BoF on “Emerging Trends in HPC Systems and Application Modernization”

SC16, November 15th, Salt Lake City

Basics

The Birds of a Feather session took place on November 15 between 5:15 and 7:00 pm. Rough number of participants in the audience was 50 – 55.

The short intro presentation and the four “feature presentations” are appended to this report. We did ask to have the discussion with the audience after the presentations.

Discussion session

Q: One solution that was not discussed in the presentations is the role of smart runtimes that receive an abstract description of a task (like compute the integral over this kernel”) and then perform the required computations in a way optimal for the available system.

A (Marc Snir): While libraries can certainly hide differences between systems, the fact remains that different systems are best suited for different tasks/algorithms.

A (Michael Bader): Moreover, libraries are hard to design in a way that finds the right balance between user needs and system capabilities. The key problem is to find the right abstraction layer here.

A (Viktor Lee): An additional complication here are IP considerations and proprietary implementations.

Q: Code modernization should not really be a topic anymore – several large programs have already targeted this topic. Why is it still important, and shouldn’t we evolve systems to match the applications instead?

A (Viktor Lee): In systems & component design, we are facing fundamental limits which make the “free lunch” scaling of delivered performance like in the past impossible. The way forward is to evolve architectures, which in turn influences software.

A (Marc Snir): Basic economics forces HPC system designers and vendors to make compromises; the “beautiful systems” custom-engineered for HPC workloads are a thing of the past.

A (Michael Bader): Development of HPC applications usually starts with smaller configurations; scaling these up to Exascale in itself requires changes to the code.

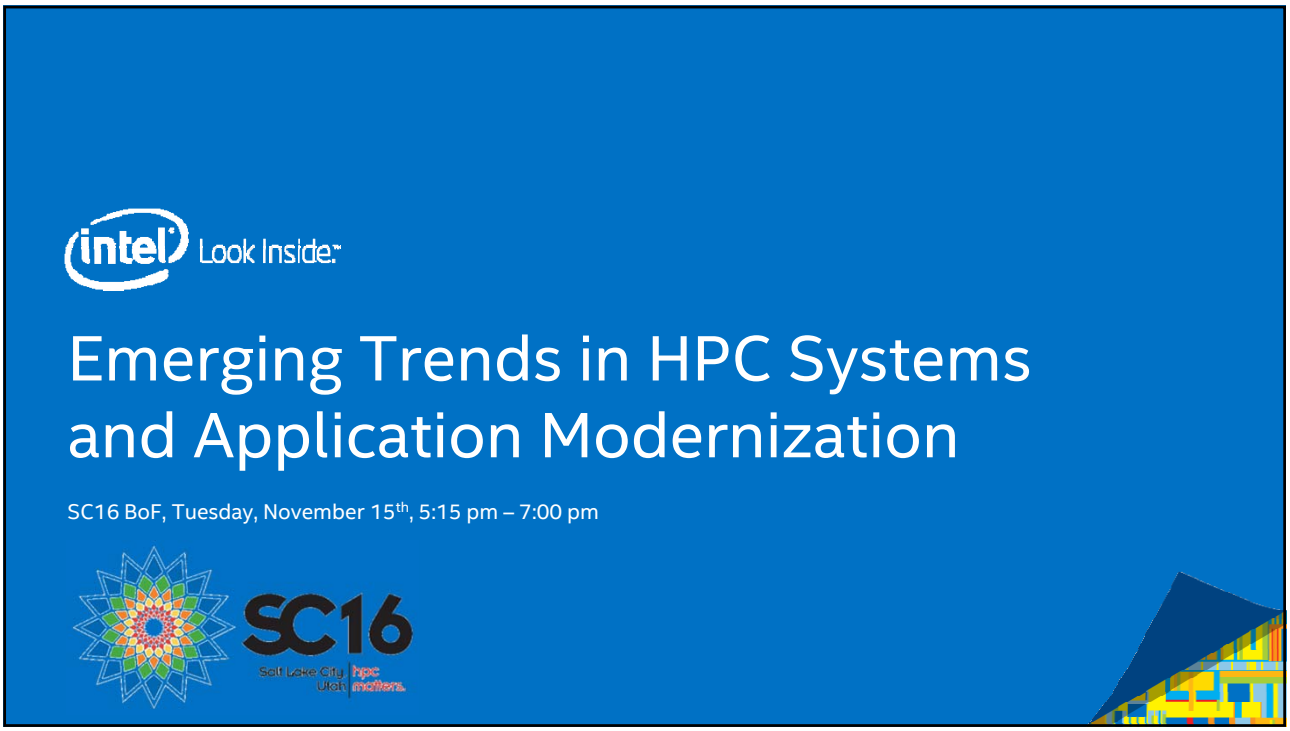
A (Christian Simmendinger): A key aspect is to differentiate between frameworks (which have to be adapted to each system) and the applications themselves, which ideally should be written only once.

Q: The pressure on the application side seems to be too high here – one important problem is the “memory wall”. Are there ways to work around this one and provide a much higher Byte/Flop ration on future systems?

A (Marc Snir): It is quite logical to see cores being used with low efficiency. Cores are comparatively cheap to implement and scale – memory systems are much more expensive. And, having cores over-designed for the available memory subsystem actually makes sense, since this ensures that the memory system itself is used efficiently.

A (Viktor Lee): Substantial improvements in memory bandwidth are very costly, and they run against limits like # of I/O pins available and power use of high-bandwidth memory. We very probably have to live with what is possible (and what we have).

A (Marc Snir): It is better to think of efficient use of a system in terms of how well one uses the memory system, and not the cores. Compute efficiency (wrt. peak compute performance) can be a misleading metric here. One can argue that HPC customers really pay for is the memory system and its delivered latency/bandwidth, and the cores are an “extra”.



Topic

Trends in HPC system architecture

- Complex, multi-level memory hierarchies
- Small fast and large, non-volatile memory
- Increasing number of cores
- Integration of high-performance interconnects

Applications will have to evolve to take advantage of these

- Many challenges around (performance) portability and code complexity

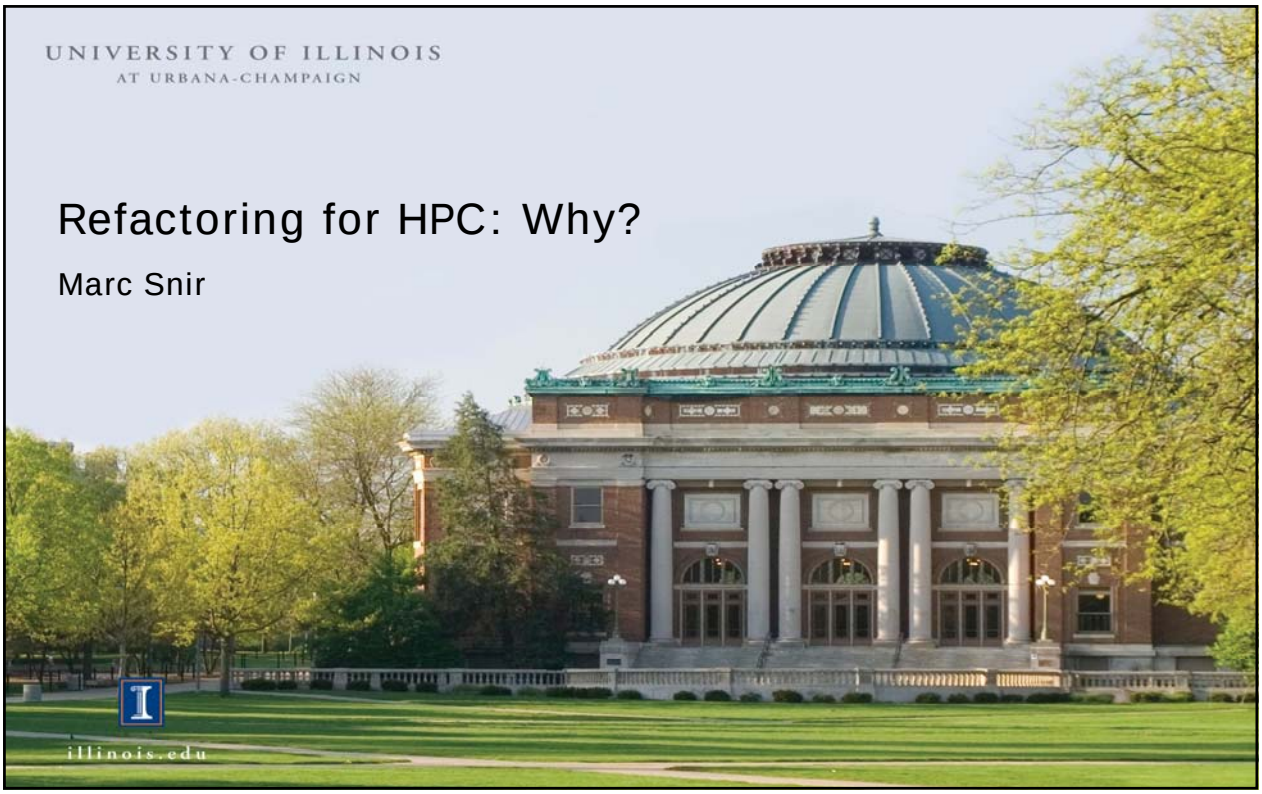
Outline

Presentations by

- Marc Snir, University of Illinois
- Christian Simmendinger, T-Systems Solutions for Research
- Michael Bader, Technical University of Munich
- Victor Lee, Intel Labs

Interactive discussion with the audience

- Moderators: Hans-Christian Hoppe & Marie Sawley, Intel



Computers are Becoming more Complex

- Very large thread count (100's-1000's peer node)
- Heterogeneous cores
- Multiple memories
  - GPU/CPU (IBM)
  - slow (persistent), fast, very fast (Intel)
- Power management & asynchrony

- Good reasons to believe that this trend will continue

Computers are Becoming More Different from each other

- CPU+GPU vs. KNL
- ARM/Power/X86
- New accelerators (FPGAs, ML…)
- Coral: 5k nodes vs. 50k nodes; 100's of threads per nodes vs. 1000's of threads per node.

- No reason to expect convergence

Why is HW Becoming More Complicated?

- Limited opportunities for cost/performance improvements within current paradigm: "The end of Moore's Law"
  - The economics of Moore's Law: The number of devices per chip that provide best cost/performance double every two years (or so)
  - The politics of Moore's Law: All chip manufacturers and their suppliers follow the same technology trajectory

Moore's Law

- Dennard scaling (up to 2004): Moore's Law essentially requires one thing: improvements in lithography.
- Since 2004:
  - No improvement in clock speed
  - Further scaling down required a stream of new inventions: new materials, new transistor shapes
  - ☞Significant increases in cost of production lines and cost per chip
  - ☞Significant industry consolidation to achieve economies of scale.

The End?

- Consolidation has reached an end
- The feature size that provides best cost/performance is not decreasing
- The best possible feature size is a couple of generations away
- No new technology is likely to emerge in less than a decade
- ITRS was disbanded: Vendors are not following anymore the same trajectory

Not the End

- There are still significant gains to be achieved by moving away from general purpose processors (fat CPUs)
  - throughput vs. latency computing
  - near-memory computing
  - scratchpads vs. caches
  - simplified memory management
  - specialization

Specialization

- It has always been the case that a specialized architecture can improve performance by orders of magnitude – but they have always been a niche business; why should it be different now?
  - The have not always been a niche: Cray 1, APS…; they were killed by the “invasion of the killer micros”
  - The world has changed

The New World

- Specialization is more viable if general-purpose systems evolve more slowly
- There is a lot of investments in accelerating the design of SoCs by using higher-level modules, standard interfaces and higher-level design tools
  - Much driven by the mobile world or the military
- There are multiple technologies at the packaging level that facilitate the integration of multiple “chiplets”
- FPGAs are becoming more capable
- “Dark silicon” reduces the energy cost of specialized accelerators

Software Consequences

- ”Write once, run everywhere” is less and less an option for HPC code
  - Performance portability is on the horizon … and will continue to be there
- Codes do not need to be prepared for the next architecture; they need to be prepared for more frequent change

A Lot of "Conventional" Software Engineering

- Modularity, good layering, separation of concerns, documentation, encapsulation of low-level constructs…
- A better use of tools
  - Autotuning
- Better performance engineering

Performance Engineering

- Documentation captures the logic of the code; is there documentation that captures the performance characteristics?
  - Is there a performance model that can predict where will bottlenecks arise on a future system?
- Code tuning is a sequence of experiments
  - How are these experiments planned?
  - Where is the lab notebook?
  - Is there a memory of what worked and what failed and why in the previous set of experiments?

Refactoring for Performance

- Code tuning is a refactoring: A change in code syntax that does not affect semantics.
  - Can we use refactoring tools to facilitate code tuning?
  - Keep track of code evolution
  - Use refactoring tool to reconcile the evolution of different branches
  - Use refactoring tools to modify code in a structured way

Abstract Data Layout

- Data movement (memory to CPU, memory to memory, node to node) is, increasingly, the main performance bottleneck in modern systems
- Data movement overhead is determined by data layout & data distribution
- Different layouts and distributions are needed on different systems (or different components of the same system)
- Need to separate the logical data structure (array) from its instantiation (stored by blocks, distributed block cyclic…)

T · · Systems · · ·

The impact of ManyCore on the critical path in communication

Emerging Trends in HPC Systems and Application Modernization

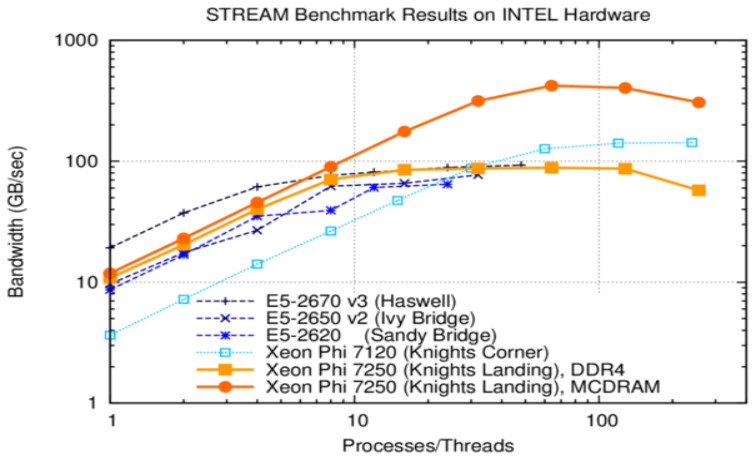
Christian Simmendinger, T-Systems SfR

ManyCore - Trends

Emerging Trends in HPC Systems: ManyCore Architectures

- Exascale
  - 10.000-100.000 nodes,10.000-100.000 threads/node
  - 10^9 threads.
  - MPI + X , PGAS + MPI, PGAS + X, …
- Heterogenous ManyCore Architectures
  - Power efficient
  - Weak cores.
  - Moderate bandwith per core / thread

ManyCore - Bandwidth

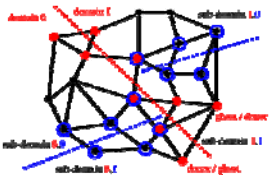


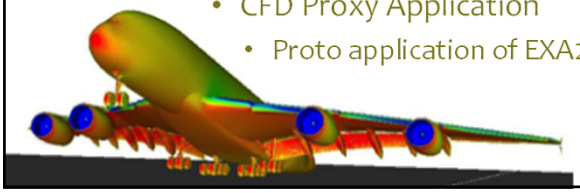
http://www.karlrupp.net/

CFD-Proxy

Application Modernization in CFD

- CFD Application
  - Today: 50M mesh points
  - In ten years: 500-1000M
- ExaScale Computers
  - 10M cores
  - Hence 100 mesh points per core
- CFD Proxy Application
  - Proto application of EXA2CT (FP7)





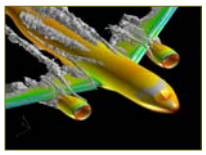
T-Systems

DLR Deutsches Zentrum für Luft- und Raumfahrt e.V. in der Helmholtz-Gemeinschaft

CFD Proxy

CFD-Proxy Application

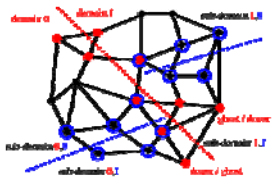
- Multithreaded OpenMP/MPI/GASPI calculation of green gauss gradients for aircraft mesh with ghost cell exchange and subsequent flux computation.
- Unstructured mesh with geometric multigrid.
- Goal: Evaluate threading models, strong scaling communication patterns.
- Strong scaling benchmark with
  ~100-200 cells/thread domain.
  (SOTA ~ 5.000-10.000 cells/MPI domain)



CFD Proxy

Threading model

- 2-stage domain decomposition with SPMD thread domains
- One-sided faces between thread domains.
- Mutual dependencies between thread domains.

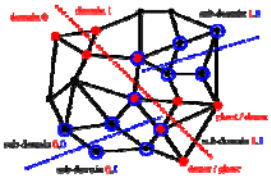


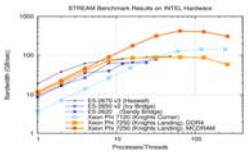
2-level decomposition (a vertex corresponds to a cell, an edge to a face)

CFD Proxy

Considerations for strong scaling

- A good threading model
- Point/face – reordering, outer to inner.
- Data locality considerations for packing
- Multithreaded packing of linear buffer
- Multithreaded send, last contributor sends.
- Mimimize time spent in critical path of communication.

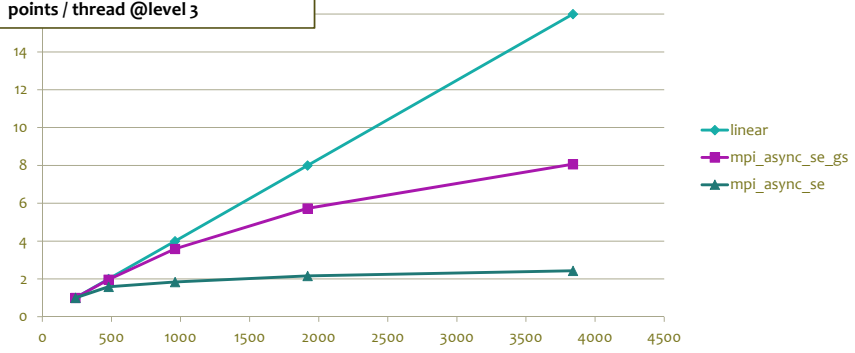




Multithreaded packing - MPI

F6 2 mio mesh points, 3 V Multigrid
- 520 mesh points / thread @level 1
- 70 mesh points / thread @level 2
- 10 mesh points / thread @level 3

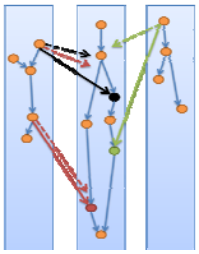
To pack (gather/scatter,gs) or not to pack



CFD Proxy

Considerations for strong scaling

- Minimal communication layer overhead.



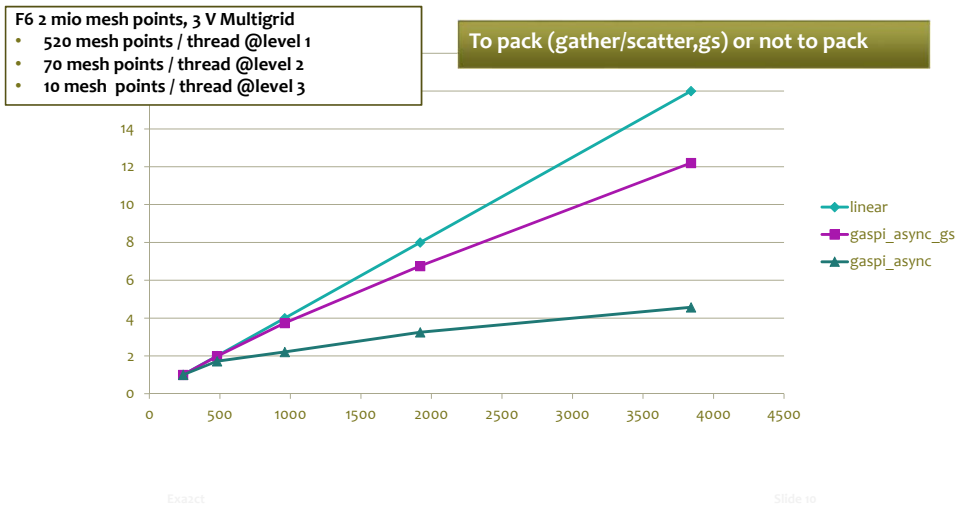
GASPI (PGAS API)

- Bundled (notified) communication with ~zero overhead. (no match making, no book keeping, thin abstraction layer on top of hardware queues, distributed locks)

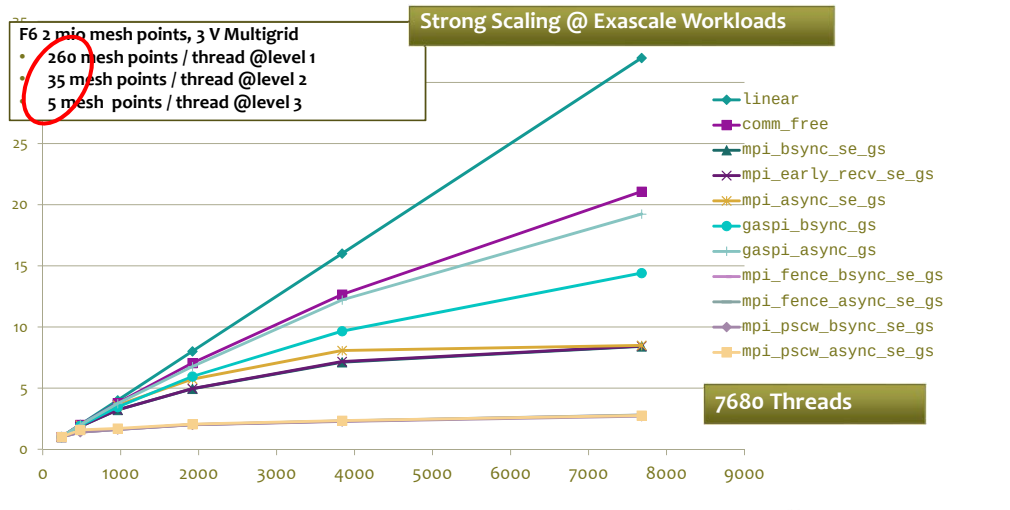
GASPI notification pattern.

- Async. double buffered notified communication
- Best MPI implementation as emulation of GASPI notification pattern via MPI_Waitany.

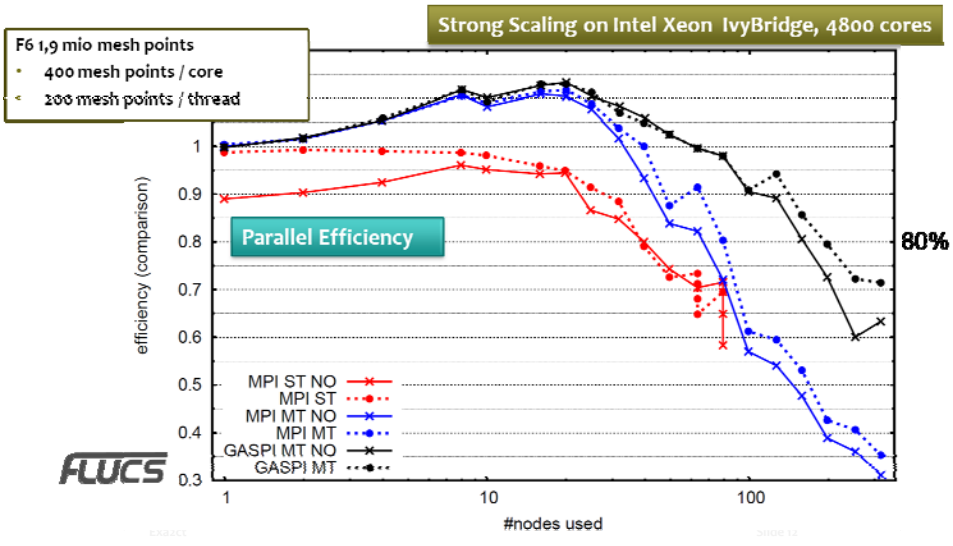
Multithreaded packing GASPI



CFD-Proxy on Xeon Phi



Back in the real world …

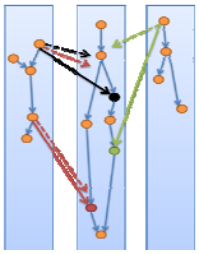


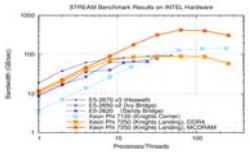
Conclusion

On ManyCore architectures minimal communication time in the critical path requires

- Fast hardware (low latency/high bandwidth)
- Minimal communication overhead.
- Asynchonous bundled communication.
- Multithreaded read/write.
- Multithreaded packing.

- **Contrary to common belief, strong scaling beyond the currently established SOTA appears as doable.**





Backup

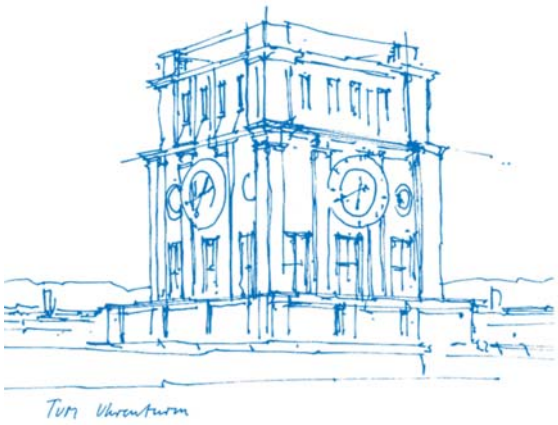
Towards an Exascale Hyperbolic PDE Engine

BoF Session on Emerging Trends in HPC Systems and Application Modernization

Michael Bader
Technical University of Munich

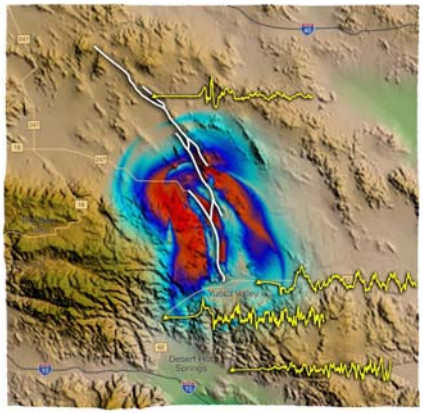
SC16

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



Part I

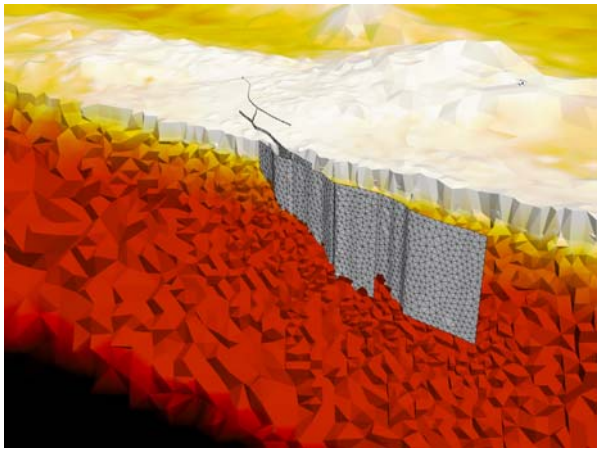
Earthquake Simulation with SeisSol

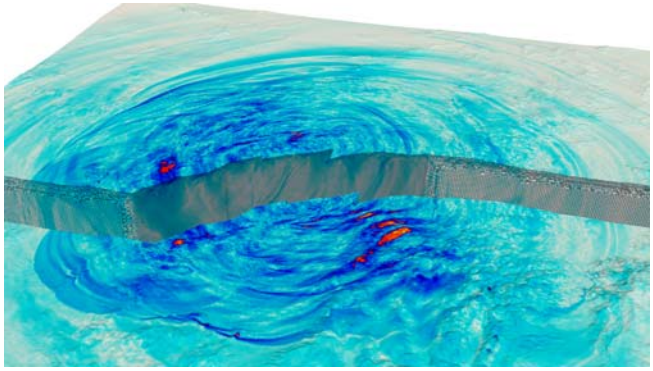


Dynamic Rupture Simulations:

- multiphysics simulation of dynamic rupture, resulting seismic waves and resulting ground motion
- goal: physics-based rupture simulation, insight into earthquake processes and resulting ground shaking

Earthquake Simulation with SeisSol





SeisSol – ADER-DG for seismic simulations (www.seissol.org):

- adaptive tetrahedral meshes (static refinement at fault and near surface)
  $\rightarrow$ complex geometries, heterogeneous media, multiphysics
- complicated fault systems with multiple branches
  $\rightarrow$ non-linear multiphysics dynamic rupture simulation
- ADER-DG: high-order discretisation in space and time

SeisSol in a Nutshell – ADER-DG

Update scheme

$$
\begin{aligned}
Q_k^{n+1} = Q_k - \frac{|S_k|}{|J_k|} M^{-1} \Bigg(& \sum_{i=1}^{4} F^{-,i} I(t^n, t^{n+1}, Q_k^n) N_{k,i} A_k^+ N_{k,i}^{-1} \\
& + \sum_{i=1}^{4} F^{+,i,j,h} I(t^n, t^{n+1}, Q_{k(i)}^n) N_{k,i} A_{k(i)}^- N_{k,i}^{-1} \Bigg) \\
& + M^{-1} K^\xi I(t^n, t^{n+1}, Q_k^n) A_k^* \\
& + M^{-1} K^\eta I(t^n, t^{n+1}, Q_k^n) B_k^* \\
& + M^{-1} K^\zeta I(t^n, t^{n+1}, Q_k^n) C_k^*
\end{aligned}
$$

Cauchy Kovalewski

$$
I(t^n, t^{n+1}, Q_k^n) = \sum_{j=0}^{J} \frac{(t^{n+1} - t^n)^{j+1}}{(j+1)!} \frac{\partial^j}{\partial t^j} Q_k(t^n)
$$

$$
(Q_k)_t = -M^{-1} \left((K^\xi)^T Q_k A_k^* + (K^\eta)^T Q_k B_k^* + (K^\zeta)^T Q_k C_k^* \right)
$$

SeisSol in a Nutshell – ADER-DG

Update scheme

$$Q_k^{n+1} = Q_k - \frac{|S_k|}{|J_k|} M^{-1} \Bigg(\sum_{i=1}^{4} F^{-,i} I(t^n, t^{n+1}, Q_k^n) N_{k,i} A_k^+ N_{k,i}^{-1} + \sum_{i=1}^{4} F^{+,i,j,h} I(t^n, t^{n+1}, Q_{k(i)}^n) N_{k,i} A_{k(i)}^- N_{k,i}^{-1} \Bigg) + M^{-1} K^\xi I(t^n, t^{n+1}, Q_k^n) A_k^* + M^{-1} K^\eta I(t^n, t^{n+1}, Q_k^n) B_k^* + M^{-1} K^\zeta I(t^n, t^{n+1}, Q_k^n) C_k^*$$

Cauchy Kovalewski

$$I(t^n, t^{n+1}, Q_k^n) = \sum_{j=0}^{J} \frac{(t^{n+1} - t^n)^{j+1}}{(j+1)!} \frac{\partial^j}{\partial t^j} Q_k(t^n)$$

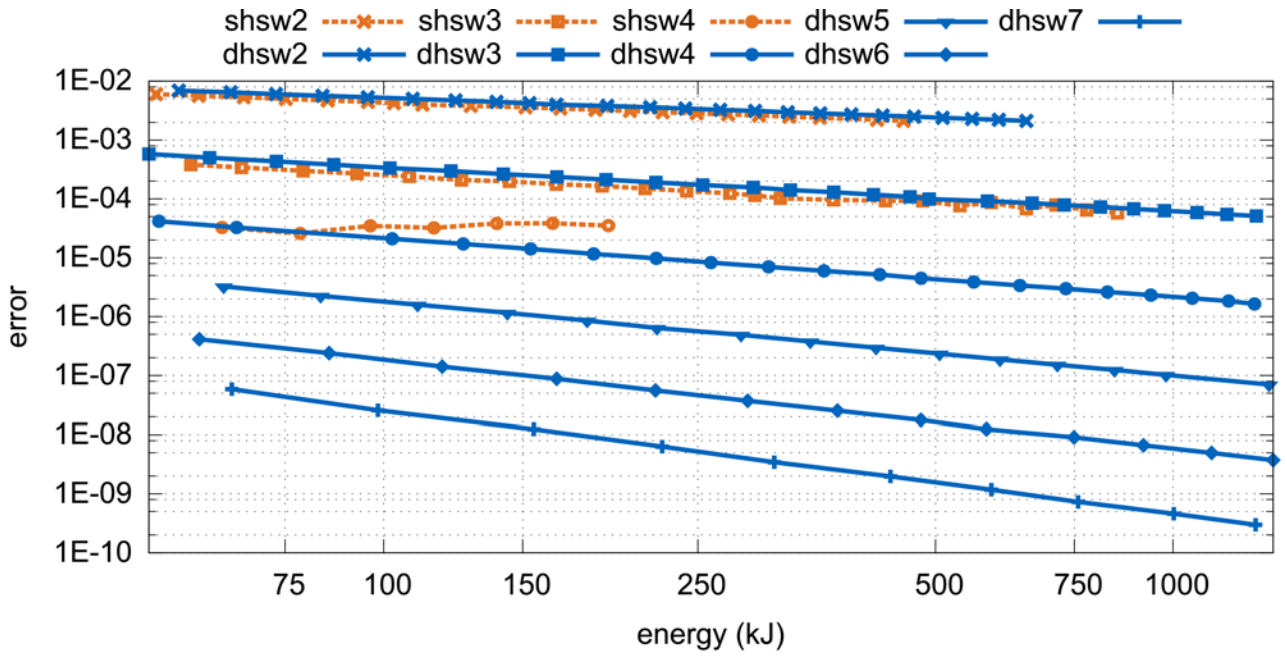
$$(Q_k)_t = -M^{-1} \left((K^\xi)^T Q_k A_k^* + (K^\eta)^T Q_k B_k^* + (K^\zeta)^T Q_k C_k^* \right)$$

Key to Performance Optimization: small matrix multiplication

- "effective sparsity pattern" determined via graph problem
- code generation for sparse matrices → hardwired sparsity structure
- dense and block-sparse multiplications mapped to **libxsmm** library
  → SC16: Heinecke et al.: LIBXSMM . . . ; Thu 4.30pm (255-EF)

High Order Maximises "Science per Watt"

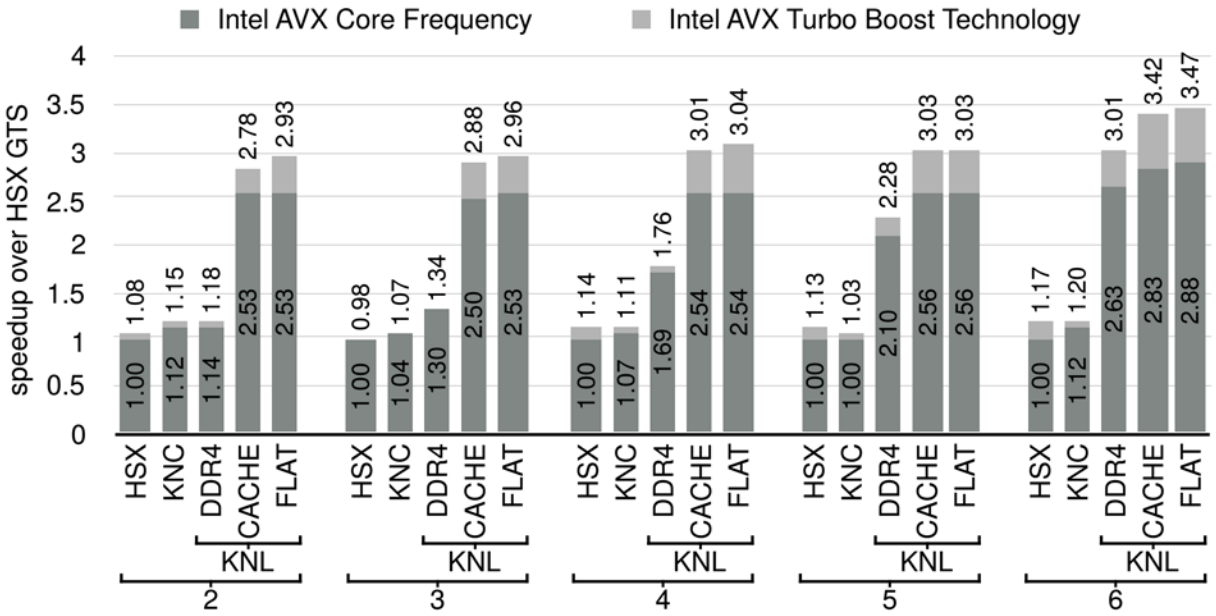
Energy consumption for increasing order (on Intel Haswell):



Breuer, Heinecke, Rannabauer, Bader:
High-Order ADER-DG Minimizes Energy- and Time-to-Solution of SeisSol, ISC'15

Performance Results on Knights Landing

Landers scenario with 466,574 elements – performance for various platforms and orders



Heinecke, Breuer, Bader, Dubey: *High Order Seismic Simulations on the Intel Xeon Phi Processor (Knights Landing)*, ISC High Performance 2016, LNCS 9697

Part II

ExaHyPE – Building an Exascale Hyperbolic PDE Engine

"They are skating to where the puck will be . . . "
(David Keyes)

Wayne Gretzky (image source: Wikipedia)

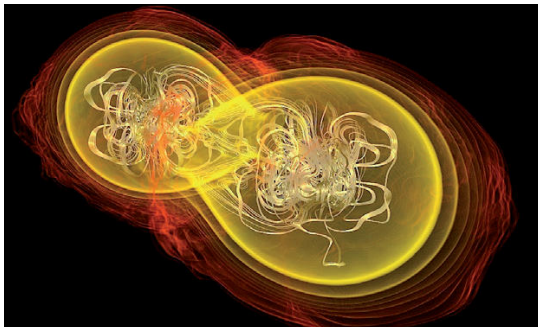
Towards an Exascale Hyperbolic *PDE Engine*

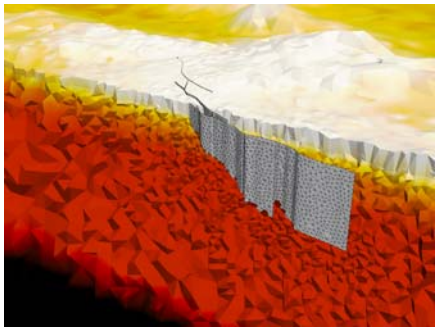
Key Assumption on Exascale Software:

Grand challenge applications require tailoring of existing codes to the specific challenge and cannot rely on general-purpose solutions.

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

- enable medium-sized interdisciplinary research teams to realise 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*

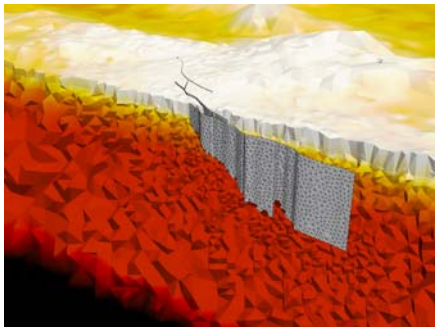
Key Assumption on Exascale Software:

Grand challenge applications require tailoring of existing codes to the specific challenge and cannot rely on general-purpose solutions.

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

- enable medium-sized interdisciplinary research teams to realise 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, tackled/presented by LMU



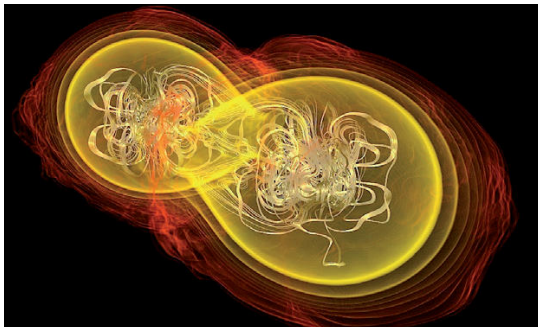
Towards an Exascale Hyperbolic *PDE Engine*

Key Assumption on Exascale Software:

Grand challenge applications require tailoring of existing codes to the specific challenge and cannot rely on general-purpose solutions.

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

- enable medium-sized interdisciplinary research teams to realise 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:



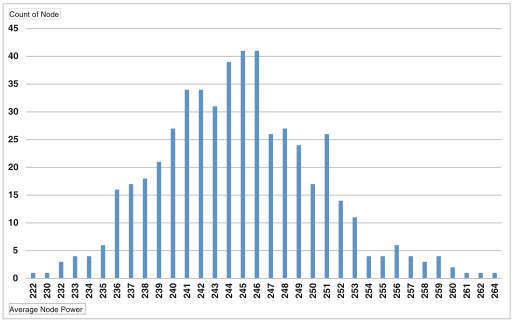
Astrophysics:
merger of binary system of neutro stars tackled/presented by FIAS

Towards an *Exascale* Hyperbolic PDE Engine

Key Assumptions on Exascale Hardware:

Equal work load will no longer lead to balanced computation time.

Moving data is the thriving constraint for performance and energy consumption.



Results by Wilde et al., ISC’15

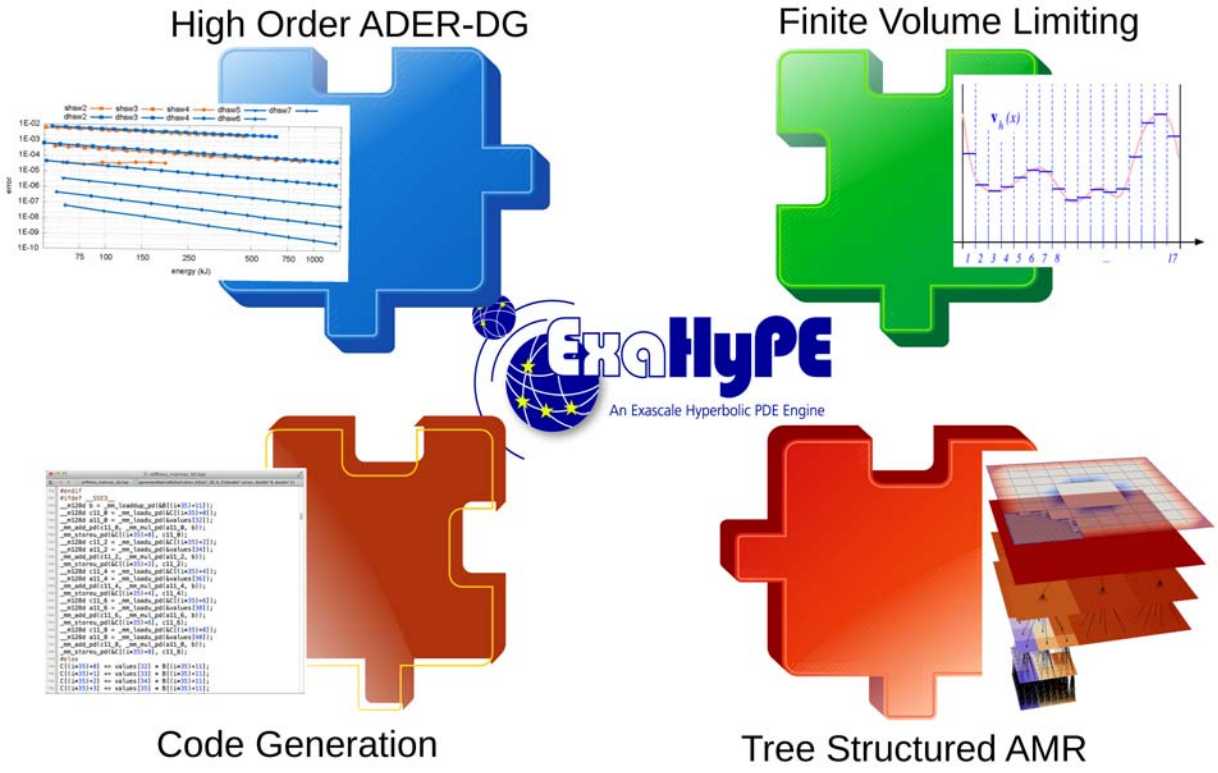
Requirements for Exascale Algorithms:

- avoid data-transfer/communication and synchronisation
- maximise arithmetic intensity
  plus: maximise “science per flop”/“science per Watt”
- dynamic load balancing with lightweight adaptive response

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

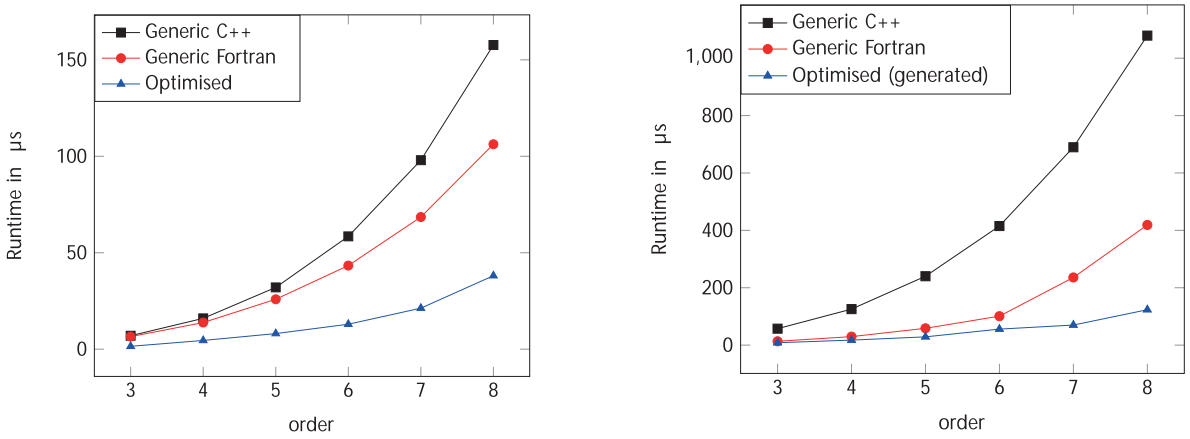
High-Order ADER-DG with Finite Volume Limiting

EU Horizon 2020 – FET-PROACTIVE "Towards Exascale HPC"



Code Generation for High Order ADER-DG

Performance Achieved by Generated Kernels:



Kernel performace for a system with 9 quantities; Haswell (left) vs. KNL (right)

Extrapolation for 56 variables (CCZ4) and 5th order:

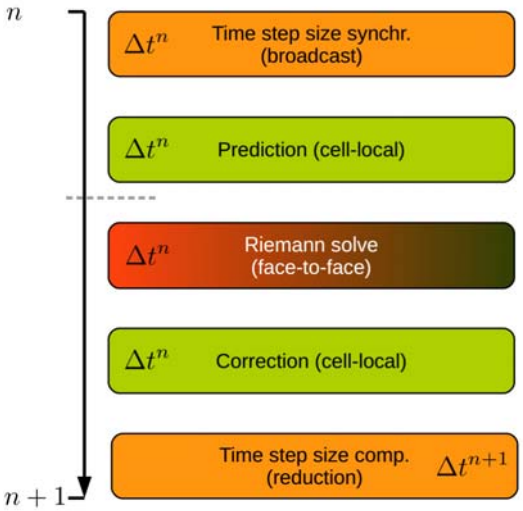
- 15.3 GFlop/s per core (36 % of peak performance)
- current work: integrate into spacetree traversals; optimise memory movements and communication

Increase Arithmetic Intensity

D. Charrier & T. Weinzierl (Durham University)

ADER-DG Computational Steps:

1. compute time step size **for all elements**
2. compute space-time predictor **for all elements** and project data onto face
3. determine numeric fluxes **for all faces**
4. check predicted solutions **for all elements** and flag “troubled” cells
5. diffuse limiter flag as proper boundary conditions for the limited cells are required
6. run a Finite Volume scheme (limiter) **for all troubled cells**



Increase Arithmetic Intensity

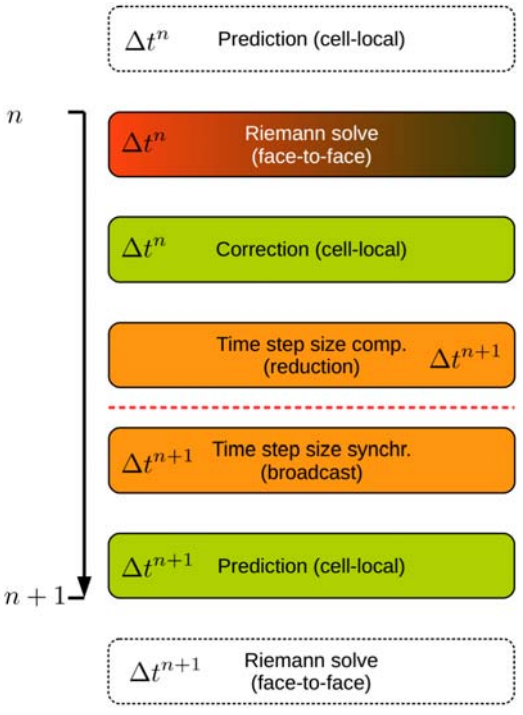
D. Charrier & T. Weinzierl (Durham University)

ADER-DG Computational Steps:

1. compute time step size **for all elements**
2. compute space-time predictor **for all elements** and project data onto face
3. determine numeric fluxes **for all faces**
4. check predicted solutions **for all elements** and flag “troubled” cells
5. diffuse limiter flag as proper boundary conditions for the limited cells are required
6. run a Finite Volume scheme (limiter) **for all troubled cells**

→ **combine into a single tree traversal**

→ with high arithmetic intensity!



Towards and Exascale Hyperbolic PDE Engine . . .

Special thanks go to:

- Intel for supporting our Parallel Computing Centre: “Extreme Scaling on MIC/x86–KNL”
- European Commission for funding the H2020 project ExaHyPE (FET-PROACTIVE “Towards Exascale HPC” – grant No 671698)

Further Readings:

[1] A. Heinecke, A. Breuer and M. Bader: High Performance Seismic Simulations. In J. Jeffers, J. Reinders and A. Sodani (ed.), Intel Xeon Phi Processor High Performance Programming – Knights Landing Edition, p. Chapter 21.

[2] C. Uphoff, M. Bader: *Generating high performance matrix kernels for earthquake simulations with viscoelastic attenuation*. 2016 Int. Conf. High Perf. Computing & Simulation (HPCS 2016).

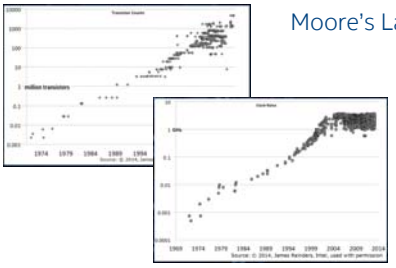
[3] `www.seissol.org`

[4] `www.exahype.eu`

Code Modernization

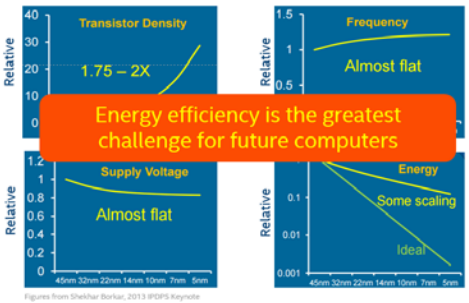
Victor Lee
Intel Corporation

Motivation



Moore's Law ✓

Dennard's Law ✘



Energy efficiency is the greatest challenge for future computers

New energy efficient architecture requires code modernization

Code Modernization

Redesign application codes to better utilize design of modern computer systems

1. How is modern systems different from before
2. What needs to be changed

Changes in Computer Systems

Old computer platforms:

- System with many nodes
- Low network speed and bandwidth
- Nodes with single core processors
- Small memory (but fewer levels of hierarchy)
- ~0.2 Bytes / Flop (memory to CPU)

Modern computer platforms:

- System with many more nodes
- Increased network speed (but network latency is still high, BW per core per node is still low)
- Nodes with many-core processors or potentially hybrid processors
- Bigger memory (but more levels of hierarchy)
- ~0.2 Byte / Flop (memory to CPU)

Code Modernization Challenges

Multi-Node

(Explore coarse-grain parallelism)
1. Top level domain
2. Functional decomposition

Key issues:
- Communication
- Load Balance
- Synchronization

Single-Node

(Explore fine-grain parallelisms)
1. sub-domain decomposition (2nd or 3rd level parallelism)
2. data level parallelism

Key issues:

Utilizing Cores
- What to parallelize
- How to parallelize (OpenMP, TBB, Pthreads)
- Load Balance
- Scalar performance

Utilizing SIMD
- Enabling auto-vectorization
- Data layout to enable contiguous access
- Reduce unaligned accesses

Utilizing Memory
- How to increase data reuse
- Block for caches
- Prefetches

Other Challenges

- Programmer Productivity – reduce effort to increase performance
- Application Portability – run application on different platforms
- Reliability – application / algorithmic changes to improve reliability