Georgia Tech

Accelerating CHAMPS on GPUs

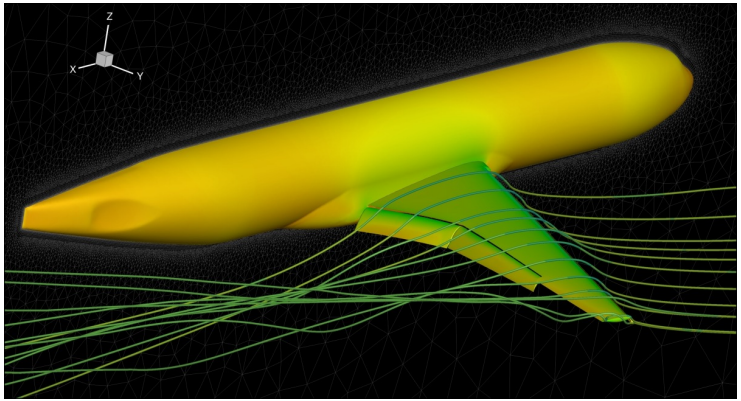
CHIUW2022, June 10, 2022

Akihiro Hayashi (Georgia Tech)

Sri Raj Paul (Intel)

Vivek Sarkar (Georgia Tech)

CHAMPS

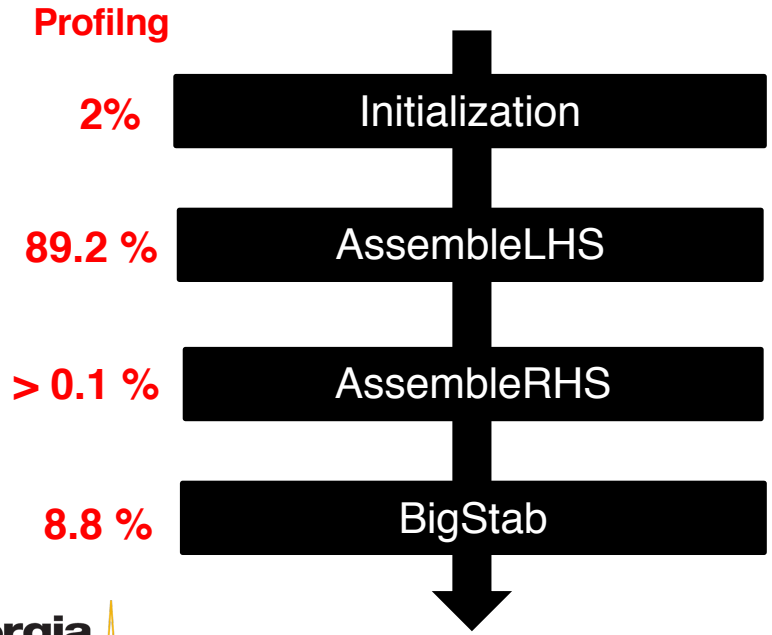


- Summary
  - A CFD application that is originally written entirely in Chapel
    - ✓ 3D unstructured finite-volume Reynolds Average Navier-Stokes (RANS) and Potential flow solver using a cell-centered discretization
  - Developed at Polytechnique Montreal
  - Our current focus is the potential solver

1. In this talk, we discuss our experience in accelerating CHAMPS on GPUs using our GPUAPI and GPUIterator
2. While our CHIUW'21 talk was more about proposing the GPUAPI module and used CHAMPS as an example, this year's talk 1) dives into more details of our GPU implementation, 2) proposes an additional optimization, and 3) presents performance numbers on different platforms including a new platform

Program Structure

- End-to-end flow



Prepares data (Read a model file)

Constructs the LHS of the equation

Constructs the RHS of the equation

Solve the equation

Note: the profiling result is obtained on a single node of Cori (Xeon Gold 6148)

High-level summary of our GPU implementations

- assembleLHS: 89.2%
  - Contains two forall loops

  → For each parallel loop, we implemented a CUDA version and call it from Chapel using GPUIterator and GPU API modules
- BigStab: 8.8%
  - LAPACK's dgesv

  → We utilized a highly-optimized GPU solver (cuSOLVER and cuBLAS)

Georgia Tech

An overview of Chapel GPU Implementation (assembleLHS)

- Used the GPUIterator module to invoke the GPU portion

```
1  proc assembleLHS(…) {
2    for zone in zones {
3      computeGeormetryInfluence(zone);
4      computeWakeInfluence(zone);
5    }
6  }
```

```
1  proc computeGeometryInfluence(zone) {
2    // original
3    forall i in zone.elements { … }
4
5    // w/ GPUIterator
6    forall i in GPU(zone.elements,
7                    callback, CPUPercent) {
8       // CPU version
9    }
10 }
```

Note: For presentation purposes, we omit the computeWakeFluence part because it is almost equivalent to the computeGeormetryInfluence

assembleLHS on GPUs: Interfacing Chapel objects to the CUDA part

```
1  class zone {
2    var scalar: int;
3    var n: int;
4    var array: [0..#n] int;
5  }
6  // Array of Structure
7  var zones = [zone0, zone1, …, zone7];
```

- CHAMPS uses an array of structure to maintain mesh information
- The forall loops typically access the array of structure in the following ways:
  - Access the “scalar” of all the zones (Array of Scalar pattern)
    - ✓ zones[0..#8].scalar
  - Access the “array” of all the zones (Array of Array pattern)
    - ✓ zones[0..#8].array

Our GPUAPI module facilitates the device memory allocation for such objects

assembleLHS on GPUs: The Array of Scalar pattern

```
// The array of scalar pattern
zones[0..#8].scalar

// preprocess
ref scalars = [zones[0..#8].scalar];
```

MID-level API

```
1  var dA = new GPUArray(scalars);
2  dA.toDevice();
```

MID-LOW level API

```
1  var dA: c_void_ptr;
2  const size = scalars.size:size_t*c_sizeof(int);
3  Malloc(dA, size);
4  Memcpy(dA, c_ptrTo(scalars), size, 0);
```

- MID-level version significantly simplifies Array of Scalar Allocation on GPUs

assembleLHS on GPUs: The Array of Array Pattern

```
// The array of array pattern
zones[0..#8].array
```

MID-level API

```
1  var dAs = new GPUJaggedArray(zones[0..#8].array);
2  dAs.toDevice();
```

MID-LOW level API

```
1   var dA: [0..#8] c_void_ptr;
2   var dAs: c_ptr(c_void_ptr);
3   for i in 0..#8 {
4     const size = zones[i].array.size:size_t*c_sizeof(int);
5     Malloc(dA[i], size);
6     Memcpy(dA[i], c_ptrTo(zones[i].array), size, 0);
7   }
8   const size = 8 * c_sizeof(c_ptr(c_void_ptr));
9   Malloc(dAs, size);
10  Memcpy(dAs, c_ptrTo(dA), size, 0);
```

- MID-level version significantly simplifies Array of Array Allocation on GPUs

assembleLHS on GPUs: Kernel

- Each forall-loop is technically a triply-nested forall loop
- In the current GPU implementation, we only parallelized the outermost loop, where each CUDA thread sequentially executes the inner loops
  - We have not aggressively applied any optimizations to the kernel (e.g., memory coalescing, shared memory utilization)
  - We will discuss the performance of the kernel in the performance evaluation part

```
1  proc computeGeometryInfluence(zone) {
2    // w/ GPUIterator
3    forall i in GPU(zone.elements,
4                    callback, CPUPercent) {
5       for …
6         for …
7    }
8  }
```

Georgia Tech

An overview of Chapel GPU Implementation (BigStab)

- The original CPU version uses LAPACK's `dgesv`
  - AX = B
  - The LU decomposition is used to factor A
- We used a combination of cuBLAS and cuSOLVER to perform a matrix transpose and an LU decomposition
  - `cublasDgeam`
  - `cusolverDnDgetrs`
- In the current implementation, we only prepared a full CUDA version (LOW-level) and call it from Chapel

Georgia Tech

Additional optimization opportunity: Data transfer optimization

- We have completely 1) hoisted CPU to GPU transfers and 2) deferred GPU to CPU transfers

```
1  // CHIUW'21 version
2  proc assembleLHS(…) {
3    CPUtoGPU();
4    for zone in zones {
5      CPUtoGPU();
6      computeGeormetryInfluence();
7      CPUtoGPU();
8      computeWakeInfluence();
9      GPUtoCPU();
10   }
11 }
```

```
1  // CHIUW'22 version
2  proc assembleLHS(…) {
3    CPUtoGPU();
4    for zone in zones {
5      computeGeormetryInfluence();
6      computeWakeInfluence();
7    }
8    GPUtoCPU();
9  }
```

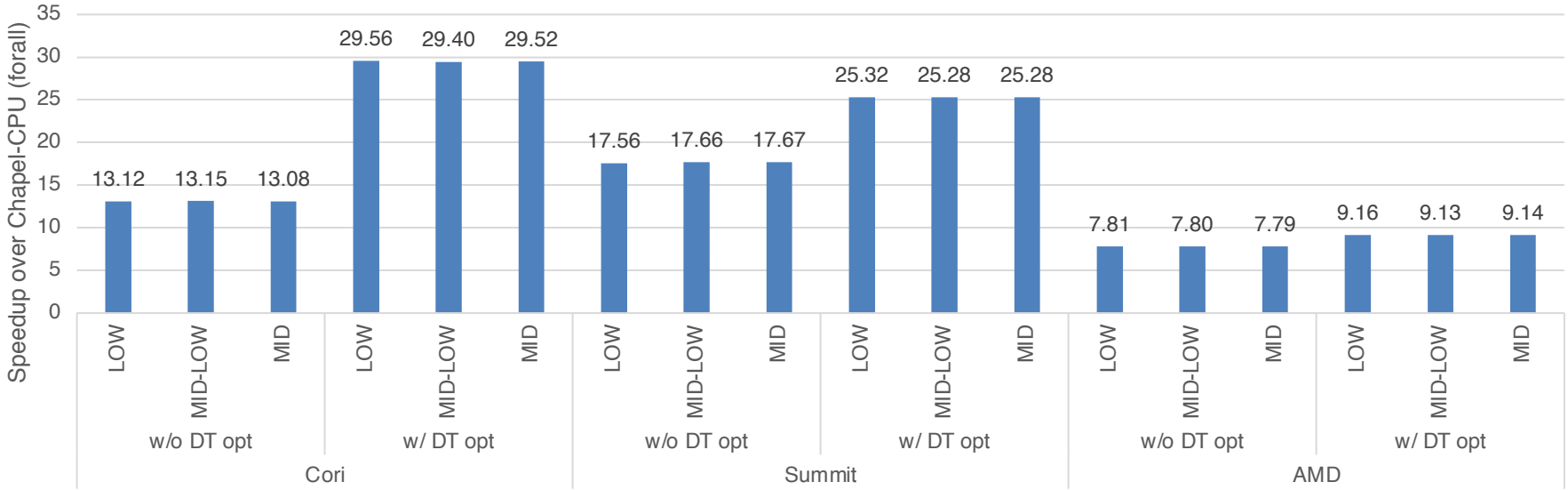
Georgia Tech

Performance & Productivity Evaluations

- Platforms (Single-node)
  - 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
- Chapel Compilers & Options
  - Champs: Chapel Compiler 1.22 with the --fast option
- GPU Compilers
  - CUDA: NVCC 11.1 (Cori), 11.1 (Summit) with the -O3 option
  - AMD: ROCM 2.9.6, HIPCC 2.8 with the -O3 option
- BigStab for CPU
  - IBM ESSL 6.3 (Summit)
  - Intel MKL 2020.2.254 (Cori), 2020.1.217 (AMD)

Georgia Tech

End-to-end Performance Improvements (1node, 1GPU/node)



- There is no significant performance degradation when GPUAPI is used
- The data transfer optimizations give 2.25x improvement on Cori, 1.39x improvement on Summit, 1.17x improvement on AMD (end-to-end performance)

Georgia Tech

Can we get further performance improvements on the GPU?

	Entire assembleLHS	Data Transfer Time	Kernel Time
w/o DT Opt	**1 (normalized)**	0.78	0.22
w/ DT Opt	0.28	0.06	0.22

HW counter numbers obtained with NVPROF (on Cori-GPU):
Achieved Occupancy: 12% (100% is the best)
Global Load Transactions per Request : 1.4 (1 is the best, 32 is the worst)
Global Store Transactions per Request: 31.5 (1 is the best, 32 is the worst)

- Further optimization opportunities
  - Hierarchical parallelization/Loop collapse
  - Global memory access optimization (particularly for “store”)

Georgia Tech

Productivity Evaluation

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

Georgia Tech

Conclusions & Future work

- We discussed our experience in accelerating CHAMPS on GPUs using our GPUAPI and GPUIterator
  - dived into details of our GPU implementation
  - proposed a data transfer optimization
  - presented performance numbers on different platforms (Cori, Summit, an AMD machine)
    - ✓ 29.5x on Cori
    - ✓ 25.3x on Summit
    - ✓ 9.1x on the AMD machine
- Future work:
  - Optimize the GPU kernel
  - Accelerate the RANS Solver

Georgia Tech

Acknowledgements

- The CHAMPS team
  - Eric Laurendeau
  - Matthieu Parenteau
  - Anthony Bouchard
- 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 0.4 is available!
  - Now supports SYCL (DPC++)
  - The repository
    - ✓ https://github.com/ahayashi/chapel-gpu
  - Detailed Documents
    - ✓ https://ahayashi.github.io/chapel-gpu/index.html
- Our community is growing!

POLYTECHNIQUE MONTRÉAL
TECHNOLOGICAL UNIVERSITY

UNIVERSITÉ DU LUXEMBOURG

Inria

Georgia Tech

Contact: ahayashi “at” gatech.edu