CHAPEL-ON-HSA: TOWARDS SEAMLESS ACCELERATION OF CHAPEL PROGRAMS USING HSA

ABHISEK PAN & MICHAEL CHU • MAY 27, 2016

OUTLINE

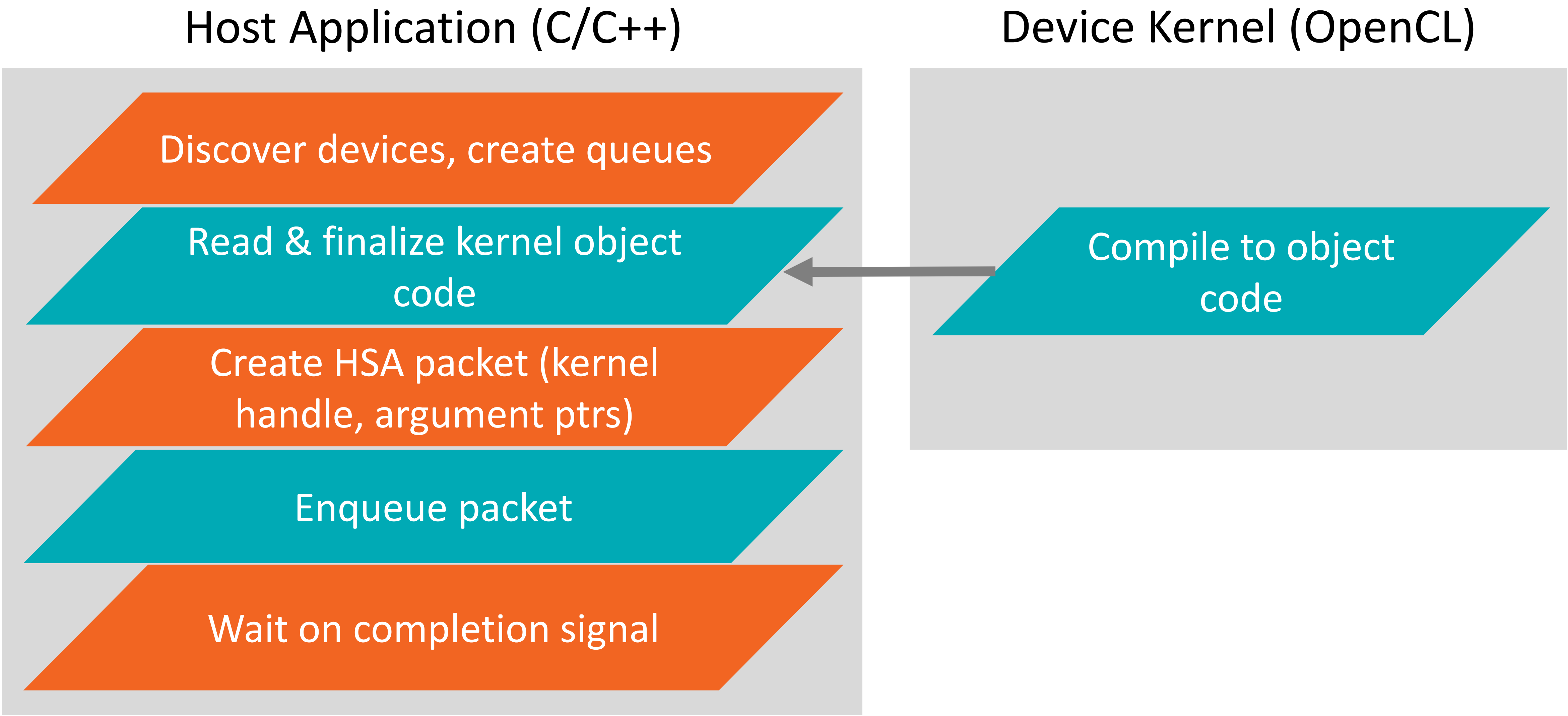
- Heterogeneous System Architecture (HSA)
- Exposing GPUs in Chapel – GPU sublocales
- Reduction
- Foralls
- Future Plans

THE HETEROGENEOUS SYSTEM ARCHITECTURE (HSA)

OPEN STANDARD PLATFORM SPECIFICATION

- Enables efficient, portable management of heterogeneous systems
- Shared address space abstraction
  - No explicit data movement
  - Single pointer across all devices
- Fast user-mode task dispatch
  - Shared memory queues for user-space direct packet enqueue
  - Fast user-space synchronization
- Multi-device support: GPUs, DSPs, FPGAs, NICs, PIM, etc.
  - A single task dispatch packet format across all devices
- Pre-emptive context-switching
- Open-source implementation
- Support for multiple higher-level languages

EXECUTING A GPU KERNEL



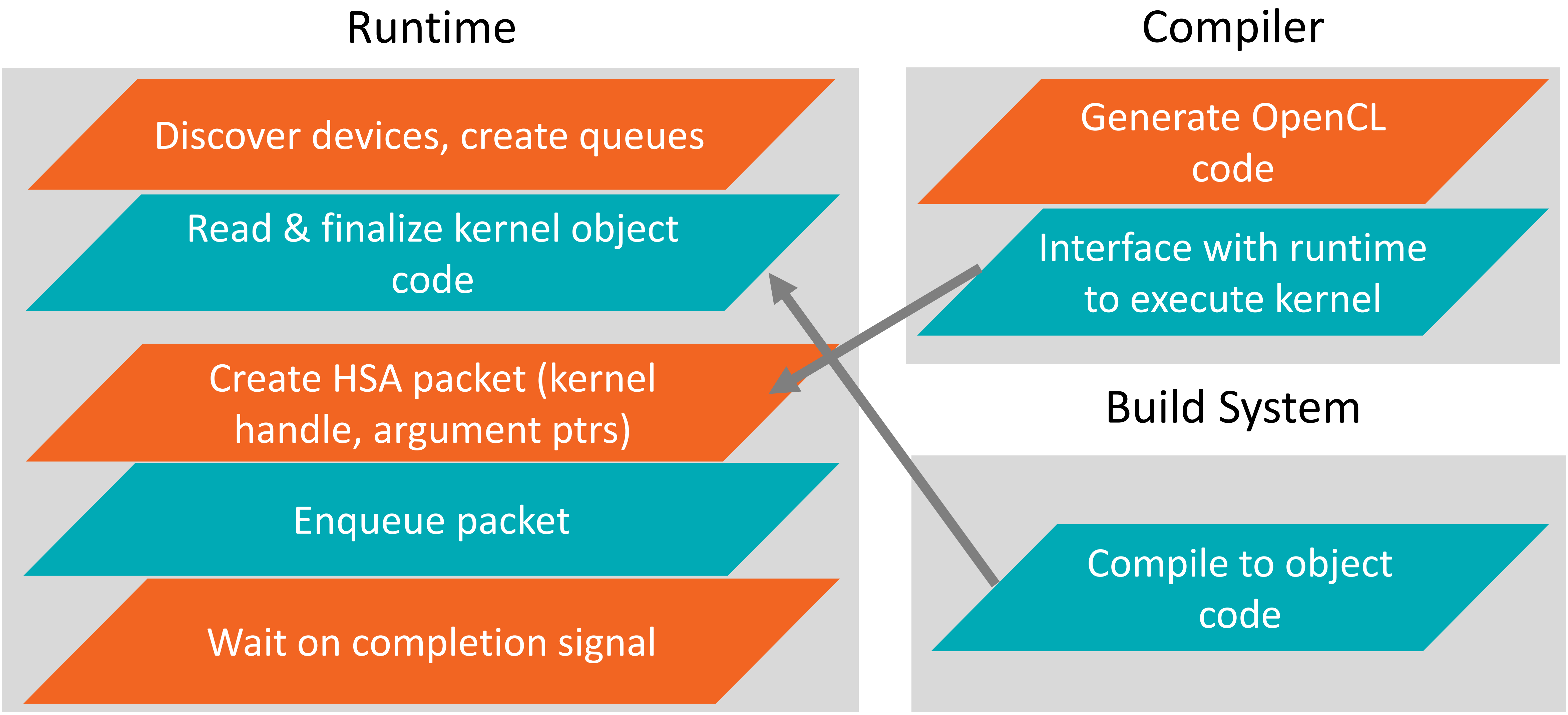
CHAPEL + HSA

GPU OFFLOAD CAPABILITY FOR DATA-PARALLEL CONSTRUCTS

- Native single-source GPU execution support
- Expose GPU execution capabilities in the language
  - Expose GPU as a sublocale
  - New “HSA” hierarchical locale with CPU and GPU sublocales
  - Any operation executed *on* a GPU sublocale gets executed on a GPU

```
on Locales[0] do {
    var A: [1..3] int = (1,2,3);
    on (Locales[0]:LocaleModel).GPU do {
       //Data-parallel constructs
       var sum = + reduce A;
   }
}
```

GPU EXECUTION IN CHAPEL



COMPILER MODIFICATIONS

CODE GENERATION AND KERNEL EXECUTION

- Parse
- Create Task Functions
- Parallel Transforms
- Optimizations
- C-Codegen

COMPILER MODIFICATIONS

AMD

Parse

- Insert new Chapel block for GPU offload
- Conditional execution of GPU code if sublocale is GPU

Create Task Functions

- Create new task functions for GPU blocks

Create GPU Functions

- Bundle core GPU executable in a new GPU function
- Maintain unique ids
- Capture arguments and bundle into 1 parameter

Parallel Transforms

Optimizations

C-Codegen

- Generate OpenCL code for GPU functions in .cl file
- Use runtime enqueue calls to enqueue functions using ids

OFFLOADING NODE-LOCAL REDUCTIONS

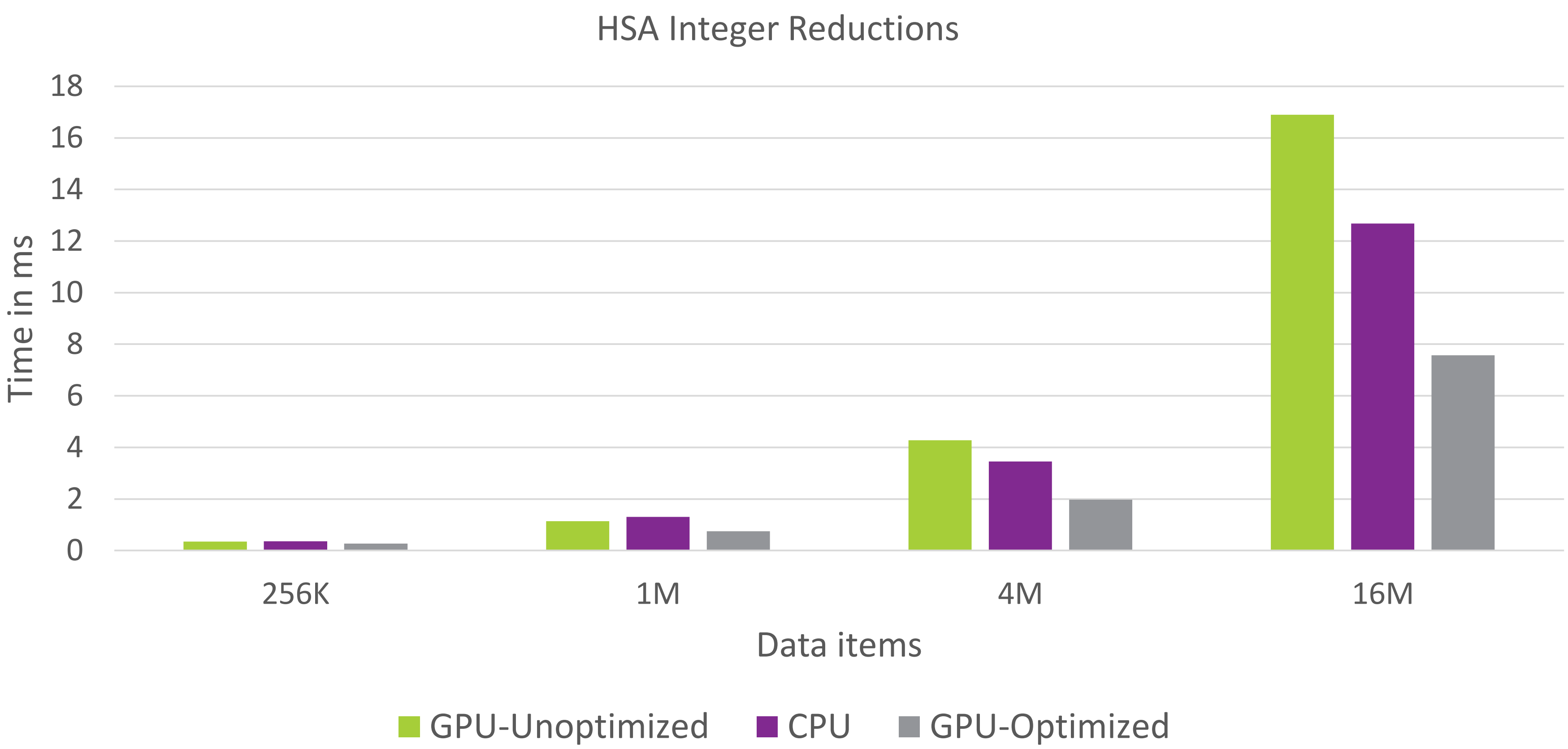
- Predefined reduction operators to reduce aggregate expressions to a single result

```
var A: [1..3] int = (1,2,3);
var sum = + reduce A;
```

- CPU reductions are pre-defined in the ChapelReduce module
- Similarly, we use precompiled OpenCL kernels
  - Separate OpenCL kernel for every <operator, data-type> pair
- Parser replaces a reduction expression with a call to the Chapel runtime routine
- GPU reductions are tricky!
  - Multiple calls to the kernels followed by processing on the CPU
  - Runtime orchestrates execution of multiple kernels
  - Direct translation of CPU code not appropriate

REDUCTION RESULTS

A10-7850K WITH RADEON™ R7 SERIES (4 CPU CORES @3.7 GHZ, 512 GPU CORES@720MHZ)



OFFLOADING NODE-LOCAL FORALLS

Data-parallelism

```
var A: [1..256] int;
forall i in {1..256} do
   A[i] = i;
```

CPU task-parallelism

```
var A: [1..256] int;
coforall j in {1..4} do {
   const lo = 1+(i–1)*64;
   const hi = lo + 63;
   for i in {lo..hi}
      A[i] = i;
}
```

4 tasks in parallel
Each task does 64 serial iterations

GPU thread-parallelism

```
var A: [1..256] int;
size_t i = get_global_id(0)
   A[i] = i;
```

256 work-items in parallel
4 workgroups
Each workgroup has 64 workitems

OFFLOADING NODE-LOCAL FORALLS

- Bundle the loop body in a new GPU-targeted function
- Estimate work-items and work group size
- Replace loop variables with OpenCL calls to obtain thread id
- Insert OpenCL specific keywords(“kernel”, “global”)
- Emit kernel code in .cl file

COMPILER

- Build system compiles the kernels to a GPU ISA using a llvm-based tool-chain

BUILD

- Kernel execution requests are sent to the runtime using the kernel-id

RUNTIME

FUTURE WORK

- Multi-node reductions and coforalls
- Multi-dimensional arrays
- Expose workgroup based resources
  - Local memory
  - Barriers
- Benchmarking
- Testing

Thank You!

We Are Hiring!

Abhisek Pan Abhisek.Pan@amd.com

Mike Chu Mike.Chu@amd.com

Interns, Co-ops, Post-docs

(Fall 2016, Spring 2017,…)

DISCLAIMER & ATTRIBUTION

The information presented in this document is for informational purposes only and may contain technical inaccuracies, omissions and typographical errors.

The information contained herein is subject to change and may be rendered inaccurate for many reasons, including but not limited to product and roadmap changes, component and motherboard version changes, new model and/or product releases, product differences between differing manufacturers, software changes, BIOS flashes, firmware upgrades, or the like. AMD assumes no obligation to update or otherwise correct or revise this information. However, AMD reserves the right to revise this information and to make changes from time to time to the content hereof without obligation of AMD to notify any person of such revisions or changes.

AMD MAKES NO REPRESENTATIONS OR WARRANTIES WITH RESPECT TO THE CONTENTS HEREOF AND ASSUMES NO RESPONSIBILITY FOR ANY INACCURACIES, ERRORS OR OMISSIONS THAT MAY APPEAR IN THIS INFORMATION.

AMD SPECIFICALLY DISCLAIMS ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE. IN NO EVENT WILL AMD BE LIABLE TO ANY PERSON FOR ANY DIRECT, INDIRECT, SPECIAL OR OTHER CONSEQUENTIAL DAMAGES ARISING FROM THE USE OF ANY INFORMATION CONTAINED HEREIN, EVEN IF AMD IS EXPRESSLY ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

ATTRIBUTION