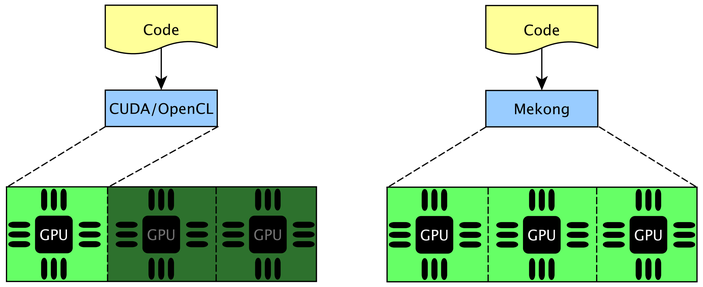
GPU Mekong: Simplified Multi-GPU Programming using Automated Partitioning

Alexander Matz, Holger Fröning
Heidelberg University, Institute of Computer Engineering
{alexander.matz, holger.froening}@ziti.uni-heidelberg.de

UNIVERSITÄT HEIDELBERG
ZUKUNFT SEIT 1386

Motivation

- GPUs provide excellent performance
- Massive amounts of exposed parallelism due to data-parallel languages (e.g. CUDA, OpenCL)
- Single GPU programming straight forward
- Multi GPU programming not so much
  - Manual orchestration of data movements & execution
  - Manual resolution of data dependencies
  - Tedious and error prone



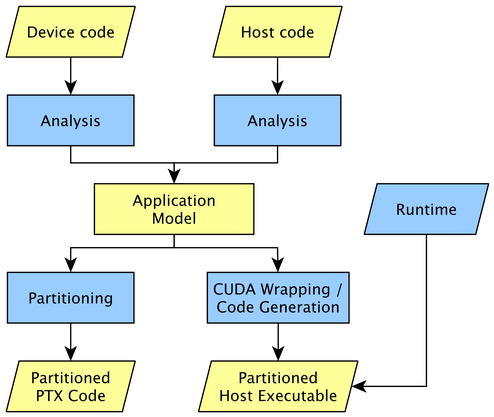
Mekong Objectives

- **Execute single GPU applications on multiple GPUs**
- Maximize productivity, i.e. **minimize required user input**
- **Exploit static analysis** as much as possible
- Provide **familiar interface** similar to nvcc

Compilation Pipeline

Based on LLVM [1]

1. Analyze device code
   - Extract polyhedral model of Memory Accesses
   - Propose efficient partitioning scheme
2. Analyze host code
   - Check for supported GPU operations
3. Transform device code
   - Extend kernel arguments with partitioning information
   - Adjust calculation of thread ID & block ID
   - Adjust global memory accesses
4. Transform host code
   - Generate code to query memory access patterns
   - Embed Mekong runtime system
   - Replace all calls to GPU functions with runtime equivalents
   - Code Generation for Scanning Polyhedra



Runtime System

- Distribute to and collect data from each GPU
- Resolves data dependencies between iterations
- Launches modified kernels on multiple GPUs

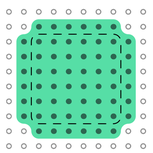
1) Kernel

```
_1 = [y*n + x]
_2 = (x > 0     ? in[y*n + x - 1] : 0);
_3 = (x < (n-1) ? in[y*n + x + 1] : 0);
_4 = (y > 0     ? in[y*n + x - n] : 0);
_5 = (y < (n-1) ? in[y*n + x + n] : 0);
out[y*n + x] = f(_1, _2, _3, _4, _5);
```

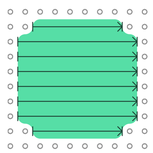
2) Polyhedral Model

```
[n] -> {
  [z, y, x] -> [y, x];
  [z, y, x] -> [y, x-1] : x > 0;
  [z, y, x] -> [y, x+1] : x < n;
  [z, y, x] -> [y-1, x] : y > 0;
  [z, y, x] -> [y+1, x] : y < n;
}
```

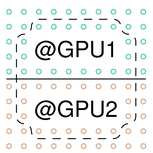
3) Partitioned Model



4a) Scanner



4b) Tracker

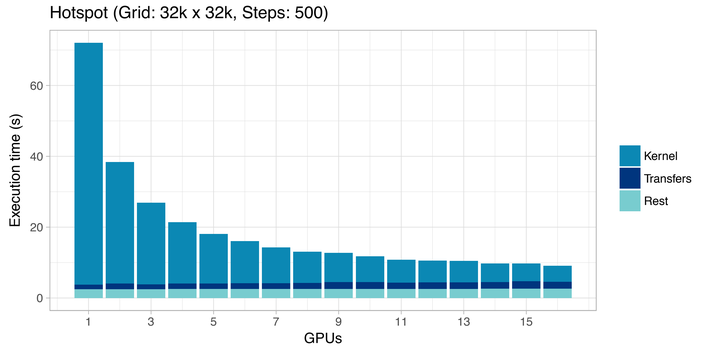


Data Dependencies

- Create Polyhedral Model of Read Sets and Write Set of the kernel
- Generate Code to scan of ranges of contiguous elements
- Combine adjacent element ranges for efficient transfer
- Red-Black-Tree based Tracker to locate most recently updated elements
- Libraries for optimized data transfer

Performance

- Benchmark: 2D 5-point stencil, grid size: 32k*32k
- Test System: 2x Intel Xeon E5-2667, 8x NVIDIA K80, 256GiB DDR4
- Significant performance improvement depending on the application
- As expected, kernel scales well with number of GPUs



- Scanning + Tracking keep communication overhead acceptable
- Overall scalability requires suitable minimum problem size

Conclusion

- Mekong enables automated and correct partitioning of data-parallel codes, resulting in promising speed-ups
- Accurate memory access patterns are vital for efficient partitioning
- Promising features in future work (e.g. array reshaping for scalability)
- Mekong can assist application and library programmers in migrating codes to multi-GPU systems with minimal effort

Selected Related Work

- APOLLO: Parallelization at runtime of nested loops [2]
- MKMD: Compile-time parallelization of GPU kernels, without using polyhedral compilation [3]
- MAPS-Multi: Parallelization of GPU kernels with user input [4]

Acknowledgements

Research contributions: Christoph Klein (Heidelberg University)
Discussions: Sudha Yalamanchili (Georgia Tech), Mark Hummel (NVIDIA), Peter Zaspel (University of Basel), Tobias Grosser (ETH Zürich), Johannes Dörfert and Sebastian Hack (Saarland University)
Sponsors: BMBF, Google, NVIDIA, German Excellence Initiative

References

[1] Alexander Matz, Mark Hummel, Holger Fröning, Exploring LLVM Infrastructure for Simplified Multi-GPU Programming, MULTIPROG-2016, in conjunction with HiPEAC 2016.
[2] Sukumaran-Rajam, Aravind, et al. "Speculative runtime parallelization of loop nests: Towards greater scope and efficiency." HIPS+ LSPP 176, 2015.
[3] Lee, Janghaeng, Mehrzad Samadi, and Scott Mahlke. "Orchestrating multiple data-parallel kernels on multiple devices." 2015 IEEE International Conference on Parallel Architecture and Compilation (PACT), 2015.
[4] Ben-Nun, Tal, et al. "Memory access patterns: the missing piece of the multi-GPU puzzle." Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis. ACM, 2015.