Hewlett Packard Enterprise

RECENT GPU PROGRAMMING IMPROVEMENTS IN CHAPEL

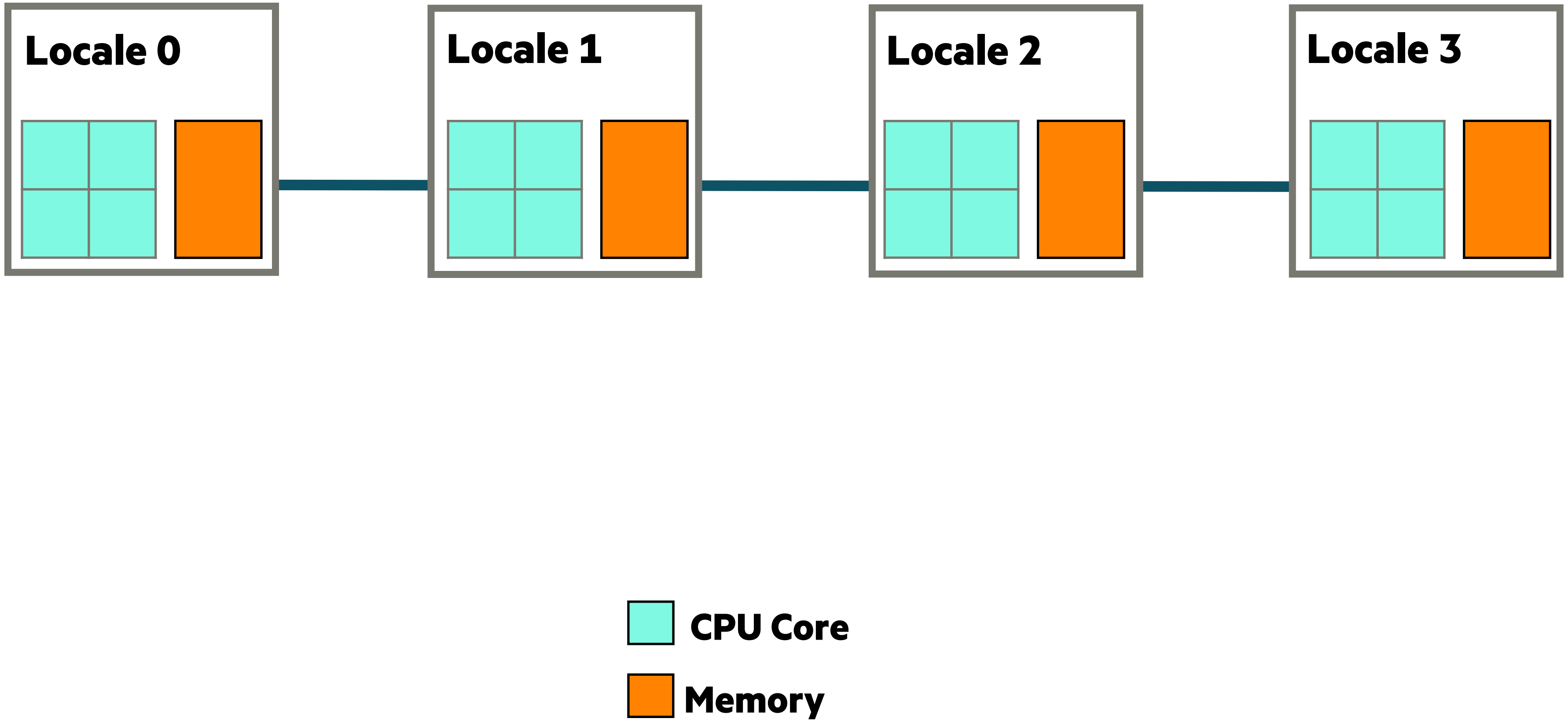
Engin Kayrakliioglu, Andy Stone, Daniel Fedorin

CHIUW 2023 - June 2nd, 2023

GPU PROGRAMMING IN CHAPEL

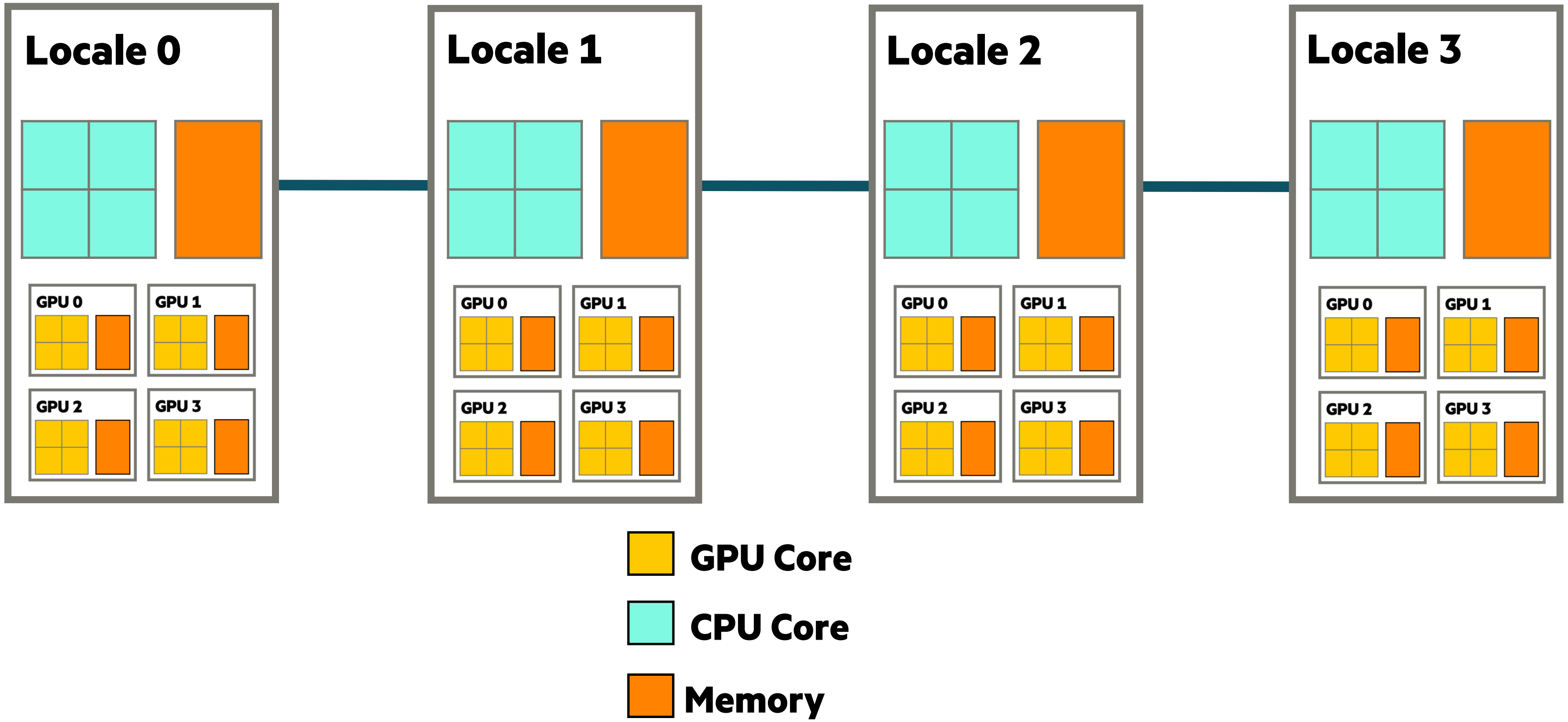
KEY CONCERNS FOR SCALABLE PARALLEL COMPUTING

1. **parallelism:** What tasks should run simultaneously?
2. **locality:** Where should tasks run? Where should data be allocated?
   - complicating matters, compute nodes now often have GPUs with their own processors and memory

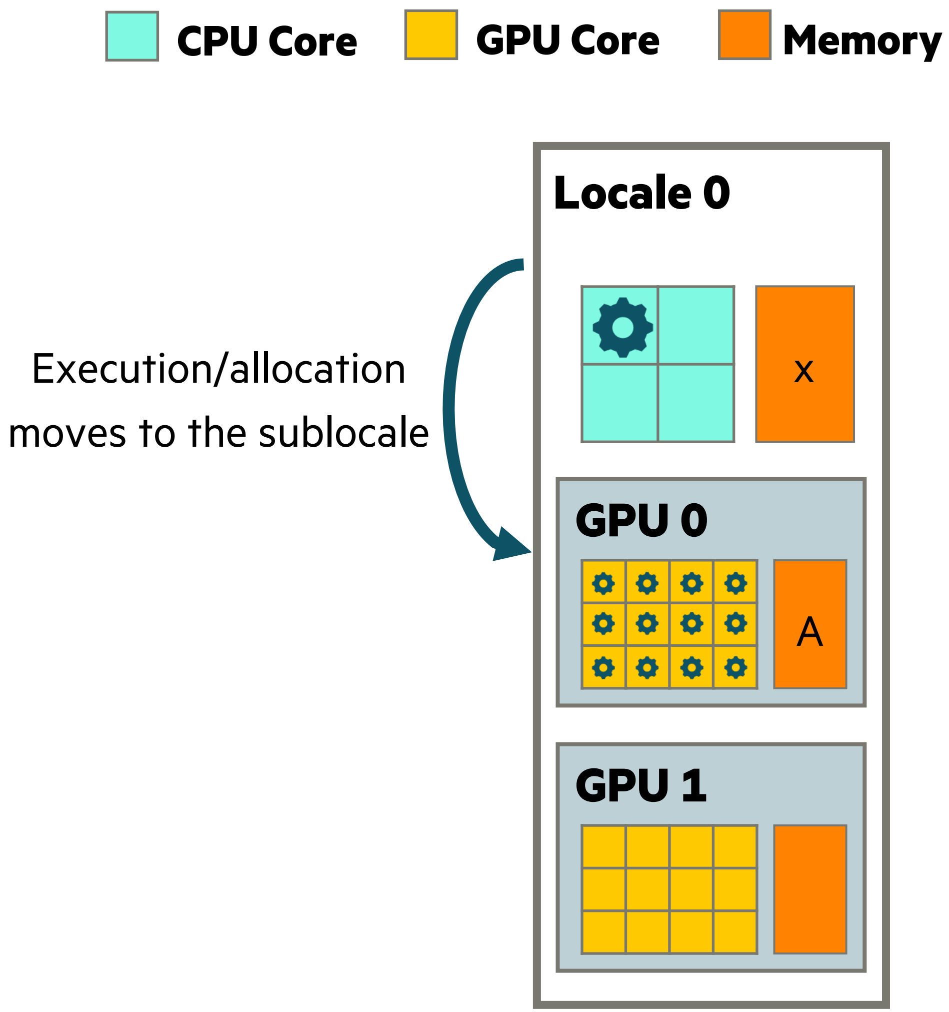


KEY CONCERNS FOR SCALABLE PARALLEL COMPUTING

1. **parallelism:** What tasks should run simultaneously?
2. **locality:** Where should tasks run? Where should data be allocated?
   - complicating matters, compute nodes now often have GPUs with their own processors and memory
   - we represent these as *sub-locales* in Chapel



PARALLELISM AND LOCALITY IN THE CONTEXT OF GPUS

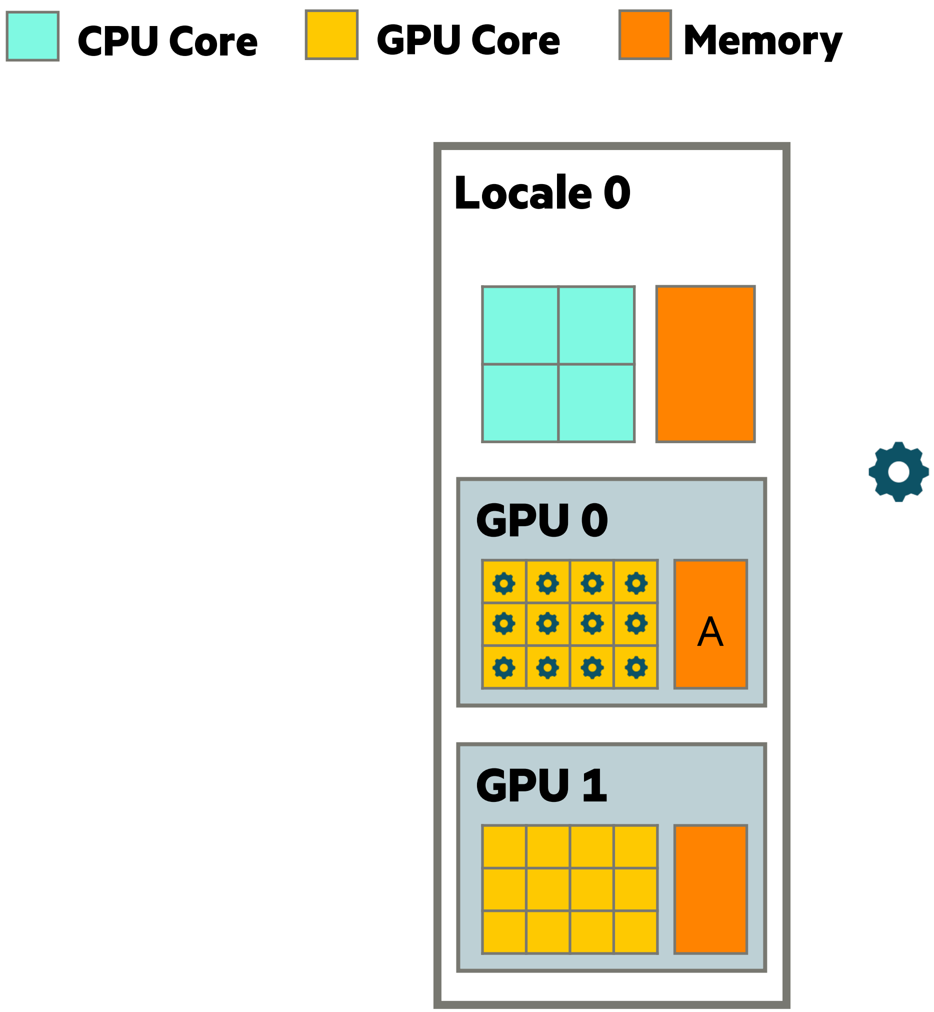



```
var x = 10;

  on here.gpus[0] {
   var A = [1, 2, 3, 4, 5, ...];
   A += 1;
  }

writeln(x);
```

PARALLELISM AND LOCALITY IN THE CONTEXT OF GPUS



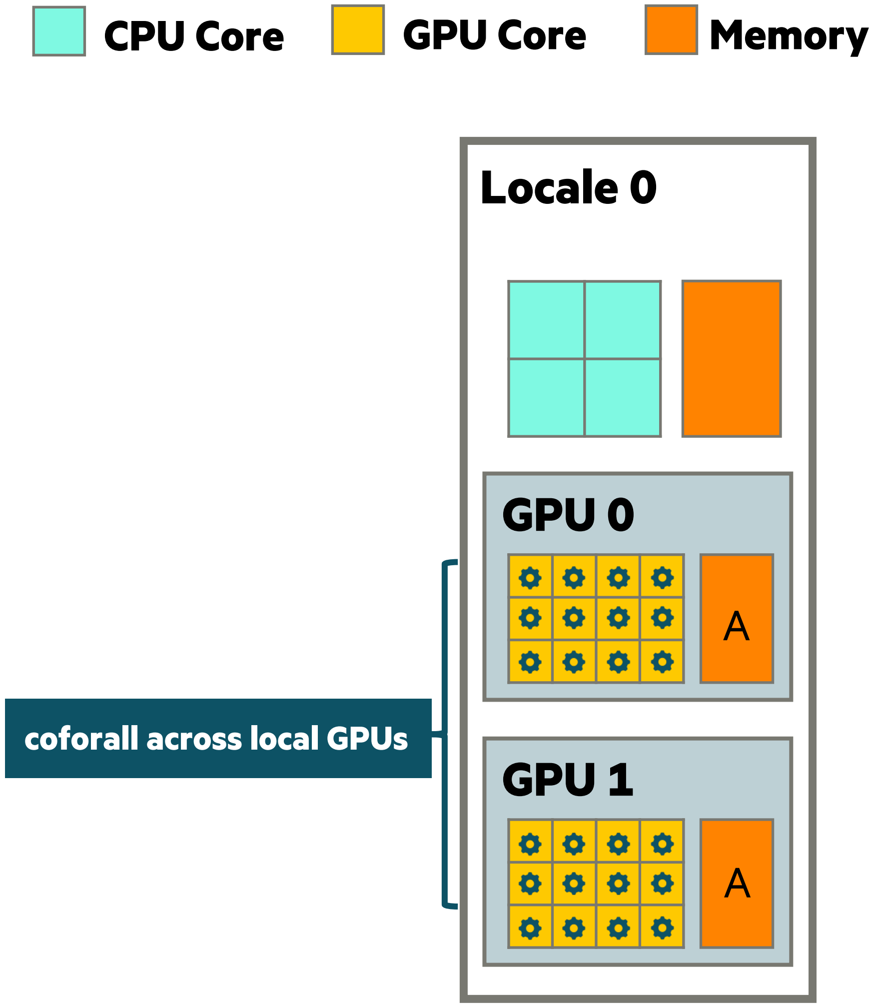

```
var x = 10;


  on here.gpus[0] {
   var A = [1, 2, 3, 4, 5, ...];
   A += 1;
  }




writeln(x);
```

PARALLELISM AND LOCALITY IN THE CONTEXT OF GPUS

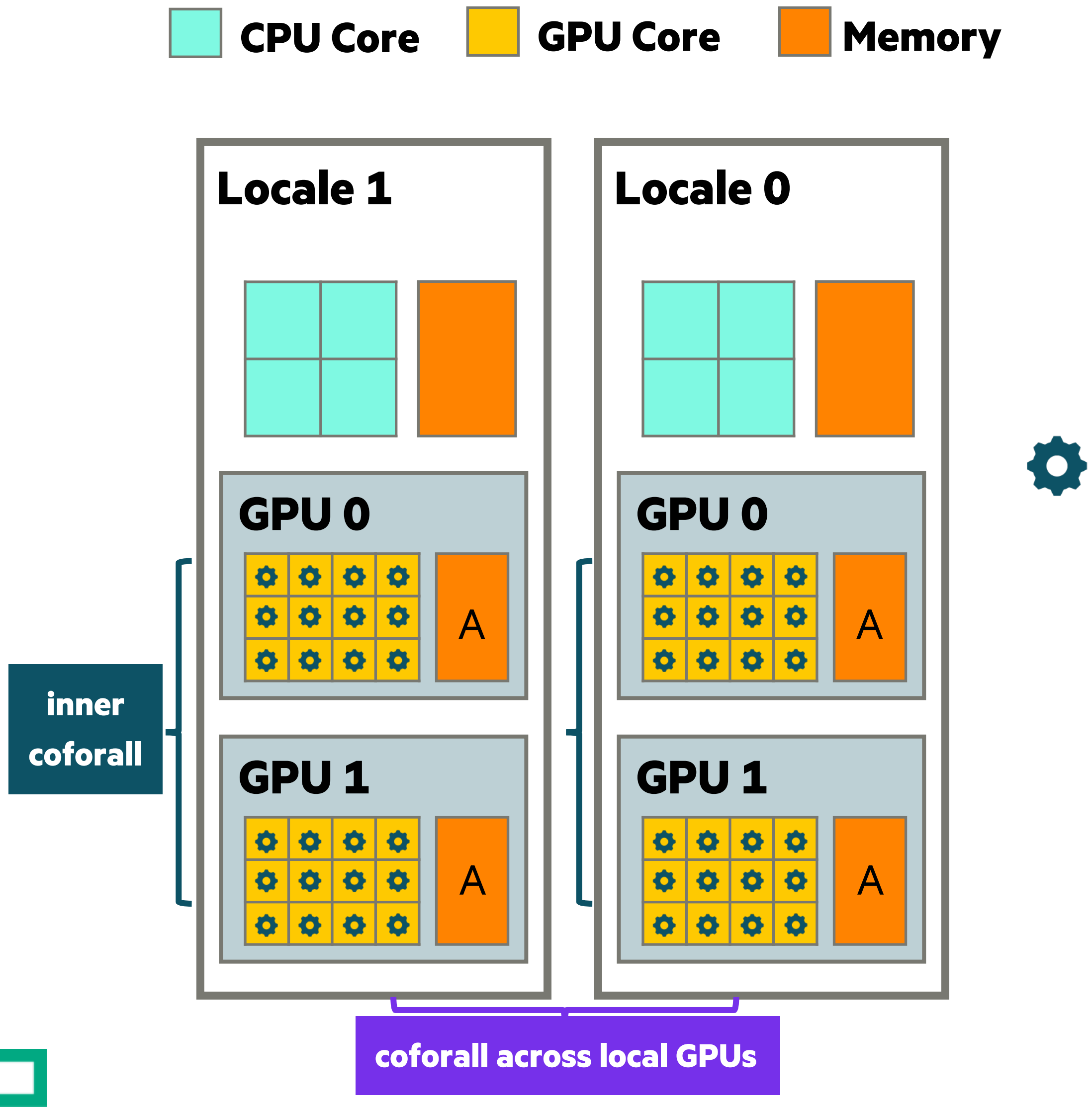



```
var x = 10;

  coforall g in here.gpus do on g {
   var A = [1, 2, 3, 4, 5, ...];
   A += 1;
  }

writeln(x);
```

PARALLELISM AND LOCALITY IN THE CONTEXT OF GPUS

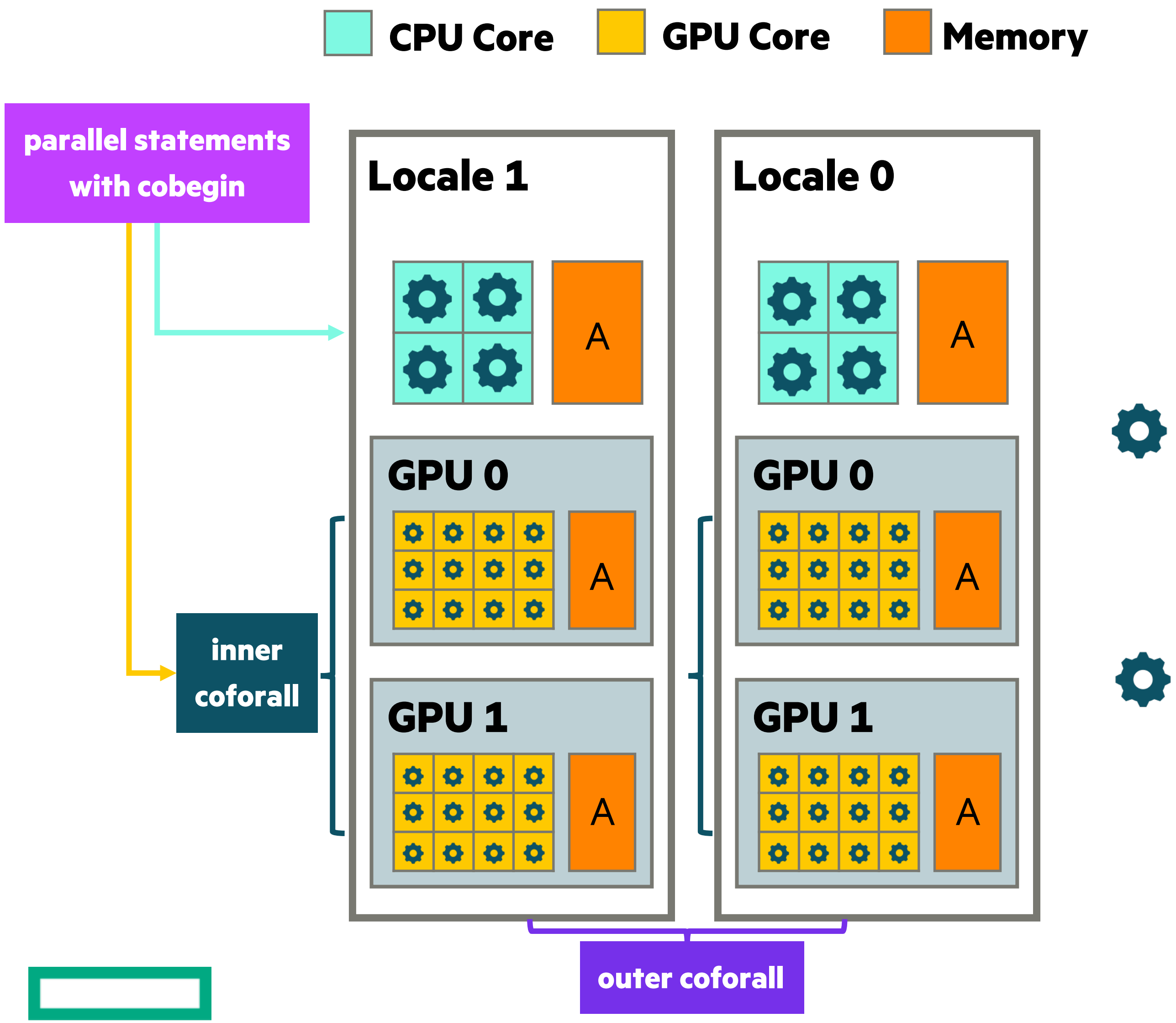



```
var x = 10;
coforall l in locales do on l {

  coforall g in here.gpus do on g {
   var A = [1, 2, 3, 4, 5, ...];
   A += 1;
  }

}
writeln(x);
```

PARALLELISM AND LOCALITY IN THE CONTEXT OF GPUS




```
var x = 10;
coforall l in locales do on l {
 cobegin {
  coforall g in here.gpus do on g {
   var A = [1, 2, 3, 4, 5, ...];
   A += 1;
  }
  {
   var A = [1, 2, 3, 4, 5, ...];
   A += 1;
  }
 }
}
writeln(x);
```

RECENT IMPROVEMENTS

FEATURES

CHIUW '22

- Fundamentals working

CHIUW '23

- GPU Module
  - assertOnGpu
  - setBlockSize
  - createSharedArray
  - atomic operations
- GpuDiagnostics Module
  - Count number of launches
  - Verbose kernel launches
- Initial profiler support

Future

- Portable features for forall/foreach
- foreach intents

GPU MODULE

- Has basic introspection/debugging support
  - assertOnGpu
    - Compilation fails if the loop is not GPU-eligible, execution fails if it is not run on a GPU sublocale
  - gpuWrite/gpuWriteln
  - gpuClock/gpuClocksPerSec
- A new standard module to support fundamental GPU operations
  - setBlockSize
  - createSharedArray
  - syncThreads

Next steps:

- We expect most of the functionality here to be implemented in existing Chapel features
  - i.e., writeln should replace gpuWriteln
- Designing new features for foreach/forall loops for fundamental GPU operations in a portable way
  - i.e., syncThreads can be implemented with some form of a barrier
  - i.e., the user should be able to query the GPU thread ID or Chapel task ID from inside a forall loop

GPU DIAGNOSTICS MODULE

- Basic diagnostics support modeled after CommDiagnostics module
  - Currently only supports counting/reporting kernel launches

```
startVerboseGpu();          // print a message out everytime a kernel is launched
startGpuDiagnostics();      // count kernel launches
on here.gpus[0] {
  foreach i in 1..10 do     // gpu-eligible operations
}
stopGpuDiagnostics();
stopVerboseGpu();
writeln(getGpuDiagnostics());
```

Output:

```
0 (gpu 0): foo.chpl:4: kernel launch (block size: 512x1x1) # via startVerboseGpu()
(kernel_launch: 1)   # via getGpuDiagnostics()
```

PROFILER SUPPORT

Background:

- Debugging and profiling GPU kernels are typically more difficult than CPU applications
  - I/O support is typically poor, execution model is less intuitive, esoteric challenges
- NVIDIA has numerous profilers, where NSight Compute is used for profiling kernel performance
  - While using profilers for Chapel in general is not very straightforward, focusing on kernels is easier
- Out-of-the-box: NSight Compute was able to show line-by-line hardware counters when '-g' was used
  - However, '--fast -g' thwarted assembler optimizations → reduced kernel performance → less valuable profiling

This Effort:

- Added the '--gpu-ptxas-enforce-optimizations' flag to ensure that assembler optimizations are enabled

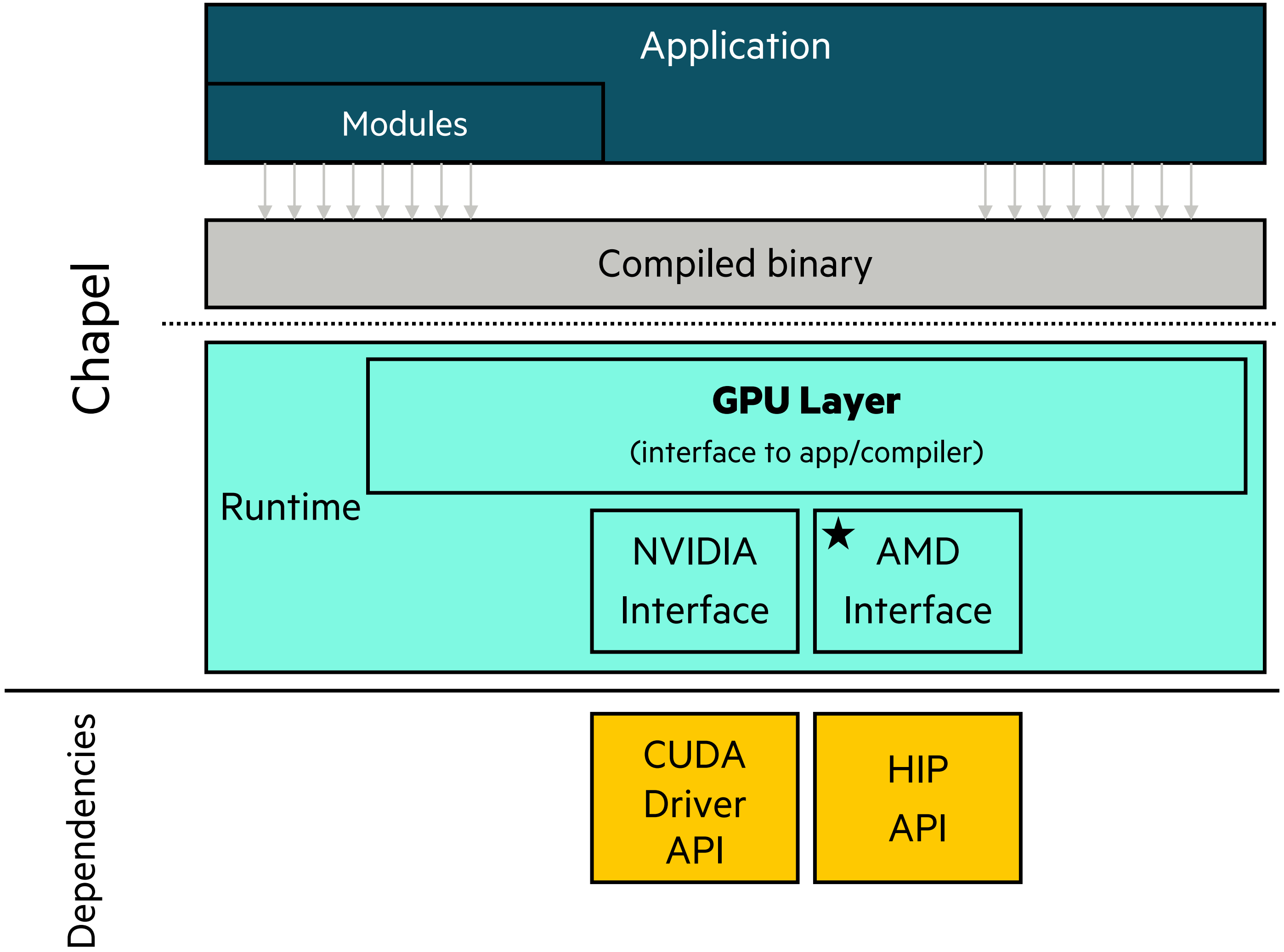
Impact:

- Significant help while trying to understand performance of compiler-generated kernels
  - Kernel performance is virtually unaffected
  - Profiler shows line-by-line information accurately
- Can compare performance behavior of a reference version against the Chapel version

PORTABILITY

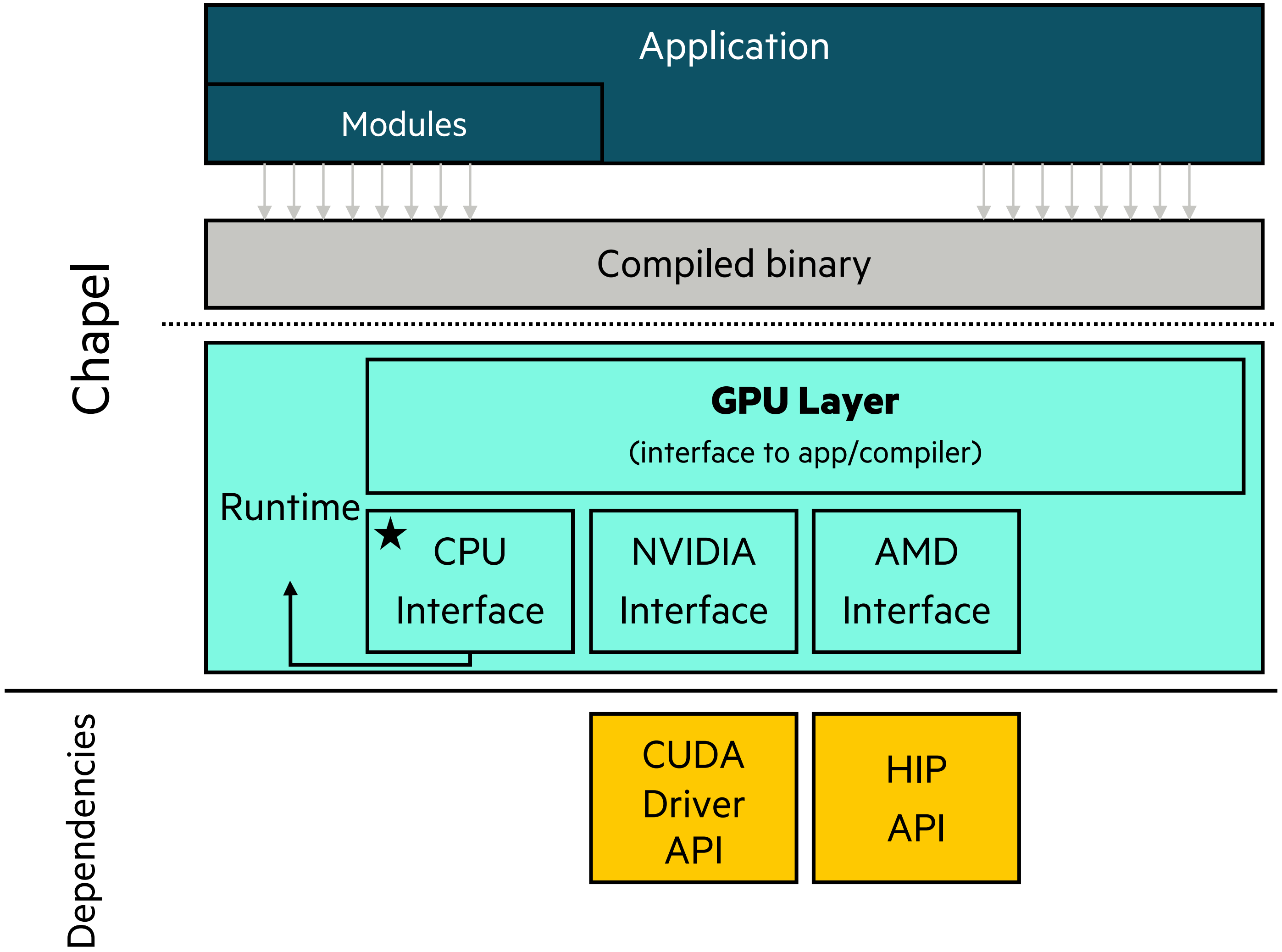
CHIUW '22	CHIUW '23	Future
• Monolithic runtime • CUDA Driver API wrapper	• Modular runtime • AMD support • "cpu-as-device" mode	• Intel support

AMD SUPPORT



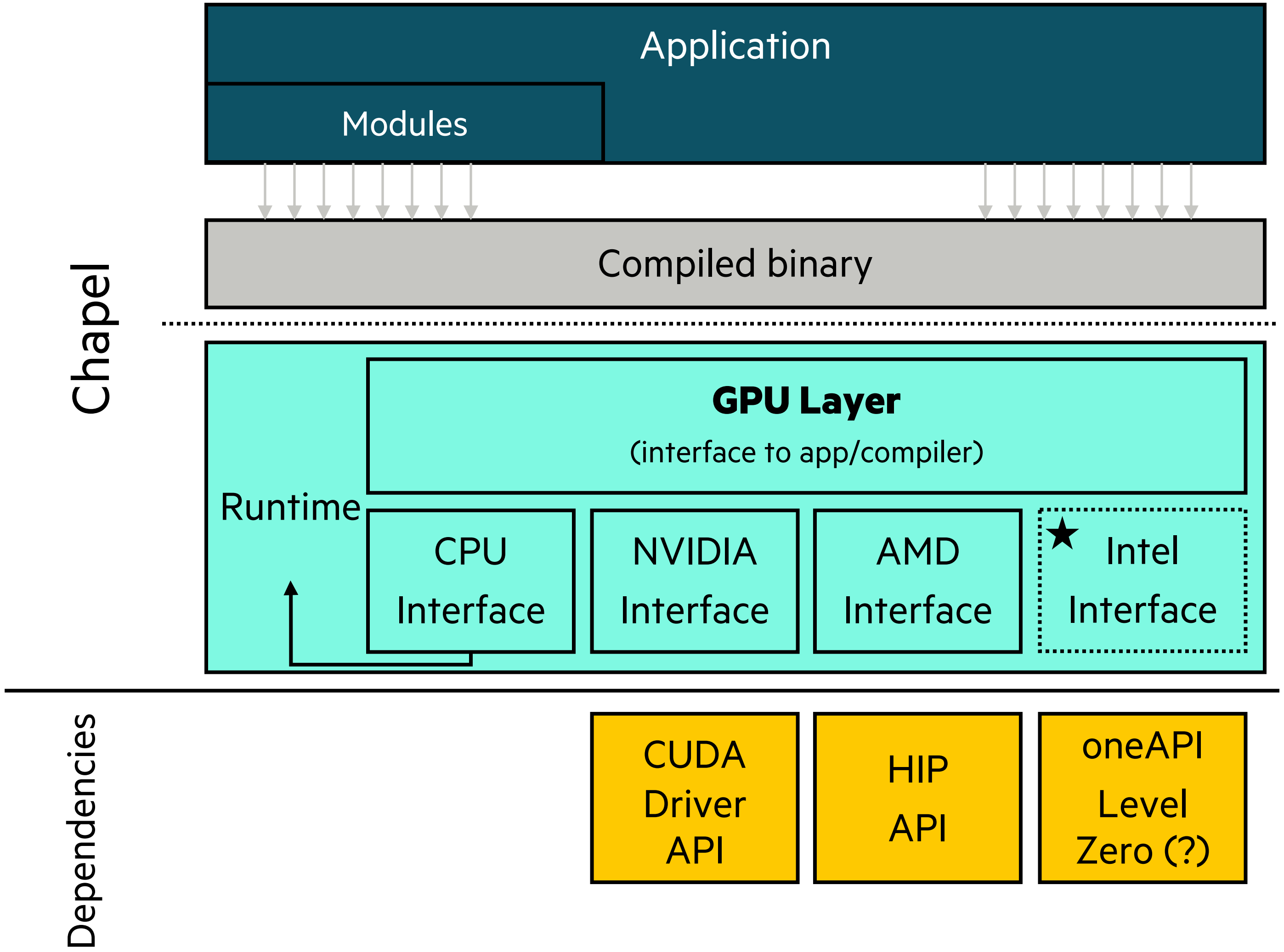
- AMD GPUs are now supported
  - **1.30:** single-locale
  - **1.31 pre-release:** multilocale
- Features/correctness
  - Mostly on-par with NVIDIA
  - Some 64-bit math functions aren't supported
- Performance
  - Only initial studies so far
  - (more on this later)

CPU-AS-DEVICE SUPPORT



- Supports GPU locale model w/o GPUs
  - GPU functionality is diverted to CPU
- Loops are outlined for kernel generation
  - however, they are not compiled/launched
  - loop always runs for equivalent correctness
- Use-cases
  - assertOnGpu to confirm GPU offload
  - GpuDiagnostics to count/report "launches"

FUTURE WORK: INTEL GPU SUPPORT



- Intel GPU support is the next target
  - Modular runtime should help
  - Compilation is a bigger question
    - GPU transformations should be the same across vendors
    - LLVM generates the code for us;
    - but no Intel GPU target, yet
    - Intel's LLVM fork can target Intel GPUs
    - Potential solution: rely on system LLVM that's Intel-GPU-enabled

- Implementation has not started, yet

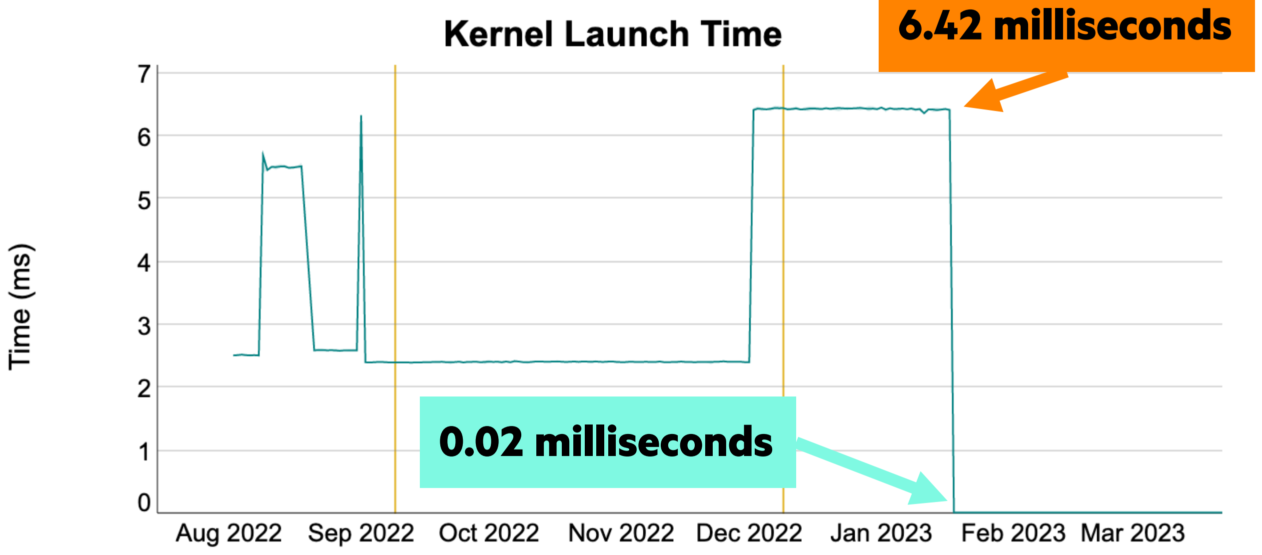
PERFORMANCE

CHIUW '22	CHIUW '23	Future

• Initial studies	• Significant performance improvements
	  • Faster kernel launch
	  • Faster kernel execution
	• "array-on-device" mode
	• New benchmarks
	• Initial AMD experiments
		• "array-on-device"
		• Compiler optimizations

GENERAL PERFORMANCE IMPROVEMENTS

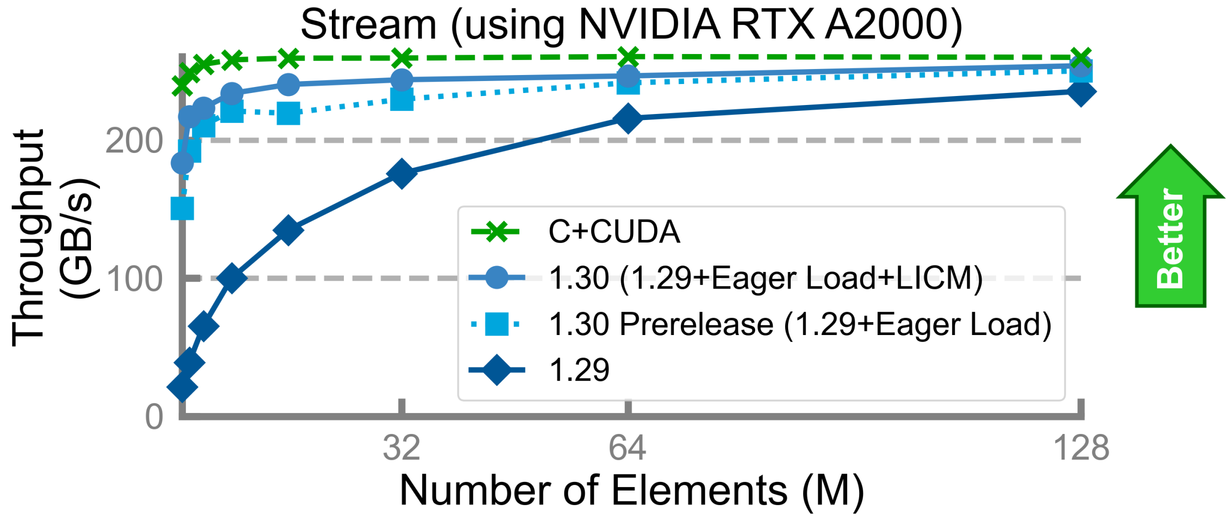
- **Eager binary loading**
  - GPU binary is loaded at application launch
  - ~300x faster kernel launch times
- **Loop-invariant code motion before GPU pass**
  - Loops are optimized before turning into kernels
  - Faster kernel execution
- **Application-level optimizations with ChOp***



N-Queens Performance with ChOp

(1x NVIDIA P100)

N	Interop (s)	Native (s)	Off by
15	0.30	0.36	*19%*
16	1.79	2.06	*15%*
17	12.47	14.76	*18%*
18	94.94	110.98	*17%*



*Tiago Carneiro , Nouredine Melab, Jan Gmys, Guillaume Helbecque. et al. — INRIA Lille, France; Imec, Belgium; University of Mons, Belgium; et al.

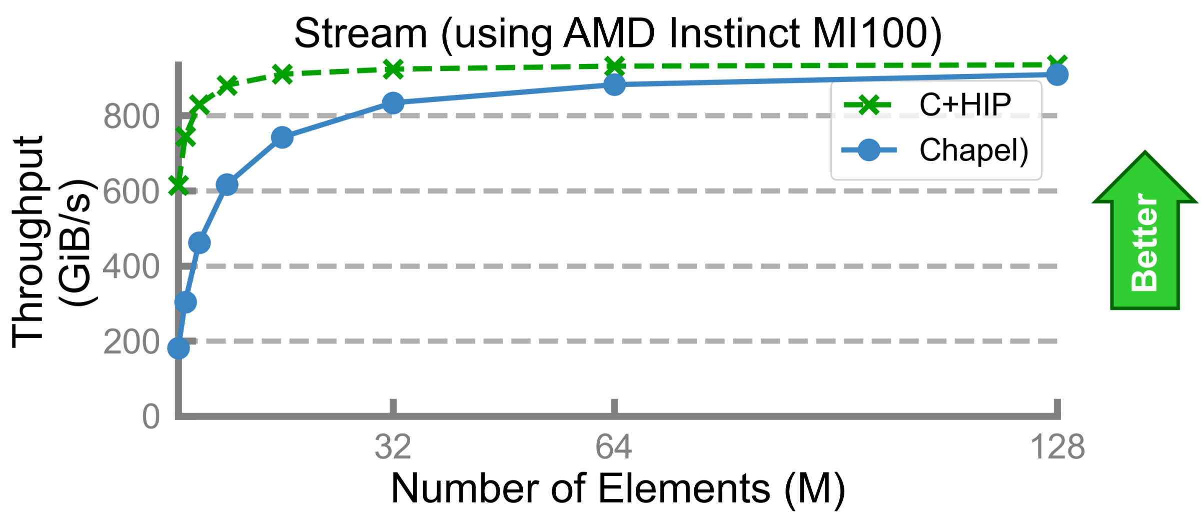
WORK IN PROGRESS: AMD PERFORMANCE

HPCC-Stream: Similar to NVIDIA

- Maybe a little lower with smaller data
  - potentially due to higher kernel launch costs

ChOp: Performance drops with larger data

- Investigation is pending



N-Queens Performance with ChOp

(1x AMD MI100)

N	Interop (s)	Native (s)	Off by
15	0.40	0.55	*36%*
16	1.14	2.18	*91%*
17	6.36	13.28	*209%*
18	47.04	115.51	*246%*

FUTURE WORK

Time (s)

(RTX A2000)

Array initialization on CPU is the current focus

Unified Memory	Array on Device
0.12	18.16
0.038	0.018

Faster initialization on GPU

Unified Memory	Array on Device
0.25	0.033
0.14	0.034

Faster data movement

```
var CpuArr: [1..n] int;

on here.gpus[0] {

  var GpuArr: [1..n] int;

  GpuArr = CpuArr;

  CpuArr = GpuArr;
}
```

How memory is allocated

	Unified Memory	Array on Device
metadata	host	host
data	host	host (registered)

metadata	managed	managed
data	managed	device

SUMMARY

Status so far:

- Can target NVIDIA and AMD GPUs in single- and multilocale
- The performance has been significantly improved, but there's more room
- Fundamental GPU operations are supported via a standard module
- Diagnostics and introspection tools can help performance analysis and optimization

Next steps:

- Target Intel GPUs
- More performance improvements
- New features to support portable programming between GPU- and vector-based execution

THANK YOU

engin@hpe.com