



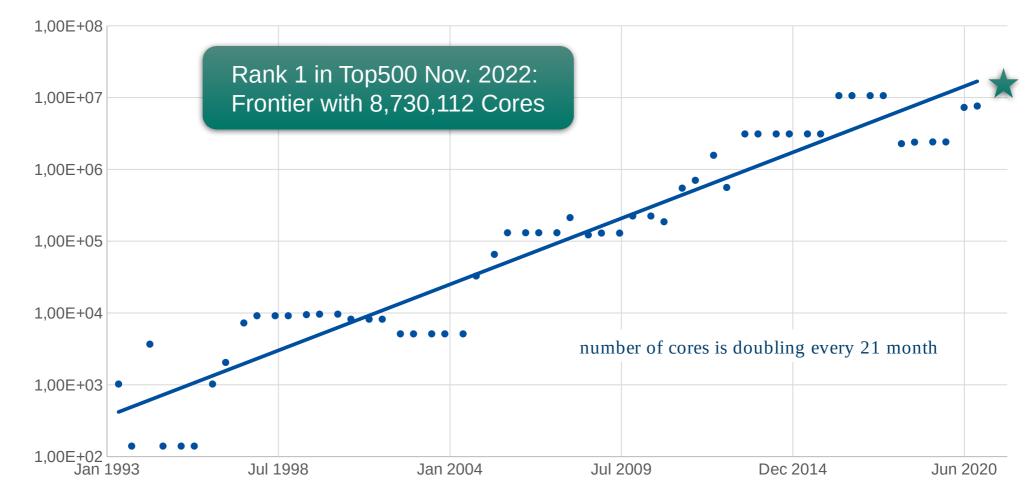
Betriebseinheit für technisch-wissenschaftliche Infrastruktur

Introduction to High-Performance Computing

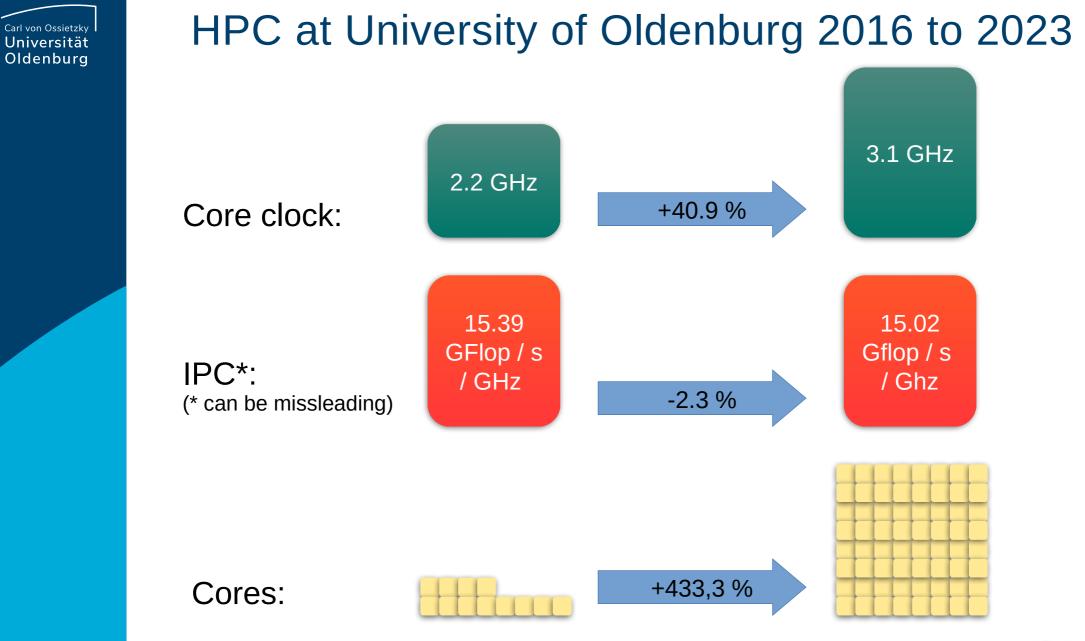
Session 04 Introduction to Parallel Computing Carl von Ossietzky Universität Oldenburg

Why Parallel Computing?

Number of Cores in the fastest Supercomputers





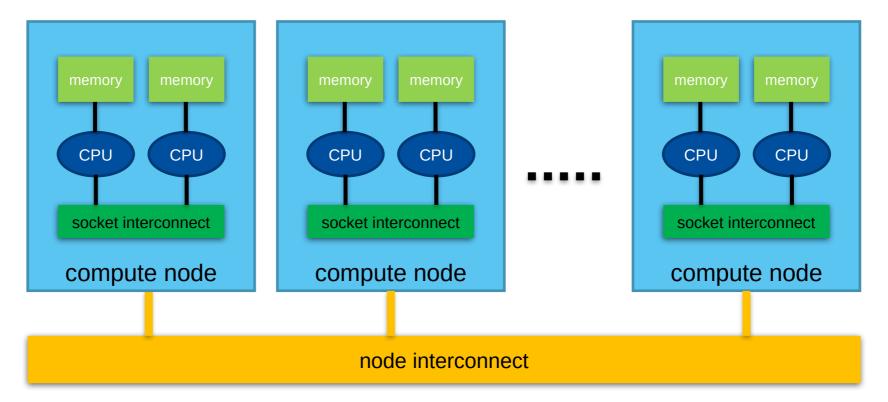






Parallel Hardware Architectures

 most modern HPC systems (e.g. CARL and EDDY) are clusters of SMP/ccNUMA nodes





Parallelization Considerations

- two major resources for computations
 - processor
 - memory
- parallelization means
 - distributing the work
 - distributing the data (on distributed memory machines)
 - synchronization of work
 - communication of data (on distributed memory machines)
- parallel programming models provide the methods to achieve the above goals

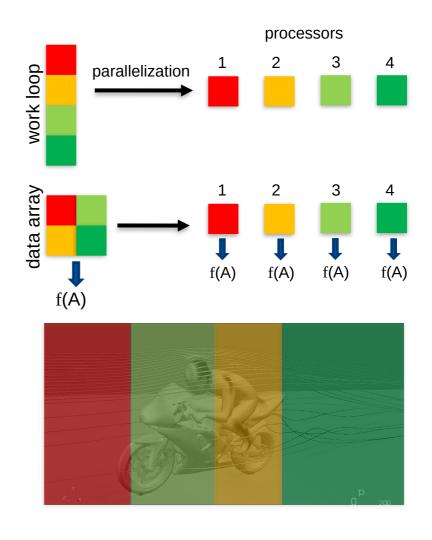




Distributing Work and Data

- Work decomposition •
 - based on loop decomposition
- Data decomposition •
 - all the work for a local chunk of the data is done by the local processor

- Domain decomposition ۲
 - work and data are distributed according to a higher model, e.g. reality





Parallel Programming Models

- two dominating programming models:
 - OpenMP: uses directives to define work decomposition
 - MPI: standardized message-passing interface
- other programming models
 - HPF (high-performance Fortran)
 - PGAS (Partitioned Global Address Space), e.g. Co-Array Fortran UPC (Unified Parallel C)
- programming models for compute devices
 - CUDA
 - OpenCL

. . .

- OpenACC



Parallel Programming Models

- two dominating programming models:
 - OpenMP: uses directives to define work decomposition
 - MPI: standardized message-passing interface
- other programming models
 - HPF (high-performance Fortran)
 - PGAS (Partitioned Global Address Space), e.g. Co-Array Fortran UPC (Unified Parallel C)
- programming models for compute devices
 - CUDA
 - OpenCL

. . .

- OpenACC





History of MPI

- MPI is a standard with the prime goals
 - to provide a message-passing interface
 - to provide source-code portability
 - to allow efficient implementations
- MPI exists for more than 25 years
 - MPI-1.0 was released in June, 1994
 - MPI-2.0 was released in July, 1997 and provided additional functionality
 - MPI-3.0 was released in October, 2012 and was developed for better platform and application support (in particular clusters of SMP nodes)
 - MPI-4.0 was released June 2021.

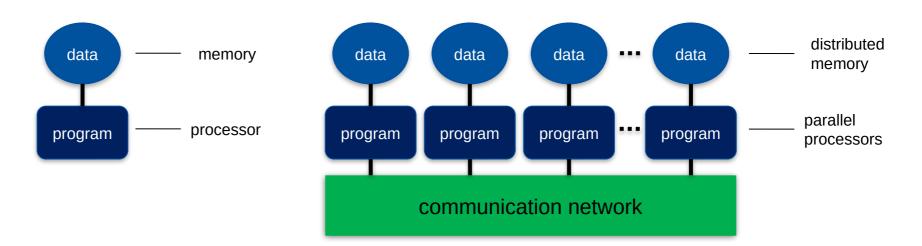
http://mpi-forum.org/docs/





A Message-Passing Interface

• sequential program vs. message-passing program

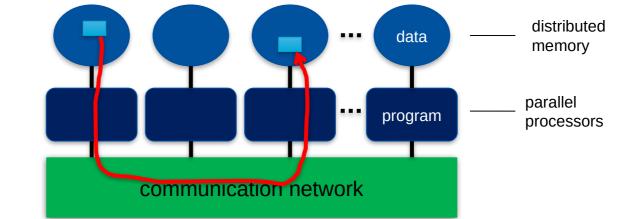


- message-passing programming paradigm:
 - each processor runs a (sub)program, typically the same (SPMD)
 - variables of subprograms have the same name but different (distributed) data
 - communication by special library routines [] message passing



Message Passing

- messages are passed through the communication network
- messages require the following information:
 - sending and receiving process
 - data location
 - data type
 - data size



- in order to use the message-passing interface the program must be
 - connected to the MPI library (at compile time)
 - started with the MPI startup tool (mpirun or mpiexec)
 - at runtime MPI is initialized with special library calls (**MPI_Init()**)



Example MPI Program in C/C++

#include <mpi.h>

```
using namespace std;
```

other examples for MPI programs are shown in the videos

```
int main(int argc, char *argv[]) {
    // initialization of MPI
    MPI_Init(&argc, &argv);
```

```
// do some computation in parallel
int partial_result = some_computation();
int global_result = 0;
```

```
// finalization of MPI
MPI_Finalize();
```



Parallel Programming Models

- two dominating programming models:
 - OpenMP: uses directives to define work decomposition
 - MPI: standardized message-passing interface
- other programming models
 - HPF (high-performance Fortran)
 - PGAS (Partitioned Global Address Space), e.g. Co-Array Fortran UPC (Unified Parallel C)
- programming models for compute devices
 - CUDA
 - OpenCL

. . .

- OpenACC



GPUs in HPC

- GPUs appeared in the early 2000s in HPC
 - good cost/performance ratio due to mass production for gaming
- initially consumer-grade graphic cards were used
 - limited general-purpose computing
 - algorithms had to mimic graphics display
- today special GPUs are used in HPC
 - no display port
 - run real algorithms

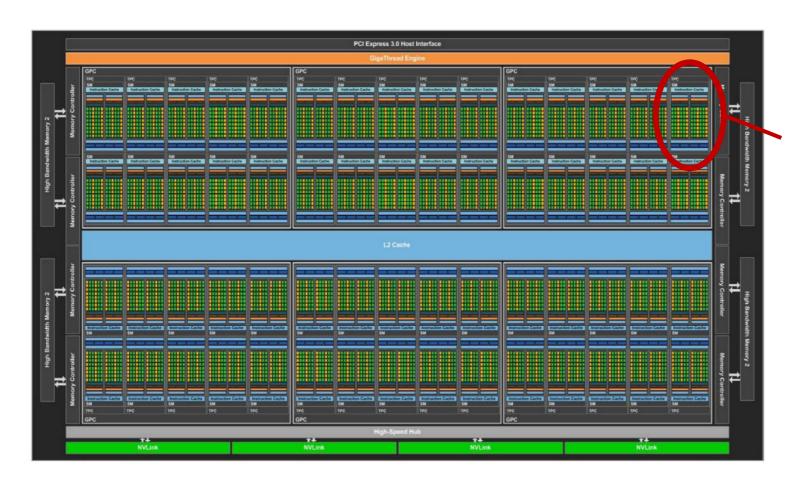






Design of GPUs

Example NVIDIA P100 •



- organized in Graphics (GPCs) and Texture (TPCs) **Processing Clusters**
- 60 streaming multiprocessor (SM)
 - basic compute \succ resource
 - \geq each SM has 64 CUDA cores
- 4 MB L2 Cache
 - accessed by 8 memory \geqslant controllers



Design of GPUs (GP100)

							Instructio	on Cache											
Instruction Buffer									Instruction Buffer										
Warp Scheduler								Warp Scheduler											
	Dispato			Dispatch Unit					Dispato		Dispatch Unit								
		Regist	er File (3	32,768 x	32-bit)					Regist	er File (3	32,768 x	32-bit)						
Core	Core	DP Unit	Core	Core	DP Unit	LD/ST	SFU	Core	Core	DP Unit	Core	Core	DP Unit	LD/ST	SFU				
Core	Core	DP Unit	Core	Core	DP Unit	LD/ST	SFU	Core	Core	DP Unit	Core	Core	DP Unit	LD/ST	SFU				
Core	Core	DP Unit	Core	Core	DP Unit	LD/ST	SFU	Core	Core	DP Unit	Core	Core	DP Unit	LD/ST	SFU				
Core	Core	DP Unit	Core	Core	DP Unit	LD/ST	SFU	Core	Core	DP Unit	Core	Core	DP Unit	LD/ST	SFL				
Core	Core	DP Unit	Core	Core	DP Unit	LD/ST	SFU	Core	Core	DP Unit	Core	Core	DP Unit	LD/ST	SFL				
Core	Core	DP Unit	Core	Core	DP Unit	LD/ST	SFU	Core	Core	DP Unit	Core	Core	DP Unit	LD/ST	SFL				
Core	Core	DP Unit	Core	Core	DP Unit	LD/ST	SFU	Core	Core	DP Unit	Core	Core	DP Unit	LD/ST	SFU				
Core	Core	DP Unit	Core	Core	DP Unit	LD/ST	SFU	Core	Core	DP Unit	Core	Core	DP Unit	LD/ST	SFU				
							/ Texture /	L1 Cache											
Tex						ex			Te	∋x	Tex								

- the SM is divided into two blocks
 - each has 32 SP core and 16 DP cores
 - 8 Special Function Units (SFUs)
- 64kB of shared memory



Carl von Ossietzky Universität Oldenburg

Design of GPUs (GH100 – New HPC Cluster)

M															
					L1 Instr	uctio	on Cácl	18				_			
					-		1								
		_	Instruct	_						L0 In	struct	ion C	ache		
	v	and the second second	Contractor of the local division of the loca		read/clk)						eduler	-			
		Dispati	ch Unit	(32 thre	ead/clk)				Dis	patch	Unit	32 th	read/	cik)	
	R	legiste	r File (1	16,384	x 32-bit)				Regis	ster f	File (1	6,384	1 x 3 2	2-bit)	
INT32	FP32	FP32	FP6	4			INT32	FP32	FP3	2	FP6	4	-	_	
INT32		FP32	FP6				INT32	FP32			FP6				
INT32		FP32	FP6				INT32	FP32			FP6				
INT32		FP32	FP6				INT32	FP32			FP6				
INT32 INT32		FP32 FP32	FP6 FP6				INT32 INT32	FP32 FP32			FP6 FP6				
INT32 INT32		FP32 FP32	FP6				INT32	FP32		_	FP6				
INT32		FP32	FP6		TENSOR CORE		INT32	FP32			FP6		T	INSC	RCOR
INT32		FP32	FP6		4th GENERATION		INT32	FP32			FP6				ERATIO
INT32		FP32	FP6		DENERGATION		INT32	FP32			FP6				
INT32	FP32	FP32	FP6	4			INT32	FP32	FP3		FP6				
INT32		FP32	FP6				INT32	FP32			FP6				
INT32		FP32	FP6				INT32	FP32			FP6				
INT32		FP32	FPG				INT32	FP32			FP6				
INT32 INT32	FP32 FP32	FP32	FP6 FP6				INT32 INT32	FP32 FP32			FP6 FP6				
							-						-	l un	1
LD/ ST	LDV LD ST S		LD/ ST	LD/ ST	ST ST SFU		LD/ ST	LD/ ST	LD/ ST	LD/ ST	LD/ ST	LD/ ST	LD/ ST	LD/ ST	SFU
		Varp Sc		r (32 th	read/clk)				Warp	Sch	struct eduler	(32 ti	hread		
	10 20	Varp Sc Dispati	hedule ch Unit	r (32 th (32 thre	read/clk) ead/clk)				Warp Disp	o Schi patch	eduler Unit ((32 ti (32 th	hread read/	clk)	
INT22	R	Varp Sc Dispati tegiste	heduler ch Unit r File (1	r (32 th (32 thre 16,384	read/clk)				Warp Disp Regis	ster f	eduler Unit (File (1	(32 th 32 th 6,384	hread read/	clk)	
	R FP32	Varp So Dispati tegiste FP32	heduler ch Unit r File (1 FP6	r (32 th (32 thre 16,384	read/clk) ead/clk)		INT32	FP32	Warp Disp Regis	o Schi patch ster F	eduler Unit (File (1 FP6	(32 th 32 th 6,384	hread read/	clk)	
INT32	R FP32 FP32	Varp Sc Dispati tegiste	heduler ch Unit r File (1	r (32 th (32 thre 16,384	read/clk) ead/clk)		INT32 INT32 INT32		Warp Disp Regis	Schi patch ster F	eduler Unit (File (1	(32 th 32 th 6,384 4 4	hread read/	clk)	
INT32 INT32 INT32	R FP32 FP32 FP32 FP32 FP32	Varp So Dispate tegiste FP32 FP32 FP32 FP32 FP32	hedule ch Unit r File (1 FP6 FP6 FP6 FP6	r (32 th (32 thro 16,384	read/clk) ead/clk)		INT32 INT32 INT32	FP32 FP32 FP32 FP32	Warp Disp Regis : FP3: : FP3: : FP3: : FP3:	2 2 2 2 2	eduler I Unit (File (1 FP6 FP6 FP6	(32 th 32 th 6,384 4 4 4 4	hread read/	clk)	
INT32 INT32 INT32 INT32	R FP32 FP32 FP32 FP32 FP32 FP32	Varp Sc Dispate tegiste FP32 FP32 FP32 FP32 FP32 FP32	hedule ch Unit r File (1 FP6 FP6 FP6 FP6 FP6	r (32 th (32 thro 16,384 14 14 14 14 14	read/clk) ead/clk)		INT32 INT32 INT32 INT32	FP32 FP32 FP32 FP32 FP32	Warp Disp Regis : FP3: : FP3: : FP3: : FP3: : FP3:	Sch patch ster F	eduler Unit (File (1 FP6 FP6 FP6 FP6	(32 th 32 th 6,384 4 4 4 4 4 4	hread read/	clk)	
INT32 INT32 INT32 INT32 INT32	R FP32 FP32 FP32 FP32 FP32 FP32 FP32	Varp Sc Dispati tegiste FP32 FP32 FP32 FP32 FP32 FP32 FP32 FP32	hedule ch Unit r File (1 FP6 FP6 FP6 FP6 FP6 FP6 FP6	r (32 thr (32 thre (32 thre (4 thre))))))))))))))))))))))))))))))))))))	read/clk) ead/clk)		INT32 INT32 INT32 INT32 INT32	FP32 FP32 FP32 FP32 FP32 FP32	Warp Disp Regis FP3: FP3: FP3: FP3: FP3: FP3:	Sch patch ster F 2 2 2 2 2 2 2 2 2 2 2 2	eduler Unit (File (1 FP6 FP6 FP6 FP6 FP6	(32 th (32 th) 6,384 4 4 4 4 4 4 4 4	hread read/	clk)	
INT32 INT32 INT32 INT32 INT32 INT32	R FP32 FP32 FP32 FP32 FP32 FP32 FP32 FP32	Varp Sc Dispati tegiste FP32 FP32 FP32 FP32 FP32 FP32 FP32 FP32	hedule ch Unit r File (1 FP6 FP6 FP6 FP6 FP6 FP6 FP6 FP6	r (32 thr (32 thro 16,384 14 14 14 14 14 14 14 14	read/clk) ead/clk) x 32-bit)		INT32 INT32 INT32 INT32 INT32 INT32	FP32 FP32 FP32 FP32 FP32 FP32 FP32	Warp Disp Regis : FP3: : FP3: : FP3: : FP3: : FP3: : FP3: : FP3: : FP3:	Schupatch ster F 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2	eduler I Unit (File (1 FP6 FP6 FP6 FP6 FP6 FP6 FP6	(32 th (32 th) 6,384 4 4 4 4 4 4 4 4 4 4	hread/ read/ 4 x 3:	clk) 2-bit)	
INT32 INT32 INT32 INT32 INT32 INT32 INT32	R FP32 FP32 FP32 FP32 FP32 FP32 FP32 FP32	Varp So Dispate Registe FP32 FP32 FP32 FP32 FP32 FP32 FP32 FP32	n File (1 FP6 FP6 FP6 FP6 FP6 FP6 FP6 FP6 FP6	r (32 thr (32 thro 16,384 14 14 14 14 14 14 14 14	read/clk) ead/clk) x 32-bit) TENSOR CORE		INT32 INT32 INT32 INT32 INT32 INT32 INT32	FP32 FP32 FP32 FP32 FP32 FP32 FP32 FP32	Warp Disp Regis : FP3: : FP3: : FP3: : FP3: : FP3: : FP3: : FP3: : FP3: : FP3: : FP3:	2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2	eduler I Unit (File (1 FP6 FP6 FP6 FP6 FP6 FP6 FP6 FP6	(32 th (32 th) (32 th)	hread/ i x 3;	clk) 2-bit) ENSC	
INT32 INT32 INT32 INT32 INT32 INT32 INT32 INT32	R FP32 FP32 FP32 FP32 FP32 FP32 FP32 FP32	Varp Sc Dispati tegiste FP32 FP32 FP32 FP32 FP32 FP32 FP32 FP32	hedule ch Unit r File (1 FP6 FP6 FP6 FP6 FP6 FP6 FP6 FP6	r (32 th (32 thra (32 thra 16,384 (4) (4) (4) (4) (4) (4) (4) (4) (4) (4	read/clk) ead/clk) x 32-bit)		INT32 INT32 INT32 INT32 INT32 INT32	FP32 FP32 FP32 FP32 FP32 FP32 FP32	Warp Disp Regis PP3 FP3 FP3 FP3 FP3 FP3 FP3 FP3 FP3 FP3	ster F	eduler I Unit (File (1 FP6 FP6 FP6 FP6 FP6 FP6 FP6	(32 th (32 th) (32 th)	hread/ i x 3;	clk) 2-bit) ENSC	
INT32 INT32 INT32 INT32 INT32 INT32 INT32 INT32 INT32 INT32	R FP32 FP32 FP32 FP32 FP32 FP32 FP32 FP32	Varp So Dispati Registe FP32 FP32 FP32 FP32 FP32 FP32 FP32 FP32	r File (1 FP6 FP6 FP6 FP6 FP6 FP6 FP6 FP6 FP6 FP6	r (32 th (32 thra (32 thra 16,384 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4	read/clk) ead/clk) x 32-bit) TENSOR CORE		INT32 INT32 INT32 INT32 INT32 INT32 INT32 INT32 INT32 INT32	FP32 FP32 FP32 FP32 FP32 FP32 FP32 FP32	Warp Disp Regis 2 FP3 2 FP3	ster F	eduier Unit (File (1 FP6 FP6 FP6 FP6 FP6 FP6 FP6	(32 th (32 th) (32 th)	hread/ i x 3;	clk) 2-bit) ENSC	
INT32 INT32 INT32 INT32 INT32 INT32 INT32 INT32 INT32 INT32 INT32	R FP32 FP32 FP32 FP32 FP32 FP32 FP32 FP32	Warp So Dispati Registe FP32 FP32 FP32 FP32 FP32 FP32 FP32 FP32	heduler ch Unit r File (1 FP6 FP6 FP6 FP6 FP6 FP6 FP6 FP6 FP6 FP6	r (32 th (32 thra (32 thra (32 thra (32 thra (4 (4 (4 (4 (4 (4 (4 (4 (4 (4 (4 (4 (4	read/clk) ead/clk) x 32-bit) TENSOR CORE		INT32 INT32 INT32 INT32 INT32 INT32 INT32 INT32 INT32 INT32 INT32	FP32 FP32 FP32 FP32 FP32 FP32 FP32 FP32	Warp Disp Regis (FP3) (FP3)	Schi patch ster I 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2	eduler Unit (File (1 FP6 FP6 FP6 FP6 FP6 FP6 FP6 FP6	(32 th (32 th (33 th) (33 th) (33 th) (34 th) (35 th) (35 th) (35 th) (36 th) (37 th) (37 th) (38 th)(hread/ i x 3;	clk) 2-bit) ENSC	
INT32 INT32 INT32 INT32 INT32 INT32 INT32 INT32 INT32 INT32 INT32 INT32	R FP32 FP32 FP32 FP32 FP32 FP32 FP32 FP32	Varp Sc Dispati Registe FP32 FP32 FP32 FP32 FP32 FP32 FP32 FP32	hedule ch Unit r File (1 FP6 FP6 FP6 FP6 FP6 FP6 FP6 FP6 FP6 FP6	r (32 th (32 thru (32 thru 16,384 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4	read/clk) ead/clk) x 32-bit) TENSOR CORE		INT32 INT32 INT32 INT32 INT32 INT32 INT32 INT32 INT32 INT32 INT32 INT32	FP32 FP32 FP32 FP32 FP32 FP32 FP32 FP32	Warp Disp Regis FP3: FP3: FP3: FP3: FP3: FP3: FP3: FP3:	schi patch ster P 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2	eduler Unit (File (1 FP6 FP6 FP6 FP6 FP6 FP6 FP6 FP6 FP6 FP6	(32 th 6,38 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4	hread/ i x 3;	clk) 2-bit) ENSC	
INT32 INT32 INT32 INT32 INT32 INT32 INT32 INT32 INT32 INT32 INT32 INT32	R FP32 FP32 FP32 FP32 FP32 FP32 FP32 FP32	Varp Sc Dispati Registe FP32 FP32 FP32 FP32 FP32 FP32 FP32 FP32	hedule ch Unit r File (1 FP6 FP6 FP6 FP6 FP6 FP6 FP6 FP6 FP6 FP6	r (32 th (32 thr (32 thr 16,384 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4	read/clk) ead/clk) x 32-bit) TENSOR CORE		INT32 INT32 INT32 INT32 INT32 INT32 INT32 INT32 INT32 INT32 INT32 INT32	FP32 FP32 FP32 FP32 FP32 FP32 FP32 FP32	Warp Disp Regis PP3: PP3: PP3: PP3: PP3: PP3: PP3: PP3	schi patch ster P 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2	eduler I Unit (File (1 FP6 FP6 FP6 FP6 FP6 FP6 FP6 FP6 FP6 FP6	4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4	hread/ i x 3;	clk) 2-bit) ENSC	
INT32 INT32 INT32 INT32 INT32 INT32 INT32 INT32 INT32 INT32 INT32 INT32 INT32	R FP32 FP32 FP32 FP32 FP32 FP32 FP32 FP32	Varp Sc Dispati egiste FP32 FP32 FP32 FP32 FP32 FP32 FP32 FP32	hedule ch Unit r File (1 FP6 FP6 FP6 FP6 FP6 FP6 FP6 FP6 FP6 FP6	r (32 th (32 thru (32 thru 16,384 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4	read/clk) ead/clk) x 32-bit) TENSOR CORE		INT32 INT32 INT32 INT32 INT32 INT32 INT32 INT32 INT32 INT32 INT32 INT32 INT32	FP32 FP32 FP32 FP32 FP32 FP32 FP32 FP32	Warp Disp Regis : FP3; : FP3;	ster f 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2	eduler I Unit (File (1 FP6 FP6 FP6 FP6 FP6 FP6 FP6 FP6 FP6 FP6	(32 th) (32 th) (32 th) (33 th) (33 th) (34 th	hread/ i x 3;	clk) 2-bit) ENSC	
INT32 INT32 INT32 INT32 INT32 INT32 INT32 INT32 INT32 INT32 INT32 INT32 INT32 INT32	R FP32 FP32 FP32 FP32 FP32 FP32 FP32 FP32	Varp Sc Dispati egiste FP32 FP32 FP32 FP32 FP32 FP32 FP32 FP32	hedule ch Unit r File (1 FP6 FP6 FP6 FP6 FP6 FP6 FP6 FP6 FP6 FP6	r (32 thr (32 thr (32 thr (32 thr (32 thr (32 thr (44 (44 (44 (44 (44 (44 (44 (44))))))))	read/clk) ead/clk) x 32-bit) TENSOR CORE 4 th GENERATION		INT32 INT32 INT32 INT32 INT32 INT32 INT32 INT32 INT32 INT32 INT32 INT32 INT32 INT32 INT32	FP32 FP32 FP32 FP32 FP32 FP32 FP32 FP32	Warp Disp Disp Regis *	2 Schr patch ster F 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2	eduler File (1 FP6 FP6 FP6 FP6 FP6 FP6 FP6 FP6	(32 th) (32 th) (32 th) (33 th) (33 th) (34 th	hread fread/ 4 x 3 TI 4 th	ENSC GEN	ERATIO
INT32 INT32 INT32 INT32 INT32 INT32 INT32 INT32 INT32 INT32 INT32 INT32 INT32	R FP32 FP32 FP32 FP32 FP32 FP32 FP32 FP32	Varp Sc Dispate Registe FP32 FP32 FP32 FP32 FP32 FP32 FP32 FP32	hedule ch Unit r File (1 FP6 FP6 FP6 FP6 FP6 FP6 FP6 FP6 FP6 FP6	r (32 th (32 thru (32 thru 16,384 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4	read/clk) ead/clk) x 32-bit) TENSOR CORE		INT32 INT32 INT32 INT32 INT32 INT32 INT32 INT32 INT32 INT32 INT32 INT32 INT32	FP32 FP32 FP32 FP32 FP32 FP32 FP32 FP32	Warp Disp Regis : FP3: : FP3:	ster f 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2	eduler I Unit (File (1 FP6 FP6 FP6 FP6 FP6 FP6 FP6 FP6 FP6 FP6	(32 th) (32 th) (32 th) (33 th) (33 th) (34 th	hread/ i x 3;	clk) 2-bit) ENSC	
INT32 INT32 INT32 INT32 INT32 INT32 INT32 INT32 INT32 INT32 INT32 INT32 INT32 INT32 INT32 INT32	R FP32 FP32 FP32 FP32 FP32 FP32 FP32 FP32	Varp Sc Dispate Registe FP32 FP32 FP32 FP32 FP32 FP32 FP32 FP32	heduled	r (32 thr (32 thr (32 thr (32 thr (33 thr (34 the (34 the (34 the (34 the))))))))))))))))))))))))))))))))))))	read/clk) sad/clk) x 32-bit) TENSOR CORE 4 th GENERATION LOY LOY SFU TENSOR MEM	_	INT32 INT32	FP32 FP32 FP32 FP32 FP32 FP32 FP32 FP32	Warp Disg Regis PP3: PP3: PP3: PP3: PP3: PP3: PP3: PP3	Schur patch ster f 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2	eduler File (1 FP6 FP6 FP6 FP6 FP6 FP6 FP6 FP6 FP6 FP6	(32 th (32 th 6,384 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4	tread t x 3; t x 3;	cik) 2-bit) GEN	ERATIO
INT32 INT32 INT32 INT32 INT32 INT32 INT32 INT32 INT32 INT32 INT32 INT32 INT32 INT32 INT32 INT32	R FP32 FP32 FP32 FP32 FP32 FP32 FP32 FP32	Varp Sc Dispate Registe FP32 FP32 FP32 FP32 FP32 FP32 FP32 FP32	heduled	r (32 thr (32 thr (32 thr (32 thr (33 thr (34 the (34 the (34 the (34 the))))))))))))))))))))))))))))))))))))	read/clk) sad/clk) x 32-bit) TENSOR CORE 4 th GENERATION	_	INT32 INT32	FP32 FP32 FP32 FP32 FP32 FP32 FP32 FP32	Warp Disg Regis PP3: PP3: PP3: PP3: PP3: PP3: PP3: PP3	Schur patch ster f 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2	eduler File (1 FP6 FP6 FP6 FP6 FP6 FP6 FP6 FP6 FP6 FP6	(32 th (32 th 6,384 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4	tread t x 3; t x 3;	cik) 2-bit) GEN	ERATIO
	R FP32 FP32 FP32 FP32 FP32 FP32 FP32 FP32	Varp Sc Dispatz degiste FP32 FP32<	heduled	r (32 thr (32 thr (32 thr (32 thr (33 thr (34 the (34 the (34 the (34 the))))))))))))))))))))))))))))))))))))	read/clk) sad/clk) x 32-bit) TENSOR CORE 4 th GENERATION LOY LOY SFU TENSOR MEM	_	INT32 INT32	FP32 FP32 FP32 FP32 FP32 FP32 FP32 FP32	Warp Disg Regis PP3: PP3: PP3: PP3: PP3: PP3: PP3: PP3	Schur patch ster f 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2	eduler File (1 FP6 FP6 FP6 FP6 FP6 FP6 FP6 FP6 FP6 FP6	(32 th (32 th 6,384 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4	tread t x 3; t x 3;	cik) 2-bit) GEN	SFU

- the SM is divided into 4 blocks, each block has
 - 32 FP32 and 16 FP64 units
 - 16 INT32 units
 - 1 Tensor core
 - 4 SFUs
- 256kB of shared L1-cache



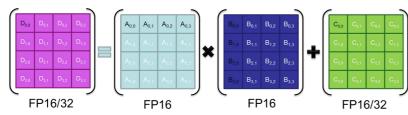
Carl von Ossietzky Universität Oldenburg

CPU Core Control Logic ALU ALU ALU ALU Cache **CUDA** Core **Dispatch Port** Operand Collector **FP Unit INT Unit** Result Queue

• CPU cores perform tasks independently

- control logic is included in the core
- designed for serial computation
- modern CPU cores include small SIMD units (e.g. AVX)
- CUDA cores have no control logic
 - control logic is in SM only
 - all cores must perform same instruction
 - SM is a SIMD unit

CPU Core vs. CUDA Core vs. Tensor Core



• Tensor cores operate on 4x4 matrices or larger



Hybrid Parallel Programming Models

- parallel programming models can be combined in a hybrid approach for better performance or special needs
- common approach is MPI + OpenMP to reduce the number of MPI process (communication overhead)
 - example: use MPI to start a parallel program on multiple dual-socket nodes, one MPI process per socket and OpenMP to utilize the available cores per socket
- MPI + CUDA/OpenACC to use GPUs across multiple nodes or OpenMP + CUDA for multiple GPUs in a single node
 - NVLink (or similar) may allow you to address multiple GPUs within a node as a single device





Slurm Options for Parallel Computing



Slurm Options for Parallel Computing

a Slurm job can request to run multiple tasks

- the option --ntasks or a combination of --nodes and
 - **--tasks-per-node** can be used to set the number of tasks
- tasks can be executed using with **srun** (but this is not a typical use case)
- a process in a parallel MPI programs corresponds to a task and **mpirun** is aware of the requested number of tasks
- a Slurm job can also request multiple (logical) cores per task
- the option **--cpus-per-task** can be used for that
- a Slurm **cpu** can be a physical core or a logical (hyper)thread



Variables in Job Scripts

if you have a parallel application and you have requested multiple tasks and/or CPUs you can use corresponding variables in your job script

- SLURM_JOB_NODELIST:

List of nodes allocated to the job

- **SLURM_JOB_NUM_NODES**: Total number of nodes in the

job's resource allocation

- SLURM_NTASKS: Number of tasks requested
- SLURM_NTASKS_PER_NODE: Number of tasks requested per node
- SLURM_CPUS_PER_TASK:

Number of CPUs requested per task



Slurm Options for GPU Computing to use the GPU nodes your job script should include selection of an appropriate partition **#SBATCH** --partition mpcg.p **# or mpcb.p or cfdg.p** request for one or two GPUs (Generic RESource in Slurm) **#SBATCH** --gres=gpu:1 # 1 or 2 gpus you also need to load the CUDA Toolkit module load CUDA # add version if needed

note that the driver (and as a result libcuda.so) is only available on the GPU nodes

