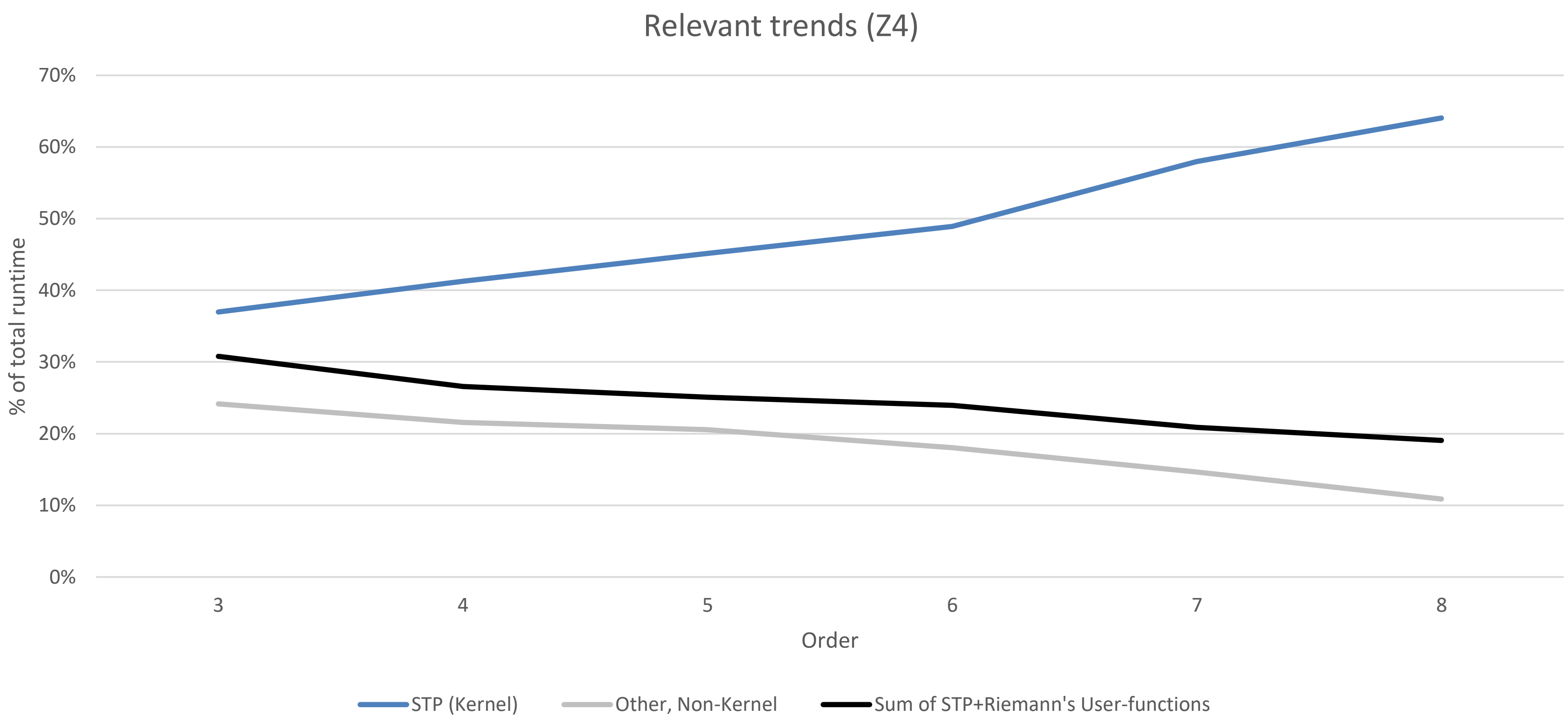
Runtime Breakdown



	3	4	5	6	7	8
STP (Kernel)	36.99%	41.28%	45.14%	48.90%	57.97%	64.03%
STP (NCP)	4.29%	3.71%	3.67%	3.81%	3.66%	3.71%
STP (Flux)	19.47%	16.77%	16.68%	16.70%	14.81%	13.54%
STP (Source)	2.64%	3.25%	3.14%	2.58%	1.97%	1.55%
Boundary Cond.	0.81%	1.57%	1.56%	1.44%	1.20%	0.95%
Volume Integral	1.05%	1.35%	1.28%	1.30%	1.07%	1.35%
Surface Integral	1.05%	1.35%	1.28%	1.30%	1.07%	1.35%
Riemann (Kernel)	4.23%	5.15%	3.99%	4.07%	2.37%	1.78%
Riemann (NCP)	4.40%	2.86%	1.60%	0.85%	0.44%	0.24%
Solution Up.	0.43%	0.58%	0.59%	0.58%	0.45%	0.34%
StableTimeStep	0.49%	0.56%	0.54%	0.46%	0.38%	0.26%
Other, Non-Kernel	24.15%	21.57%	20.54%	18.03%	14.63%	10.89%

Runtime Breakdown trends

- Overhead and user function cost goes down (relatively)
- SpaceTimePredictor becomes more and more dominant



Outlook

- Optimized kernels make ExaHyPE faster than the Fortran prototype from Trento.
- User functions not negligible anymore.
- Bottleneck with libxsmm identified, potential solution proof of concept tested successfully.
- Still need good scaling tests once current problem with generic code identified and solved