STATUSUPDATE DES MEKONG-PROJEKTS

MEKONG: AUTOMATISIERTE PARTITIONIERUNG FÜR HETEROGENE SYSTEME DURCH CODE-ANALYSE UND -TRANSFORMATION

Holger Fröning, Lorenz Braun, **Simon Gawlok**, Vincent Heuveline
Ruprecht-Karls University of Heidelberg, Germany
http://www.ziti.uni-heidelberg.de/compeng
holger.froening@ziti.uni-heidelberg.de

BMBF HPC Statuskonferenz, 04.12.2017, HLRS Stuttgart

BACKGROUND: GPU SOFTWARE VIEW

Massive amount of scalar threads

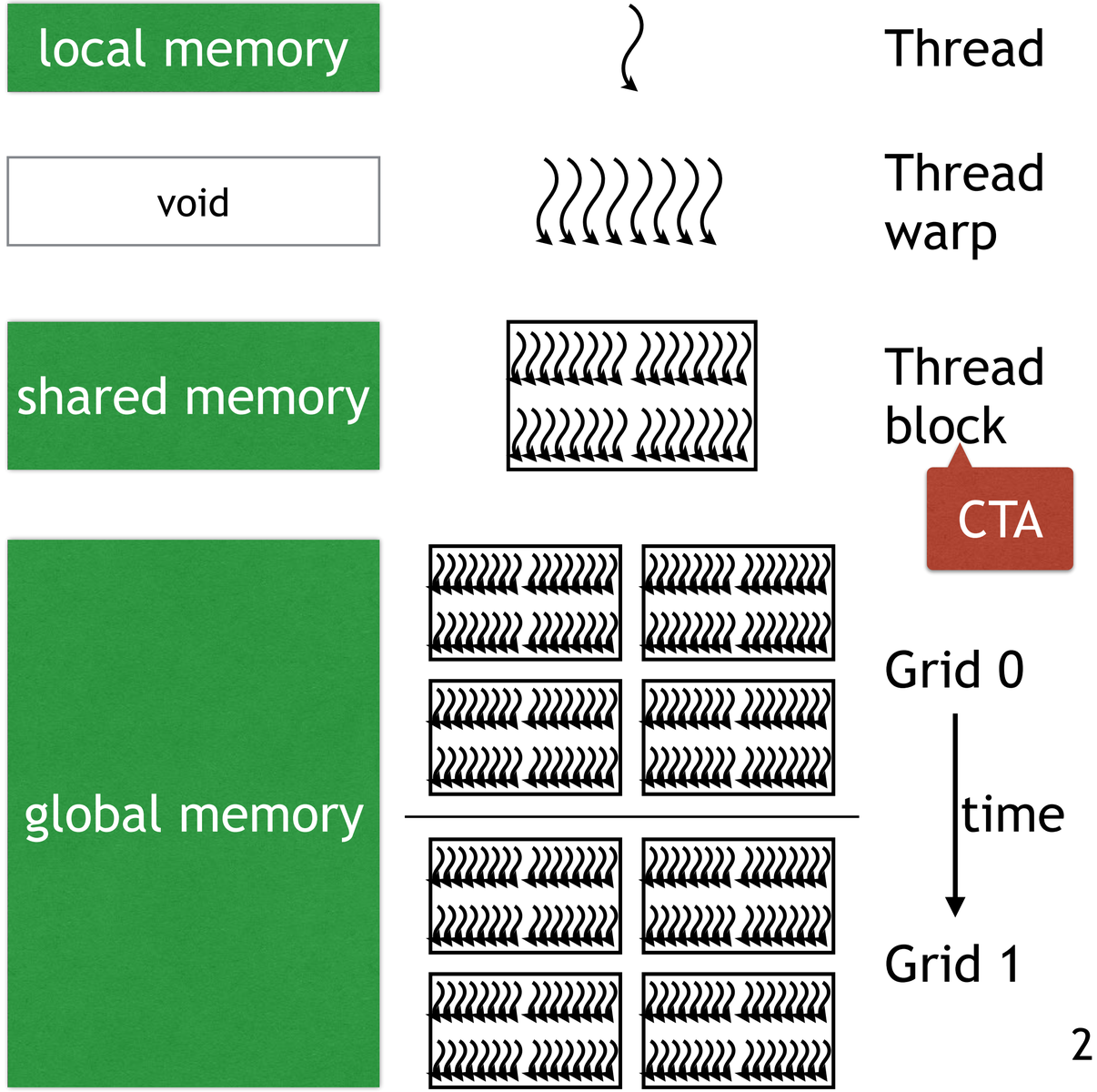
- Collaborative compute
- Collaborative memory access

Thread hierarchy

- Each thread has local memory
- Parallel threads packed in blocks (CTAs)
- Grid executes independent groups

Foundation: BSP

- #threads >> #cores
- => One thread per output element



COMPLEXITY OF MULTI-GPU

GPUs are excellent proxies for future processors

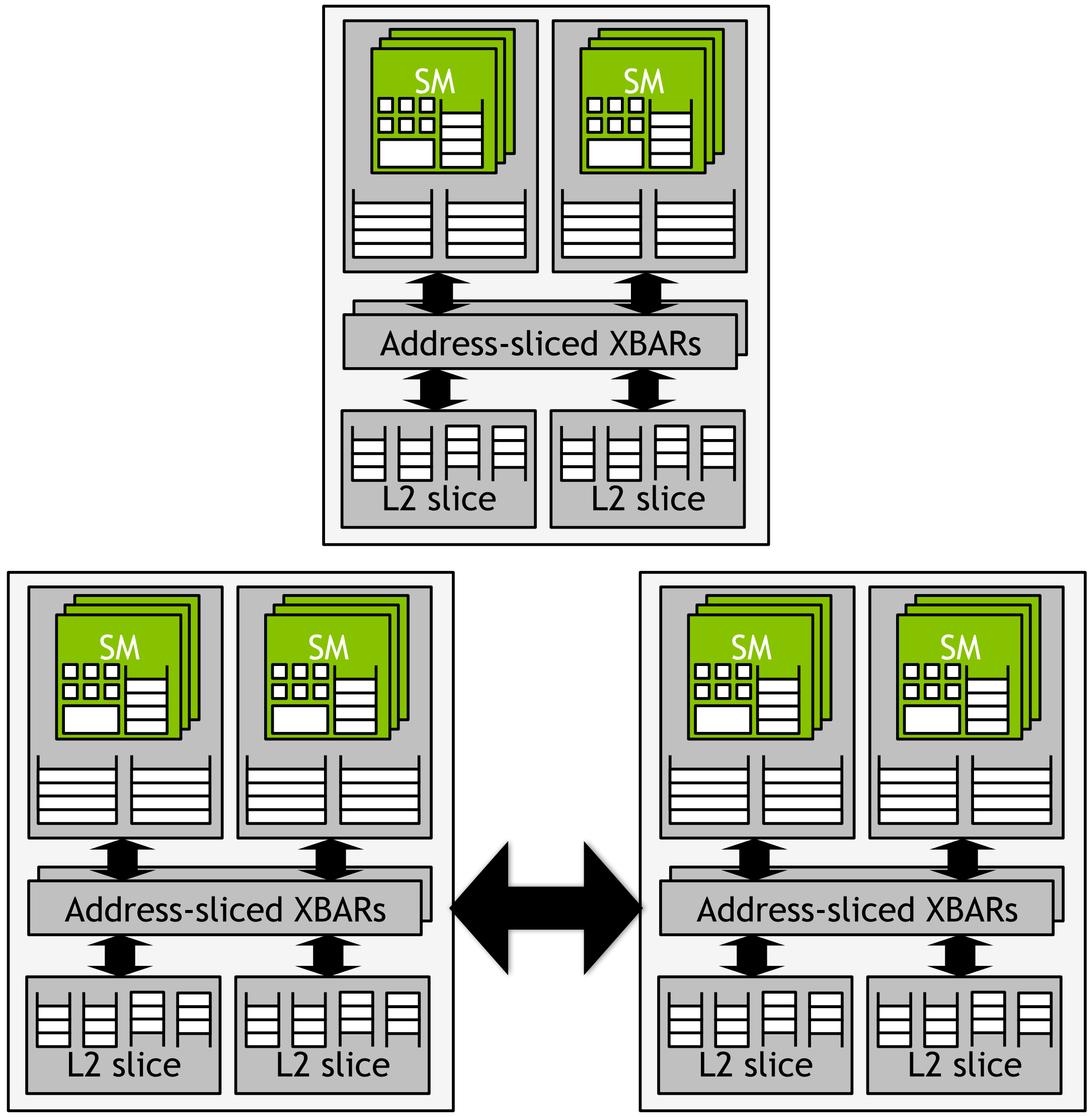
- (+) Fast, energy efficient
- (-) Memory capacity, power consumption
- (+) Sane programming semantics, sane scalability, inline with technology trends

Applications demand for many processors

- Processing power & memory capacity
- Massively parallel communication is structured, selective & fine-grained [1]

Multi-GPU: beauty of simplicity is lost

- Orthogonal extensions, scattered through host and device code, breaking the BSP model



[1] *Benjamin Klenk, Holger Fröning, An Overview of MPI Characteristics of Exascale Proxy Applications, International Supercomputer Conference ISC 2017. (**best paper finalist**)*

OBSERVATIONS ABOUT PARTITIONING

Control

No guarantees exist for interactions among CTAs unless a kernel completion boundary is encountered

=> Kernels can be safely partitioned along CTA boundaries

Corporative Thread Groups (introduced in CUDA 9.0) might break this assumption in the future

Memory

Strong NUMA effects prohibit latency tolerance for remote accesses

Good partitioning mainly depends on memory access pattern

Language

Data-parallel languages help in identifying areas of interest (kernels)

Parallel slackness helps for scalability (larger core count due to multi-GPU)

MEKONG'S BASIC IDEA

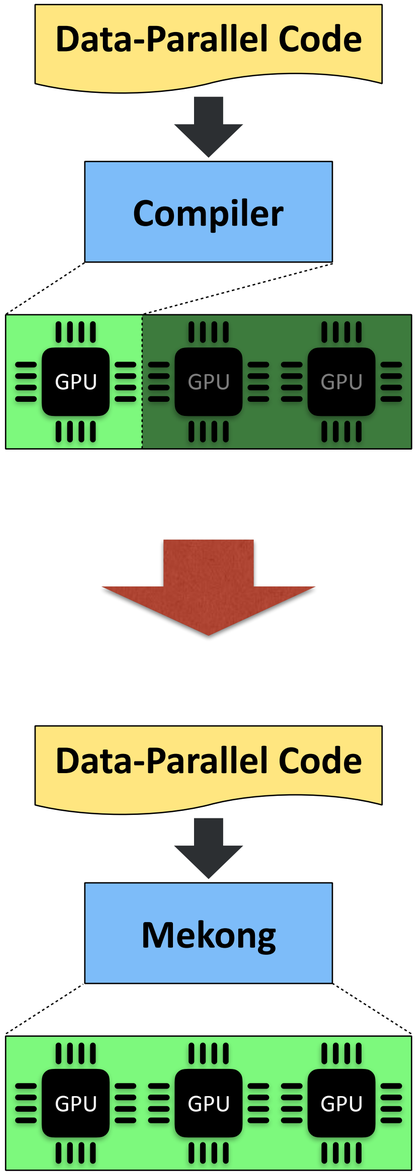
Automatically transform a single-device CUDA program into a multi-device program

- No user intervention
- Key: automated creation of communication tasks
- Initial target one multi-GPU node, but not limited in principle

Code analysis/code generation at compile time

- Minimize run-time overhead
- Partitioning along CTA boundaries
- => Analysis inter-CTA, not intra-CTA (e.g., no shared memory analysis)

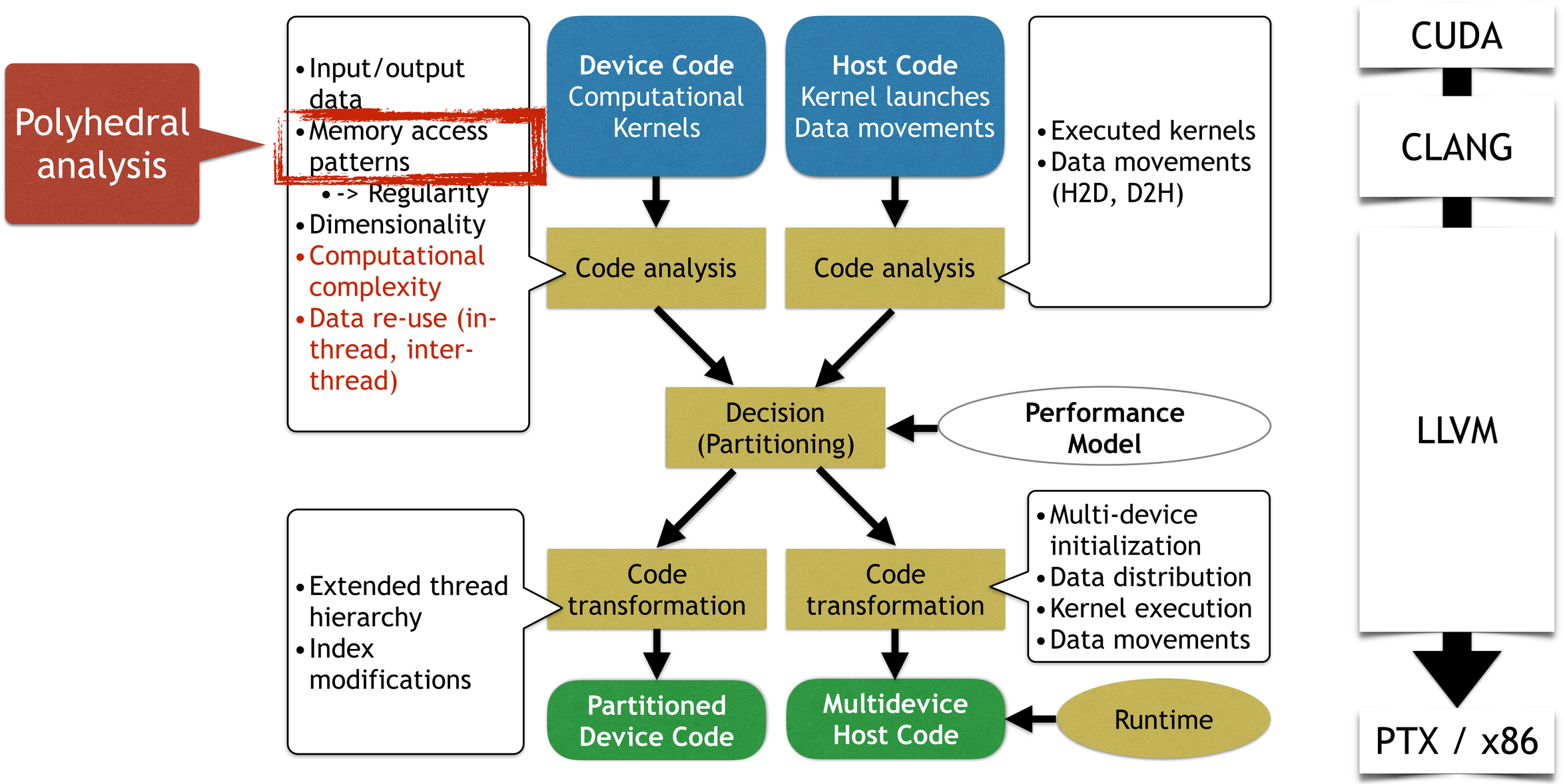
This BMBF project: polyhedral compilation for compile-time analysis of memory access patterns



TODAY: UPDATE ON DECISIONS MADE

1. Compile stack
2. Early integration of polyhedral compilation
3. Partitioning concept for stencil codes
4. App selection
5. Energy instrumentation

1./2. MEKONG'S TOOL STACK



BACKGROUND: POLYHEDRAL COMPILATION

Polyhedral model represents iterative executions, one dimension per (nested) loop => multi-dimensional iteration space

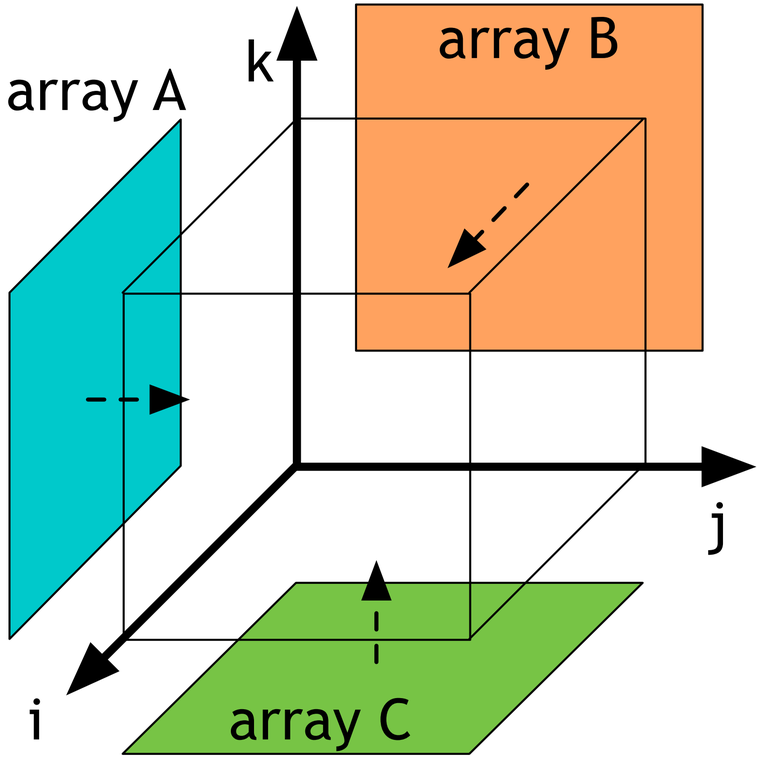
- Z-Polyhedra: described by linear constraints on the universe set, or maps from one set to another
- Example for a matrix-matrix multiplication

Quasi-affine access function (ISL map) + iteration domain (ISL set) -> read/write set (ISL set)

- a[3*i+1] (affine)
- a[(3*i)/d+1] (quasi-affine for d being integer constant)
- a[i*i] (non-affine)

=> Reasoning about multi-dimensional computations and data structures; avoiding explicit unrolling

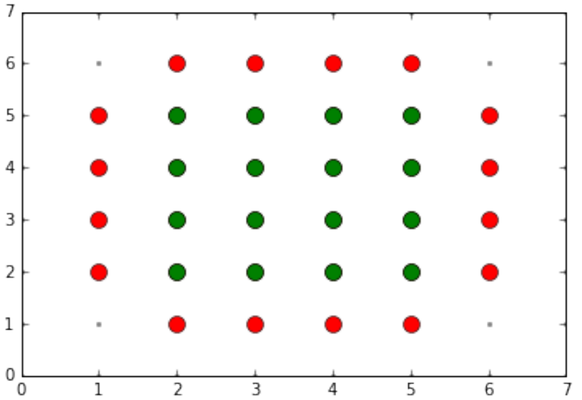
- Inline with n-dimensional thread grid (GPUs)



SIMPLIFIED HOTSPOT EXAMPLE

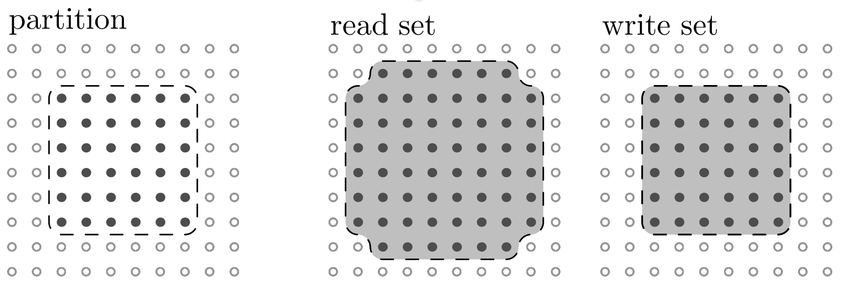
```
if (tx < N && ty < N) {
  acc = A[ty*N+tx]/2;
  acc+=(tx>0   ? A[ty*N+tx-1] : 0)/8;
  acc+=(tx<N-1 ? A[ty*N+tx+1] : 0)/8;
  acc+=(ty>0   ? A[ty*N+tx-N] : 0)/8;
  acc+=(ty<N-1 ? A[ty*N+tx+N] : 0)/8;
  B[ty*N+tx] = acc;
}
```

CUDA kernel code
+
1/2/3D iteration domain (thread grid)



$$
\begin{array}{cccccc}
 & \{ & [y,x] & : & [y,x] & \} \\
\cup & \{ & [y,x] & : & [y,x-1], & x-1 \geq 0 \} \\
\cup & \{ & [y,x] & : & [y,x+1], & x+1 \leq N-1 \} \\
\cup & \{ & [y,x] & : & [y-1,x], & y-1 \geq 0 \} \\
\cup & \{ & [y,x] & : & [y+1,x], & y+1 \leq N-1 \}
\end{array}
$$

ISL map
(simplified)



read/write set
+
set algebra
=
data movements

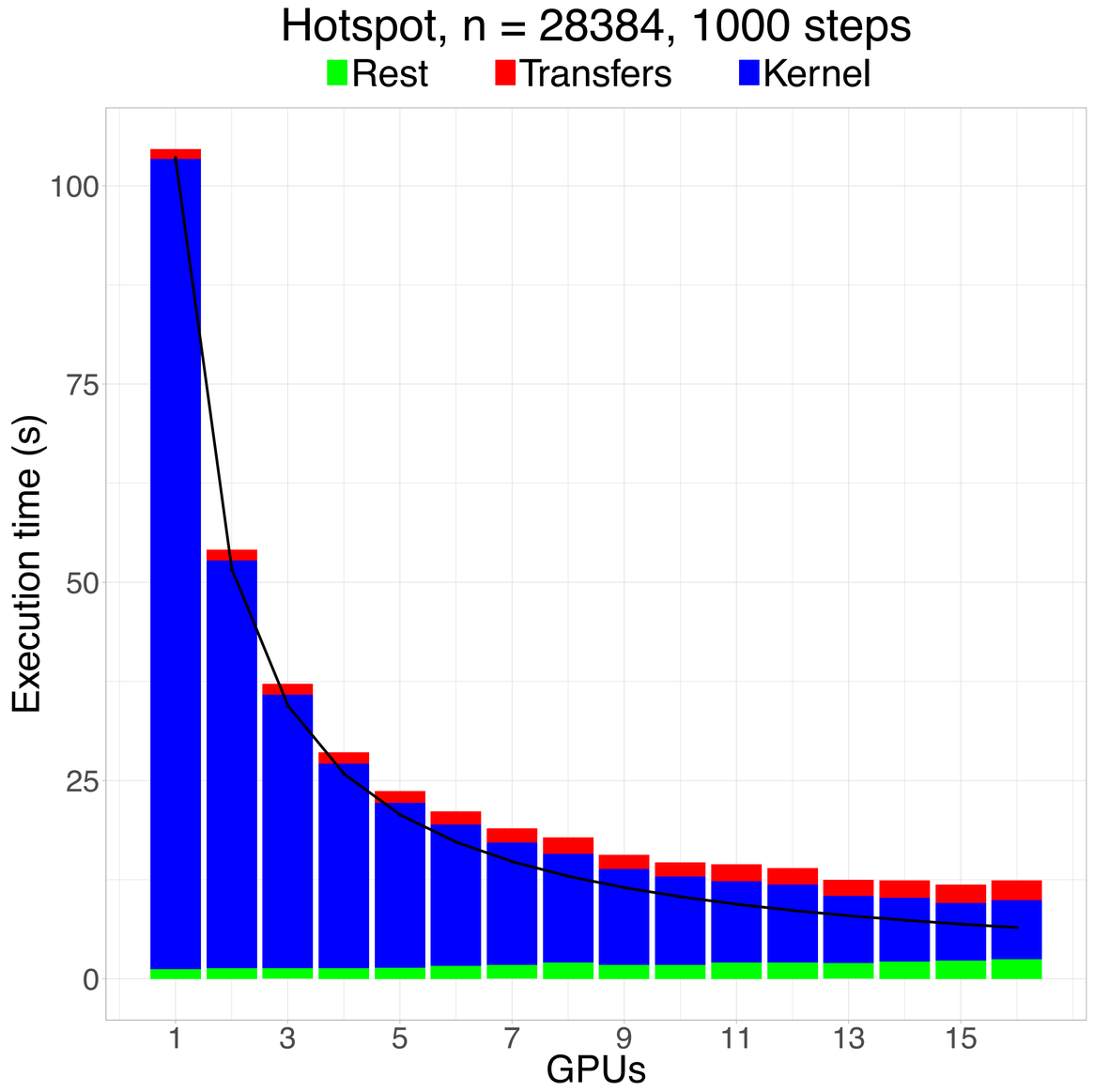
EARLY RESULTS (PROTOTYPE STACK)

Proxy app: stencil code

- No residual, manually defined number of iterations
- No CUDA driver overhead

8x NVIDIA K80

- 16 discrete GPUs total

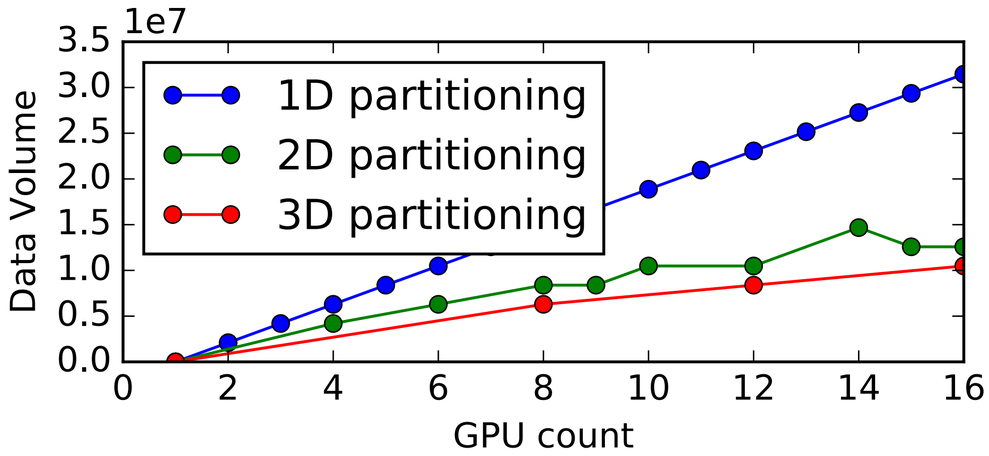


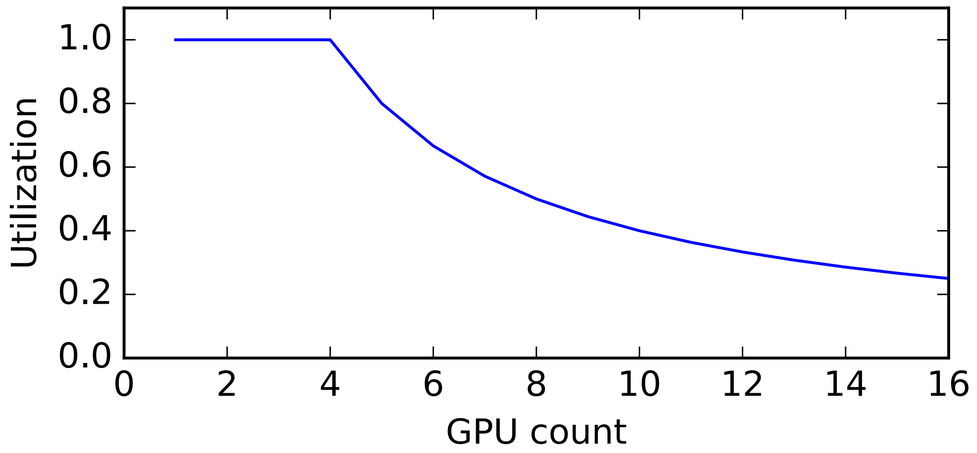
3. PARTITIONING CONCEPT

Consequences of partitioning

- Strong scaling assumed
- #GPUs = #processors = p

1. Communication overhead
   - Common observation: communication overhead increases with p
   - n-dimensional partitioning: volume vs. alignment
2. Reduced utilization usually reduces compute efficiency
   - Sustained performance/peak performance
   - Fixed problem size, increasing p
     => work/p decreasing





4. APP SELECTION

Methods for the discretization and numerical solution of PDEs

- Numerical linear algebra: vector/vector, vector/scalar, reduction sum, sparse-matrix/vector

Mini-app 1: Poisson’s equation in 2D with finite differences

- Three different CG, Jacobi or SSOR variants as solvers

Mini-app 2: Poisson’s equation in 3D with finite differences

- Similar variants to (1)

Mini-app 3: Poisson’s equation in 2D with finite elements

- For unstructured/less structured domains, similar variants to (1), possibly sparse data structures

Mini-app 4: Linear convection-diffusion equation in 2D with finite elements

- (Flexible) Generalized Minimum Residual Method ((F)GMRES) instead of CG
- Challenge: efficient scalar product of two vectors

5. ENERGY MEASUREMENT

Score-P for time measurement

Wall-plug power: Zimmer Electronic Systems (ZES) LMG450

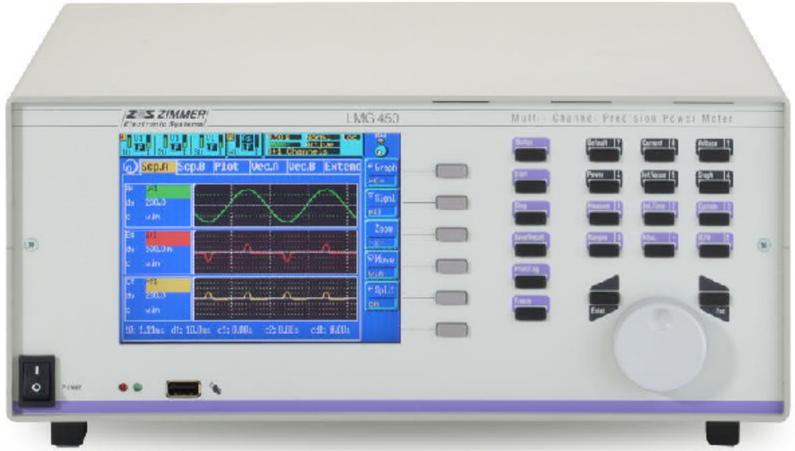
- High temporal resolution, single device

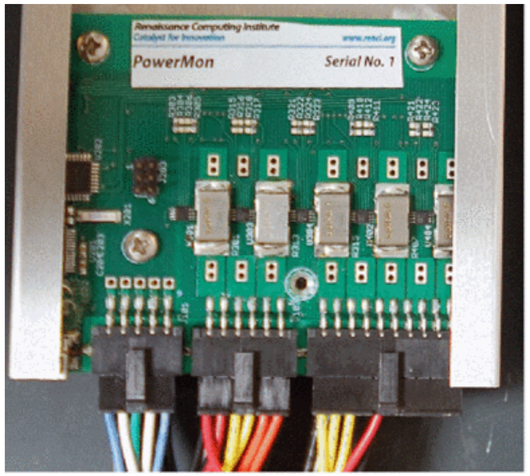
Component power: RAPL (Intel CPUs & mem) & NVML (NVIDIA GPUs)

- Low temporal resolution, assumed to be correct, ubiquitous

Component power (optional): PowerMon

- High temporal resolution, integration complexity





D. Bedard, M. Y. Lim, R. Fowler and A. Porterfield, "PowerMon: Fine-grained and integrated power monitoring for commodity computer systems," Proceedings of the IEEE SoutheastCon 2010, doi: 10.1109/SECON.2010.5453824

SUMMARY

Mekong simplifies multi-GPU programming using polyhedral compilation techniques

- Automated creation of communication tasks

Compile stack, concept and apps defined

Initial results very promising (overhead, scalability)

Next: automated tool stack for fixed partitioning, partitioning decision, overlap by sub-partitioning & scheduling, energy implications

Acknowledgements

- Support by polyhedral compilation community: Johannes Doerfert & Sebastian Hack (discussion & compile pass), Tobias Grosser (associated member), …
- Support by NVIDIA Germany (associated member) & NVIDIA Research (gifts, grants)
- BMBF funding (2017-2020)
- Google Faculty Research Award (2014)

Alexander Matz, Holger Fröning, GPU Mekong: Simplified Multi-GPU Programming using Automated Partitioning, SC2017 Poster
Alexander Matz, Holger Fröning, Automated Partitioning of Data-Parallel Kernels using Polyhedral Compilation, CGO'18 SRC