Georgia Tech

GPUAPI: Multi-level Chapel Runtime API for GPUs

CHIUW2021, June 4, 2021

Akihiro Hayashi (Georgia Tech)

Sri Raj Paul (Georgia Tech)

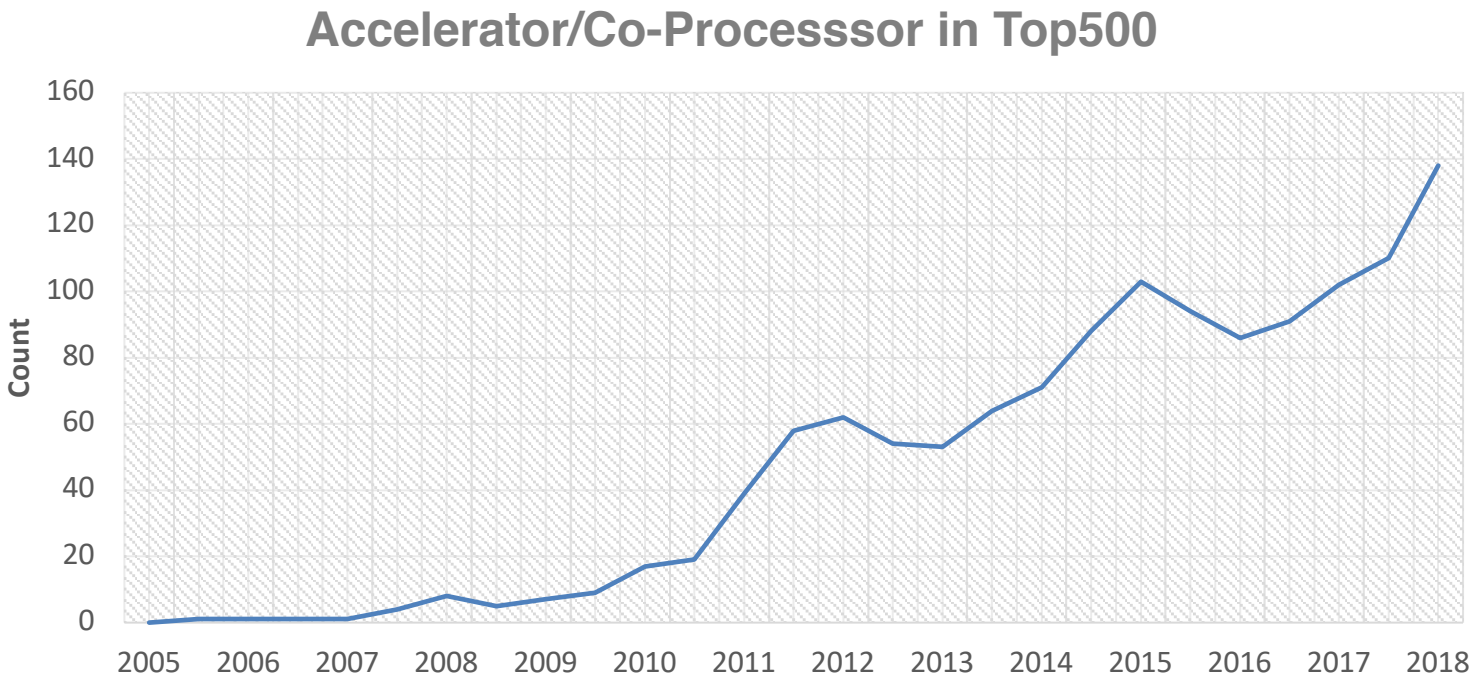
Vivek Sarkar (Georgia Tech)

GPUAPI: Multi-level Chapel Runtime API for GPUs

MOTIVATION

Georgia Tech

GPUs are “essential” in HPC



All plan to include GPUs!

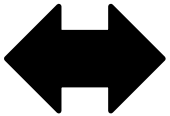
2021-

Source: https://www.top500.org/statistics/list/
Image Sources: ornl.gov, anl.gov, llnl.gov

GPU Programming in Chapel: no “intermediate” programming model

Highest-level Chapel-GPU Programming
(Prototype GPU code generator in Chapel 1.24)

```
1 forall i in 1..n {
2   A(i) = B(i) + alpha * C(i);
3 }
```



A huge gap!

Lowest-level Chapel-GPU Programming
(C Interoperability only or GPUIterator)

```
1  // separate C file
2  __global__ void stream(float *dA, float *dB, float *dC,
3                         float alpha, int N) {
4      int id = blockIdx.x * blockDim.x + threadIdx.x;
5      if (id < N) {
6              dA[id] = dB[id] + alpha * dC[id];
7      }
8  }
9  void GPUST(float *A, float *B, float *C, float alpha,
10            int start, int end, int GPUN) {
11   float *dA, *dB, *dC;
12   CudaSafeCall(cudaMalloc(&dA, sizeof(float) * GPUN));
13   CudaSafeCall(cudaMalloc(&dB, sizeof(float) * GPUN));
14   CudaSafeCall(cudaMalloc(&dC, sizeof(float) * GPUN));
15   CudaSafeCall(cudaMemcpy(dB, B + start, sizeof(float) *
16                           GPUN, cudaMemcpyHostToDevice));
17   CudaSafeCall(cudaMemcpy(dC, C + start, sizeof(float) *
18                           GPUN, cudaMemcpyHostToDevice));
19
20   stream<<<ceil(((float)GPUN)/1024), 1024>>>
21                                  (dA, dB, dC, alpha, GPUN);
22   CudaSafeCall(cudaDeviceSynchronize());
23   CudaSafeCall(cudaMemcpy(A + start, dA, sizeof(float) *
24                           GPUN, cudaMemcpyDeviceToHost));
25   CudaSafeCall(cudaFree(dA));
26   CudaSafeCall(cudaFree(dB));
27   CudaSafeCall(cudaFree(dC));
28 }
```

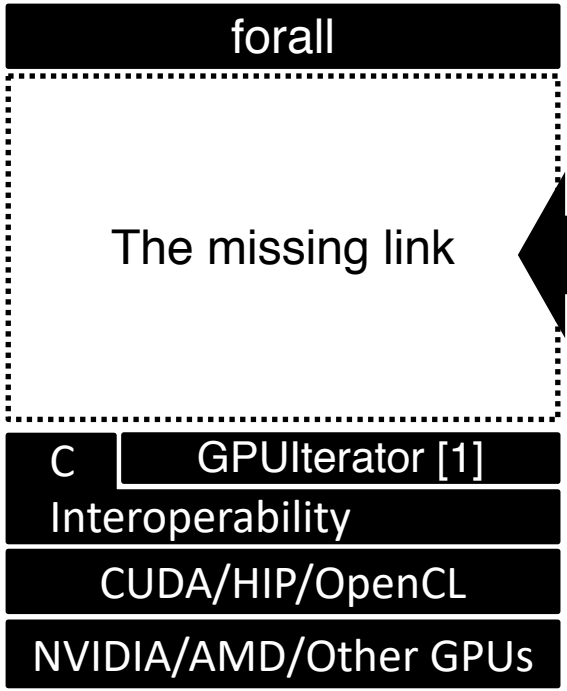
Research Question:
What is an appropriate and portable programming interface that bridges the ”forall” and GPU versions?

Georgia Tech

Big Picture: A Multi-level Chapel GPU Programming Model

HIGH-level:
The compiler compiles forall to CUDA, HIP, and OpenCL

LOW-level:
The user prepares full GPU programs and invokes them from Chapel (w/ or w/o the GPUIterator)



Our proposal

Chapel programmer friendly GPU APIs :
MID-level
```
var dA = new GPUArray(A);
dA.toDevice();
```

Thin wrappers for low-level GPU APIs:
MID-LOW-level
```
Malloc(); Memcpy();
```

Goal: increase productivity with no performance loss

[1] Akihiro Hayashi, Sri Raj Paul, Vivek Sarkar
"GPUIterator: Bridging the gap between Chapel and GPU Platforms" (CHIUW'19)

Contributions

- Why Chapel-level GPU API?
  - For improving productivity

    Our observation: The complexity in GPU programming comes not only from writing GPU kernels in the device part, but also from writing the host part

    ✓ Our GPUAPI is designed to simplify the host part
  - For improving portability

    Our observation: There are different GPU programming models from different vendors

    ✓ Our GPUAPI is implemented to work on different platforms (NVIDIA, AMD, Intel, …)
- Contributions:
  - Design and implementation of two tiers of Chapel Runtime GPU API
    - ✓ MID-level: Chapel programmer friendly GPU APIs
    - ✓ MID-LOW-level: Thin wrappers for low-level GPU APIs
  - Performance and productivity evaluations on different distributed and single-node platforms using micro benchmark and real-world applications

Georgia Tech

GPUAPI: Multi-level Chapel Runtime API for GPUs

DESIGN

Georgia Tech

Chapel GPU API Design: Summary

- Use case:
  - The user would like to 1) write GPU kernels, or 2) utilize highly-tuned GPU libraries, and would like to stick with Chapel for the other parts (allocation, data transfers)
- Provides two levels of GPU API
  - MID-LOW: Provides wrapper functions for raw GPU APIs
    Example: `var ga: c_void_ptr = GPUAPI.Malloc(sizeInBytes);`
  - MID: Provides more user-friendly APIs
    Example: `var ga = new GPUArray(A);`
- Note
  - The user is still supposed to write kernels in CUDA/HIP/OpenCL
  - The APIs significantly facilitates the orchestration of:
    - ✓ Device memory (de)allocation, and host-to-device/device-to-host data transfers,
  - The use of the APIs does not involve any modifications to the Chapel compiler
  - Can work with the prototype GPU code generator in Chapel 1.24

Georgia Tech

Chapel GPU API Design: MID-LOW GPU API

- Summary
  - Provides the same functionality as CUDA/HIP/OpenCL
  - The user is still supposed to write CUDA/HIP/OpenCL kernels
  - The user is supposed to handle both C types and Chapel types
- Key APIs
  - Device Memory Allocation
    - ✓ `Malloc(…);`
    - ✓ `MallocPitch(…);`
  - Host-to-device, and device-to-host data transfers
    - ✓ `Memcpy(…);`
    - ✓ `Memcpy2D(…);`
  - Ensuring the completion of GPU computations
    - ✓ `DeviceSynchronize();`
  - Device Memory deallocation
    - ✓ `Free(…);`

Georgia Tech

Chapel GPU API Design: MID GPU API

- Summary
  - More natural to Chapel programmers
  - The user is still supposed to write CUDA/HIP/OpenCL kernels
- Key APIs
  - Device Memory Allocation
    - ✓ `var dA = new GPUArray(A);`
    - ✓ `var dA = new GPUJaggedArray(A);`
  - Host-to-device, and device-to-host data transfers
    - ✓ `ToDevice(dA:GPUArray, …); FromDevice(dA: GPUArray, …);`
    - ✓ `dA.ToDevice(); dA.fromDevice();`
  - Implicit Device Memory deallocation
    - ✓ Automatically “freed” when a GPUArray/GPUJaggedArray object is deleted
  - Explicit Device Memory deallocation
    - ✓ `delete`

Georgia Tech

Chapel GPU API Design: MID-LOW/MID GPU API Example

MID-LOW Level

```
1  use GPUAPI;
2  var A: [1..n] real(32);
3  var B: [1..n] real(32);
4  var C: [1..n] real(32);
5  var dA, dB, dC: c_void_ptr;
6  var size: size_t =
7     (A.size:size_t * c_sizeof(A.eltType));
8  Malloc(dA, size);
9  Malloc(dB, size);
10 Malloc(dC, size);
11 Memcpy(dB, c_ptrTo(B), size, TODEVICE);
12 Memcpy(dC, c_ptrTo(C), size, TODEVICE);
13 LaunchST(dA, dB, dC, alpha, N: size_t);
14 DeviceSynchronize();
15 Memcpy(c_ptrTo(A), dA, size, FROMDEVICE);
16 Free(dA); Free(dB); Free(dC);
```

MID-level

```
1  use GPUAPI;
2  var A: [1..n] real(32);
3  var B: [1..n] real(32);
4  var C: [1..n] real(32);
5  var dA = new GPUArray(A);
6  var dB = new GPUArray(B);
7  var dC = new GPUArray(C);
8  toDevice(dB, dC);
9  LaunchST(dA.dPtr(), dB.dPtr(),
10          dC.dPtr(), alpha,
11          dN: size_t);
12 DeviceSynchronize();
13 FromDevice(dA);
14 Free(dA, dB, dC);
```

Georgia Tech

For more details, please see https://ahayashi.github.io/chapel-gpu/index.html

Example: Single-node execution of STREAM (MID-level w/ GPUIterator)

```
1
2   var A: [1..n] real(32);
3   var B: [1..n] real(32);
4   var C: [1..n] real(32);
5   var GPUCallBack = lambda (lo: int, hi: int, nElems: int) {
6     var dA = new GPUArray(A);
7     var dB = new GPUArray(B);
8     var dC = new GPUArray(C);
9     toDevice(dB, dC);
10    LaunchST(dA.dPtr(), dB.dPtr(),
11             dC.dPtr(), alpha,
12             dN: size_t);
13    DeviceSynchronize();
14    FromDevice(dA);
15    Free(dA, dB, dC);
16  };
17  forall i in GPU(1..n, GPUCallBack,
18                  CPUPercent) {
19    A(i) = B(i) + alpha * C(i);
20  }
```

The user has the option of writing device functions, device lambdas, or library calls

```
1  // separate C file (CUDA w/ device lambda)
2  void LaunchST(float *dA, float *dB,
3                float *dC, float alpha, int N) {
4    GPU_FUNCTOR(N, 1024, NULL,
5      [=] __device__ (int i) {
6        dA[i] = dB[i] + alpha * dC[i];
7      });
8  }
```

Example: Distributed execution of STREAM (MID-level w/ GPUIterator)

```
1  var D: domain(1) dmapped Block(boundingBox={1..n}) = {1..n};
2  var A: [D] real(32);
3  var B: [D] real(32);
4  var C: [D] real(32);
5  var GPUCallBack = lambda (lo: int, hi: int, nElems: int) {
6    var dA = new GPUArray(A.localSlice(lo..hi));
7    var dB = new GPUArray(B.localSlice(lo..hi));
8    var dC = new GPUArray(C.localSlice(lo..hi));
9    toDevice(dB, dC);
10   LaunchST(dA.dPtr(), dB.dPtr(),
11            dC.dPtr(), alpha,
12            dN: size_t);
13   DeviceSynchronize();
14   FromDevice(dA);
15   Free(dA, dB, dC);
16 };
17 forall i in GPU(D, GPUCallBack,
18                 CPUPercent) {
19   A(i) = B(i) + alpha * C(i);
20 }
```

The user has the option of writing device functions, device lambdas, or library calls

```
1  // separate C file (CUDA w/ device lambda)
2  void LaunchST(float *dA, float *dB,
3                float *dC, float alpha, int N) {
4    GPU_FUNCTOR(N, 1024, NULL,
5      [=] __device__ (int i) {
6        dA[i] = dB[i] + alpha * dC[i];
7      });
8  }
```

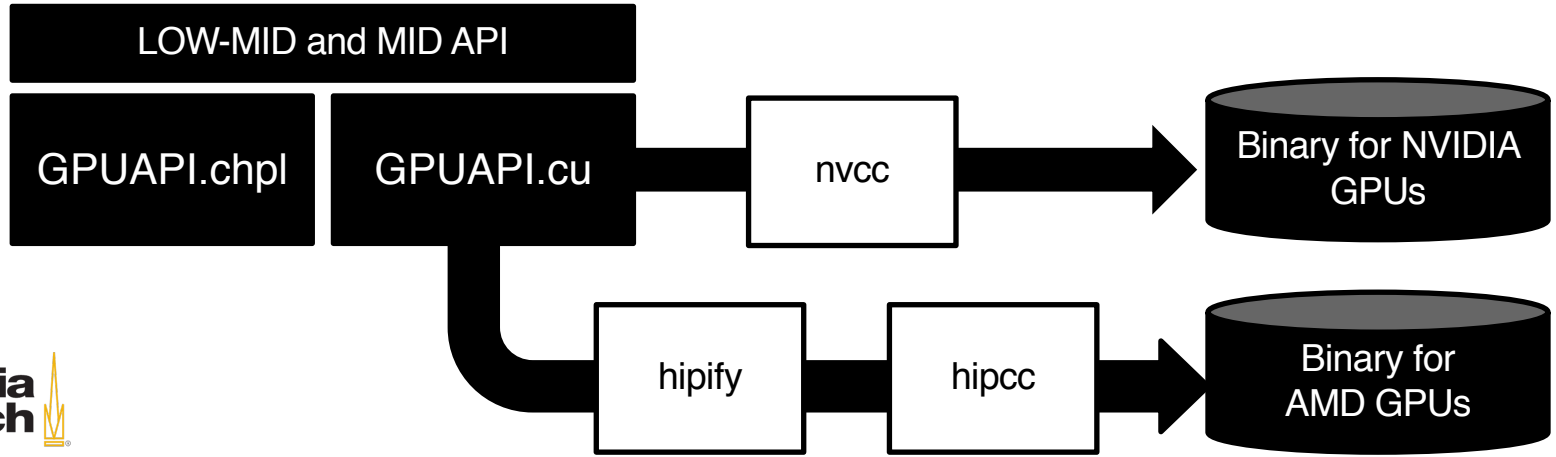
GPUAPI: Multi-level Chapel Runtime API for GPUs

IMPLEMENTATION

Georgia Tech

Implementation

- Provides an external module (GPUAPI)
  - Can be used either stand-alone or with the GPUIterator module
  - Provides a cmake-based build system for building GPU-dependent codes
- Currently supports NVIDIA and ROCM-ready AMD GPUs
  - OpenCL is also supported
  - SYCL backend is under development



GPUAPI: Multi-level Chapel Runtime API for GPUs

PERFORMANCE & PRODUCTIVITY EVALUATIONS

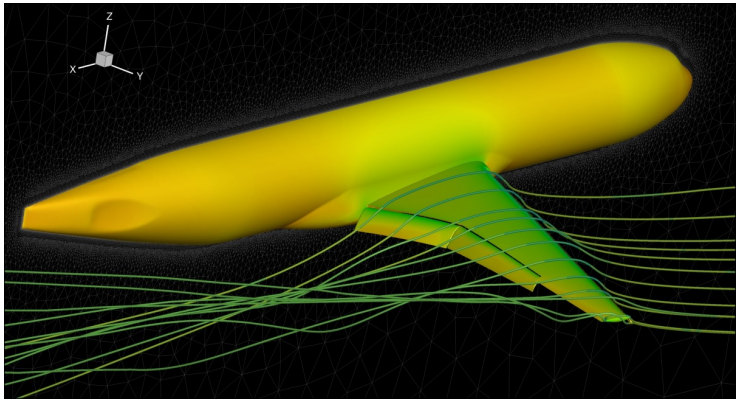
Georgia Tech

Performance & Productivity Evaluations

- Platforms
  - Cori GPU@NERSC: Intel Xeon (Skylake) + NVIDIA V100 GPU
  - Summit@ORNL: IBM POWER9 + NVIDIA Tesla V100 GPU
  - A single-node AMD machine: Ryzen9 3900 + Radeon RX570
- Applications
  - Micro-benchmark: STREAM, BlackScholes, Matrix Multiplication, Logistic Regression
  - Real-world applications: Champs, Distributed Exact Optimization
- Chapel Compilers & Options
  - Micro-benchmark: Chapel Compiler 1.20.0 with the --fast option
  - Champs: Chapel Compiler 1.22.0 with the --fast option
  - Distributed exact optimization: 1.24.0 with the --fast option
- GPU Compilers
  - CUDA: NVCC 10.2 (Cori), 10.1 (Summit) with the -O3 option
  - AMD: ROCM 2.9.6, HIPCC 2.8 with the -O3 option

Georgia Tech

CHAMPS

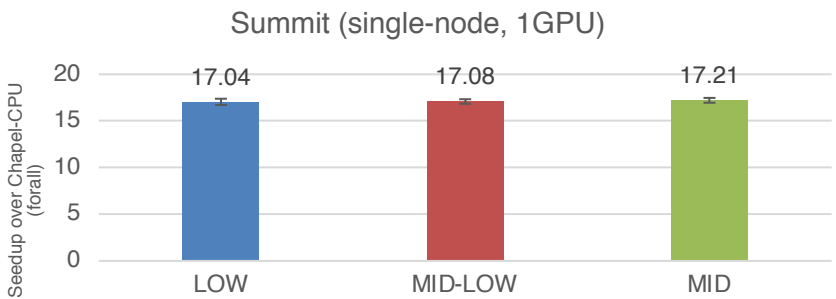


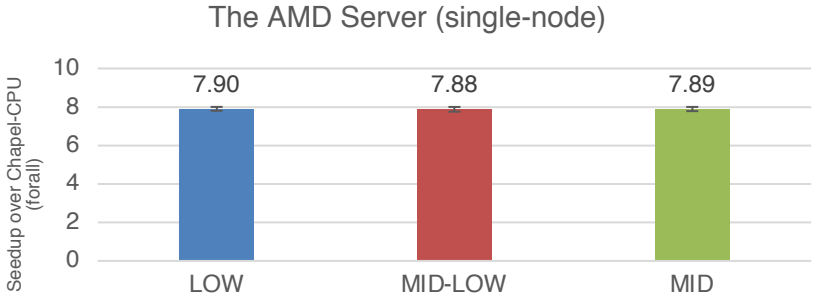
- Summary
  - 3D unstructured finite-volume Reynolds Average Navier-Stokes (RANS) and Potential flow solver using a cell-centered discretization
    - ✓ The potential solver is used
  - Developed at Polytechnique Montreal
  - Originally written entirely in Chapel
    - ✓ We prepared a GPU version of two most time-consuming forall loops
    - ✓ We utilized cuSolver for the solver part

Matthieu Parenteau, Simon Bourgault-Cote, Frédéric Plante, Engin Kayraklioglu, and Eric Laurendeau. "Development of Parallel CFD Applications with the Chapel Programming Language" (AIAA-Scitech 2021 Forum)

Champs (potential solver): Productivity & Performance

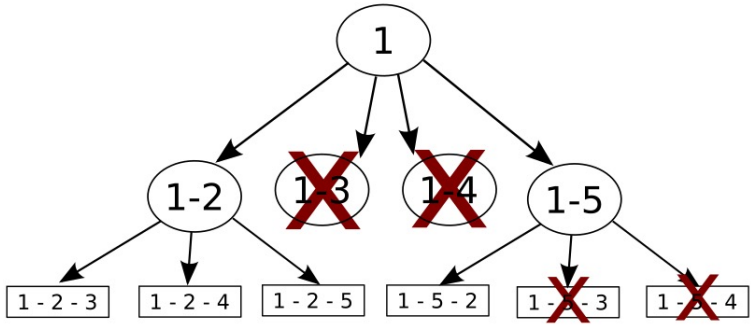
SLOC Added	Chapel	Host (CUDA)	Kernel (CUDA)
LOW	**172**	**117**	361
MID-LOW	**265**	**5**	361
MID	**161**	**5**	361





- Points
  - The use of GPUAPI significantly reduces SLOC for the host part
  - The MID-level API further reduces SLOC for the Chapel part
  - There is no statistically significant performance degradation when GPUAPI is used

Distributed Exact Optimization



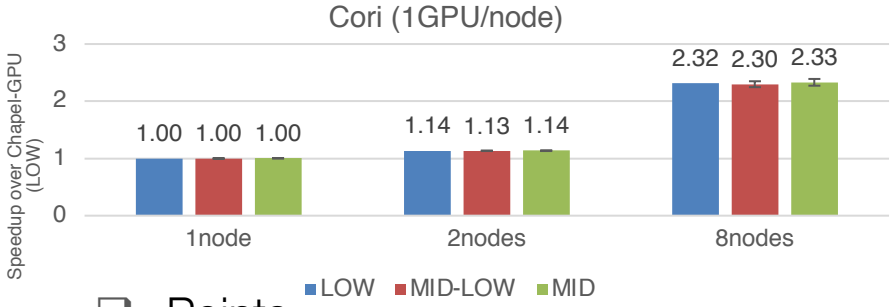
- Summary
  - Tree-based exact algorithms: Backtracking, Branch-and-bound
  - Developed at Inria Lille and University of Luxembourg
    - ✓ https://github.com/tcarneirop/ChOp
  - Originally written in Chapel+CUDA (what we call “LOW-level”)
    - ✓ Highly compute-intensive and irregular
    - ✓ We prepared MID and MID-LOW versions

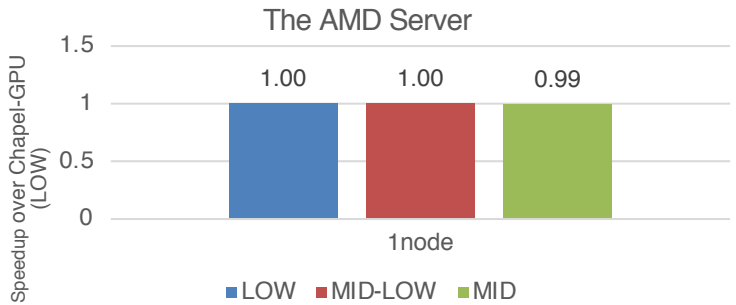
Georgia Tech

Tiago Carneiro, Nouredine Melab, Akihiro Hayashi, Vivek Sarkar,
“Towards Chapel-based Exascale TreeSearch Algorithms: dealing with multiple GPU accelerators” (HPCS2020)

Distributed Exact Optimization: Productivity & Performance

SLOC Added	Chapel	Host (CUDA)	Kernel (CUDA)
LOW	**2**	**16**	71
MID-LOW	**13**	**4**	71
MID	**9**	**4**	71





- Points
  - The use of GPUAPI significantly reduces SLOC for the host part
  - The MID-level API further reduces SLOC for the Chapel part
  - There is no statistically significant performance degradation when GPUAPI is used

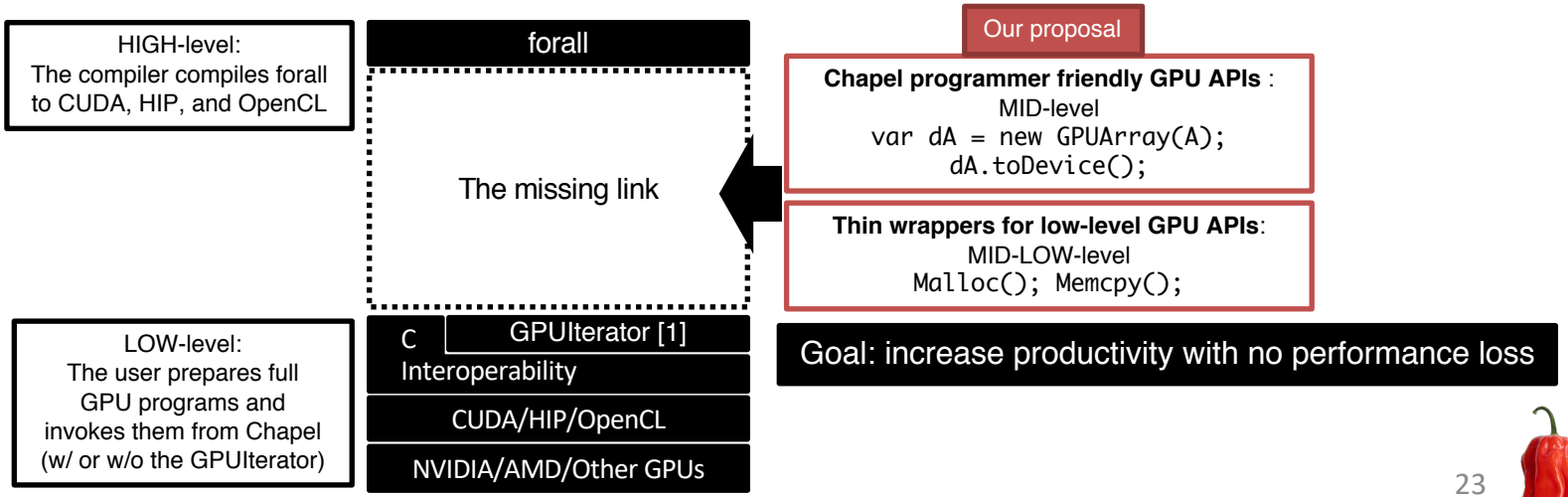
GPUAPI: Multi-level Chapel Runtime API for GPUs

CONCLUSIONS

Georgia Tech

Conclusions

- Introduced multi-level GPUAPI for Chapel
  - Improves both productivity and portability
  - Verified with microbenchmark and real-world applications
  - Verified on Summit@ORNL, Cori-GPU@NERSC, and an AMD server



Acknowledgements

- The Champs team
  - Eric Laurendeau
  - Matthieu Parenteau
  - Anthony Bouchard
- The BONUS team
  - Tiago Carneiro
  - Nouredine Melab
- The Chapel team
- The Habanero Research Group @ Georgia Tech

This research used resources of the National Energy Research Scientific Computing Center,
which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231.

Also, this research used resources of the Oak Ridge Leadership Computing Facility at the Oak Ridge National Laboratory,
which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC05-00OR22725.

Georgia Tech

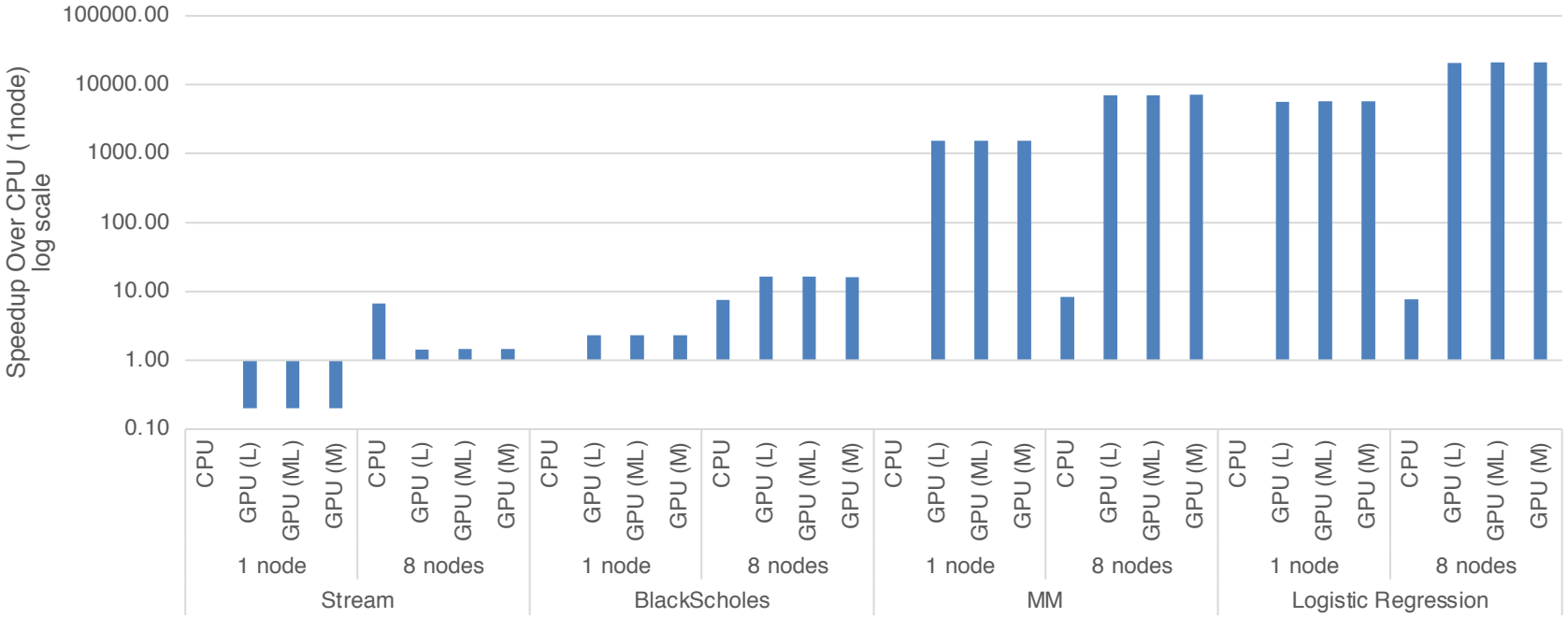
Join our community

- GPUAPI+GPUIterator 0.3 is released!
  - The repository
    - ✓ https://github.com/ahayashi/chapel-gpu
  - Detailed Documents
    - ✓ https://ahayashi.github.io/chapel-gpu/index.html
- Our community is growing!

GPUAPI: Multi-level Chapel Runtime API for GPUs

BACKUP SLIDES

Micro-benchmark Performance (Cori)



Micro-benchmark Performance (Summit)