

GPU ACCELERATORS AT JSC SUPERCOMPUTING INTRODUCTION COURSE

1 June 2023 | Andreas Herten | Forschungszentrum Jülich



Member of the Helmholtz Association

Outline

GPUs at JSC JUWELS JUWELS Cluster **JUWELS Booster** JURECA DC **GPU** Architecture **Empirical Motivation** Comparisons **GPU** Architecture Summary

Programming GPUs Libraries Directives CUDA C/C++ Performance Analysis Advanced Topics Advanced Topics





JUWELS Cluster – Jülich's Scalable System

- 2500 nodes with Intel Xeon CPUs (2 \times 24 cores)
- 46 + 10 nodes with 4 NVIDIA Tesla V100 cards (16 GB memory)
- 10.4 (CPU) + 1.6 (GPU) PFLOP/s peak performance (Top500: #86)







JUWELS Booster – Scaling Higher!

- = 936 nodes with AMD EPYC Rome CPUs (2 \times 24 cores)
- Each with 4 NVIDIA A100 Ampere GPUs (each: FP64TC: 19.5 FP64: 9.7 TFLOP/s, 40 GB memory)
- InfiniBand DragonFly+ HDR-200 network; $4 \times 200 \text{ Gbit/s per node}$







Top500 List Nov 2020:

- #1 Europe
- #7 World
- #4* Top/Green500



- = 936 nodes with AMD EPYC Rome CPUs (2 \times 24 cores)
- Each with 4 NVIDIA A100 Ampere GPUs (each: FP64TC: 19.5 FP64: 9.7 TFLOP/s, 40 GB memory)
- InfiniBand DragonFly+ HDR-200 network; 4 \times 200 Gbit/s per node





JURECA DC – Multi-Purpose

- 768 nodes with AMD EPYC Rome CPUs (2 \times 64 cores)
- 192 nodes with 4 NVIDIA A100 Ampere GPUs
- InfiniBand DragonFly+ HDR-100 network



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



Slide 7141

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





Host

Device







A100

40 GB RAM, 1555 GB/s



Member of the Helmholtz Association

1 June 2023

Slide 9141

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 9141



Scalar

A ₀	+	B_0	=	<i>C</i> ₀
A_1	+	B_1	=	C_1
<i>A</i> 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)



Member of the Helmholtz Association



Slide 10|41

CPU:

 $SIMT = SIMD \oplus SMT$

SIMT

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



Vector







CPU:

- Single Instruction, Multiple Data (SIMD)
- Simultaneous Multithreading (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 \cong 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





Graphics: img:amperepictures





SIMT

$SIMT = SIMD \oplus SMT$



Vector





Graphics: img:amperepictures





SIMT



Multiprocessor

м		L1 Instruc	tion Cache				
L0 Ir	struction C	ache		LO	Instruction (lache	
Warp Scheduler (\$2 threadicik)			Warp Scheduler (32 thread/clk)				
Dispatch	i Unit (\$2 th	read/cik)		Dispate	th Unit (32 ti	read/clk)	
Register	File (16,38	4 x 32-bit)		Register	File (16,38	4 x 32-bit)	
INT32 INT32 FP32 FP32			NT32 INT32	FP32 FP32	FP64		
INTARINTAR FP32 FP32			INT32 INT32	FP32 FP32	FP84		
INTAR INTAR FP32 FP32			INT32 INT32	FP32 FP32	FP64		
INT32 INT32 FP32 FP32		TENSOR CORE	INT32INT32	FP32 FP32	FP84	TENSOR CORE	
INTAZ INTAZ FP32 FP32		TENSOR CORE	INT32 INT32	FP32 FP32	FP84		
INT32 INT32 FP32 FP32			INT32INT32	FP32 FP32	FP64		
INT32 INT32 FP32 FP32			INT32 INT32	FP32 FP32	FP84		
INT32 INT32 FP32 FP32			INTERINTER	FP32 FP32	FP64		
LEV LEV LEV LEV	LOY LOY ST ST		ST ST	101 101 57 57	100 L00 51 51	SFU	
144							
Warp Sch	L0 Instruction Cache Warp Scheduler (32 thread/clk)			Warp Scheduler (32 thread/clk)			
Dispatel	Unit (32 Ih	read/clk}		Dispate	th Unit (32 ti	weedk(lk)	
Register	File (16,38	4 x 32-bit)		Register	File (16,38	i4 x 32-bit)	
INTERINTER FP32 FP32	IP64		NT321NT32	FP32 FP32	FP64		
INTAZ INTAZ FP32 FP32			INT32INT32	FP32 FP32	FP64		
INTAZINTAZ FP32 FP32			INTERINTER	FP32 FP32	FP64		
INTRE INTRE FP32 FP32		TENSOR CORE	INT32INT32	FP32 FP32	FP64	751000 0005	
INT32 INT32 FP32 FP32		TENSOR CORE	INT32INT32	FP32 FP32	FP64	TENSOR CORE	
INT32 INT32 FP32 FP32			INT32INT32	FP32 FP32	FP64		
INT32 INT32 FP32 FP32			INT32INT32	FP32 FP32	FP84		
INT32 INT32 FP32 FP32			INT32 INT32	FP32 FP32	FP64		
LDV LDV LDV LDV ST ST ST ST	LDY LDI ST ST	ST ST SFU	ST ST	LDI LDI ST ST	100 L00 81 81	ST ST SFU	
		192KB L1 Data Cac	he / Shared M	emory			
				-		T	



Vector





Graphics: img:amperepictures





Slide 10141

A100 vs H100

Comparison of current vs. next generation

A100



H100





Slide 11|41

A100 vs H100

Comparison of current vs. next generation



A100 vs H100

Comparison of current vs. next generation

A100

SM						
		L1 Instru	tion Cache			
	atruction C	acter	LUCE IN CO.	struction Ca	che	
Werp Echeduler (22 threadsile)			Warg Scheduler (32 threadicity)			
Dispatch Unit (32 Ihreadicik)		Dispatch Unit (32 thread/clb)				
		Register File (16,384 x 32-bit)				
1001 1001 1002 1002			NT31 NT33 FP33 FP33	1944		
NOTING LEAD LEAD			INTERINTER FROM FROM	FP64		
60116012 0022 0022			INTERINTER PERSON PERSON	1764		
6733 6772 7792 7792		TINSOR CORE	11733 11733 1793 1793	1754	TENSOR CORE	
нтанта <mark>1192</mark> (192			INTERINTER FROM FROM	1754		
1021002 FP22 FP22			INTERINTER FROM FROM	1794		
1022 AUX 1022 AUX			NT32 NT32 FP32 FP32	1944		
1021 HTTL 1122 1123			NT33 NT33 FP32 FP32	1984		
****	401 401 91 91	un un sru	****	81 81 10. 10.	ម្លាំម្ហី ទោប	
Lil Instruction Cache						
1.01		ache	LOIN	struction Ca	che	
Werp Sch	natruction C Advier (221	iche Hreadich)	Warg Sch	elrection Ca Maler (32.0)	chei reard/c@j	
Werp Sch Dispatch	NOVER 122 11	eche Freedicik) reedicik)	L0 in Ward Bolt Dispatch	dra clien Ca Maler (32 th Unit (32 the	(fm read/c0) rach(c0)	
Register	nduner (121) 1 Unit (121) File (16,36)	acha Freedicth) Freedicth) K x 32-bit)	Litin Wang Bolt Dispetch Register I	Hardten Ca Make (32 Ho Unit (32 Ho Tile (16,304	cher receditabi anditabi x 32-biti	
King Sch Dispatch Register 81133 81132 FF32 FF32	File (16,38) File (16,38)	acha Presidicita) Residicita) S x 32-bittj	Li In Wara Gub Diapatch Rogister I NT22 NT22 FP32 FP33	Unit (32 the Unit (32 the Tile (16,204 FP64	cha read/ch) red/ch) x 32-bit)	
E 11 Winys 547 Citiganic Regission 8(132) 8(132) 87922 87922 8(132) 8(132) 87922 87922	File (16,36) File (16,36) File (16,36)	actor Breadly(1) Preadly(1) 5 x 32-brit)	Libin Wang Sch Dapatch Register I INT23 INT23 1923 1923 INT23 INT23 1923 1923	Hindiso Da elektr (52 Ho Unit (52 Ho Tile (16,304 FP64 FP64	cha madd(ch) add(ch) x 32-bit)	
611 Werp Sci Gisparce Register 1733 11732 17932 17932 11732 11732 17932 17933 11732 11732 17932 17933	File (16,30) File (16,30) File (16,30) File (16,30) File (16,30)	Acte	Udina Wang Schw Dispetch Register I 1973 1973 1932 1933 1973 1973 1973	Hindler (32 the Hale (16,304 File (16,304 File (16,304 File (16,304	che mentică (andică (x 32-bit)	
4 119 Wing Sel Delayark Register H1732 H1732 H932 H732 H732 H932 H732 H732 H932 H732 H732 H932 H732 H732 H932	File (16,38) File		Vierg 505 Cospetiti Register 1 9723 9723 923 923 9723 9723 923 923 9723 9723 923 923 9723 9723 923 923 9723 9723 9723 9723	Houston Ca Hainer (32 the Unit (32 the Tale (16,304 FP44 FP44 FP44 FP44 FP44	the medical solitical x 32-bit)	
HTT22 HTT22 FF22 FF22 HTT22 HTT22 FF22 FF22	File (16,38) File	References	Using Solution View Solution Comparison Register 1 0123 (1023) 1923 1923 0123 (1023) 1923 1923 0123 (1023) 1923 1923 0123 (1023) 1923 1923 0123 (1023) 1923 1923	1994 (10,304) 1994 (10,304) 1994 (10,304) 1994 (10,304) 1994 (10,304) 1994 (10,304) 1994 (10,304) 1994 (10,304) 1994 (10,304)	the meditabl x 32 bit) TENSOR CORE	
HTT3 HT23 HT23 <td< th=""><th>Riverson C Robert (22 Ib Unit (22 Ib File (16,30) File (1</th><th>texection texesterion texasterion texasterion TENSOR CORE</th><th>University of the second secon</th><th>1944 (32 04) 1944 (32 04) 1944 (32 04) 1944 (32 04) 1944 (16,204) 1944 (16,204) 1944 (16,204) 1944 (16,204) 1944 (16,204)</th><th>the read(5) and(16) x 32-bit) TENSOR CORE</th></td<>	Riverson C Robert (22 Ib Unit (22 Ib File (16,30) File (1	texection texesterion texasterion texasterion TENSOR CORE	University of the second secon	1944 (32 04) 1944 (32 04) 1944 (32 04) 1944 (32 04) 1944 (16,204) 1944 (16,204) 1944 (16,204) 1944 (16,204) 1944 (16,204)	the read(5) and(16) x 32-bit) TENSOR CORE	
HTTL:	EVENIE (16,30) File (16,30) Fil	tereschi tereschi te 32-http://www.second tensorcore	Horse Solv Ward Solv Krazi Krazi Warzi Krazi	1794 1794 1794 1794 1794 1794 1794 1794	der eentitikk aktivel x 32-bit) TENSOR CORE	
111 Warp bei Clapset Register Register 1921 0021 1022 1022 1022 0022 1022 1022 1022 0021 1022 1022 1022 1022 1022 0021 1022 1022 1022 1022 0021 1022 1022 1022 1022 1022 0021 1022 1022 1022 1022 1022 1022 0021 1022 1022 1022 1022 1022 1022 0021 1022 1022 1022 1022 1022 1022	File (16,38) File	ternediti ternediti s 22 ktl	Home Box Cosynchin Home Box Cosynchin Home Box Cosynchin Home Box Home Box Harris Home Home Box HEID HATED HOME HOME	HILLION CO HILLION (32 HW URR (32 HW URR (32 HW FP44 FP44 FP44 FP44 FP44 FP44 FP44 FP4	der eeedta 3 aktabi x 32-bit) TEMSOR CORE	
1100 1000 1000 1000 1000 1000 1000 100	File (16,30) File	TENSOR CORE	Horse Bas Register	HILLING COM HILLING COLUMN HILLING C	ter end(3) x 32 bit) TENSOR CORE	
113 113 113 113 113 113 113 113 113 113	File (16,30) File		Line Head State Head Head Head State Head State Head State Head	10000000000000000000000000000000000000	ter seatball seatball x 322 bit) TENSOR CORE	
1000 1000 1000 1000 1000 1000 1000 100	File (16,30) File	TENSOR CORE		HILLING COLUMN	ter (1997) Halva) x 32-bit) TEMSOR CORE	

H100





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

Preface: CPU

A simple CPU program!

```
SAXPY: \vec{y} = a\vec{x} + \vec{y}, with single precision
Part of LAPACK BLAS Level 1
void saxpy(int n, float a, float * x, float * y) {
  for (int i = 0; i < n; i++)
    y[i] = a * x[i] + v[i];
}
int a = 42:
int n = 10:
float x[n], y[n];
// fill x, v
saxpy(n, a, x, y);
```



Summary of Acceleration Possibilities





Summary of Acceleration Possibilities






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], y[n];
// fill x, y
cublasHandle t handle:
cublasCreate(&handle):
float * d x, * d y;
cudaMallocManaged(&d_x, n * sizeof(x[0]));
cudaMallocManaged(\delta d v. n * sizeof(v[0])):
cublasSaxpv(handle. n. a. d x. 1. d v. 1):
cublasGetVector(n, sizeof(v[0]), d v, 1, v, 1);
```

```
cudaFree(d_x); cudaFree(d_y);
cublasDestroy(handle);
```



cuBLAS

Code example

int a = 42; int n = 10;

<pre>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 * sizeof(x[0]));● cudaMallocManaged(&d_y, n * sizeof(y[0]));</pre>	Allocate GPU memory
cublasSaxpy(handle, n, a, d_x, 1, d_y, 1);	Call BLAS routine
cublasGetVector(n, sizeof(y[0]), d_y, 1, y, 1);	Copy result to host
cudaFree(d_x);	Finalize



Slide 19|41

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 25141

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



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								
CUDA API Statistics:								
Time(%)	Total Time (ns)	Num Calls	Average	Minimum	Maximum	Name		
90.9	160,407,572	30	5,346,919.1	1,780	25,648,117	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_iterat	ion_352_gpu 🔻	🛛 👻 🖌 Add B	aseline 👻 Apply	Rules						Save as in	nage
urrent 1194 Time: 25	82 msecond Cycle	s: 28.191.300	Regs: 144 G	PU: A100-SXM	40GB	SM Frequenc	y: 1.09 cycle/nsecond	CC: 8.0	Process:	[31938] SON	(A 🕀	Θ
100 126 Time: 77.	17 msecond Cycle	s: 101.123.609	Regs: 144 G	PU: Tesla V100-	SXM2-16GB	SM Frequency	y: 1.31 cycle/nsecond	CC: 7.0	Process:	[30412] SON	1A	
3PU Speed Of Light 🛕										AI		ρ
n-level overview of the utili pretical maximum. High-lev	ization for compute a vel overview of the uti	nd memory reso lization for com	purces of the G pute and mem	PU. For each un ory resources of	t, the Speed the GPU pre	Of Light (SOL) r sented as a roof	eports the achieved pe line chart.	ercentage	of utilization	n with respec	t to the	
. SM [%]			20.5	1 (+160.09%)	Duration	[msecond]				25.82	(-66.	54%)
Memory [%]			55.8	2 (+1.26%)	Elapsed C	ycles [cycle]			28191300	(-72.)	12%)
L1/TEX Cache [%]			40.1	9 (+5.49%)	SM Active	Cycles [cyc	le]		27	784365.64	(-72.)	30%)
L2 Cache [%]			61.8	9 (+149.01%)	SM Freque	ncy [cycle/n	second]			1.09	(-16.)	67%)
DRAM [%]			31.5	2 (-42.82%)	DRAM Freq	uency [cycle	/nsecond]			1.21	(+38.)	21%)
				GPU U	tilization							
SM [%]		•										
lemory [%]						-						
0,0	10,0 20	,0	30,0	40,0	50,0	60,0	70,0	80	,0	90,0		100,0
				Sp	eed Of Light	[%]						
	SOL SM B	reakdown					SOL Memory	Breakdo	own			
SOL SM: Issue Active [%]			20	0.51 (+160.09%)	SOL L2:)	(bar2lts Cycles)	Active [%]			55.82	+124.58	1%)
SOL SM: Inst Executed [%]		20.46 (+160.10%)		SOL L2: T Tag Requests [%]		45.27 (+178.82%)			(%)		
SOL SM: Pipe Shared Cycl	les Active [%]		18	.36 (+165.69%)	SOL L1: N	L1tex2xbar Red	Cycles Active [%]			39.61	(+107.93	
SOL SM: Pipe Fp64 Cycles	s Active [%]		18	36 (+165.69%)	SOL L2: 1	Sectors [%]				38.80	(+71.23	1%)
SOL SM: Inst Executed Pig	pe Lsu [%]		10	.80 (+127.96%)	SOL L1: D	ata Pipe Lsu Wa	vefronts [%]			34.10	(+76.03	(%)
SOL SM: Pipe Alu Cycles A	Active [%]		10	0.37 (+161.23%)	SOL GPU	: Dram Through	put [%]			31.52	(-42.82	!%)
SOL SM: Inst Executed Pig	e Cbu Pred On Any [%]		8.13 (+71.98%)	SOL L1: L	su Writeback Ac	tive [%]			24.83	(+63.67	%)
SOL SM: Mio2rf Writeback	Active [%]		1	8.10 (+161.91%)	SOL L2: D	Sectors [%]				22.64	(+115.71	1%)
SOL SM: Mio Pa Read Cyc	les Active [%]		8	10 (+105.96%)	SOL L2: 0	Sectors Fill De	vice [%]			12.19	(-12.29	1%)

SOL SM. Pipe Shared Cycles Active [76]	10.30 (+100.03%)	SOL LI. M LITEXZXDar Red Cycles Active [76]	33.01 (+107.3376)
SOL SM: Pipe Fp64 Cycles Active [%]	18.36 (+165.69%)	SOL L2: T Sectors [%]	38.80 (+71.23%)
SOL SM: Inst Executed Pipe Lsu [%]	10.80 (+127.96%)	SOL L1: Data Pipe Lsu Wavefronts [%]	34.10 (+76.03%)
SOL SM: Pipe Alu 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%)

Programming GPUs Advanced Topics

Advanced Topics

So much more interesting things to show!

- Optimize memory transfers to reduce overhead
- Optimize applications for GPU architecture
- Drop-in BLAS acceleration with NVBLAS (\$LD_PRELOAD)
- Tensor Cores for Deep Learning
- Libraries, Abstractions: Kokkos, Alpaka, Futhark, HIP, SYCL, C++AMP, C++ pSTL, ...
- Use multiple GPUs
 - On one node
 - Across many nodes \rightarrow MPI
- ...
- Some of that: Addressed at dedicated training courses



Using GPUs on JSC Systems

Compiling

- CUDA Module: module load CUDA/11.7
 - Compile: nvcc file.cu
 - Example cuBLAS:g++ file.cpp -I\$CUDA_HOME/include -L\$CUDA_HOME/lib64
 -lcublas -lcudart
- **OpenACC • Module:** module load NVHPC/23.1
 - Compile: nvc++ -acc=gpu file.cpp
 - MPI CUDA-aware MPIs (with direct Device-Device transfers) ParaStationMPI module load ParaStationMPI/5.8.0-1 MPI-settings/CUDA OpenMPI module load OpenMPI/4.1.4 MPI-settings/CUDA



Running

 Dedicated GPU partitions JUWELS

```
--partition=gpus 46 nodes (Job limits: ≤1 d)
--partition=develgpus 10 nodes (Job limits: ≤2 h, ≤ 2 nodes)
JUWELS Booster
--partition=booster 926 nodes
--partition=develbooster 10 nodes (Job limits: ≤1 d, ≤ 2 nodes)
JURECA DC
```

--partition=dc-gpu 192 nodes

• Needed: Resource configuration with --gres=gpu:4

 \rightarrow See online documentation



Running

JUWELS Booster Topology

- JUWELS Booster: NPS-4 (in total: 8 NUMA Domains)
- Not all have GPU or HCA affinity!
- Network is structured into two levels: In-Cell and Inter-Cell (DragonFly+ network)







ightarrow Documentation:

apps.fz-juelich.de/jsc/hps/juwels/



Slide 38141

Example

- 16 tasks in total, running on 4 nodes
- Per node: 4 GPUs

```
#!/bin/bash -x
#SBATCH --nodes=4
#SBATCH --ntasks=16
#SBATCH --ntasks-per-node=4
#SBATCH --output=gpu-out.%j
#SBATCH --error=gpu-err.%j
#SBATCH --time=00:15:00
#SBATCH --partition=gpus
#SBATCH --gres=gpu:4
```

```
0 01
```

srun ./gpu-prog



Conclusion

- GPUs provide highly-parallel computing power
- We have many devices installed at JSC, ready to be used!



- GPUs provide highly-parallel computing power
- We have many devices installed at JSC, ready to be used!
- Training courses by JSC next year
- See online documentation and sc@fz-juelich.de



- GPUs provide highly-parallel computing power
- We have many devices installed at JSC, ready to be used!
- Training courses by JSC next year
- See online documentation and sc@fz-juelich.de
- Further consultation via our lab: NVIDIA Application Lab in Jülich; contact me!



- GPUs provide highly-parallel computing power
- We have many devices installed at JSC, ready to be used!
- Training courses by JSC next year
- See online documentation and sc@fz-juelich.de
- Further consultation via our lab: NVIDIA Application Lab in Jülich; contact meta





Appendix

Appendix Glossary References



Glossary I

- AMD Manufacturer of CPUs and GPUs. 52, 53, 54, 55, 56, 57, 88, 90 Ampere GPU architecture from NVIDIA (announced 2019). 4, 5, 6
 - API A programmatic interface to software by well-defined functions. Short for application programming interface. 52, 53, 54, 55, 56, 57
 - CUDA Computing platform for GPUs from NVIDIA. Provides, among others, CUDA C/C++. 2, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 77, 90
 - HIP GPU programming model by AMD to target their own and NVIDIA GPUs with one combined language. Short for Heterogeneous-compute Interface for Portability. 52, 53, 54, 55, 56, 57





Glossary II

- JSC Jülich Supercomputing Centre, the supercomputing institute of Forschungszentrum Jülich, Germany. 2, 82, 83, 84, 85, 89
- JURECA A multi-purpose supercomputer at JSC. 6
- JUWELS Jülich's new supercomputer, the successor of JUQUEEN. 3, 4, 5, 78
 - MPI The Message Passing Interface, a API definition for multi-node computing. 75, 77
 - NVIDIA US technology company creating GPUs. 3, 4, 5, 6, 26, 27, 28, 52, 53, 54, 55, 56, 57, 70, 82, 83, 84, 85, 88, 90
- OpenACC Directive-based programming, primarily for many-core machines. 46, 47, 48, 49, 50, 77



Glossary III

- OpenCL The Open Computing Language. Framework for writing code for heterogeneous architectures (CPU, GPU, DSP, FPGA). The alternative to CUDA. 52, 53, 54, 55, 56, 57
- OpenMP Directive-based programming, primarily for multi-threaded machines. 46, 47, 48, 49, 50
 - ROCm AMD software stack and platform to program AMD GPUs. Short for Radeon Open Compute (*Radeon* is the GPU product line of AMD). 52, 53, 54, 55, 56, 57
 - SAXPY Single-precision $A \times X + Y$. A simple code example of scaling a vector and adding an offset. 34, 67, 68
 - Tesla The GPU product line for general purpose computing computing of NVIDIA. 3





Glossary IV

- CPU Central Processing Unit. 3, 6, 10, 11, 12, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 34, 52, 53, 54, 55, 56, 57, 88, 90
- GPU Graphics Processing Unit. 2, 3, 4, 5, 6, 7, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 33, 37, 38, 39, 40, 41, 42, 45, 46, 47, 48, 51, 52, 53, 54, 55, 56, 57, 68, 69, 70, 74, 75, 76, 78, 79, 80, 82, 83, 84, 85, 88, 89, 90
- SIMD Single Instruction, Multiple Data. 19, 20, 21, 22, 23, 24, 25, 26, 27, 28
- SIMT Single Instruction, Multiple Threads. 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28
 - SM Streaming Multiprocessor. 19, 20, 21, 22, 23, 24, 25, 26, 27, 28
- SMT Simultaneous Multithreading. 19, 20, 21, 22, 23, 24, 25, 26, 27, 28



Slide 618



- [2] Karl Rupp. Pictures: CPU/GPU Performance Comparison. URL: https://www.karlrupp.net/2013/06/cpu-gpu-and-mic-hardwarecharacteristics-over-time/ (pages 8, 9).
- [6] Wes Breazell. Picture: Wizard. URL: https://thenounproject.com/wes13/collection/its-a-wizards-world/ (pages 37-41).



References: Images, Graphics I

- [1] Forschungszentrum Jülich GmbH (Ralf-Uwe Limbach). JUWELS Booster.
- [3] Mark Lee. Picture: kawasaki ninja. URL: https://www.flickr.com/photos/pochacco20/39030210/ (pages 10, 11).
- [4] Shearings Holidays. Picture: Shearings coach 636. URL: https://www.flickr.com/photos/shearings/13583388025/ (pages 10, 11).
- [5] Nvidia Corporation. Pictures: Volta GPU. Volta Architecture Whitepaper. URL: https://images.nvidia.com/content/volta-architecture/pdf/Volta-Architecture-Whitepaper-v1.0.pdf.

