THE UNIVERSITY of TENNESSEE UT CHATTANOOGA

CPU-GPU Programming: Performance Portability on Modern HPC Platforms

Thomas Gorham
Computer Science & Engineering Department
University of Tennessee at Chattanooga
Spring 2022

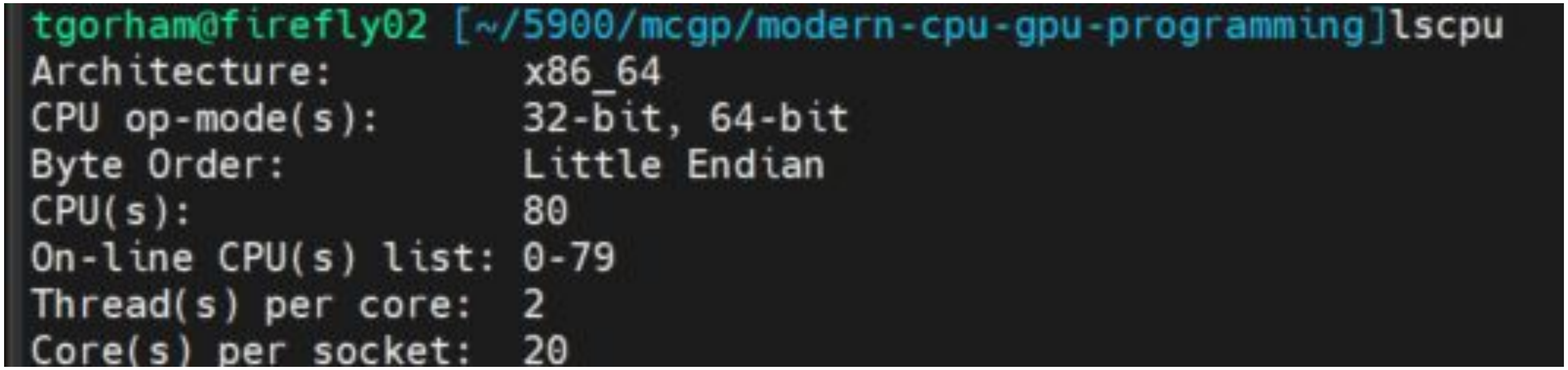
We Shall Achieve

THE UNIVERSITY of TENNESSEE UT
CHATTANOOGA

Why we are here today

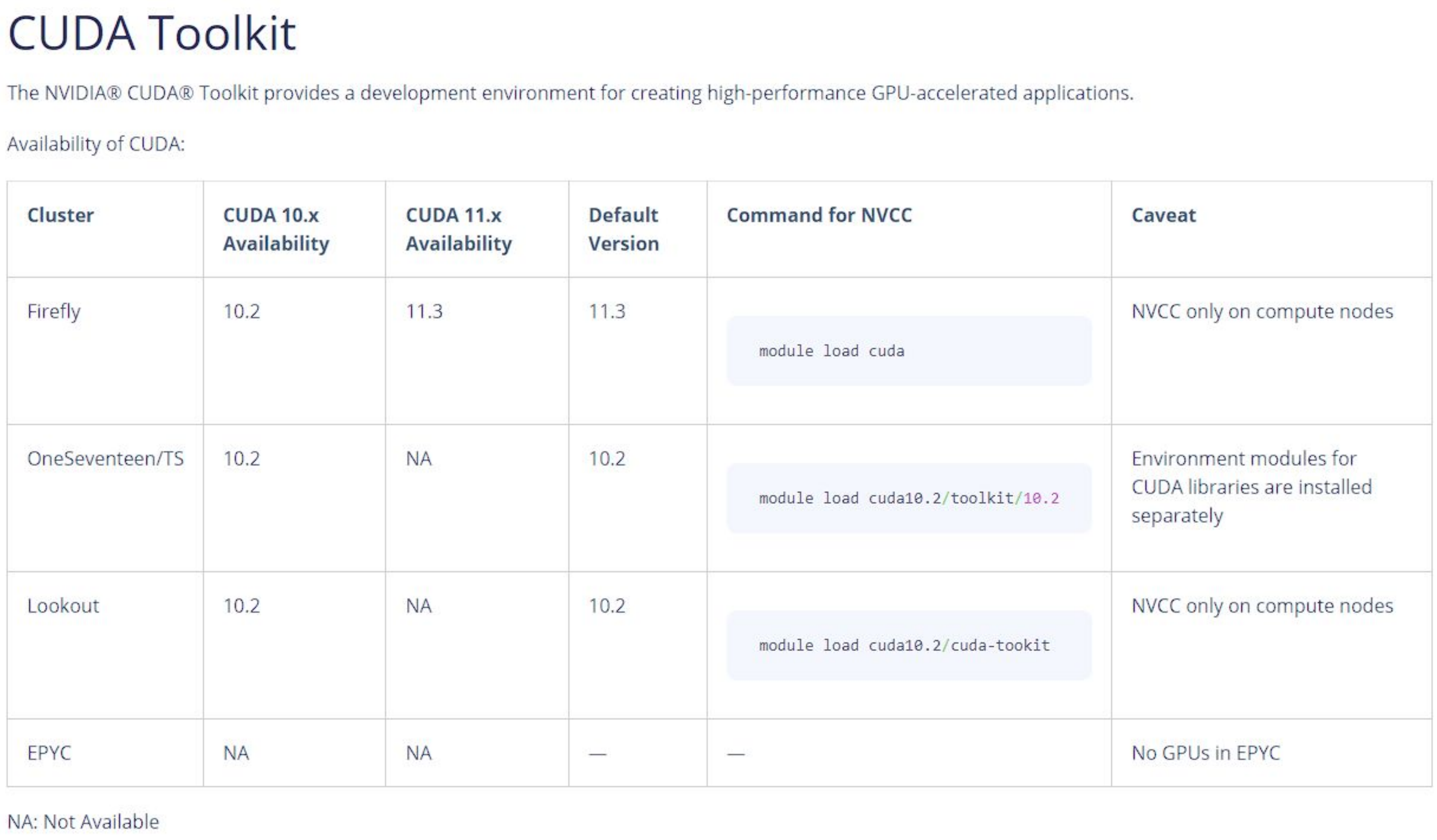
- The **Purpose** of this work was to create a useful code base and wiki page of programming expertise on heterogeneous (CPU+GPU) architectures by utilizing Kokkos on a manycore node with attached GPU accelerators.
  - Built on Firefly 02 with Cuda 11.3, GCC 10.2, and CMake 3.19.4
  - Have also built on TS and locally

Firefly02 Specs



```
tgorham@firefly02 [~/5900/mcgp/modern-cpu-gpu-programming]lscpu
Architecture:          x86_64
CPU op-mode(s):        32-bit, 64-bit
Byte Order:            Little Endian
CPU(s):                80
On-line CPU(s) list:   0-79
Thread(s) per core:    2
Core(s) per socket:    20
```

We Shall Achieve



CUDA Toolkit

The NVIDIA® CUDA® Toolkit provides a development environment for creating high-performance GPU-accelerated applications.

Availability of CUDA:

Cluster	CUDA 10.x Availability	CUDA 11.x Availability	Default Version	Command for NVCC	Caveat
Firefly	10.2	11.3	11.3	`module load cuda`	NVCC only on compute nodes
OneSeventeen/TS	10.2	NA	10.2	`module load cuda10.2/toolkit/10.2`	Environment modules for CUDA libraries are installed separately
Lookout	10.2	NA	10.2	`module load cuda10.2/cuda-tookit`	NVCC only on compute nodes
EPYC	NA	NA	—	—	No GPUs in EPYC

NA: Not Available

*Quick Aside: No GPUs on Epyc

Why we are here today

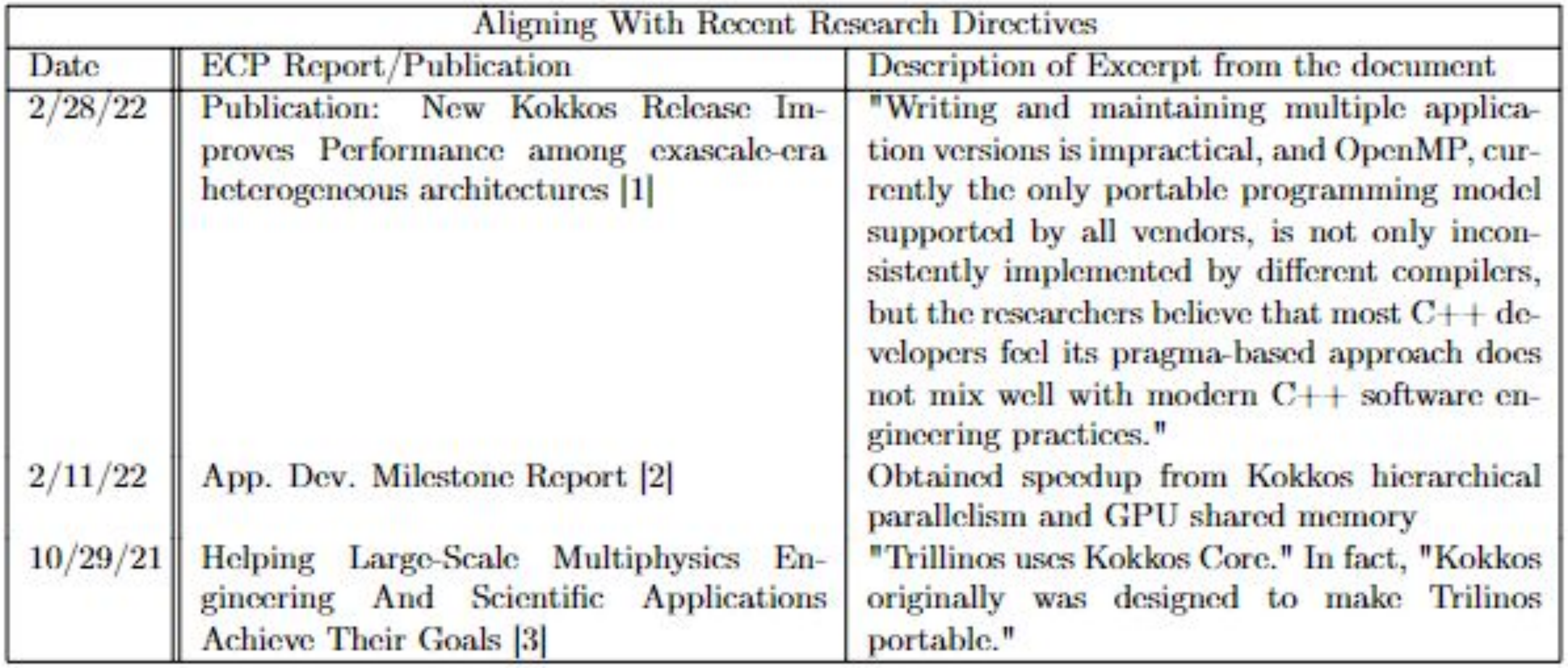
- The **Goal** is to show you how Kokkos simplifies the challenges of programming heterogeneous architectures with abstractions that optimize data storage and parallel execution, and how different programming model backends can be set at compile time for maximum portability
  - Controlling Kokkos Execution Spaces and Memory Spaces at Compile Time vs in the Code

- An **execution space** is the place and way to run the code
  - Kokkos::Serial, Kokkos::OpenMP, Kokkos::Threads, Kokkos::Cuda, Kokkos::Hip, etc...

- A **memory space** is where the data is stored
  - Kokkos::HostSpace, Kokkos::CudaSpace, Kokkos::CudaUVMSpace, Kokkos::HIPSpace, etc...

Why we are here today

- The **Significance** of this work is rooted in its alignment with current research regarding heterogeneous computing for performance portability.



Aligning With Recent Research Directives

Date	ECP Report/Publication	Description of Excerpt from the document
2/28/22	Publication: New Kokkos Release Improves Performance among exascale-era heterogeneous architectures [1]	"Writing and maintaining multiple application versions is impractical, and OpenMP, currently the only portable programming model supported by all vendors, is not only inconsistently implemented by different compilers, but the researchers believe that most C++ developers feel its pragma-based approach does not mix well with modern C++ software engineering practices."
2/11/22	App. Dev. Milestone Report [2]	Obtained speedup from Kokkos hierarchical parallelism and GPU shared memory
10/29/21	Helping Large-Scale Multiphysics Engineering And Scientific Applications Achieve Their Goals [3]	"Trillinos uses Kokkos Core." In fact, "Kokkos originally was designed to make Trilinos portable."

HPC Background

"**High Performance Computing" (HPC)** most generally refers to the practice of aggregating computing power in a way that delivers much higher performance in order to efficiently solve massive computational problems

- "if one computer can solve a problem, 5000 computers can solve it even faster"

  -Tim Carrol Director, HPC & AI for Research at Microsoft

more compute resources = more computational power = the ability to solve problems faster

HPC enables us to solve difficult problems across many scientific fields that were once considered "out of reach".

One example of a problem that we have solved thanks to developments in HPC was decoding the Human Genome [5]

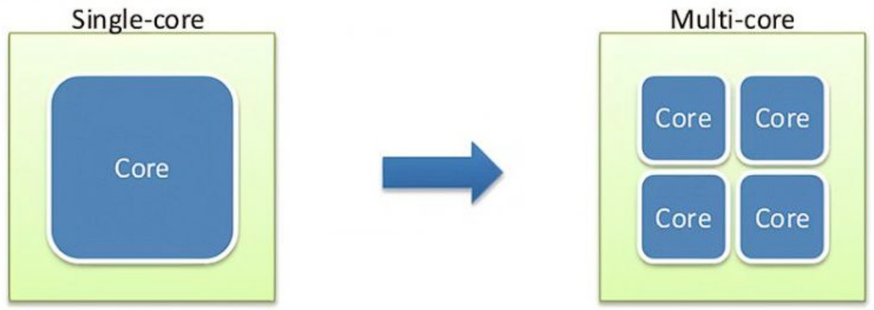
How are these Problems solved?

Up until the early 2000s:

smaller transistors = faster processors = increased power consumption = solve problems faster

Around 2005:

MultiCore Processors



core = a distinct execution unit of a CPU with its own instruction stream

How are these Problems solved?

Caveat: adding more CPU processors did not magically improve the performance of the vast majority of serial programs. For instance,

Instead, exploiting the increased power of multiple processors on a single chip for increased performance required *parallelism*.

How are these Problems solved?

parallel computing is the simultaneous use of multiple compute resources to solve a computational problem.

1) Break the computational problem into discrete parts that can be solved concurrently

2) Reduce each part of the problem further into a series of instructions

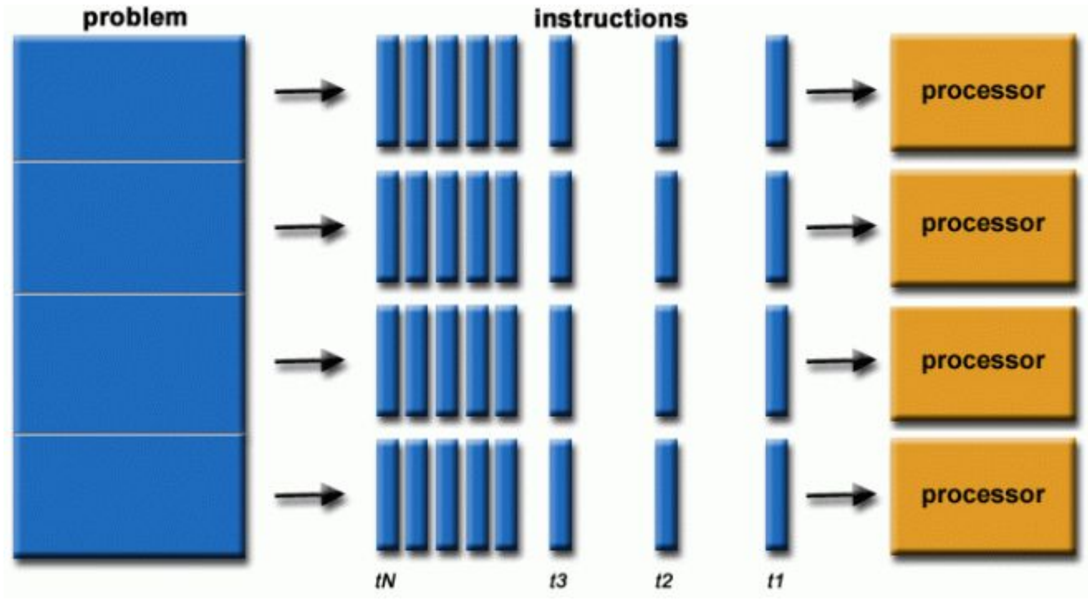
3) Execute the instructions from each part simultaneously on multiple cores/CPUs

THE UNIVERSITY of TENNESSEE CHATTANOOGA

How are these Problems solved?

Example: C:\Users\TM\pi.c
Example2: C:\Users\TM\pipar.c

Wiki link



How are these Problems solved?

In previous example, we used OpenMP, an API for writing multithreaded applications.

1 -> 2 threads

Kokkos is a shared memory programming model

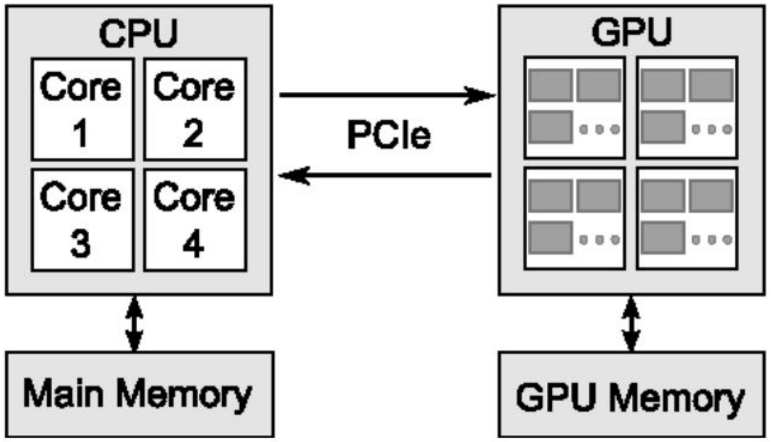
Shared memory machines are computers that are composed of multiple processing elements that ***share*** an address space. These are further divided into two classes

The class that is relevant today is:

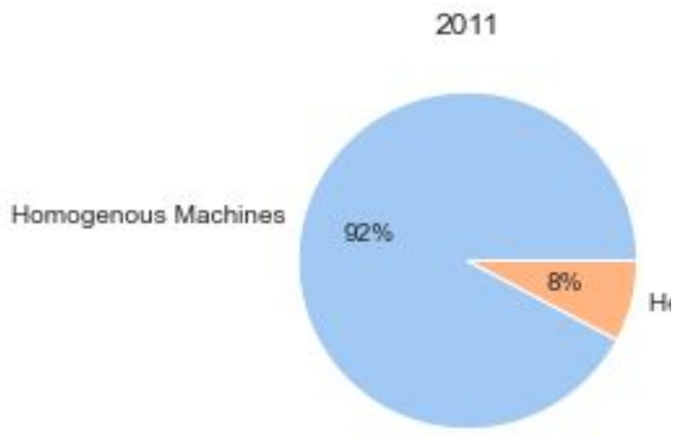
- **Non Uniform address space multiprocessor (NUMA):** different memory regions have different access costs … think of memory segmented into “Near” and “Far” memory

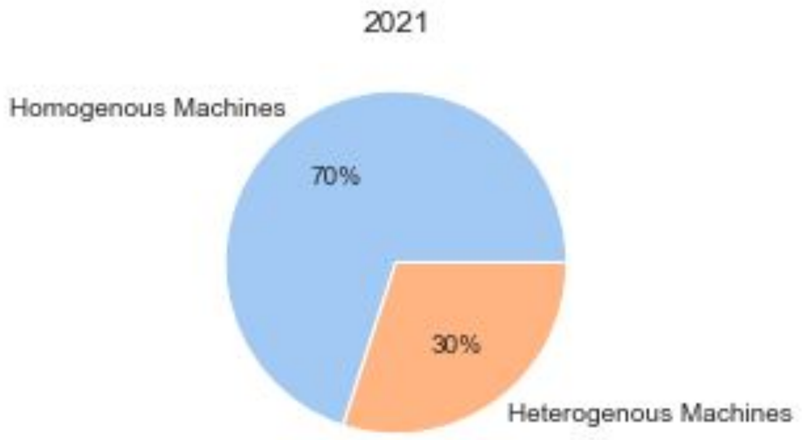
Heterogeneous Architectures

Machines that contain at least one accelerator, such as Sierra or Summit are considered to have **heterogenous (CPU-GPU) architectures,** in that they contain different types of computational units. Heterogeneous memory systems provide the potential benefit of significantly increasing system performance and reducing power consumption at the cost of complexity in their architectures.



The number of these machines is increasing rapidly





Heterogeneity

Hardware: CPUs, GPUs, FPGA

- Increasing complexity of modern machines such as multiple GPU accelerators

Software: Compilers, Operating Systems, Libraries

How can we get applications to target underlying hardware given the reality of diverse complexity in modern systems?

Heterogeneity Challenges

Challenge 1: optimizing data access patterns in CPU-GPU environments.

This is because these devices store data quite differently, and the data structure layouts in GPU memory often lead to sub-optimal performance for programs designed with a CPU memory interface. In other words, **application performance is highly sensitive to irregularity in memory access pattern**

Heterogeneity Challenges

Challenge 2: Portability

This derives from lack of uniformity

its not quite clear how performance portable a program written for an AMD-based machine will be to another (e.g., an Intel-based machine).

What is clear is that if you are use the standard programming model for a heterogenous NVIDIA-based machine and you want to run on an AMD-based machine, you will need to change to AMD's supported programming model

Challenge 2 cont

Changing programming models to run on different machines requires code to be rewritten per the specifications of the machine's respective programming model. Ultimately, this becomes a particular problem for large-scale HPC applications that on average consist of 300,000 - 600,000 lines of code. In other words, rewriting code is expensive.

Existing Solutions

- Programming models for shared/distributed memory
  - Make programmers learn openmp, OpenACC, OpenCV, pthreads, TBB, Cilk++, MPI, UPC, Fortran,

What if instead we could just write one solution in C++ and change our target at compile time?

Solving These Challenges With Kokkos

Programming two execution targets (the CPU and the GPU) require a set of mechanisms for performance portability across the range of potential and final system designs. Kokkos is a shared memory programming model that emphasizes performance portability accross all modern HPC architectures. The Kokkos API allows developers to *express* parallelism that is able to compile and execute

- In Serial on the CPU
- In Parallel on the CPU
- In Parallel on the CPU and GPU

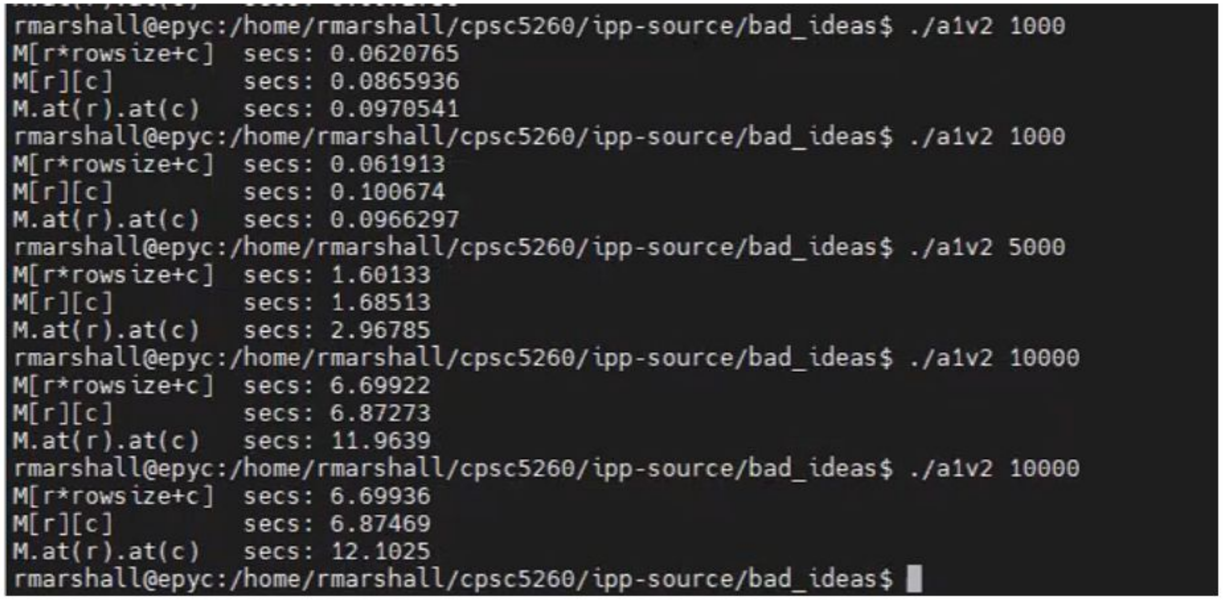
regardless of the manufacturer of either chip. Thus, this removes the need to refactor the source code itself, and enables HPC applications to run on theoretically any HPC machine.

tommygorham/modern-cpu-gpu-programming: Wiki page and code base to support my final project and paper with regards to a MS in Computer Science at the University of Tennessee at Chattanooga. (github.com)

Architecture Specific Optimizaitons · tommygorham/modern-cpu-gpu-programming Wiki (github.com)

Optimize memory Access in C

From a parallel computing class I took under Mr. Ryan Marshall, we had experimented with matrices access time in C, and we found that one-dimensional row-major access was the fastest. Below is an image from his in-class experimentation.

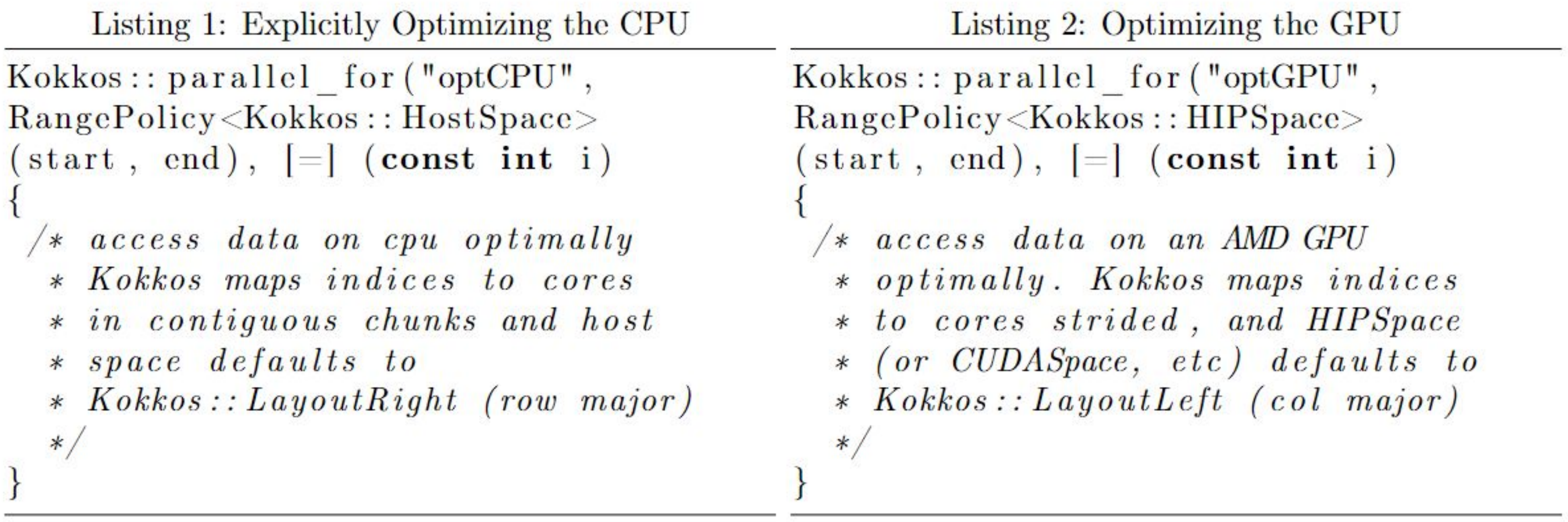


```
rmarshall@epyc:/home/rmarshall/cpsc5260/ipp-source/bad_ideas$ ./a1v2 1000
M[r*rowsize+c]  secs: 0.0620765
M[r][c]         secs: 0.0865936
M.at(r).at(c)   secs: 0.0970541
rmarshall@epyc:/home/rmarshall/cpsc5260/ipp-source/bad_ideas$ ./a1v2 1000
M[r*rowsize+c]  secs: 0.061913
M[r][c]         secs: 0.100674
M.at(r).at(c)   secs: 0.0966297
rmarshall@epyc:/home/rmarshall/cpsc5260/ipp-source/bad_ideas$ ./a1v2 5000
M[r*rowsize+c]  secs: 1.60133
M[r][c]         secs: 1.68513
M.at(r).at(c)   secs: 2.96785
rmarshall@epyc:/home/rmarshall/cpsc5260/ipp-source/bad_ideas$ ./a1v2 10000
M[r*rowsize+c]  secs: 6.69922
M[r][c]         secs: 6.87273
M.at(r).at(c)   secs: 11.9639
rmarshall@epyc:/home/rmarshall/cpsc5260/ipp-source/bad_ideas$ ./a1v2 10000
M[r*rowsize+c]  secs: 6.69936
M[r][c]         secs: 6.87469
M.at(r).at(c)   secs: 12.1025
rmarshall@epyc:/home/rmarshall/cpsc5260/ipp-source/bad_ideas$
```

Optimize memory access in Kokkos

Can control where memory lives and executes 1 of 2 ways
1) Compile time parameters and which set the default execution/memory spaces

2) Hard set in the Code as shown below in the code, beginning with Program 1.
The code snippet below shows how these spaces can be explicitly controlled to set optimal access for the CPU or the GPU from within the source code



Listing 1: Explicitly Optimizing the CPU

```
Kokkos::parallel_for("optCPU",
RangePolicy<Kokkos::HostSpace>
(start, end), [=] (const int i)
{
  /* access data on cpu optimally
   * Kokkos maps indices to cores
   * in contiguous chunks and host
   * space defaults to
   * Kokkos::LayoutRight (row major)
   */
}
```

Listing 2: Optimizing the GPU

```
Kokkos::parallel_for("optGPU",
RangePolicy<Kokkos::HIPSpace>
(start, end), [=] (const int i)
{
  /* access data on an AMD GPU
   * optimally. Kokkos maps indices
   * to cores strided, and HIPSpace
   * (or CUDASpace, etc) defaults to
   * Kokkos::LayoutLeft (col major)
   */
}
```

HPC Program One: Matrix-Vector Multiplication

Algorithm: Matrix-Vector Multiplication

Formal Equation: $y_m = A_{mn} x_n$

***Initial m,n*:** 10,000

Solution Size of *y*: 100,000

Link to results Program 1,2,4 here

Link to results for Program 3 here

Architecture Problem(s) Solved	MACHINE	CPU ARCHITECTURE	# CPUs	GPU ARCHITECTURE	# GPUs
Portability: -I need to compile for CPU/GPU and I don't have time to change my code Performance: -I want to make use of multiple compute resources for faster execution	Firefly (Node2)	Intel® Xeon® Gold 6148	80	NVIDIA Tesla V100 SXM2	4

Question: How can I reproduce this on my cluster?

Write <u>one</u> solution and make compile time changes for the compute resource you want to leverage

Compile and run for CPU

Cmake Build Commands

-DCMAKE_INSTALL_PREFIX = <path-to-kokkos-install>
-DCMAKE_CXX_COMPILER = <path-to-c++_compiler>

Compile and run for GPU

Cmake Build Commands

-DCMAKE_INSTALL_PREFIX = <path-to-kokkos-install>
-DCMAKE_CXX_COMPILER
= <path-to-nvcc_compiler>
-DKokkos_ENABLE_HIP=ON

Most Notable Takeaways

- Link to page 6

References

[1] ECP "New Kokkos Release Improves Performance Portability Among Exascale-era Heterogeneous Architectures

[2] ECP "2021 Application Development Milestone Report"

[3] ECP "Helping Large-scale Multiphysics Engineering and Scientific Applications Achieve Their Goals

[4] ECP "ECP Software Technology Capability Assessment Report Public"

[5] National Library of Medicine "Deconding the Human Genome"

[6] BIOWULF High Performance Computing At The NIH "Scientific Applications on NIH HPC Systems"

[7] Science Direct "Programming languages for data-Intensive HPC applications: A systematic mapping study"

[8] International Journal of Innovative Science and Research Technology "Programming Language for Data Intensive HPC Applications: Applications and its Relevance"