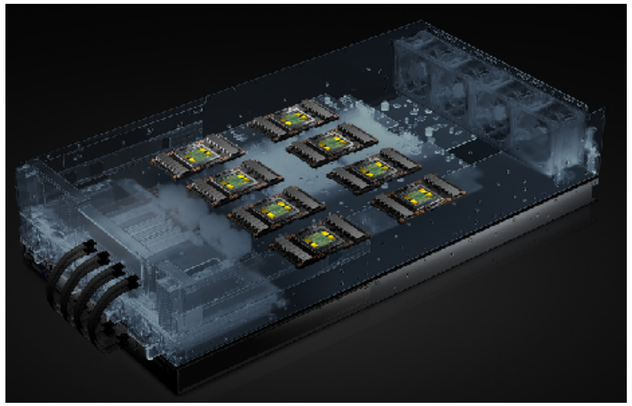
UNIVERSITÄT
HEIDELBERG
ZUKUNFT
SEIT 1386

Enabling Automatic Partitioning of Data-Parallel Kernels with Polyhedral Compilation

Alexander Matz, Holger Fröning
Heidelberg University, Germany

LLVM Performance Workshop @CGO 2018
Sat 24 Feb 2018, Vienna, Austria

Multi GPU in the Real World



[1]

P2-Instance-Details

Name	GPUs	vCPUs	RAM (GiB)
p2.xlarge	1	4	61
p2.8xlarge	8	32	488
p2.16xlarge	16	64	732

[2]

- NVIDIA DGX-1 and HGX-1
  8 Tesla GPUs
- Amazon AWS P2
  up to 16 Tesla GPUs
- Google Cloud Platform
  up to 8 Tesla GPUs

[1] https://www.nvidia.de/content/dam/en-zz/Solutions/Data-Center/hgx-1/data-center-nvidia-hgx-1-update-2-hero-desktop@2x.jpg

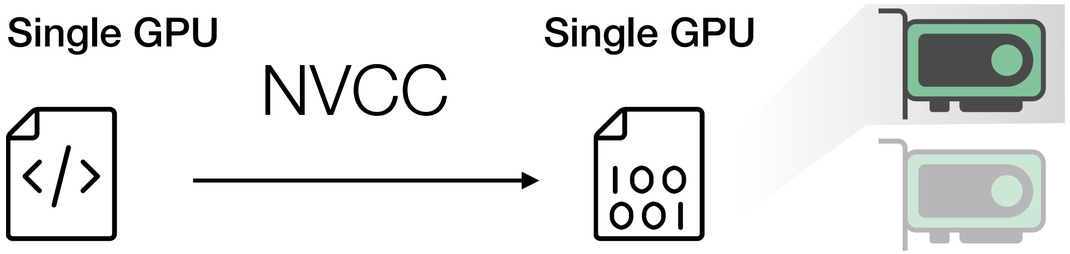
[2] https://aws.amazon.com/de/ec2/instance-types/p2/

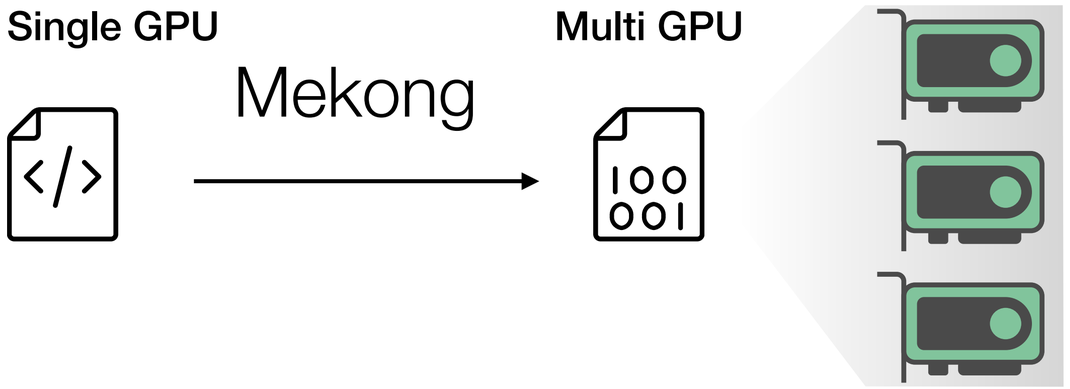
Observations GPU Programming

UNIVERSITÄT
HEIDELBERG
ZUKUNFT
SEIT 1386

- Execution model
  - No guarantees exist for interactions among CTAs until kernel completion
    => Kernels can be safely partitioned along CTA boundaries (usually)
- 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)

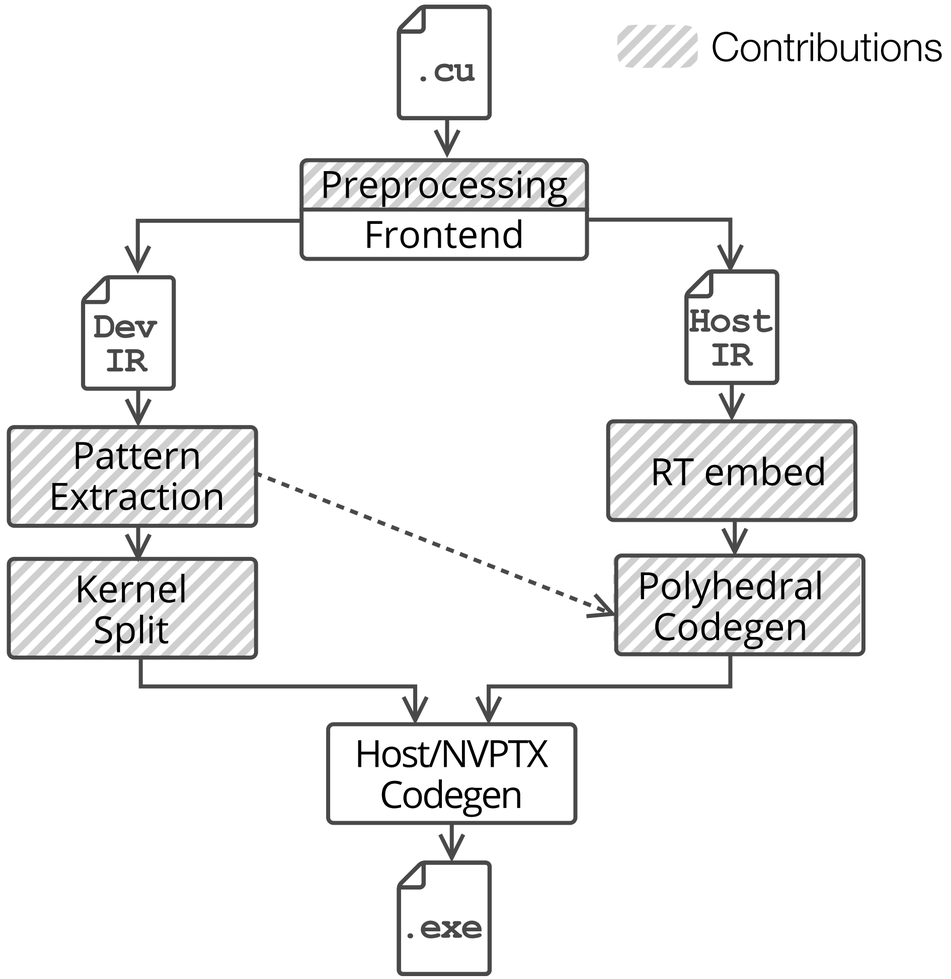
Basic Idea





- Keep clear data ownership and movements of single GPU programming
- Automatically sync buffers
- Hybrid compile time / run time approach
- Minimize runtime overhead

Pipeline Overview



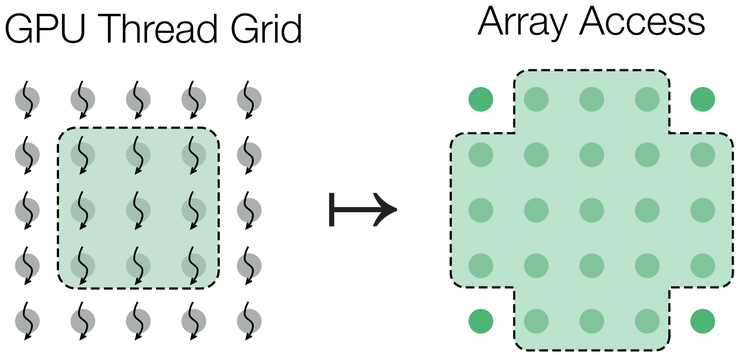
- Based on LLVM (gpucc)
- Preprocessing based on text substitution
- Majority of functionality implemented as passes
- Not fully integrated yet

Kernel Analysis & Code Generation

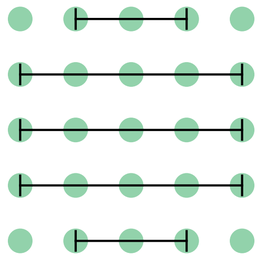
1. Kernel Code

```
b[y*N + x] += a[y*N + x];
b[y*N + x] += a[y*N + x + 1];
b[y*N + x] += a[y*N + x - 1];
b[y*N + x] += a[y*N + x + N];
b[y*N + x] += a[y*N + x - N];
b[y*N + x] *= 0.2;
```

2. Application Model



3. Memory Range



Polyhedral Analysis

Polyhedral Code Generation

Kernel Analysis

UNIVERSITÄT
HEIDELBERG
ZUKUNFT
SEIT 1386

- Based on Polyhedral Value & Memory Analysis [1]
- Model should intuitively map Global ID ↦ Array Element, so $\mathbb{Z}^3 \mapsto \mathbb{Z}^d$
- CUDA Expression “threadIdx + blockIdx * blockDim” not affine
- Workaround
  - Replace product with new input dimension “blockOffset”
  - Limit “threadIdx” to [0..“blockDim”], then project out
- Model is now: $\mathbb{Z}^6 \mapsto \mathbb{Z}^d$, with three pairs of two dependent dimensions

```
[N] -> {
  I[y, x] -> S[o1=y, o2=x] : 0 <= o1,o2 < N;
  I[y, x] -> S[o1=y, o2=x-1] : 0 <= o1,o2 < N;
  I[y, x] -> S[o1=y, o2=x+1] : 0 <= o1,o2 < N;
  I[y, x] -> S[o1=y-1, o2=x] : 0 <= o1,o2 < N;
  I[y, x] -> S[o1=y+1, o2=x] : 0 <= o1,o2 < N;
}
```

[1] http://www.llvm.org/devmtg/2017-10/#src2

Code Generation

UNIVERSITÄT
HEIDELBERG
ZUKUNFT
SEIT 1386

- Purpose A: Encode buffer dimension sizes and type information
- Purpose B: Implement efficient iterators for array accesses
  - Tracking buffer state requires iterators for write accesses
  - Synchronizing buffers for kernels requires iterators for read accesses

Iterator Code Generation

2D 5-point stencil, read map

```
[N] -> {
  I[y, x] -> S[o1=y, o2=x] : 0 <= o1,o2 < N;
  I[y, x] -> S[o1=y, o2=x-1] : 0 <= o1,o2 < N;
  I[y, x] -> S[o1=y, o2=x+1] : 0 <= o1,o2 < N;
  I[y, x] -> S[o1=y-1, o2=x] : 0 <= o1,o2 < N;
  I[y, x] -> S[o1=y+1, o2=x] : 0 <= o1,o2 < N;
}
```

2D domain

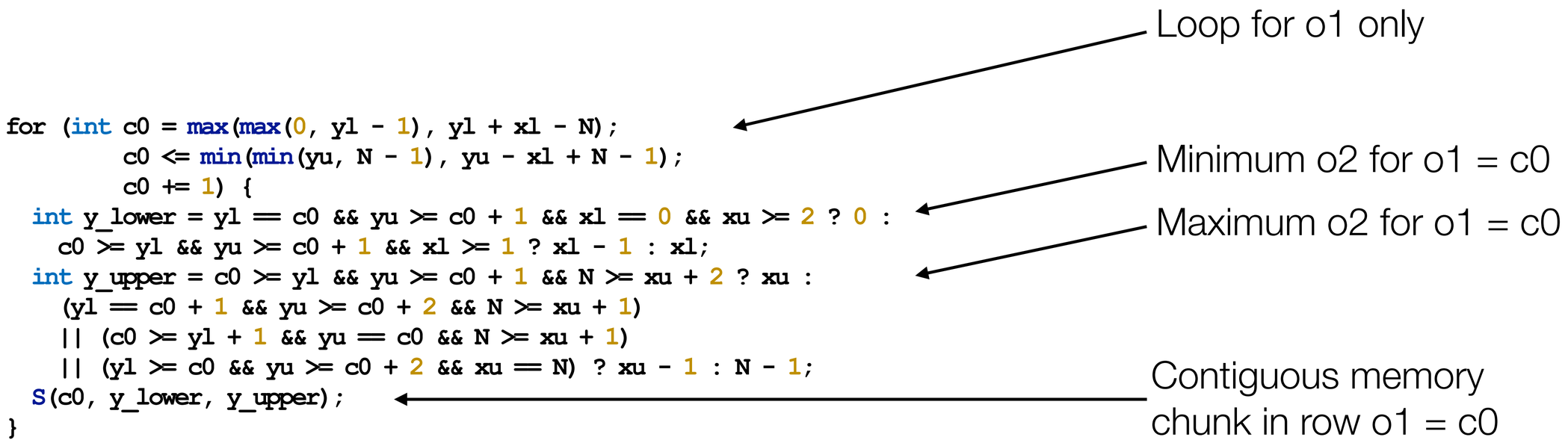
```
[yl, yu, xl, xu] -> {
  I[y, x] : 0 <= yl <= y < yu and 0 <= xl <= x < xu
}
```

Identity schedule of map range

```
for (int c0 = max(max(0, yl - 1), yl + xl - N);
         c0 <= min(min(yu, N - 1), yu - xl + N - 1);
         c0 += 1)
  for (int c1 = max(max(max(0, xl - 1), yl + xl - c0 - 1), -yu + xl + c0);
           c1 <= min(min(min(xu, N - 1), yu + xu - c0 - 1), -yl + xu + c0);
           c1 += 1)
    S(c0, c1);
```

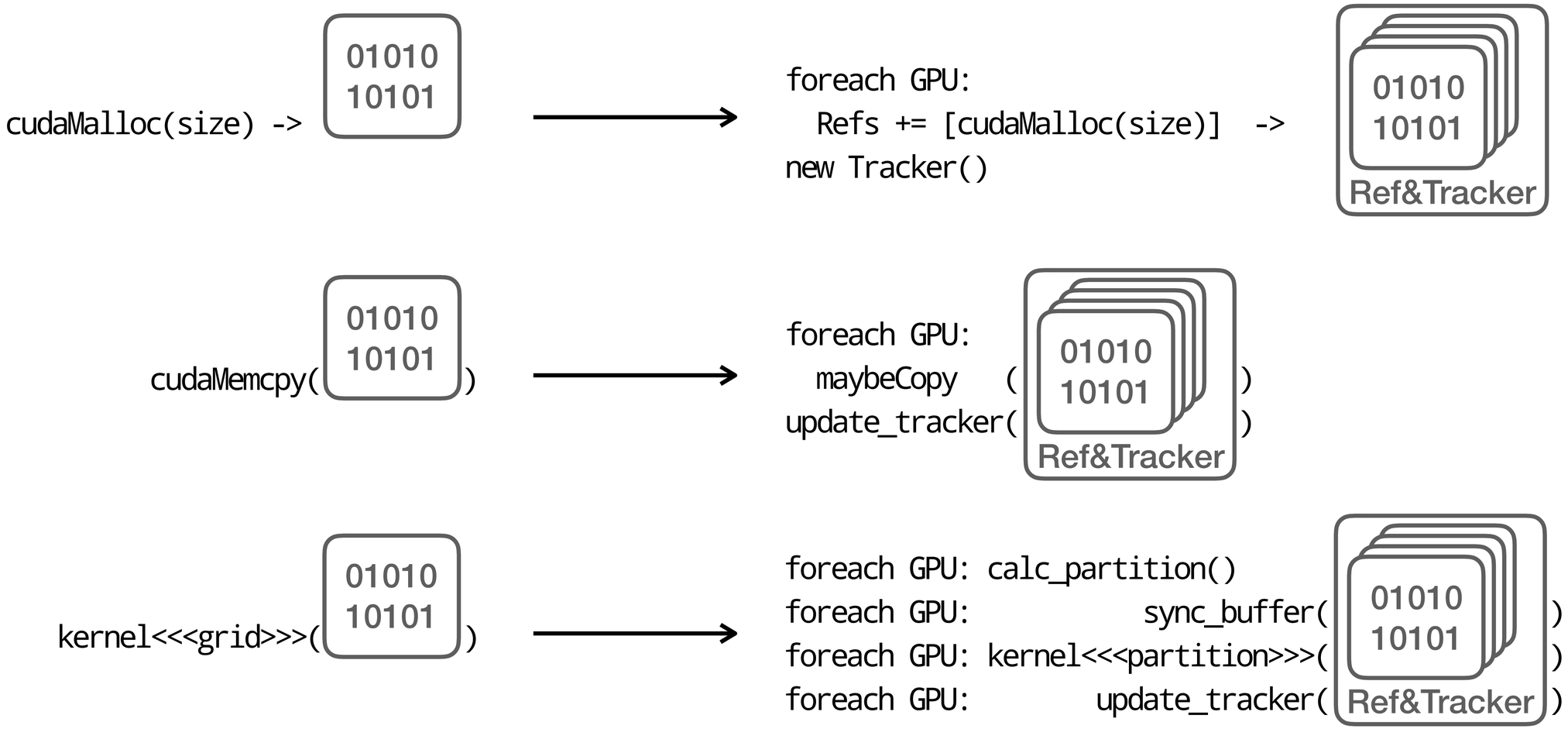
- Based on isl AST generation
- Accurate but inefficient
- Reads don’t need 100% accuracy
- Last dimension is stored contiguous in memory in C

Iterator Code Generation



- Replace one loop with closed-form lower/upper expressions (optimized by LLVM)
- Good estimate for read maps
- Write maps need extra checks (modulo, non-convex sets) to verify accuracy
- Allows more efficient tracking and data transfers

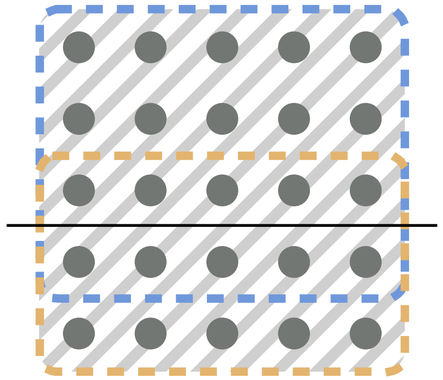
Runtime Buffer Management



Runtime Buffer Synchronization

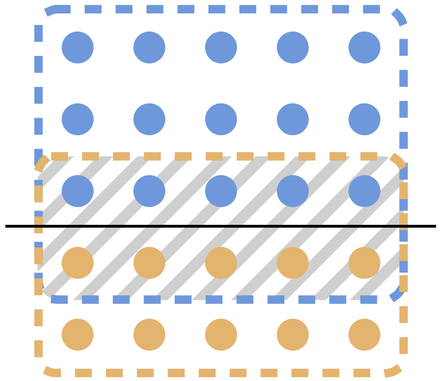
UNIVERSITÄT HEIDELBERG
ZUKUNFT SEIT 1386

First Kernel Launch



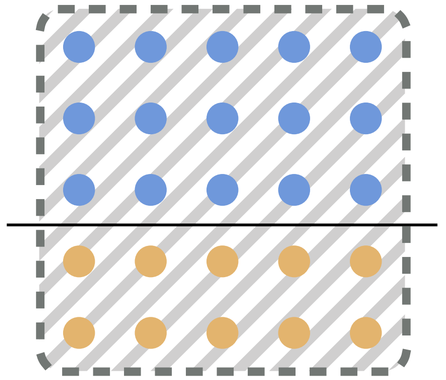
- Data is in host memory
- Each GPU transfers its whole read set

Kernel Iteration

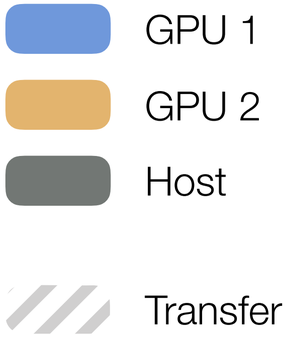


- Data is distributed on GPUs
- Each GPU only transfers stale data
- Often the most repeated part of application

Data Gathering



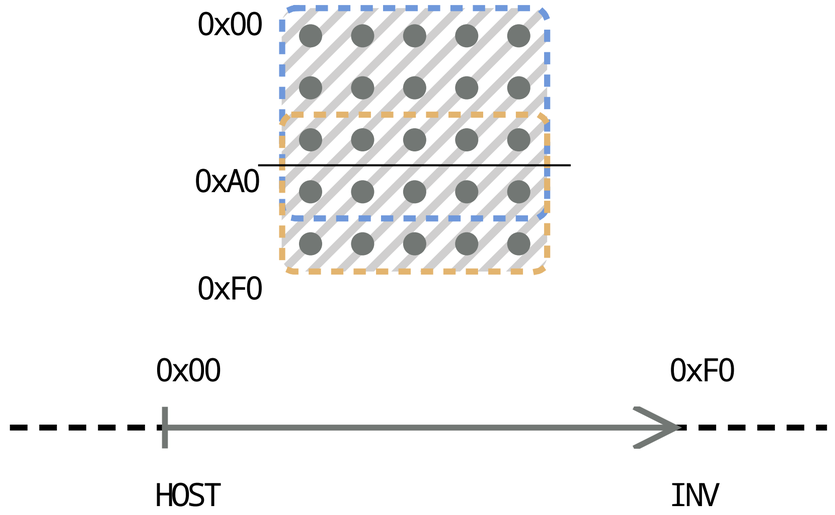
- Data is distributed on GPUs
- Host transfers most up to data chunk from each GPU

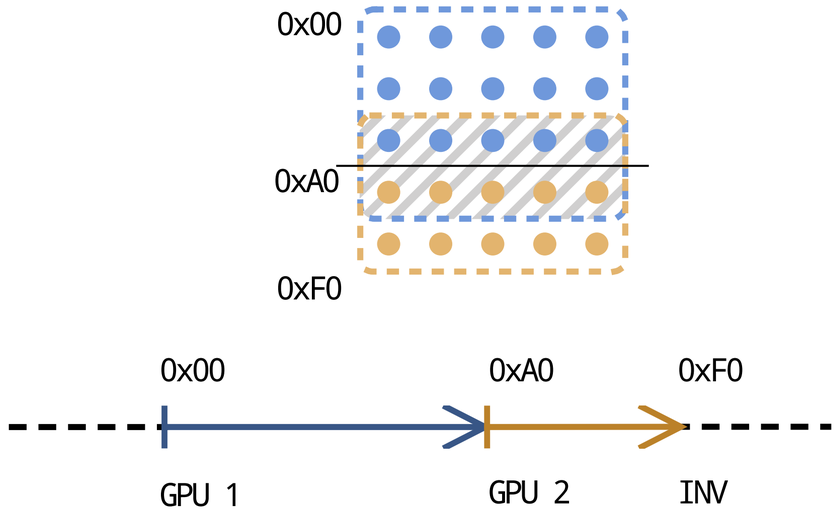


Runtime Buffer Tracking

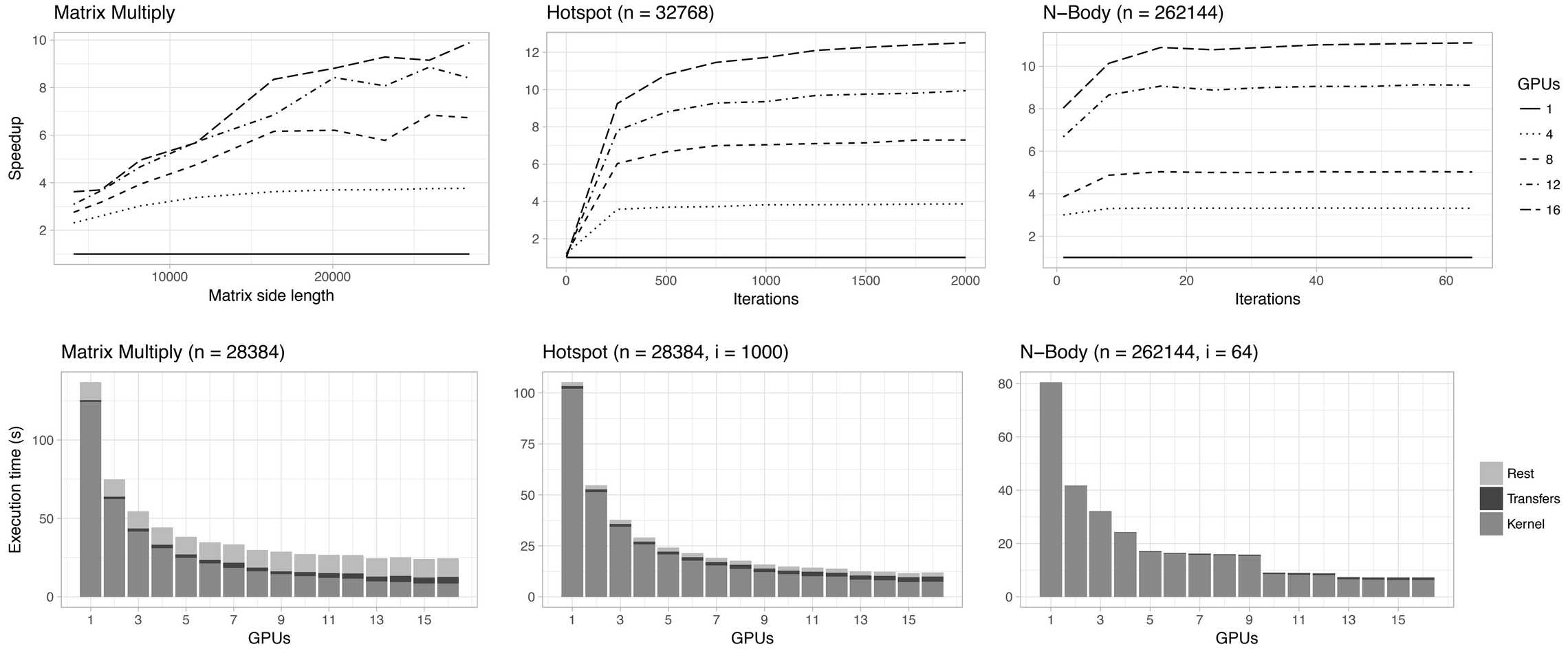
UNIVERSITÄT HEIDELBERG
ZUKUNFT SEIT 1386

- Synchronization requires tracking
- Track intervals of memory describing location of most recent update
- No overlapping intervals, implemented as b-tree based map with lower bound search
- Coalescing neighboring intervals keeps memory footprint and performance stable





Performance



Future Work

UNIVERSITÄT
HEIDELBERG
ZUKUNFT
SEIT 1386

- Fully integrated proof-of-concept
- Better handling of non-affine accesses
- More comprehensive validation using well-known benchmarks
- Array reshaping for better performance and memory utilization
- Explore shared memory optimizations (e.g. posted writes for synchronization)

Conclusion

UNIVERSITÄT
HEIDELBERG
ZUKUNFT
SEIT 1386

- Compiler based Automatic Partitioning is feasible
- Polyhedral compilation is a good fit for GPU memory access patterns
- Accuracy of extracted memory access patterns crucial for both correctness (write accesses) and performance (read accesses)
- Performance of prototype experiments very promising
- LLVM provides excellent research platform for non-traditional compiler researchers

UNIVERSITÄT
HEIDELBERG
ZUKUNFT
SEIT 1386

Thank you

We especially thank
Christoph Klein and Lorenz Braun (Heidelberg University), and
Johannes Doerfert (Saarland University) for their contributions to our research
as well as
Sudha Yalamanchili (Georgia Tech), Mark Hummel (NVIDIA), Peter Zaspel (University of Basel), Tobias Grosser (ETH Zürich), Johannes Doerfert and Sebastian Hack (Saarland University) for many helpful discussions
and our Sponsors
BMBF, Google, NVIDIA, and the German Excellence Initiative

1D-Identity Map

UNIVERSITÄT
HEIDELBERG
ZUKUNFT
SEIT 1386

1. Analysis Output

```
[boff_x, tid_x] -> {
  [] -> [boff_x + tid_x]
}
```

2. 1D Iteration Domain (CUDA Thread Grid)

```
[boffmin_x, boffmax_x, bidmin_x,
  bidmax_x, bdim_x] -> {
  [boff_x, bid_x, tid_x] :
    boffmin_x <= boff_x < boffmax_x
    and bidmin_x <= bid_x < bidmax_x
    and 0 <= tid_x < bdim_x;
}
```

3. Canonicalized Access Map

```
[boffmin_x, boffmax_x, bidmin_x,
  bidmax_x, bdim_x] -> {
  [boff_x, bid_x] -> [o0] :
    boffmin_x <= boff_x < boffmax_x
    and bidmin_x <= bid_x < bidmax_x
    and boff_x <= o0 < bdim_x + boff_x
}
```

2D 5-Point Stencil Read

UNIVERSITÄT HEIDELBERG
ZUKUNFT SEIT 1386

Analysis Output

```
[tid_x, boff_x, tid_y, boff_y, N] -> {
  [] -> A[o0, o1] :
    N > tid_x + boff_x
    and N > tid_y + boff_y
    and o0 <= tid_y + boff_y
    and -1 + tid_x + boff_x + tid_y
      + boff_y - o0 <= o1 < N
    and o1 <= 1 + tid_x + boff_x
      - tid_y - boff_y + o0;
  [] -> A[1 + tid_y + boff_y, tid_x + boff_x]
}
```

Canonicalized Access Map

```
[bdim_y, bdim_x, boffmin_y, boffmax_y, boffmin_x, boffmax_x,
    bidmin_y, bidmax_y, bidmin_x, bidmax_x, N] -> {
  [boff_y, boff_x] -> A[o0, o1] : bdim_y = 1 and bdim_x = 1
    and bidmax_y > bidmin_y and bidmax_x > bidmin_x
    and boffmin_y <= boff_y < N and boff_y < boffmax_y
    and boffmin_x <= boff_x < N and boff_x < boffmax_x
    and o0 <= boff_y and -1 + boff_y + boff_x - o0 <=
      o1 <= 1 - boff_y + boff_x + o0 and o1 < N;
  [boff_y, boff_x] -> A[o0 = 1 + boff_y, o1 = boff_x] :
    bdim_y = 1 and bdim_x = 1 and bidmax_y > bidmin_y
    and bidmax_x > bidmin_x
    and boffmin_y <= boff_y < boffmax_y
    and boffmin_x <= boff_x < boffmax_x
}
```