A Generic Profiling Infrastructure for the Hyperbolic PDE Solver Engine ExaHyPE
Part II

Fabian Güra

www.exahype.eu

November 2016

The project has received funding from the European Union's Horizon 2020 research and innovation program under grant agreement No 671698 (ExaHyPE).

Introduction

What this presentation is not about:

- **Numerics** deep dive into ADER-DG, Limiting, etc. (first $\sim$ 30 pages of the thesis)
- No beautiful **pictures**, convergence **plots**, application **examples**, . . .
- **Demo session** ;-(

Introduction

What this presentation is not about:

- **Numerics** deep dive into ADER-DG, Limiting, etc. (first $\sim$ 30 pages of the thesis)
- No beautiful **pictures**, convergence **plots**, application **examples**, . . .
- **Demo session** ;-(

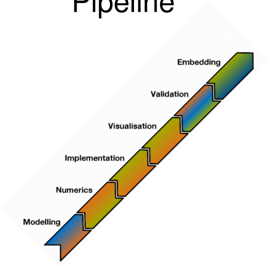
What this presentation is about:

- Review: **Context** and **motivation** behind the project
- **Hardware performance monitoring** (HPM) on x86
- A **generic profiling infrastructure** for ExaHyPE
- Preliminary **profiling results** and two **case studies** focusing on metrics-driven **performance engineering**

Context & Motivation

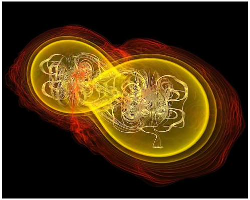
Important aspects in the context of Scientific Computing:

Simulation Pipeline



Exascale Computing

Hyperbolic Balance Laws



ExaHyPE: The Project

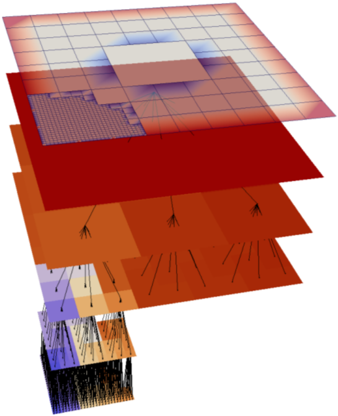
Vision:

- Three pillars of scientific progress:
  Theory, Experiment and **Simulation**
- Programming (exascale) supercomputers is a key challenge
- The ExaHyPE project seeks to address the **software aspect** of supercomputer development
  - Development of new mathematical and algorithmic approaches
  - Initial focus on applications in geo- and astrophysics
  - In correspondence with Europe's **2020 exascale strategy**
- Goal: Become **the engine** for large HCL simulations!

ExaHyPE: The Project

Approach:

- High-order space-time Discontinuous Galerkin method (**ADER-DG**) with a-posteriori FVM based subcell limiting
- Dynamically adaptive Cartesian grids (**AMR**), space filling curves and dynamic load balancing (**Peano**)
- Hardware specific optimization of dominant compute kernels (**libxsmm**)



Adaptive grids in Peano
(via Tobias Weinzierl, 2014)

Profiling for ExaHyPE: Motivation

You can't optimize for what you don't measure!

Profiling provides metrics to

- obtain a **baseline**,
- **guide** and **track progress** of optimization efforts,
- **compare** current status to other state of the art solutions.

We don't need a third one of these on campus!

Profiling and Energy-aware Computing in Modern x86 Systems

The Current Prevalence of x86 in High Performance Computing

Architecture	abs.	rel.
x86	468	93.6%
Power	23	4.6%
SPARC	7	1.4%
Sunway	2	0.4%

(a) CPU Architecture

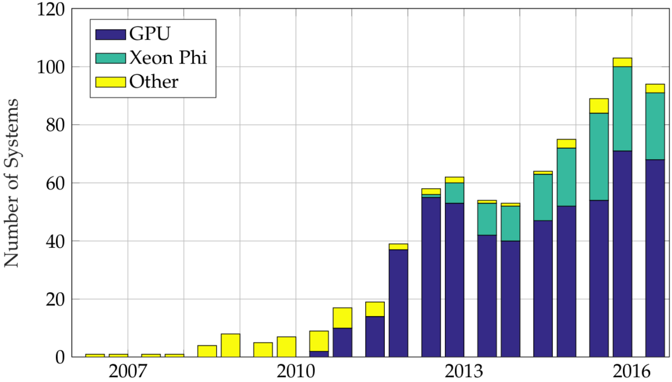
Accelerator	abs.	rel.
None	406	81.2%
GPU	66	13.2%
Xeon Phi	23	4.6%
Other	5	1.0%

(b) Accelerator Cards

Table: Distribution of CPU architectures and accelerator cards of the supercomputers listed in the June 2016 Top500 list. Systems that have both GPUs and Xeon Phi accelerator cards are listed as "Other."

Profiling and Energy-aware Computing in Modern x86 Systems

The Current Prevalence of x86 in High Performance Computing?



Hardware Performance Monitoring on x86 Platforms

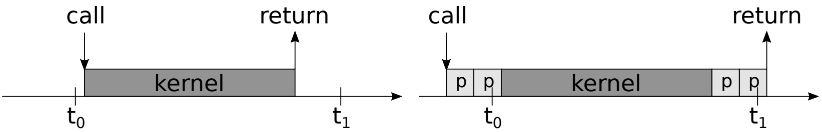
- **History**: 1st gen. **Pentium** (1993) **collects metrics** on interaction between hardware and code.
- Access **undocumented**/proprietary
  $\rightarrow$ **reverse engineering**
- General Principle: **Performance counters** programmable via **model specific registers** (MSRs)
- Today: Separate performance monitoring units (PMUs) for cores and shared resources
- Advantage: **Low overhead** and unintrusive
- Disadvantage: Not part of ISA, i.e. **unstable** and **undocumented**
- Use **libraries** (PAPI, LIKWID, ...)

Energy Monitoring in x86

- Context: **Underprovisioning** of power supply in datacenters
- History: Sandy Bridge (2011) and Bulldozer (2013) support prescription of **dynamic power limits** (RAPL, APM)
- Control task gives rise to **on-chip power estimation**
- Exposed via **MSRs**, separate for CORE, PKG and DRAM
- Good: **Precise** ($\sim \mu J$) and valid ($\rightarrow$ literature)
- Bad: **Low update rate** ($\leq$1 kHz)

Energy Monitoring in x86

Measurements close to the counter update rate:



(a) Naive approach

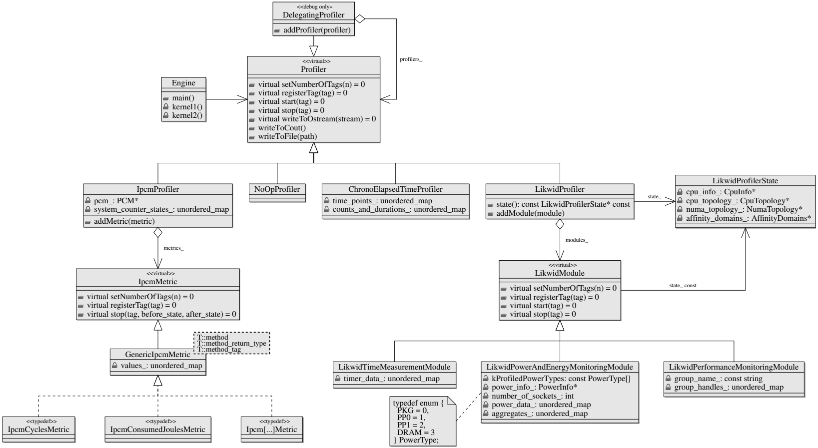
(b) Approach based on repeated counter probing

Figure: Approaches towards measuring power consumption in short code paths using RAPL counters.
Idea and original illustration: Hähnel et al., 2012.

Profiling for ExaHyPE: Design Goals

- Ease of use
- Extensibility
- Low measurement overhead
- Low maintenance effort for the team
- Flexibility towards internal and external analysis

Profiling for ExaHyPE: Architecture



Profiling for ExaHyPE: Instumentation

```
auto p = make_unique<MyProfilerImplementation>();
p->registerTag("region1");
p->start("region1");
// work, work, work, ...
p->stop("region1");
p->writeToOstream(&std::cout);
```

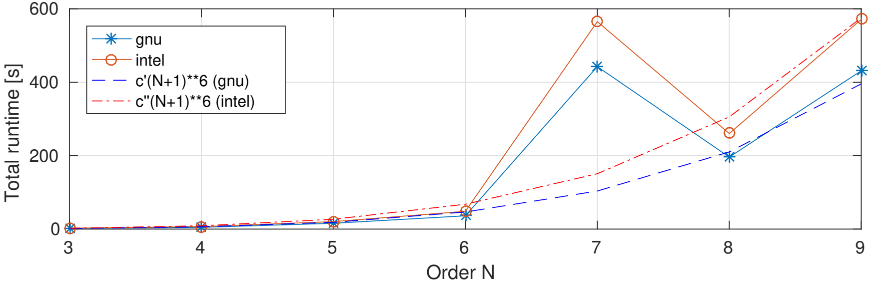
Profiling for ExaHyPE: Metrics

What we can measure at the moment:

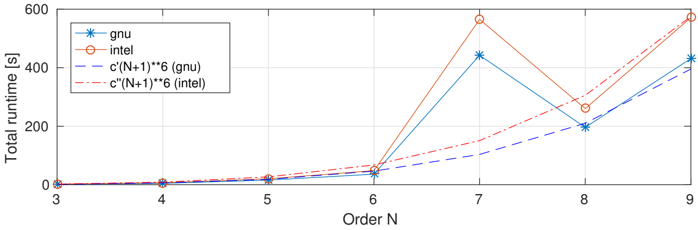
- Kernel call counts
- Cycles and wall clock time
- RAM reads/writes
- **Flop/s** (estimate)
- **Energy consumption** (Core, RAM, package; estimate)
- Cache hits/misses/ratio (I, L2, L3)
- **Cycles lost** due to cache misses (L2, L3)
- Instructions retired
- Branch prediction ratio
- …

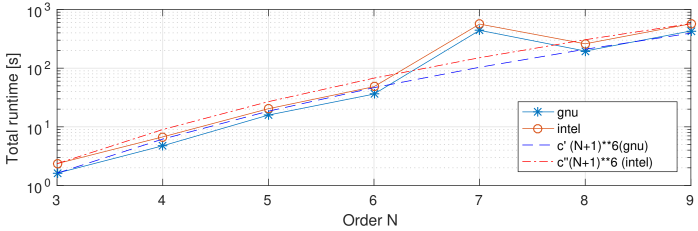
Preliminary Results

- Disclaimer: Generic kernels; for illustrative purposes only
- Setup:
  - CoolMUC2 (Haswell-EP, 2x12 cores @ 2.6 GHz)
  - “Euler200”, 3D, 9^3 cells
- Runtime is $O((N+1)^6)$ in N order of ansatz functions
- “Performance bug” at order seven



Case Study: “The 7th Order Itch”





Case Study: “The 7th Order Itch”

Order	3	4	5	6	7	8	9
GNU	2.0	5.1	12.3	25.7	292.6	79.0	131.9
Intel	2.9	7.6	16.7	35.8	371.9	104.7	178.0

Table: Duration in ms of a complete ADER-DG element update in the Euler200 model.

Findings:

- Anomaly is observable independent of compiler
- Bug does not seem to be in Peano, but in the kernels or the “data transfer logic”

Case Study: “The 7th Order Itch”

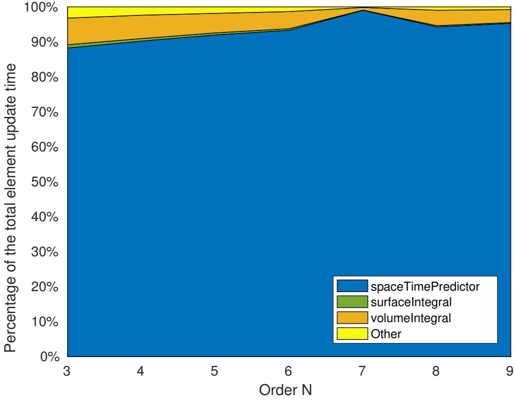
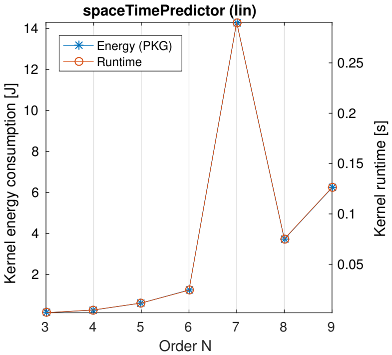
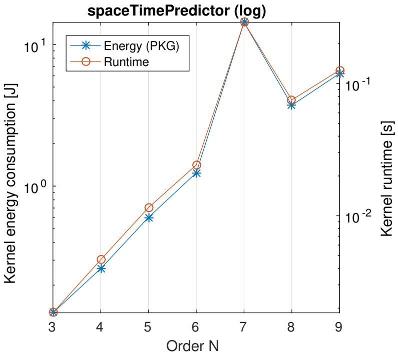


Figure: Share of the individual ADER-DG kernels (generic, unoptimized) with respect to the total element update time (stacked).

Case Study: “The 7th Order Itch”

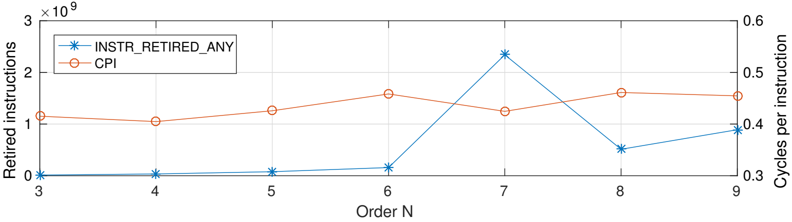




Findings:

- Element update time is dominated by space-time predictor
- Space-time predictor indeed has a performance issue for order seven

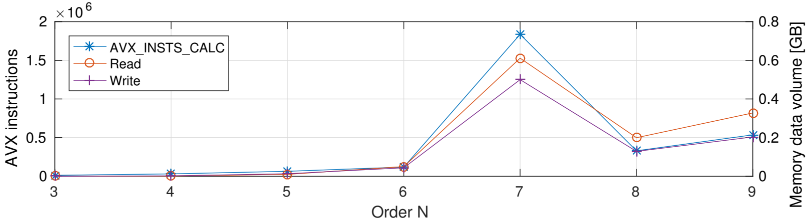
Case Study: “The 7th Order Itch”



Findings:

- There is no intrinsic performance problem at order seven
- There is simply more “work” to do...

Case Study: “The 7th Order Itch”



Findings:

- There is no intrinsic performance problem at order seven
- There is simply more “work” to do
- Confidence: This is not just measurement fluctuation! It has a fundamental cause inside the application code!

Case Study: "The 7th Order Itch"

Observation:

- Binary size constant irrespective of order/number of variables
- Side note: No surprise…
- Conclusion: Parts of the space-time predictor kernel get executed multiple times

Up to this point not a single line of code has been changed and no recompilations was necessary!

Case Study: “The 7th Order Itch”

Observation:

- Binary size constant irrespective of order/number of variables
- Side note: No surprise…
- Conclusion: Parts of the space-time predictor kernel get executed multiple times
- Oh wait…How about the fixed-point iteration inside the kernel?

Up to this point not a single line of code has been changed and no recompilations was necessary!

Case Study: “The 7th Order Itch”

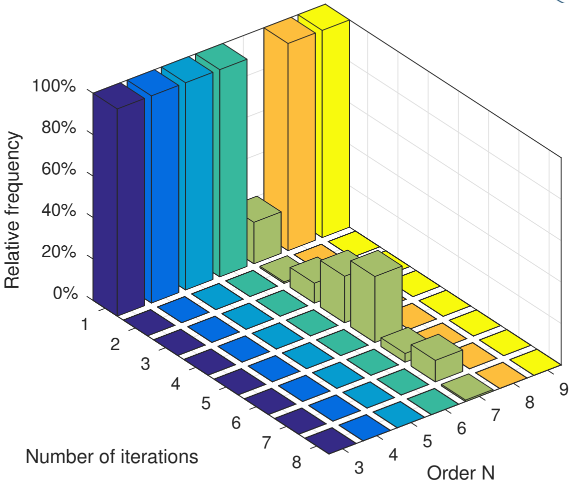


Figure: Relative frequencies of the number of iterations in the “Picard loop” fixed-point iteration within the space-time predictor kernel.

Case Study: “Finding the Juciest Fruits”

Use profiling to guide optimization efforts

- Disclaimer: Generic kernels aim for correctness, flexibility
- Aim for the low-hanging fruits
- Case study illustrates typical process
- Leitmotif: “Measure first, then optimize!”

Case Study: "Finding the Juciest Fruits"

Order	3	4	5	6	7	8	9
GNU	65.1	75.2	81.9	85.5	96.9	89.0	90.0
Intel	58.8	75.7	83.6	87.1	96.1	87.5	90.5

Table: Percentage of time spent in the ADER-DG kernels of the simulation loop and the total wall clock time.

Findings:

- Most of the time is actually spent in the compute kernels
- Peano overhead seems to be negligible
- Conclusion: Focus optimization efforts on kernels

Case Study: “Finding the Juciest Fruits”

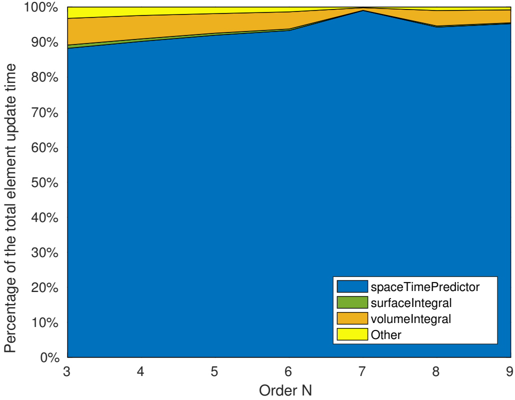
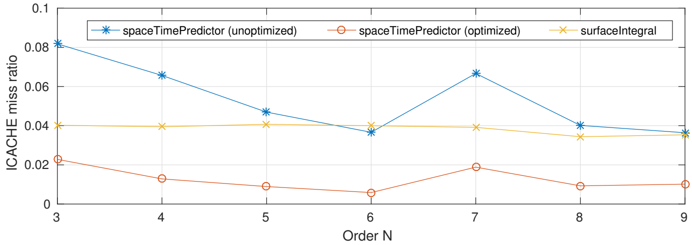


Figure: Share of the individual ADER-DG kernels (generic, unoptimized) with respect to the total element update time (stacked).

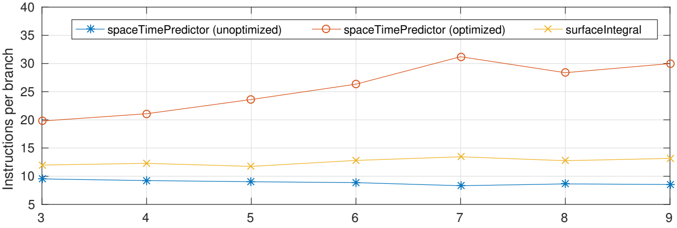
Case Study:
“Finding the Juciest Fruits”



Findings:

- Relatively high instruction cache miss rate compared to other kernels
- Typical cause:
  - High number of branching instructions in code
  - Complex ranching pattern → branch prediction fails

Case Study: "Finding the Juciest Fruits"



Findings:

- Average number of instructions per branch is low
- May cause instruction pipeline stall
- Conclusion:
  - Eliminate unnecessary branching
  - Help the compiler to optimize them away

Case Study: “Finding the Juciest Fruits”

while $r < R$ **and** $\Delta^2 < \varepsilon^2$ **do**

- Copy result of previous iteration: $\left[\hat{\boldsymbol{q}}^{K,i,\mathrm{old}}\right]_{\boldsymbol{n}lv} := \left[\hat{\boldsymbol{q}}^{K,i,\mathrm{new}}\right]_{\boldsymbol{n}lv}$.
- Copy contribution of S-III to right-hand side: $[\hat{\boldsymbol{r}}]_{l\boldsymbol{n}v} := [\hat{\boldsymbol{r}}_0]_{l\boldsymbol{n}v}$.
- **for** $l \in \mathcal{N}$ **do** // *time DOFs*
  - **for** $\boldsymbol{n} \in \boldsymbol{\mathcal{N}}$ **do** // *spatial DOFs*
    - Evalute fluxes (eq. (2.91)): $\left[\hat{\boldsymbol{F}}\right]_{l\boldsymbol{n}dv} := \left[\boldsymbol{F}\left(\left[\hat{\boldsymbol{q}}^{K,i,\mathrm{old}}\right]_{\boldsymbol{n}l}\right)\right]_{dv}$.
    - Evaluate sources (eq. (2.93)): $[\hat{\boldsymbol{s}}]_{l\boldsymbol{n}v} := \left[\boldsymbol{s}\left(\left[\hat{\boldsymbol{q}}^{K,i,\mathrm{old}}\right]_{\boldsymbol{n}l}\right)\right]_{v}$.
  - **end**
  - **for** $\boldsymbol{n} \in \boldsymbol{\mathcal{N}}$ **do** // *spatial DOFs*
    - Evaluate S-IV and subtract from right-hand side (eq. (2.91)):

$$[\hat{\boldsymbol{r}}]_{l\boldsymbol{n}v} \mathrel{:\overset{-}{=}} J_{\mathcal{T}_i}\hat{\omega}_l \sum_{d\in\mathcal{D}}\left(\frac{1}{\left[\Delta\boldsymbol{x}^K\right]_d}\hat{\omega}_{\boldsymbol{n}_{0:d-1,d+1:D-1}} \cdots \right.$$
$$\left. \cdots \sum_{n'_d\in\mathcal{N}}\left([\boldsymbol{K}]_{\boldsymbol{n}_d n'_d}\left[\hat{\boldsymbol{F}}\right]_{l[\boldsymbol{n}_{0:d-1},n'_d,\boldsymbol{n}_{d+1:D+1}]dv}\right)\right).$$

    - Evaluate S-V and add to right-hand side (eq. (2.80)): $[\hat{\boldsymbol{r}}]_{l\boldsymbol{n}v} \mathrel{:\overset{\pm}{=}} J_{\mathcal{T}_i}\hat{\omega}_{\boldsymbol{n}}\hat{\omega}_l\,[\hat{\boldsymbol{s}}]_{l\boldsymbol{n}v}$.
  - **end**
  - **for** $\boldsymbol{n} \in \boldsymbol{\mathcal{N}}, l \in \mathcal{N}$ **do** // *space-time DOFs*
    - Multiply with inverse iteration matrix: $\left[\hat{\boldsymbol{q}}^{K,i,\mathrm{new}}\right]_{\boldsymbol{n}lv} := \frac{1}{\hat{\omega}_{\boldsymbol{n}}}\left[\tilde{\boldsymbol{K}}\right]_{ll'}[\boldsymbol{r}]_{l'\boldsymbol{n}v}$.
  - **end**
  - Update squared element-wise residual (eq. (2.95)): $\Delta^2 := \sum_{\boldsymbol{n}\in\boldsymbol{\mathcal{N}}}\sum_{l\in\mathcal{N}}\sum_{v\in\mathcal{V}}\left(\left[\hat{\boldsymbol{q}}^{K,i,\mathrm{new}}\right]_{\boldsymbol{n}lv} - \left[\hat{\boldsymbol{q}}^{K,i,\mathrm{old}}\right]_{\boldsymbol{n}lv}\right)^2$.
- **end**

end

Case Study: “Finding the Juciest Fruits”

Findings:

- Branching hidden in loops vectorization and overlap checks
- Problem: Loop count is available only in different compilation unit
- Solution: Templatization
- Problem: Imperfect loop nests
- Solution: Loop unwinding to help compiler heuristics to “understand” the loops
- Problem: Pointer aliasing in C(++)
- Solution: Use __restrict__ where applicable

Case Study: “Finding the Juciest Fruits”

while $r < R$ **and** $\Delta^2 < \varepsilon^2$ **do**

  Swap $\left[\hat{\boldsymbol{q}}^{K,i,\text{old}}\right]_{\boldsymbol{n}lv}$ and $\left[\hat{\boldsymbol{q}}^{K,i,\text{new}}\right]_{\boldsymbol{n}lv}$.

  Copy contribution of S-III to right-hand side: $[\hat{\boldsymbol{r}}]_{l\boldsymbol{n}v} := [\hat{\boldsymbol{r}}_0]_{l\boldsymbol{n}v}$.

  for $l \in \mathcal{N}, \boldsymbol{n} \in \boldsymbol{\mathcal{N}}$ **do** *// space-time DOFs*

    Evalute fluxes (eq. (2.91)): $\left[\hat{\boldsymbol{F}}\right]_{l\boldsymbol{n}dv} := \left[\boldsymbol{F}\left(\left[\hat{\boldsymbol{q}}^{K,i,\text{old}}\right]_{\boldsymbol{n}l}\right)\right]_{dv}$.

    Evaluate sources (eq. (2.93)): $[\hat{\boldsymbol{s}}]_{l\boldsymbol{n}v} := \left[\boldsymbol{s}\left(\left[\hat{\boldsymbol{q}}^{K,i,\text{old}}\right]_{\boldsymbol{n}l}\right)\right]_{v}$.

  end

  for $l \in \mathcal{N}, \boldsymbol{n} \in \boldsymbol{\mathcal{N}}$ **do** *// space-time DOFs*

    Evaluate S-IV and add to right-hand side (eq. (2.91)):

$$[\hat{\boldsymbol{r}}]_{l\boldsymbol{n}v} :\overset{-}{=} J_{\mathcal{T}_i}\hat{\omega}_l \sum_{d\in\mathcal{D}} \left(\frac{1}{\left[\Delta \boldsymbol{x}^K\right]_d} \frac{1}{\hat{\omega}_{\boldsymbol{n}_d}} \sum_{n'_d \in \mathcal{N}} \left([\boldsymbol{K}]_{\boldsymbol{n}_d n'_d} \left[\hat{\boldsymbol{F}}\right]_{l[\boldsymbol{n}_{0:d-1}, n'_d, \boldsymbol{n}_{d+1:D+1}]dv} \right) \right).$$

  end

  for $l \in \mathcal{N}, \boldsymbol{n} \in \boldsymbol{\mathcal{N}}$ **do** *// space-time DOFs*

    Evaluate S-V and add to right-hand side (eq. (2.80)):

    $[\hat{\boldsymbol{r}}]_{l\boldsymbol{n}v} :\overset{+}{=} J_{\mathcal{T}_i}\hat{\omega}_l\,[\hat{\boldsymbol{s}}]_{l\boldsymbol{n}v}$.

  end

  for $l \in \mathcal{N}, \boldsymbol{n} \in \boldsymbol{\mathcal{N}}$ **do** *// space-time DOFs*

    Multiply with inverse iteration matrix: $\left[\hat{\boldsymbol{q}}^{K,i,\text{new}}\right]_{\boldsymbol{n}lv} := \left[\tilde{\boldsymbol{K}}\right]_{ll'} [\boldsymbol{r}]_{l'\boldsymbol{n}v}$.

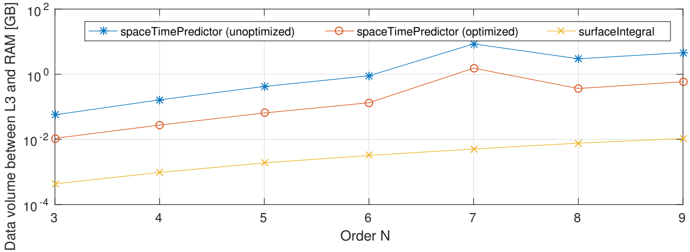
  end

  Update squared element-wise residual (eq. (2.95)):

  $\Delta^2 := \sum_{\boldsymbol{n}\in\boldsymbol{\mathcal{N}}} \sum_{l\in\mathcal{N}} \sum_{v\in\mathcal{V}} \left(\left[\hat{\boldsymbol{q}}^{K,i,\text{new}}\right]_{\boldsymbol{n}lv} - \left[\hat{\boldsymbol{q}}^{K,i,\text{old}}\right]_{\boldsymbol{n}lv}\right)^2$.

end

Case Study: “Finding the Juciest Fruits”



Findings:

- Beginning of loop: Replace copy “new $\rightarrow$ old” by swap
- Careful derivation: Some 3/4D Gauss-Legendre weights can be canceled
- Reduce pressure on registers and caches

Case Study: “Finding the Juciest Fruits”

Result:

- Simple, unintrusive changes in places where it matters
- Significant performance gains
  - Runtime: $\sim$ 3X
  - Energy: $\sim$ 2.5X
- “Measure first, then optimize!”

Conclusion

Key aspects:

1. Exascale Computing is about both hardware and **software**
2. ExaHyPE as an **answer to future challenges** in Scientific Computing
3. Profiling is important:
   - Guidance for optimization efforts: **Measure, then optimize!**
   - Debugging of (performance) bugs without manual changes: **Value your time!**

A Generic Profiling Infrastructure for the Hyperbolic PDE Solver Engine ExaHyPE
Part II

Fabian Güra

www.exahype.eu

November 2016

The project has received funding from the European Union's Horizon 2020 research and innovation program under grant agreement No 671698 (ExaHyPE).