

MANY WAYS TO GPUS GPU INTRODUCTION

13 October 2022 | Andreas Herten, Kaveh Haghighi-Mood | Forschungszentrum Jülich



Member of the Helmholtz Association

Outline

GPU Architecture **Empirical Motivation** Comparisons **GPU** Architecture Summary **Programming GPUs** Libraries Directives CUDA C/C++ **Performance Analysis** Conclusion References

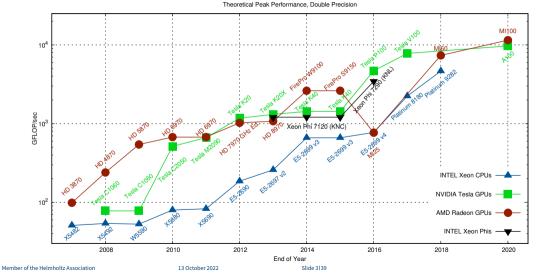
Image references are collected in **References** section at end of slides Title image: Debiève [**1**]



GPU Architecture

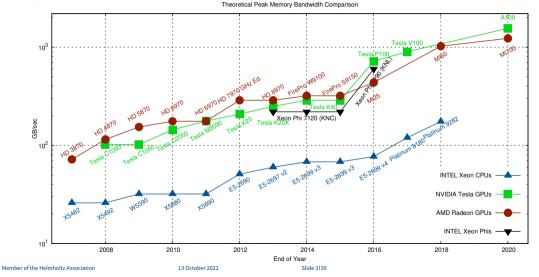
Status Quo Across Architectures

Performance



Status Quo Across Architectures

Memory Bandwidth



CPU vs. GPU

A matter of specialties







CPU vs. GPU

A matter of specialties



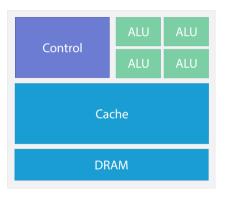
Transporting one

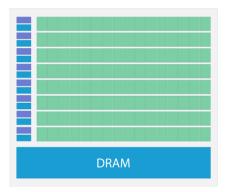


Transporting many



CPU vs. GPU _{Chip}

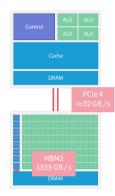






GPU optimized to hide latency

- Memory
 - GPU has small (40 GB), but high-speed memory 1555 GB/s
 - Stage data to GPU memory: via PCIe 4 bus (32 GB/s)

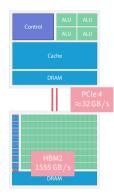


Host



GPU optimized to hide latency

- Memory
 - GPU has small (40 GB), but high-speed memory 1555 GB/s
 - Stage data to GPU memory: via PCIe 4 bus (32 GB/s)



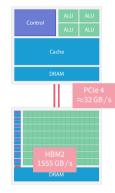
Host



GPU optimized to hide latency

- Memory
 - GPU has small (40 GB), but high-speed memory 1555 GB/s
 - Stage data to GPU memory: via PCIe 4 bus (32 GB/s)
 - Stage automatically (Unified Memory), or manually

Host

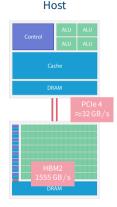




GPU optimized to hide latency

- Memory
 - GPU has small (40 GB), but high-speed memory 1555 GB/s
 - Stage data to GPU memory: via PCIe 4 bus (32 GB/s)
 - Stage automatically (Unified Memory), or manually
- Two engines: Overlap compute and copy







GPU optimized to hide latency

- Memory
 - GPU has small (40 GB), but high-speed memory 1555 GB/s
 - Stage data to GPU memory: via PCIe 4 bus (32 GB/s)
 - Stage automatically (Unified Memory), or manually
- Two engines: Overlap compute and copy



HBM2 1555 GB/s DRAM

Device

Host



V100 32 GB RAM, 900 GB/s

0



40 GB RAM, 1555 GB/s



13 October 2022

Slide 6139

GPU optimized to hide latency

- Memory
 - GPU has small (40 GB), but high-speed memory 1555 GB/s
 - Stage data to GPU memory: via PCIe 4 bus (32 GB/s)
 - Stage automatically (Unified Memory), or manually
- Two engines: Overlap compute and copy



V100

32 GB RAM, 900 GB/s

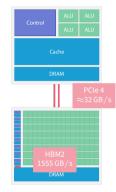


A100

40 GB RAM, 1555 GB/s



Host



Device



Slide 6139



Scalar

A ₀	+	B_0	=	<i>C</i> ₀
A_1	+	B_1	=	C_1
A_2	$^+$	<i>B</i> ₂	=	<i>C</i> ₂
A ₃	$^+$	B_3	=	<i>C</i> ₃

CPU:

Single Instruction, Multiple Data (SIMD)





Vector



CPU:

• Single Instruction, Multiple Data (SIMD)





 $SIMT = SIMD \oplus SMT$

SIMT

CPU:

Single Instruction, Multiple Data (SIMD)

13 October 2022

Simultaneous Multithreading (SMT)



B₂

B₃

+

A₂

 A_3

 C_1

 C_2

 C_3

_







CPU:

- Single Instruction, Multiple Data (SIMD)
- Simultaneous Multithreading (SMT)



SMT







CPU:

- Single Instruction, Multiple Data (SIMD)
- Simultaneous Multithreading (SMT)
- GPU: Single Instruction, Multiple Threads (SIMT)



SMT







CPU:

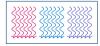
- Single Instruction, Multiple Data (SIMD)
- Simultaneous Multithreading (SMT)
- GPU: Single Instruction, Multiple Threads (SIMT)



SMT



SIMT





$\begin{array}{l} \text{SIMT} \\ \text{simt} = \text{simd} \oplus \text{smt} \end{array}$

CPU:

- Single Instruction, Multiple Data (SIMD)
- Simultaneous Multithreading (SMT)
- GPU: Single Instruction, Multiple Threads (SIMT)
 - CPU core \simeq GPU multiprocessor (SM)
 - Working unit: set of threads (32, a warp)
 - Fast switching of threads (large register file)
 - Branching if _____

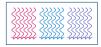


Vector

SMT



SIMT





SIMT

$\mathsf{SIMT}=\mathsf{SIMD}\oplus\mathsf{SMT}$



Vector











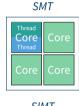
SIMT

$\mathsf{SIMT}=\mathsf{SIMD}\oplus\mathsf{SMT}$



Vector





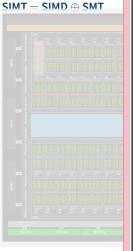
Graphics: Nvidia Corporation [6]





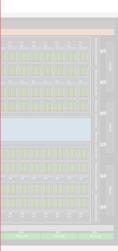


SIMT



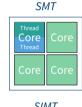
Multiprocessor

M		L1 Instruc	tion Cache			
LØI	nstruction C	ache		L0 k	nstruction C	ache
Warp Scheduler (32 threadiclk) Dispatch Unit (32 threadiclk)			Warp Scheduler (32 thread/clk) Dispatch Unit (32 thread/clk)			
	File (16,38				File (16,38	
INT32 INT32 FP32 FP32	ГРБА		INT52 INT52	FP32 FP32	FP64	
INTAR INTAR FP32 FP32			INT32 INT32	FP32 FP32		
INTAZ INTAZ FP32 FP32		TENSOR CORE	INT32 INT32	FP32 FP32		
INT32 INT32 FP32 FP32			INT32 INT32	FP32 FP32		TENSOR CORE
INT32 INT32 FP32 FP32			INT32 INT32	FP32 FP32		Contra Contra
INT32 INT32 FP32 FP32				FP32 FP32		
INT32 INT32 FP32 FP32				FP32 FP32		
INT32 INT32 FP32 FP32	EP64	10' 10' eru		FP32 FP32	FP54	LOV LOV POPU
AT AT AT AT	BT BT	W W SFU		BT BT	100° 100' 117 117	SFU SFU
LØ	L0 Instruction Cache					
Warp Scheduler (32 thread/clk) Dispatch Unit (32 thread/clk)		Warp Scheduler (32 thread/clk) Dispatch Unit (32 thread/clk)				
Register File (16,384 x 32-bit)		Register File (16,384 x 32-bit)				
INTAZ INTAZ FP32 FP32			INT32 INT32	FP32 FP32		
INT32 INT32 FP32 FP32			INT32 INT32	FP32 FP32		
INTAZ INTAZ FP32 FP32			INT32 INT32	FP32 FP32		
INT32 INT32 FP32 FP32		TENSOR CORE		FP32 FP32		TENSOR CORE
INT32 INT32 FP32 FP32				FP32 FP32		
INT32 INT32 FP32 FP32				FP32 FP32		
INT32 INT32 FP32 FP32	FP64			FP32 FP32	FP84	
INTER INTER FP32 FP32	FP64		INTRA INTRA	FP32 FP32	FP64	
ST ST ST ST	ST ST	ST ST SFU	LOY LOY ST ST	ST ST	ST ST	ST ST SFU
al al al al			the / Shared M	mory		
Tex	_	Tex	_	Tex		Tex



Vector





Graphics: Nvidia Corporation [6]





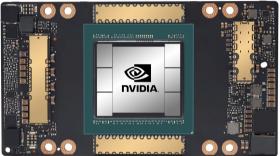
13 October 2022

Slide 7139

A100 vs H100

Comparison of current vs. next generation

A100



H100





A100 vs H100

Comparison of current vs. next generation





A100 vs H100

Comparison of current vs. next generation

A100

SM L4 Instruction Cache	uction Cache					
	L0 instruction Cache					
Werp Bcheduler (22 thread(cit)	Warp Scheduler (32 thread/dk)					
Dispatch Unit (22 threadlolk)	Dispatch Unit (32 thread/clk)					
Register File (16,384 x 32-bit)	Register File (16,304 x 32-bit)					
NT32 NT32 FF92 FF92 FF93	NT21 NT21 FP32 FP32 FP44					
NT22 NT22 FP22 FP22 FP34	NT23 NT23 FP32 FP32 FP34					
NT11 NT11 FF12 FF22 FF14	INTEL INTEL 1932 1932 1954					
BIDI BIDI FIDI FIDI FINA	NT2 NT2 NT2 TT2 TT2 TT2					
HT21 HT22 FF22 FF22 FF54	INTRINTA IPA IPA IPA					
HTTL HTTL 1912 1912 1914	NTH NTH 1938 1938 1944					
NTE NTE 192 192 194	NT1 NT1 P21 P22 FP4					
B1733 B1733 FF932 FF932 FF934	NT2 NT2 P22 P22 P23 P24					
W W W W W W W W W W W W W W W W W W W	*****					
La Instruction Cache	L0 Instruction Cache					
Werp Scheduler (22 threadlith)	Warp Scheduler (32 thread/cit)					
Dispatch Unit (22 thread/clk)	Dispatch Unit (32 thread/cib)					
Register File (16,384 x 32-bit)	Register File (16,304 x 32-bit)					
NT32 NT32 1792 1792 1994	NT3 NT3 173 173 173 1744					
NT32 NT32 FF32 FF32 FF92	NT22 INT22 1922 1922 1924					
B(T32 B(T32 FF32 FF32 FF92 FF94	NT22 (NT32 PT52 PT52 PT52 PT54					
HT22 HT22 FP22 FP32 FP34 TENSOR CORE	INTRINTAL ITAL ITAL ITAL					
HT22 HT22 FP22 FP22 FP44	INTERNESS FPER FPER					
NT22 NT22 FP22 FP22 FP24	NT22 INT22 FP32 FP32 FP34					
NT23 NT23 FF32 FF32 FF34	NT22 INT22 FP32 FP32 FP34					
NT32 NT32 FF32 FF32 FF44	15722 (5722 FF22 FF23 FF24					
11248 L1 Data Cache / Shared Memory						

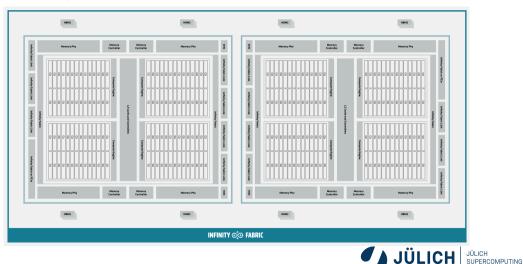
H100





AMD Instinct MI250

One GPU with Two Chiplets





Forschungszentrum

CENTRE

CPU vs. GPU

Let's summarize this!



Optimized for low latency

- + Large main memory
- + Fast clock rate
- + Large caches
- + Branch prediction
- + Powerful ALU
- Relatively low memory bandwidth
- Cache misses costly
- Low performance per watt



Optimized for high throughput

- + High bandwidth main memory
- + Latency tolerant (parallelism)
- + More compute resources
- + High performance per watt
- Limited memory capacity
- Low per-thread performance
- Extension card



Programming GPUs

State of the GPU

C/C++ C*

- Full vendor support
- Vendor support, but not (yet) entirely comprehensive
- Indirect, but comprehensive support, by vendor

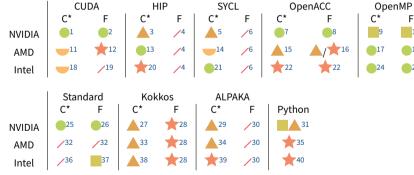
- Comprehensive support, but not by vendor
- Limited, probably indirect support but at least some

F

17

24

No direct support available, but of course one could / ISO-C-bind your way through it or directly link the libraries





Slide 12130

State of the GPU: Footnotes I

- 1: CUDA C/C++, supported through CUDA Toolkit
- 2: CUDA Fortran, proprietary Fortran extension supported by NVIDIA HPC SDK
- 3: HIP programs can directly use NVIDIA GPUs via a CUDA backend; HIP is maintained by AMD
- 4: No such thing like HIP for Fortran
- 5: SYCL can be used on NVIDIA GPUs with experimental support either in SYCL directly or in DPC++, or via hipSYCL
- 6: No such thing like SYCL for Fortran
- 7: OpenACC C/C++ supported on NVIDIA GPUs directly (and best) through NVIDIA HPC SDK; additional, somewhat limited support by GCC C compiler and Clacc
- 8: OpenACC Fortran supported on NVIDIA GPUs directly (and best) through NVIDIA HPC SDK; additional, somewhat limited support by GCC Fortran compiler and Flacc
- 9: OpenMP in C supported on NVIDIA GPUs through NVIDIA HPC SDK (but not full OpenMP feature set available), by GCC, and Clang
- 10: OpenMP in Fortran supported on NVIDIA GPUs through NVIDIA HPC SDK (but not full OpenMP feature set available), by GCC, and Flang
- 25: pSTL features supported on NVIDIA GPUs through NVIDIA HPC SDK



State of the GPU: Footnotes II

- 26: Standard Language parallel features supported on NVIDIA GPUs through NVIDIA HPC SDK
- 27: Kokkos supports NVIDIA GPUs by calling CUDA as part of the compilation process
- 28: Kokkos is a C++ model, but at least the authors provided an ISO C Binding example for Fortran
- 29: Alpaka supports NVIDIA GPUs by calling CUDA as part of the compilation process
- 30: Alpaka is a C++ model
- 31: There is a vast community of offloading Python code to NVIDIA GPUs, like CuPy, Numba, cuNumeric, and many others; NVIDIA actively supports a lot of them, but has no direct product like CUDA for Python; so, the status is somewhere in between
- 11: hipify by AMD can translate CUDA calls to HIP calls which runs natively on AMD GPUs
- 12: AMD offers a Source-to-Source translator to convert some CUDA Fortran functionality to OpenMP for AMD GPUs (gpufort); in addition, there are ROCm library bindings for Fortran in hipfort OpenACC/CUDA Fortran Source-to-Source translator gpufort: https://github.com/ROCmSoftwarePlatform/gpufort
- 13: HIP is the preferred native programming model for AMD GPUs
- 14: SYCL can use AMD GPUs, for example with hipSYCL or DPC++ for HIP AMD



State of the GPU: Footnotes III

- 15: OpenACC C/C++ can be used on AMD GPUs via GCC or Clacc; also, Intel's OpenACC to OpenMP Source-to-Source translator can be used to generate OpenMP directives from OpenACC directives
- 16: OpenACC Fortran can be used on AMD GPUs via GCC; also, AMD's gpufort Source-to-Source translator can move OpenACC Fortran code to OpenMP Fortran code, and also Intel's translator can work
- 17: AMD offers a dedicated, Clang-based compiler for using OpenMP on AMD GPUs: AOMP; it supports both C/C++ (Clang) and Fortran (Flang, example)
- S2: Currently, no (known) way to launch Standard-based parallel algorithms on AMD GPUs
- 33: Kokkos supports AMD GPUs through HIP
- 34: Alpaka supports AMD GPUs through HIP
- 35: AMD does not officially support GPU programming with Python (also not semi-officially like NVIDIA), but third-party support is avaiable, for example through Numba or a HIP version of CuPy
- 18: SYCLomatic translates CUDA code to SYCL code, allowing it to run on Intel GPUs; also, Intel's DPC++ Compatability Tool can transform CUDA to SYCL
- 19: No direct support, only via ISO C bindings, but at least an example can be found on GitHub; it's pretty scarce and not by Intel itself, though
- 20: CHIP-SPV supports mapping CUDA and HIP to OpenCL and Intel's Level Zero, making it run on Intel GPUs



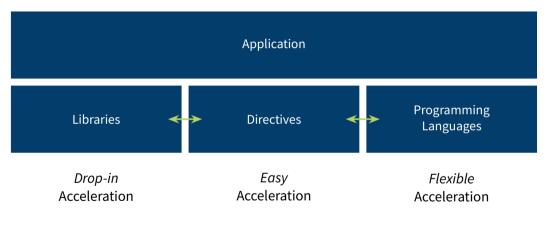
Slide 15139

State of the GPU: Footnotes IV

- 21: SYCL is the prime programming model for Intel GPUs; actually, SYCL is only a standard, while Intel's implementation of it is called DPC++ (Data Parallel C++), which extends the SYCL standard in places
- 22: OpenACC can be used on Intel GPUs by translating the code to OpenMP with Intel's Source-to-Source translator
- 24: Intel has extensive support for OpenMP through their latest compilers
- **36:** Currently, no (known) way to launch Standard-based parallel algorithms on Intel GPUs
- 37: With Intel oneAPI 2022.3, Intel supports DO CONCURRENT with GPU offloading
- 38: Kokkos supports Intel GPUs through SYCL
- 39: Alpaka v0.9.0 introduces experimental SYCL support
- 40: Not a lot of support available at the moment, but notably DPNP, a SYCL-based drop-in replacement for Numpy



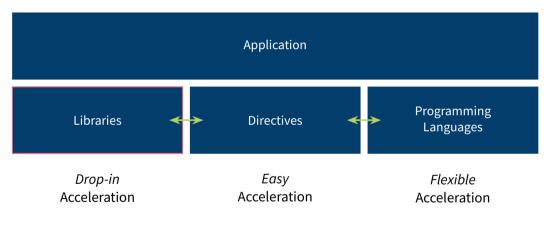
Summary of Acceleration Possibilities





Slide 17139

Summary of Acceleration Possibilities





Slide 17139

Programming GPUs Libraries



Programming GPUs is easy: Just don't!



Programming GPUs is easy: Just don't!

Use applications & libraries



Programming GPUs is easy: Just don't!

Use applications & libraries







Programming GPUs is easy: Just don't!

Use applications & libraries



JÜLICH SUPERCOMPLITING

CENTRE

Programming GPUs is easy: Just don't!

Use applications & libraries



JÜLICH SUPERCOMPLITING

CENTRE



- GPU-parallel BLAS (all 152 routines)
- Single, double, complex data types
- Constant competition with Intel's MKL
- Multi-GPU support
- → https://developer.nvidia.com/cublas http://docs.nvidia.com/cuda/cublas



cuBLAS

Code example

```
int a = 42: int n = 10:
float x[n]. v[n]:
// fill x. v
cublasHandle t handle:
cublasCreate(&handle):
float * d x. * d y:
cudaMallocManaged(\delta d x. n * sizeof(x[0])):
cudaMallocManaged(\delta d y, n * sizeof(y[0]);
cublasSetVector(n. sizeof(x[0]). x. 1. d x. 1):
cublasSetVector(n, sizeof(v[0]), v, 1, d v, 1);
cublasSaxpy(n, a, d x, 1, d y, 1);
cublasGetVector(n. sizeof(v[0]). d v. 1. v. 1):
cudaFree(d x); cudaFree(d y);
cublasDestrov(handle):
```



cuBLAS

Code example

<pre>int a = 42; int n = 10; float x[n], y[n]; // fill x, y</pre>			
cublasHandle_t handle; cublasCreate(∂handle);●			Initialize
<pre>float * d_x, * d_y; cudaMallocManaged(&d_x, n * cudaMallocManaged(&d_y, n * cublasSetVector(n, sizeof(x[cublasSetVector(n, sizeof(y[</pre>	<pre>sizeof(y[0])); 0]), x, 1, d_x, 1)</pre>		Allocate GPU memory Copy data to GPU
cublasSaxpy(n, a, d_x, 1, d_	y, 1);		Call BLAS routine
<pre>cublasGetVector(n, sizeof(y[0]), d_y, 1, y, 1);</pre>			Copy result to host
<pre>cudaFree(d_x); cudaFree(d_y); cublasDestroy(handle);</pre>			
Member of the Helmholtz Association	13 October 2022	Slide 22139	JÜLICH JÜLICH SUPERCOMPUTING CENTRE

Programming GPUs Directives

GPU Programming with Directives

Keepin' you portable

Annotate serial source code by directives

#pragma acc loop
for (int i = 0; i < 1; i++) {};</pre>



GPU Programming with Directives

Keepin' you portable

Annotate serial source code by directives

```
#pragma acc loop
for (int i = 0; i < 1; i++) {};</pre>
```

- OpenACC: Especially for GPUs; OpenMP: Has GPU support
- Compiler interprets directives, creates according instructions



GPU Programming with Directives

Keepin' you portable

Annotate serial source code by directives

```
#pragma acc loop
for (int i = 0; i < 1; i++) {};</pre>
```

- OpenACC: Especially for GPUs; OpenMP: Has GPU support
- Compiler interprets directives, creates according instructions

Pro

- Portability
 - Other compiler? No problem! To it, it's a serial program
 - Different target architectures from same code
- Easy to program

Con

- Only few compilers
- Not all the raw power available
- A little harder to debug



OpenACC / OpenMP

Code example

```
void saxpy_acc(int n, float a, float * x, float * y) {
    #pragma acc kernels
    for (int i = 0; i < n; i++)
        y[i] = a * x[i] + y[i];
}
float a = 42;
int n = 10;
float x[n], y[n];
// fill x, y
saxpy_acc(n, a, x, y);</pre>
```



OpenACC / OpenMP

Code example

```
void saxpy_acc(int n, float a, float * x, float * y) {
    #pragma omp target map(to:x[0:n]) map(tofrom:y[0:n]) loop
    for (int i = 0; i < n; i++)
        y[i] = a * x[i] + y[i];
}
float a = 42;
int n = 10;
float x[n], y[n];
// fill x, y
saxpy_acc(n, a, x, y);</pre>
```



Programming GPUs CUDA C/C++

Finally...



Finally...

OpenCL Open Computing Language by Khronos Group (Apple, IBM, NVIDIA, ...) 2009

- Platform: Programming language (OpenCL C/C++), API, and compiler
- Targets CPUs, GPUs, FPGAs, and other many-core machines
- Fully open source



Finally...

OpenCL Open Computing Language by Khronos Group (Apple, IBM, NVIDIA, ...) 2009

- Platform: Programming language (OpenCL C/C++), API, and compiler
- Targets CPUs, GPUs, FPGAs, and other many-core machines
- Fully open source

CUDA NVIDIA's GPU platform 2007

- Platform: Drivers, programming language (CUDA C/C++), API, compiler, tools, ...
- Only NVIDIA GPUs
- Compilation with nvcc (free, but not open)

clang has CUDA support, but CUDA needed for last step

Also: CUDA Fortran; and more in NVIDIA HPC SDK



Finally...

OpenCL Open Computing Language by Khronos Group (Apple, IBM, NVIDIA, ...) 2009

- Platform: Programming language (OpenCL C/C++), API, and compiler
- Targets CPUs, GPUs, FPGAs, and other many-core machines
- Fully open source

CUDA NVIDIA's GPU platform 2007

- Platform: Drivers, programming language (CUDA C/C++), API, compiler, tools, ...
- Only NVIDIA GPUs
- Compilation with nvcc (free, but not open)

clang has CUDA support, but CUDA needed for last step

- Also: CUDA Fortran; and more in NVIDIA HPC SDK
- HIP AMD's unified programming model for AMD (via ROCm) and NVIDIA GPUs 2016+

SYCL Intel's unified programming model for CPUs and GPUs (also: DPC++)





Finally...

OpenCL Open Computing Language by Khronos Group (Apple, IBM, NVIDIA, ...) 2009

- Platform: Programming language (OpenCL C/C++), API, and compiler
- Targets CPUs, GPUs, FPGAs, and other many-core machines
- Fully open source

CUDA NVIDIA's GPU platform 2007

- Platform: Drivers, programming language (CUDA C/C++), API, compiler, tools, ...
- Only NVIDIA GPUs
- Compilation with nvcc (free, but not open)
 - clang has CUDA support, but CUDA needed for last step
- Also: CUDA Fortran; and more in NVIDIA HPC SDK
- HIP AMD's unified programming model for AMD (via ROCm) and NVIDIA GPUs 2016+

SYCL Intel's unified programming model for CPUs and GPUs (also: DPC++)

- Choose what flavor you like, what colleagues/collaboration is using
- Hardest: Come up with parallelized algorithm



Finally...

OpenCL Open Computing Language by Khronos Group (Apple, IBM, NVIDIA, ...) 2009

- Platform: Programming language (OpenCL C/C++), API, and compiler
- Targets CPUs, GPUs, FPGAs, and other many-core machines
- Fully open source

CUDA NVIDIA's GPU platform 2007

- Platform: Drivers, programming language (CUDA C/C++), API, compiler, tools, ...
- Only NVIDIA GPUs
- Compilation with nvcc (free, but not open)

clang has CUDA support, but CUDA needed for last step

Also: CUDA Fortran; and more in NVIDIA HPC SDK

HIP AMD's unified programming model for AMD (via ROCm) and NVIDIA GPUs 2016+

SYCL Intel's unified programming model for CPUs and GPUs (also: DPC++)

- Choose what flavor you like, what colleagues/collaboration is using
- Hardest: Come up with parallelized algorithm



In software: Threads, Blocks

Methods to exploit parallelism:



- Methods to exploit parallelism:
 - Thread



- Methods to exploit parallelism:
 - Threads





In software: Threads, Blocks

Methods to exploit parallelism:







In software: Threads, Blocks

Methods to exploit parallelism:

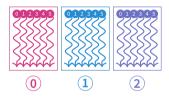






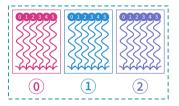


- Methods to exploit parallelism:
 - Threads \rightarrow Block
 - Blocks



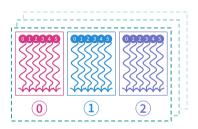


- Methods to exploit parallelism:
 - Threads \rightarrow Block
 - $\blacksquare \quad \mathsf{Blocks} \to \mathsf{Grid}$





- Methods to exploit parallelism:
 - Threads \rightarrow Block
 - $\blacksquare \quad \mathsf{Blocks} \to \mathsf{Grid}$
 - Threads & blocks in 3D



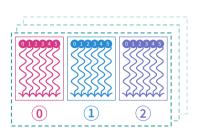


In software: Threads, Blocks

- Methods to exploit parallelism:
 - Threads \rightarrow Block
 - Blocks \rightarrow Grid
 - Threads & blocks in 3D
- Parallel function: kernel
 - __global__ kernel(int a, float * b) { }
 - Access own ID by global variables threadIdx.x, blockIdx.y,...
- Execution entity: threads
 - Lightweight \rightarrow fast switchting!
 - = 1000s threads execute simultaneously \rightarrow order non-deterministic!



Slide 28139



CUDA SAXPY

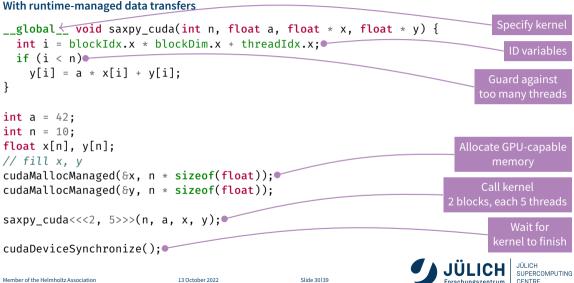
With runtime-managed data transfers

```
global void saxpy cuda(int n, float a, float * x, float * y) {
 int i = blockIdx.x * blockDim.x + threadIdx.x;
 if (i < n)
   v[i] = a * x[i] + v[i]:
}
int a = 42;
int n = 10;
float x[n], y[n];
// fill x, y
cudaMallocManaged(&x. n * sizeof(float)):
cudaMallocManaged(&y, n * sizeof(float));
saxpy cuda<<<2, 5>>>(n, a, x, y);
```

```
cudaDeviceSynchronize();
```



CUDA SAXPY



Kernel Conversion

Recipe for C Function \rightarrow CUDA Kernel

Identify Loops

```
void scale(float scale, float * in, float * out, int N) {
   for (int i = 0; i < N; i++)
        out[i] = scale * in[i];
}</pre>
```



Kernel Conversion

Recipe for C Function \rightarrow CUDA Kernel

Identify Loops

```
void scale(float scale, float * in, float * out, int N) {
    for (
        int i = 0;
        i < N;
        i++
    )
        out[i] = scale * in[i];
}</pre>
```



Recipe for C Function \rightarrow CUDA Kernel

Identify Loops Extract Index

```
void scale(float scale, float * in, float * out, int N) {
    int i = 0
    for (;
        i < N;
        i++
    )
        out[i] = scale * in[i];
}</pre>
```



Recipe for C Function \rightarrow CUDA Kernel

Identify Loops Extract Index Extract Termination Condition

```
void scale(float scale, float * in, float * out, int N) {
    int i = 0
    for (;
        ;
        i++
    )
        if (i < N)
            out[i] = scale * in[i];
}</pre>
```



Recipe for C Function \rightarrow CUDA Kernel

Identify Loops Extract Index Extract Termination Condition Remove for

```
void scale(float scale, float * in, float * out, int N) {
    int i = 0
```



Recipe for C Function \rightarrow CUDA Kernel

Identify Loops Extract Index Extract Termination Condition Remove for Add global

__global__ void scale(float scale, float * in, float * out, int N) { int i = 0



Recipe for C Function \rightarrow CUDA Kernel

Identify Loops Extract Index Extract Termination Condition Remove for Add global
Replace i by threadIdx.x
__global__ void scale(float scale, float * in, float * out, int N) {

```
int i = threadIdx.x;
```



Recipe for C Function \rightarrow CUDA Kernel

Identify Loops Extract Index Extract Termination Condition Remove for Add global
Replace i by threadIdx.x ... including block configuration
__global__ void scale(float scale, float * in, float * out, int N) {
 int i = threadIdx.x + blockIdx.x * blockDim.x;



Summary

• C function with explicit loop void scale(float scale, float * in, float * out, int N) { for (int i = 0; i < N; i++) out[i] = scale * in[i]; } • CUDA kernel with implicit loop

```
__global__ void scale(float scale, float * in, float * out, int N) {
    int i = threadIdx.x + blockIdx.x * blockDim.x;
    if (i < N)
        out[i] = scale * in[i];
}</pre>
```



Programming GPUs Performance Analysis

GPU Tools

The helpful helpers helping helpless (and others)

NVIDIA

cuda-gdb GDB-like command line utility for debugging compute-sanitizer Check memory accesses, race conditions, ... Nsight IDE for GPU developing, based on Eclipse (Linux, OS X) or Visual Studio (Windows) or VScode Nsight Systems GPU program profiler with timeline Nsight Compute GPU kernel profiler

AMD

rocProf Profiler for AMD's ROCm stack uProf Analyzer for AMD's CPUs and GPUs



Nsight Systems

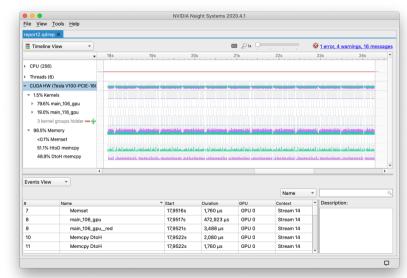
•••

\$ nsys profilestats=true ./poisson2d 10 # (shortened) CUDA API Statistics:								
Time(%) 90.9	Total Time (ns) 160,407,572		Average 5,346,919.1	Minimum 1,780	Maximum 25,648,117	Name cuStreamSynchronize		
CUDA Kernel Statistics:								
Time(%)	Total Time (ns)	Instances	Average	Minimum	Maximum	Name		
100.0 0.0	158,686,617 25,120	10 10	15,868,661.7 2,512.0	14,525,819 2,304	25,652,783 3,680	main_106_gpu main_106_gpured		



Nsight Systems

GUI



Nsight Compute

-		
G	U	н
-	-	

Details * Launch: -	mc_polymer	r_iteration_352_gpu 🔻	Add 8	aseline 👻 Apply	Rules						Save as li	mag
urrent 1194 Time: 25.82	2 msecond	Cycles: 28.191.300	Regs: 144 G	PU: A100-SXM4	1-40GB	SM Frequency	1.09 cycle/nsecon	d CC: 8.0	Process: [31938] SOM	ia 🕑	Θ
100 126 Time: 77.17	msecond	Cycles: 101.123.609	Regs: 144 G	PU: Tesla V100-	SXM2-16GB	SM Frequency	r: 1.31 cycle/nsecond	d CC: 7.0	Process: [30412] SOM	A	
PU Speed Of Light 🛕										All	÷	\$
-level overview of the utilizat retical maximum. High-level								percentage	of utilization	with respect	t to the	
SM [%]			20.5	1 (+160.09%)	Duration	[msecond]				25.82	(-66.	541
Memory [%]			55.8	2 (+1.26%)	Elapsed C	ycles [cycle]]			28191300	(-72.	12*
. L1/TEX Cache [%]			40.1			Cycles [cyc			277	84365.64	(-72.	
L2 Cache [%]						ncy [cycle/n				1.09	(-16.	
. DRAM [%]			31.5	2 (-42.82%)	DRAM Freq	uency [cycle,	/nsecond]			1.21	(+38.	21
emory [%]												
	0,0	20,0	30,0	40,0	50,0	60,0	70,0	80,	0	90,0		100
	0,0	20,0	30,0		50,0 eed Of Light		70,0	80,	0	90,0		100
		20,0 SM Breakdown	30,0				70,0 SOL Memor	,		90,0		100
iemory [%] 0,0 10 SOL SM: Issue Active [%]					eed Of Light		SOL Memor	,			+124.58	
0,0 10			2	Sp	sol L2: >	[%]	SOL Memor	,		55.82 (8%)
0,0 10 SOL SM: Issue Active [%]	SOL		20	Sp 0.51 (+160.09%)	SOL L2:) SOL L2: 1	[%] (bar2lts Cycles A Tag Requests [SOL Memor	,		55.82 (45.27 (+124.58	8%)
0,0 10 SOL SM: Issue Active [%] SOL SM: Inst Executed [%] SOL SM: Pipe Shared Cycles	SOL S		21 21 18	Sp 0.51 (+160.09%) 0.46 (+160.10%)	SOL L2: > SOL L2: > SOL L2: 1 SOL L1: M SOL L2: 1	[%] (bar2lts Cycles A Tag Requests [L1tex2xbar Req Sectors [%]	SOL Memor Active [%] %] Cycles Active [%]	,		55.82 (45.27 (39.61 (38.80	(+124.58 (+178.82 (+107.93 (+71.23	8% 2% 3% 3%
0,0 10 SOL SM: Issue Active (%) SOL SM: Inst Executed (%) SOL SM: Pipe Shared Cycles SOL SM: Pipe Fp84 Cycles A	SOL S Active [%] Active [%]		2) 20 18 18	Sp 0.51 (+160.09%) 0.46 (+160.10%) 0.36 (+165.69%)	SOL L2: > SOL L2: > SOL L2: 1 SOL L1: M SOL L2: 1	[%] (bar2lts Cycles A Tag Requests [1 L1tex2xbar Req	SOL Memor Active [%] %] Cycles Active [%]	,		55.82 (45.27 (39.61 (38.80	(+124.58 (+178.82 (+107.93	8% 2% 3% 3%
0,0 10 SOL SM: Issue Active (%) SOL SM: Inst Executed (%) SOL SM: Pipe Sp84 Cycles A SOL SM: Pipe Fp84 Cycles A SOL SM: Pipe Au Cycles Act SOL SM: Pipe Au Cycles Act	SOL S Active [%] Active [%] Lsu [%] ive [%]	SM Breakdown	2/ 2(18 18	Sp 0.51 (+160.09%) 0.46 (+160.10%) 0.36 (+165.69%) 0.36 (+165.69%)	SOL L2:) SOL L2: 1 SOL L2: 1 SOL L1: M SOL L2: 1 SOL L1: D	[%] (bar2lts Cycles A Tag Requests [L1tex2xbar Req Sectors [%]	SOL Memor	,		55.82 (45.27 (39.61 (38.80 34.10 31.52	(+124.58 (+178.82 (+107.93 (+71.23 (+76.03 (-42.82	8% 2% 3% 3% 3%
0,0 10 SOL SM: Issue Active [%] SOL SM: Inst Executed [%] SOL SM: Pipe Shared Cycles SOL SM: Pipe Fp64 Cycles A SOL SM: Inst Executed Pipe I	SOL : s Active [%] Active [%] Lsu [%] ive [%] Cbu Pred Or	SM Breakdown	2) 24 18 18 11 11	Sp 0.51 (+160.09%) 0.46 (+160.10%) 0.36 (+165.69%) 0.36 (+165.69%) 0.80 (+127.96%)	SOL L2: 3 SOL L2: 1 SOL L2: 1 SOL L1: N SOL L1: N SOL L1: D SOL GPU SOL L1: L	(%) Tag Requests [Litex2xbar Req Sectors [%] tata Pipe Lsu Wa	SOL Memory Active [%] %] Cycles Active [%] vefronts [%] put [%]	,		55.82 (45.27 (39.61 (38.80 34.10 31.52 24.83	(+124.58 (+178.82 (+107.93 (+71.23 (+76.03	8% 2% 3% 3% 2% 7%

SOL SM: Inst Executed Pipe Lsu [%]	10.80 (+127.96%)	SOL L1: Data Pipe Lsu Wavefronts [%]	34.10 (+76.03%)
SOL SM: Pipe Au Cycles Active [%]	10.37 (+161.23%)	SOL GPU: Dram Throughput [%]	31.52 (-42.82%)
SOL SM: Inst Executed Pipe Cbu Pred On Any [%]	8.13 (+71.98%)	SOL L1: Lsu Writeback Active [%]	24.83 (+63.67%)
SOL SM: Mio2rf Writeback Active [%]	8.10 (+161.91%)	SOL L2: D Sectors [%]	22.64 (+115.71%)
SOL SM: Mio Pq Read Cycles Active [%]	8.10 (+105.96%)	SOL L2: D Sectors Fill Device [%]	12.19 (-12.29%)
SOL SM: Mio Pq Write Cycles Active [%]	7.53 (+165.64%)	SOL L1: Lsuin Requests [%]	10.80 (+127.96%)
SOL SM: Pipe Fma Cycles Active [%]	7.22 (+165.80%)	SOL L2: Lts2xbar Cycles Active [%]	8.82 (-21.48%)
SOL SM: Mio Inst Issued [%]	5.55 (+123.69%)	SOL L1: M Xbar2l1tex Read Sectors [%]	6.39 (-25.58%)
SOL SM: Inst Executed Pipe Xu [%]	4.59 (+165.69%)	SOL L1: Data Bank Reads [%]	3.09 (+77.11%)
SOL SM: Inst Executed Pipe Uniform [%]	1.29	SOL L1: Data Bank Writes [%]	1.95 (+23.96%)
SOL SM: Inst Executed Pipe Adu [%]	1.18 (+165.53%)	SOL L1: Texin Sm2tex Req Cycles Active [%]	0.00 (+258.69%)
SOL IDC: Request Cycles Active [%]	0.59 (+165.37%)	SOL L1: F Wavefronts [%]	0.00 (+258.69%)
SOL SM: Inst Executed Pipe Tex [%]	0 (+0.00%)	SOL L2: D Sectors Fill Sysmem [%]	0.00 (+inf%)
SOL SM: Inst Executed Pipe Ipa [%]	0 (+0.00%)	SOL L1: Data Pipe Tex Wavefronts [%]	0 (+0.00%)

Conclusion

- GPUs provide highly-parallel computing power
- Many ways to use them

Libraries Best performance, but need to map

Directives Easy to use, but needs to fit



- GPUs provide highly-parallel computing power
- Many ways to use them

Libraries Best performance, but need to map

Directives Easy to use, but needs to fit



- GPUs provide highly-parallel computing power
- Many ways to use them

Libraries Best performance, but need to map

Directives Easy to use, but needs to fit



- GPUs provide highly-parallel computing power
- Many ways to use them

Libraries Best performance, but need to map

Directives Easy to use, but needs to fit





Appendix

Appendix Glossary References



Glossary I

- AMD Manufacturer of CPUs and GPUs. 54, 55, 56, 57, 58, 59, 92, 93
 - API A programmatic interface to software by well-defined functions. Short for application programming interface. 54, 55, 56, 57, 58, 59
- CUDA Computing platform for GPUs from NVIDIA. Provides, among others, CUDA C/C++. 2, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 93
 - HIP GPU programming model by AMD to target their own and NVIDIA GPUs with one combined language. Short for Heterogeneous-compute Interface for Portability. 54, 55, 56, 57, 58, 59

NVIDIA US technology company creating GPUs. 22, 23, 24, 54, 55, 56, 57, 58, 59, 81, 92





Glossary II

OpenACC Directive-based programming, primarily for many-core machines. 48, 49, 50, 51, 52

- OpenCL The Open Computing Language. Framework for writing code for heterogeneous architectures (CPU, GPU, DSP, FPGA). The alternative to CUDA. 54, 55, 56, 57, 58, 59
- OpenMP Directive-based programming, primarily for multi-threaded machines. 48, 49, 50, 51, 52
 - ROCm AMD software stack and platform to program AMD GPUs. Short for Radeon Open Compute (*Radeon* is the GPU product line of AMD). 54, 55, 56, 57, 58, 59
 - SAXPY Single-precision $A \times X + Y$. A simple code example of scaling a vector and adding an offset. 69, 70





Glossary III

- CPU Central Processing Unit. 6, 7, 8, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 54, 55, 56, 57, 58, 59, 92, 93
- GPU Graphics Processing Unit. 2, 3, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 30, 38, 39, 40, 41, 42, 43, 44, 47, 48, 49, 50, 53, 54, 55, 56, 57, 58, 59, 70, 80, 81, 86, 87, 88, 89, 92, 93
- SIMD Single Instruction, Multiple Data. 15, 16, 17, 18, 19, 20, 21, 22, 23, 24
- SIMT Single Instruction, Multiple Threads. 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24
 - SM Streaming Multiprocessor. 15, 16, 17, 18, 19, 20, 21, 22, 23, 24
- SMT Simultaneous Multithreading. 15, 16, 17, 18, 19, 20, 21, 22, 23, 24





References: Images, Graphics I

- [1] Alexandre Debiève. *Title Graphic: Bowels of computer*. Freely available at Unsplash. URL: https://unsplash.com/photos/F07JIlwjOtU (page 2).
- [2] Karl Rupp. Pictures: CPU/GPU Performance Comparison. URL: https://www.karlrupp.net/2013/06/cpu-gpu-and-mic-hardwarecharacteristics-over-time/ (pages 4, 5).
- [3] Mark Lee. Picture: kawasaki ninja. URL: https://www.flickr.com/photos/pochacco20/39030210/ (pages 6, 7).
- [4] Shearings Holidays. Picture: Shearings coach 636. URL: https://www.flickr.com/photos/shearings/13583388025/(pages 6, 7).





References: Images, Graphics II

- [5] Nvidia Corporation. Pictures: Volta GPU. Volta Architecture Whitepaper. URL: https://images.nvidia.com/content/volta-architecture/pdf/Volta-Architecture-Whitepaper-v1.0.pdf.
- [6] Nvidia Corporation. Pictures: Ampere GPU. Nvidia Devblogs: NVIDIA Ampere Architecture In-Depth. URL: https://devblogs.nvidia.com/nvidia-ampere-architecture-in-depth/ (pages 22-24).
- [7] Nvidia Corporation. Pictures: Hopper GPU. Nvidia Developer Technical Blog: NVIDIA Hopper Architecture In-Depth. URL: https: //developer.nvidia.com/blog/nvidia-hopper-architecture-in-depth/.
- [8] AMD. Pictures: MI250 GPU. AMD CDNA2 Architecture Whitepaper. URL: https: //www.amd.com/system/files/documents/amd-cdna2-white-paper.pdf.





References: Images, Graphics III

[9] Wes Breazell. Picture: Wizard. URL: https://thenounproject.com/wes13/collection/its-a-wizards-world/ (pages 39-43).

