HUNGARIAN ACADEMY OF SCIENCES
WIGNER RESEARCH CENTRE FOR PHYSICS

5th GPU WORKSHOP - THE FUTURE OF MANY-CORE COMPUTING IN SCIENCE 2015, Budapest, May 20-21

Methodology for heterogeneous programming on OpenCL and its applications

Olga Sudareva, olsudareva@gmail.ru
Pavel Bogdanov, bogdanov@niisi.msk.ru

Heterogeneous parallel programming group
Institute of System Research Russian Academy of Sciences
http://www.niisi.ru

What it's about?

- Hybrid System Architecture (HSA)
- Heterogeneous programming

What and why?

- The methodology of hybrid cluster programming
- Set of distributed tasks for verification with world leaders and cross-optimization

Top 500 — November 2014

	Rank	Site	Computer	Total Cores	Accelerator/ Co-Processor Cores	Rmax (TFlop/s)	Rpeak (TFlop/s)	Eff(%)	Power (kW)
Hybrid	1	National University of Defense Technology China	**Tianhe-2 (MilkyWay-2)** — TH-IVB-FEP Cluster, Intel Xeon E5-2692 12C 2.200GHz, TH Express-2, Intel Xeon Phi 31S1P, NUDT.	3120000	2736000	33862700	54902400	61,68	17808
Hybrid	2	DOE/SC/Oak Ridge National Laboratory United States	**Titan** — Cray XK7 , Opteron 6274 16C 2.200GHz, Cray Gemini interconnect, NVIDIA K20x, Cray Inc.	560640	261632	17590000	27112550	64,88	8209
Classic	3	DOE/NNSA/LLNL United States	**Sequoia** — BlueGene/Q, Power BQC 16C 1.60 GHz, Custom, IBM	1572864	0	16324751	20132659,2	81,09	7890
Classic	4	RIKEN Advanced Institute for Computational Science (AICS) Japan	**K computer**, SPARC64 VIIIfx 2.0GHz, Tofu interconnect Fujitsy	705024	0	10510000	11280384	93,17	12660
Classic	5	DOE/SC/Argonne National Laboratory United States	**Mira** — BlueGene/Q, Power BQC 16C 1.60GHz, Custom, IBM	786432	0	8162376	10066330	81,09	3945
Hybrid	6	Swiss National Supercomputer Center (CSCS) Switzerland	**Piz Daint** — Cray XC30, Xeon E5-2670 8C 2.60GHz, Aries interconnect, NVIDIA K20X, Cray Inc.	115984	0	627100	7788900	80,6	2325
Hybrid	7	Texas Advanced Computing Center/Univ. of Texas United States	**Stampede** — PowerEdge C8220, Xeon E5-2680 8C 2.700GHz, Infiniband FDR, Intel Xeon Phi, Dell	462462	366366	5168110	8520111	60,6	4510
Classic	8	Forschungszentrum Juelich (FZJ) Germany	**JUQUEEN** — BlueGene/Q, Power BQC 16C 1.600GHz, Custom Interconnect, IBM	458752	0	5008857	5872025	85,3	2301
Classic	9	DOE/NNSA/LLNL United States	**Vulcan** — BlueGene/Q, Power BQC 16C 1.60 GHz, Custom, IBM	393216	0	4293306	5033165	85,3	1972
Hybrid	10	Government United States	**Cray CS Storm** — Xeon E5-2660v2 10C 2.2 GHz, NVIDIA K40, Infiniband, Cray Inc.	72800	0	3577000	6131800	58,2	1499

2008 – (**#1**) **RoadRunner**
1.4 Pflops
(PowerXCell 8i 9C)

2009 – (**#5**) **Tianhe-1**
1.2 PFlops
(!!! ATI Radeon HD 4870 !!!)
Who is mister CUDA???

2010 – (**#2**) **Nebulae**
3 Pflops
(Nvidia Tesla C2050)

2012 – (**#7**) **Stampede**
4 Pflops
(Intel Xeon Phi)

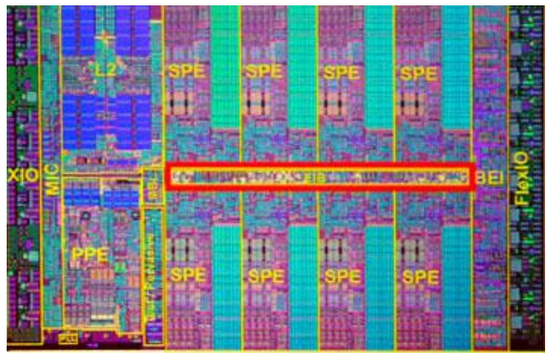
Green 500 — November 2014

	Rank	MFLOPS/W	Site	Computer	Total Power (kW)
Hybrid	1	5271.81	GSI Helmholtz Center	**L-CSC -** ASUS ESC4000 FDR/G2S. Intel Xeon E5-2690v2 10C 3GHz. Infiniband FDR. AMD FirePro S9150	57.15
Classic	2	4945.63	High Energy Accelerator Research Organization /KEK	**Suiren - ExaScaler 32U256SC Cluster.** Intel Xeon E5-2660v2 10C 2.2GHz. Infiniband FDR. PEZY-SC	37.83
Hybrid	3	4447.58	GSIC Center. Tokyo Institute of Technology	**TSUBAME-KFC - LX 1U-4GPU/104Re-1G Cluster.** Intel Xeon E5-2620v2 6C 2.100GHz. Infiniband FDR. NVIDIA K20x	35.39
Hybrid	4	3926.73	Cray Inc.	**Storm1 - Cray CS-Storm.** Intel Xeon E5-2660v2 10C 2.2GHz. Infiniband FDR. Nvidia K40m	44.54
Hybrid	5	3631.7	Cambridge University	**Wilkes - Dell T620 Cluster.** Intel Xeon E5-2630v2 6C 2.600GHz. Infiniband FDR. NVIDIA K20	52.62
Hybrid	6	3543.32	Financial Institution	**iDataPlex DX360M4.** Intel Xeon E5-2680v2 10C 2.800GHz. Infiniband. NVIDIA K20x	54.60
Hybrid	7	3517.84	Center for Computational Sciences. University of Tsukuba	**HA-PACS TCA - Cray CS300 Cluster.** Intel Xeon E5-2680v2 10C 2.800GHz. Infiniband QDR. NVIDIA K20x	78.77
Hybrid	8	3459.46	SURFsara	**Cartesius Accelerator Island - Bullx B515 cluster.** Intel Xeon E5-2450v2 8C 2.5GHz. InfiniBand 4× FDR. Nvidia K40m	44.40
Hybrid	9	3185.91	Swiss National Supercomputing Centre (CSCS	**Piz Daint - Cray XC30.** Xeon E5-2670 8C 2.600GHz. Aries interconnect . NVIDIA K20x	1753.66
Hybrid	10	3131.06	ROMEO HPC Center - Champagne-Ardenne	**Romeo - Bull R421-E3 Cluster.** Intel Xeon E5-2650v2 8C 2.600GHz. Infiniband FDR. NVIDIA K20x	81.41

Hybrid computers

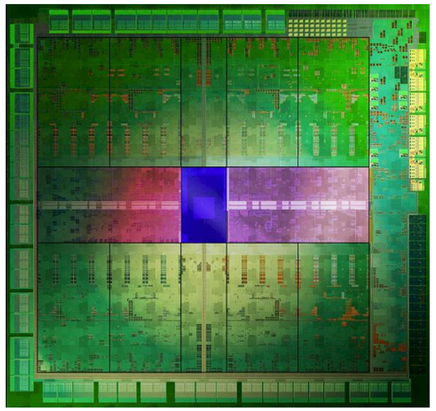
2006

IBM Cell BE



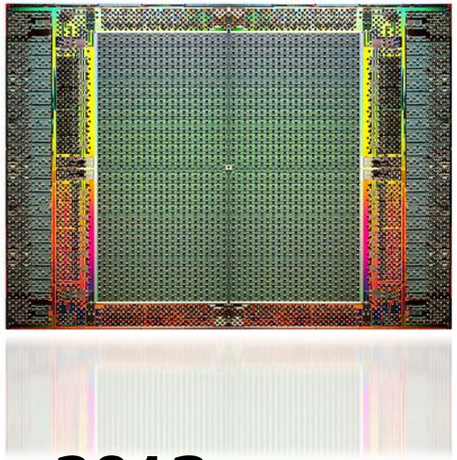
2007

GPGPU



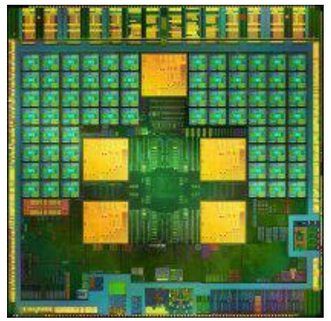
2010

FPGA (mass)



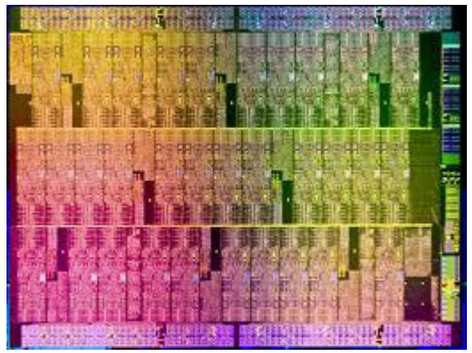
2010-2012

The era of post-PC
Tablet and smartphone revolution



2012

Intel MIC



2013-…

The era of micro servers is coming

RISC-architecture strikes back!

Summary table of popular architectures (2011-2013)

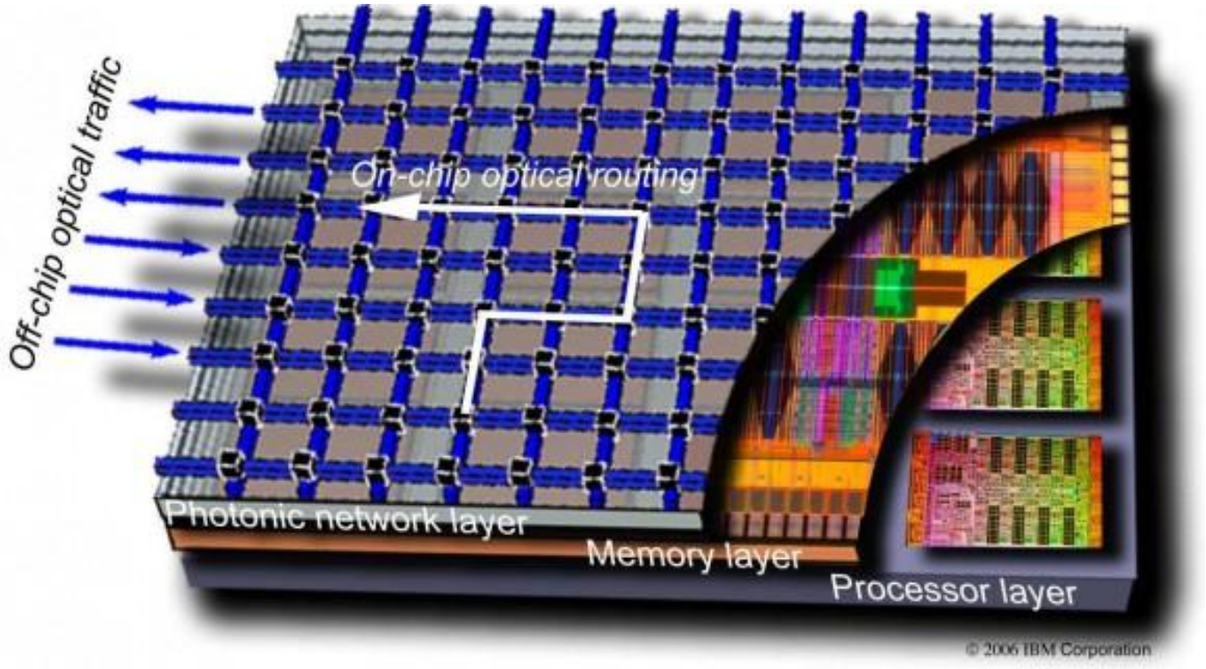
Processor	Type	GFLOPS (32bit)	GFLOPS (64bit)	Watt (TDP)	GFLOPS/Watt (32bit)	GFLOPS/Watt (64bit)
Adapteva Epiphany-IV	Epiphany	100	N/A	2	**50**	N/A
Movidius Myriad	ARM SoC: LEON3+SHAVE	15.28	N/A	0.32	**48**	N/A
ZiiLabs	ARM SoC	58	N/A	3	**20**	N/A
Nvidia Tesla K10	X86 GPU	4577	190	225	20.34	0,84
ARM + MALI T604	ARM SoC	8 + 68	N/A	4	**19**	N/A
NVidia GTX 690	X86 GPU x 2	5621	234?	300	18.74	0.78
GeForce GTX 680	X86 GPU	3090	128	195	15.85	0.65
AMD Radeon HD 7970 GHz	X86 GPU	4300	1075	300+	**14.3**	**3.58**
Nvidia Tesla K20X (Kepler)	X86 GPU	3950	1310	235	**16,8**	**5,5**
Intel Knight's Corner (Xeon Phi)	X87?	2000	1000	200	**10**	**5**
AMD A10-5800K + HD 7660D	X86 SoC	121 + 614	60	100	7.35	0.6
Intel Core i7-3770 + HD4000	X86 SoC	225 + 294,4	112 + 73.6	77	6.74	2.41
NVIDIA CARMA (complete board)	ARM + GPU	70 + 200	35	40	6.75	0.87
IBM Power A2	Power CPU	408	204	55	**7.44**	**3.72**

What's next?

Waiting for exascale…
2018-2020

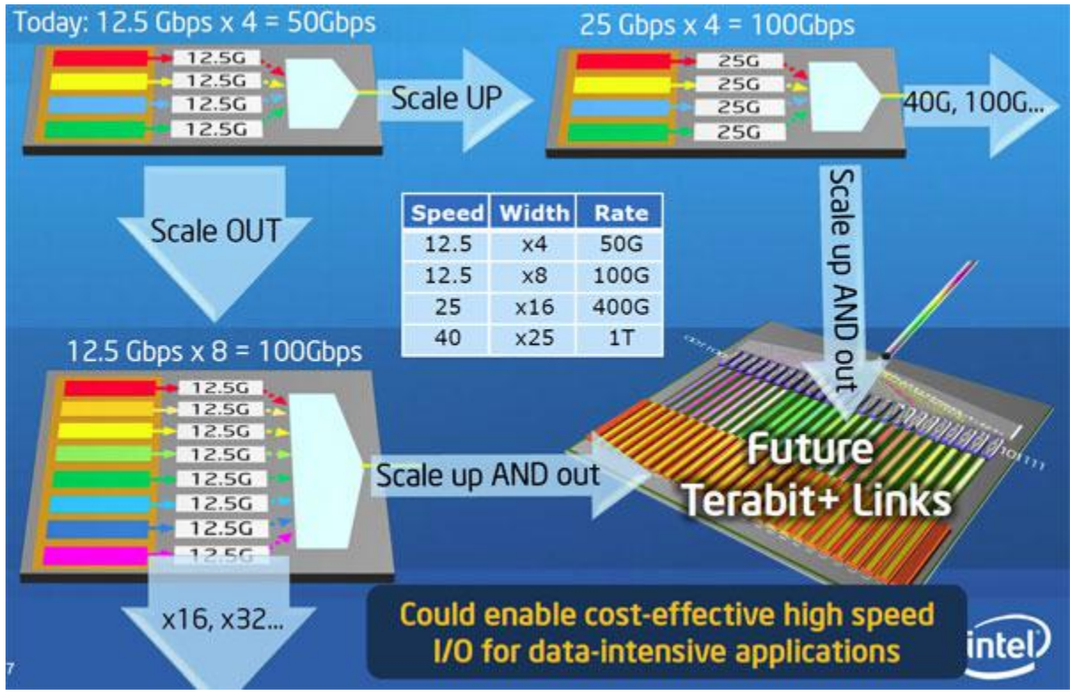
Silicon photonics (IBM, Intel)

IBM Silicon nanophotonics and 2.5D Interposer Design



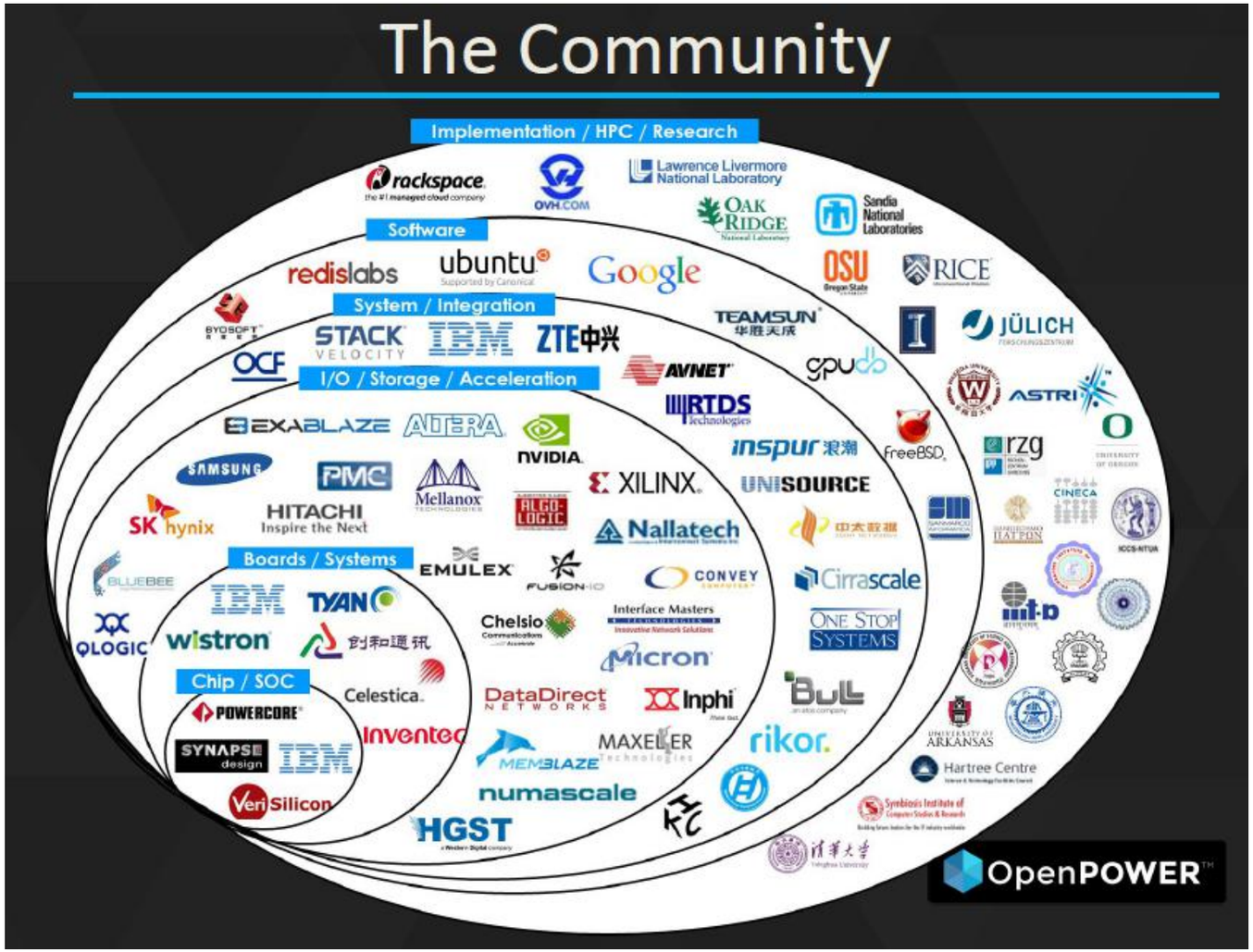
IBM's Silicon Photonics Technology Ready to Speed up Cloud and Big Data Applications
http://www-03.ibm.com/press/us/en/pressrelease/46839.wss

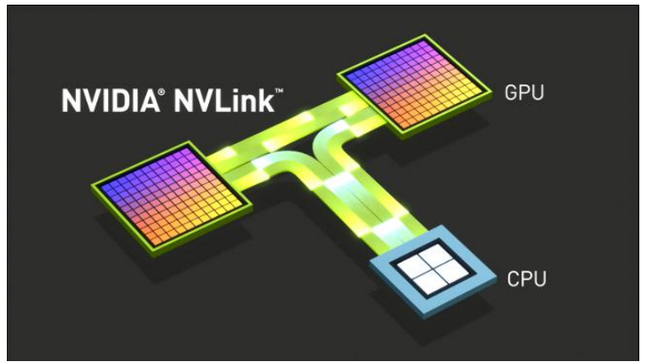
Intel Silicon Photonics Link

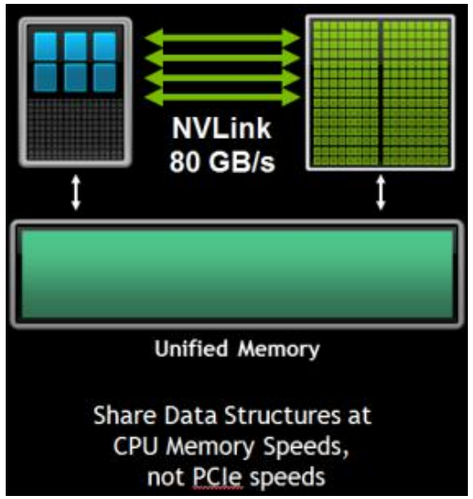


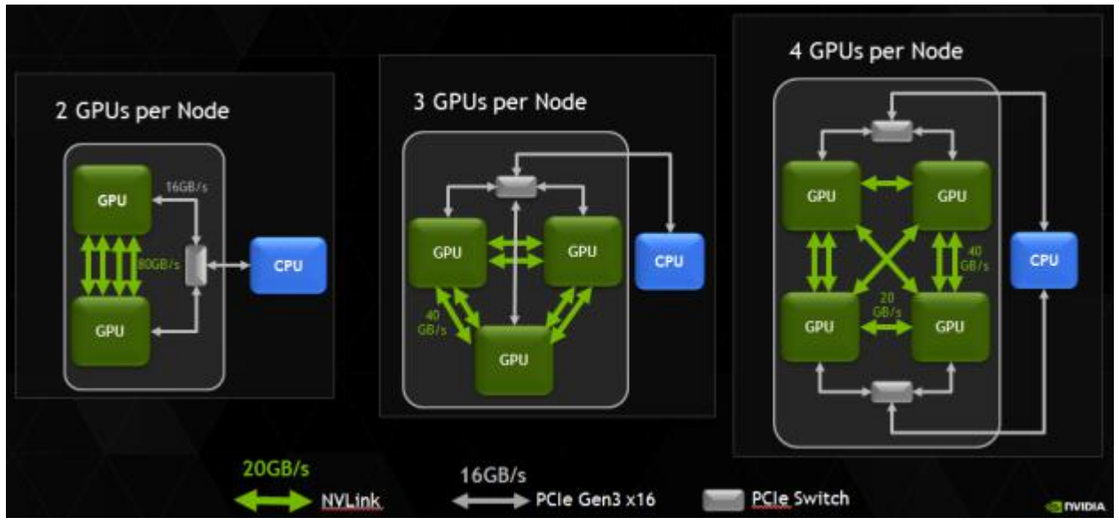
Intel® Silicon Photonics Technology
A New Era of Scalable Bandwidth
https://intel.lanyonevents.com/sf14/connect/fileDownload/session/DC78ECBD76550A7F48C1474BC2589AA6/SF14_NETS001_102f.pdf

OpenPOWER Division + NVLink







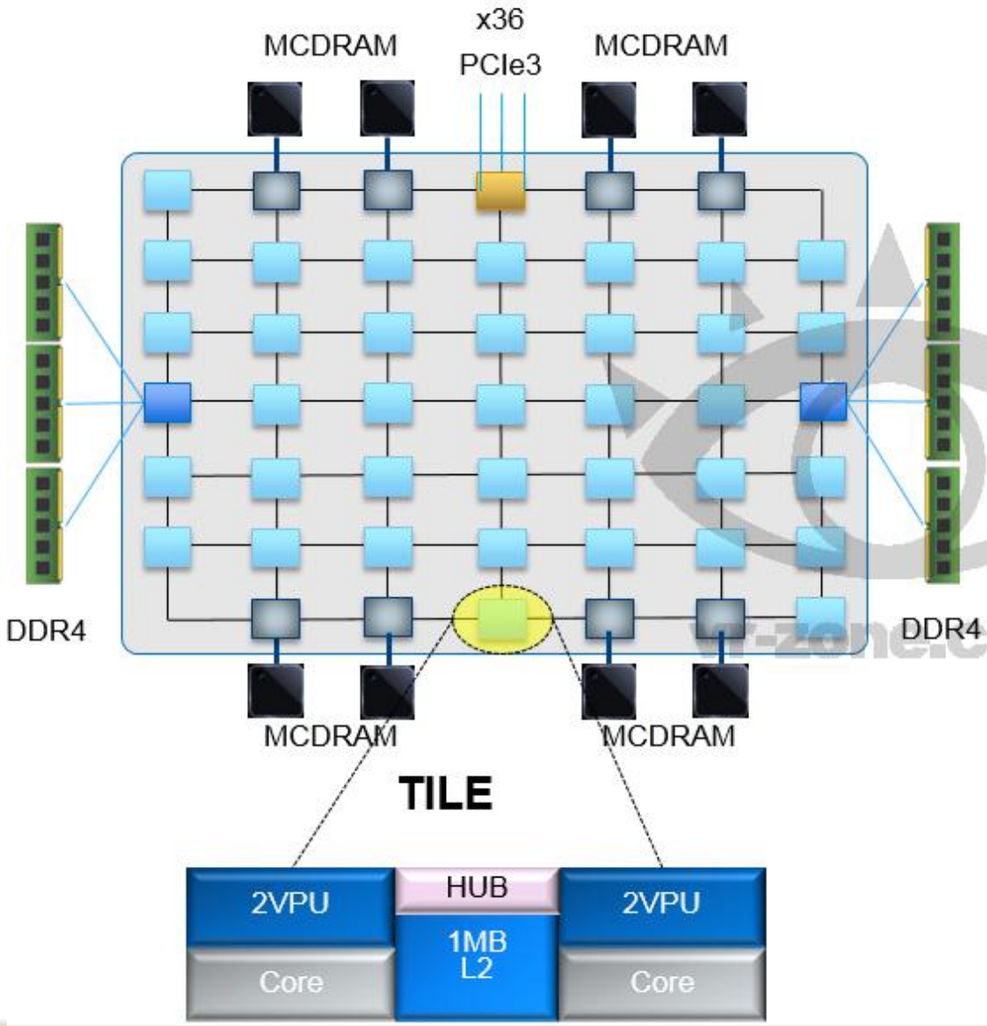


www.openpowerfoundation.org

http://www.nvidia.com/object/nvlink.html

Intel MIC 2015 (3 Tflops DP, 14 nm)

Knights Landing Processor Architecture



Up to 72 Intel Architecture cores based on Silvermont (Intel® Atom processor)

- Four threads/core
- Two 512b vector units/core
- Up to 3x single thread performance improvement over KNC generation

Full Intel® Xeon processor ISA compatibility through AVX-512 (except TSX)

6 channels of DDR4 2400 MHz -up to 384GB

36 lanes PCI Express* Gen 3

8/16GB of high-bandwidth on-package MCDRAM memory >500GB/sec

200W TDP

AMD Big APU 2015

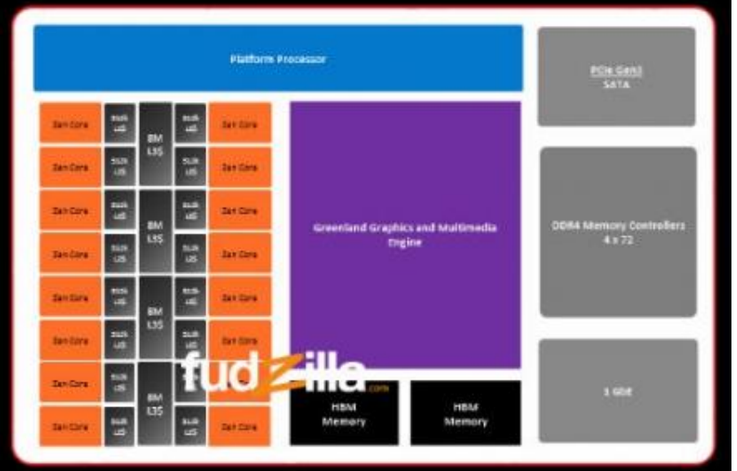
Up to 16 "Zen" cores | "Greenland" Stream Processor | 4 DDR4 Memory Channels

COMPUTE
- Up to 16 AMD Zen x86 cores (32 threads)
- 512KB L2 cache per core (8MB total L2 cache)
- 32 MB shared L3 cache
- Platform Processor
  - Secure Boot
  - Crypto Coprocessor

GRAPHICS
- "Greenland" stream processor
- Up to 16GB HBM Memory (512GB/s)
- ½ rate double precision compute
- Enhanced EC[illegible]
- HSA

MEMORY
- 4 channel DDR4 with ECC up to 3200 MHz
- SODIMM, UDIMM, RDIMM, LRDIMM 2 DIMMs/channel
- Memory capacity 256GB/channel

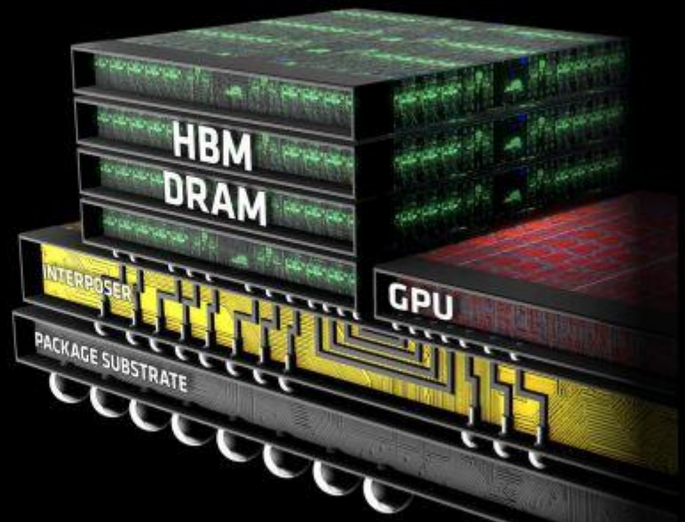
INTEGRATED I/O
- 64 lanes of PCI-Express Gen3
  - 16 lanes switchable with 2 lanes of SATA Express, 14 lanes of SATA



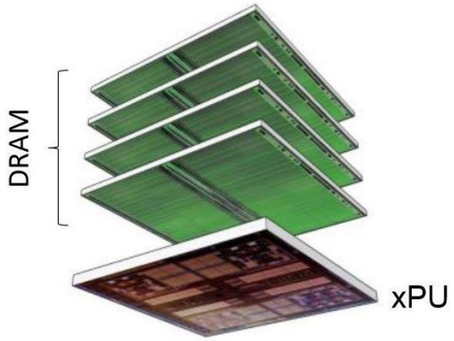
GRAPHICS TECHNOLOGY LEADERSHIP

AMD

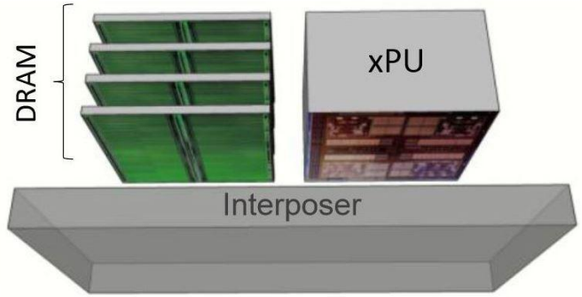
HIGH BANDWIDTH MEMORY
- First in the Industry with High Bandwidth Memory (HBM) Technology
- 3D HBM DRAM Die Stack on Silicon Interposer
- >3X Performance/Watt Compared to GDDR5[3]
- >50% Power Savings Versus GDDR5[4]



10 | 2015 FINANCIAL ANALYST DAY | MAY 6, 2015

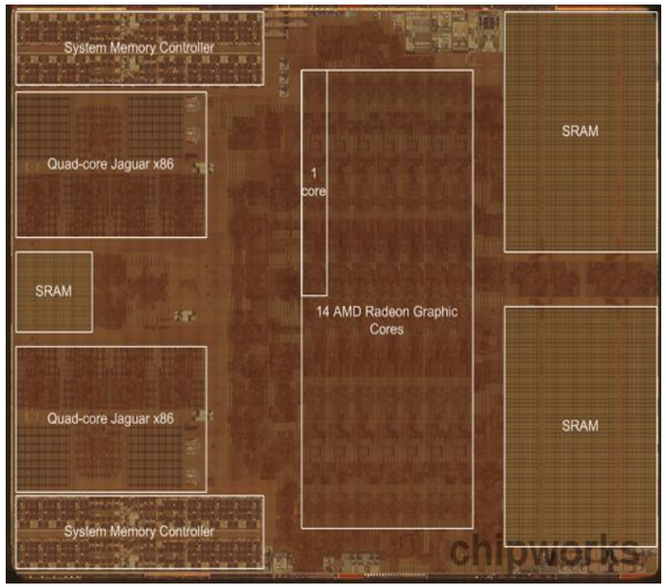


VERTICAL STACKING (3D)

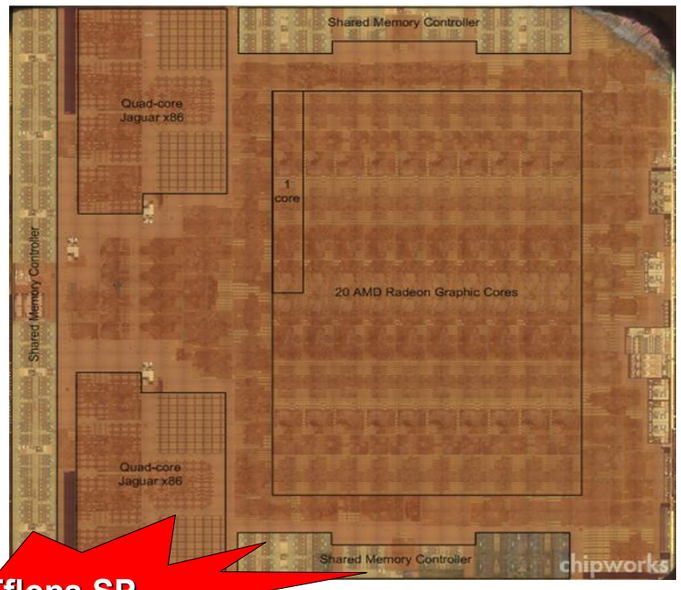


INTERPOSER STACKING (2.5D)

PS4 and Xbox APU AMD 2013



	Xbox One	Playstation 4
CPU	8 core AMD SoC	Eight x86-64 AMD Cores
GPU	Unknown	1.84 TFLOPS AMD Radeon
RAM	8 GB	8 GB
HDD	500 GB	Unknown
Camera	1080p	2 cameras with 1280 x 800
USB 3.0	YES	YES
LAN	YES	YES
Wi-Fi	802.11n Wi-Fi	802.11 b/g/n
Video	HDMI, S/PDIF	HDMI, S/PDIF, AV



1.84 Tflops SP
180 Watt, 28 nm

AMD Server CPU based on ARM architecture

THE NEW AMD OPTERON™ A-SERIES PROCESSORS

AMD

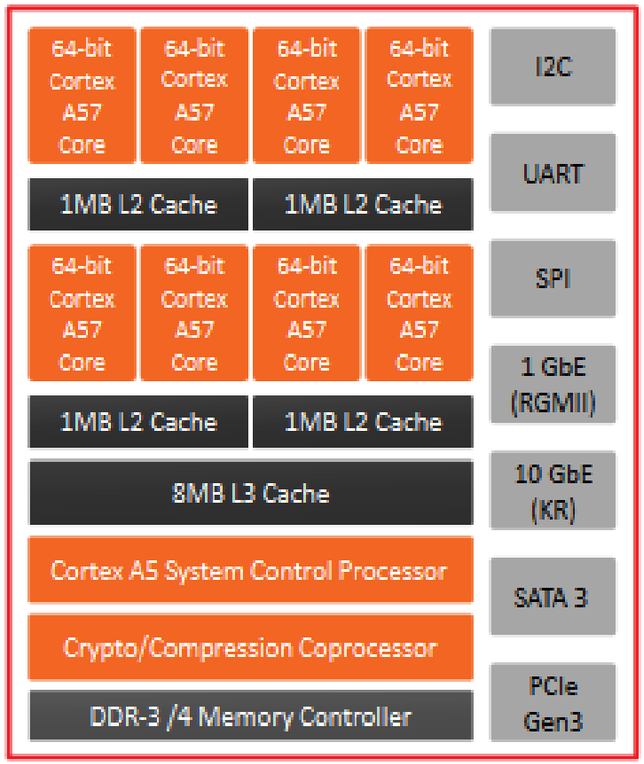
ARM EFFICIENCY | AMD OPTERON™ SERVER DNA

Industry's only 64-bit ARM Server SoC from a proven server processor supplier

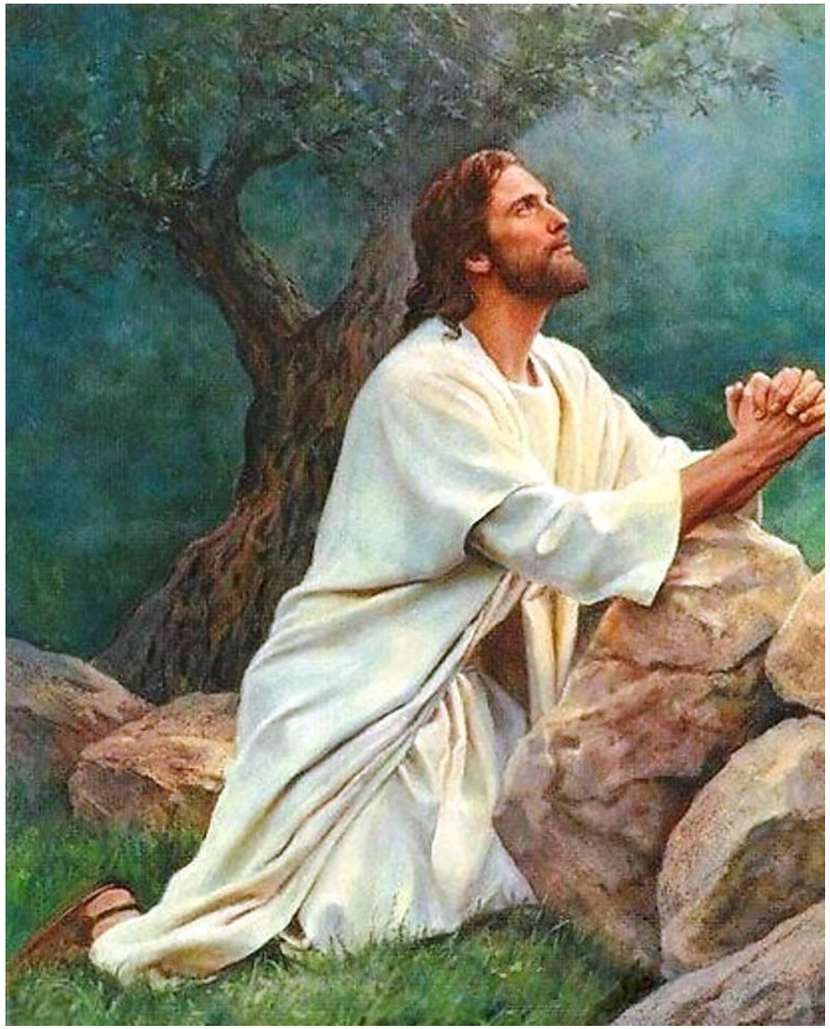
- The most server experience of any ARM licensee
- Server class IP blocks–no other competitor has

Opteron A1100 Processor – "Seattle"

- 2-4x the performance of AMD Opteron™ X-Series with significant improvement in compute/watt[3]
- 8 core SoCs with 128 GB DRAM support
- Based on ARM Cortex™-A57 cores at > = 2 GHz
- Server caliber encryption and compression
- Integrated 10GbE
- High port-count storage interfaces for big data



Dream SoC processor for exascale HPC



DreamSoC:

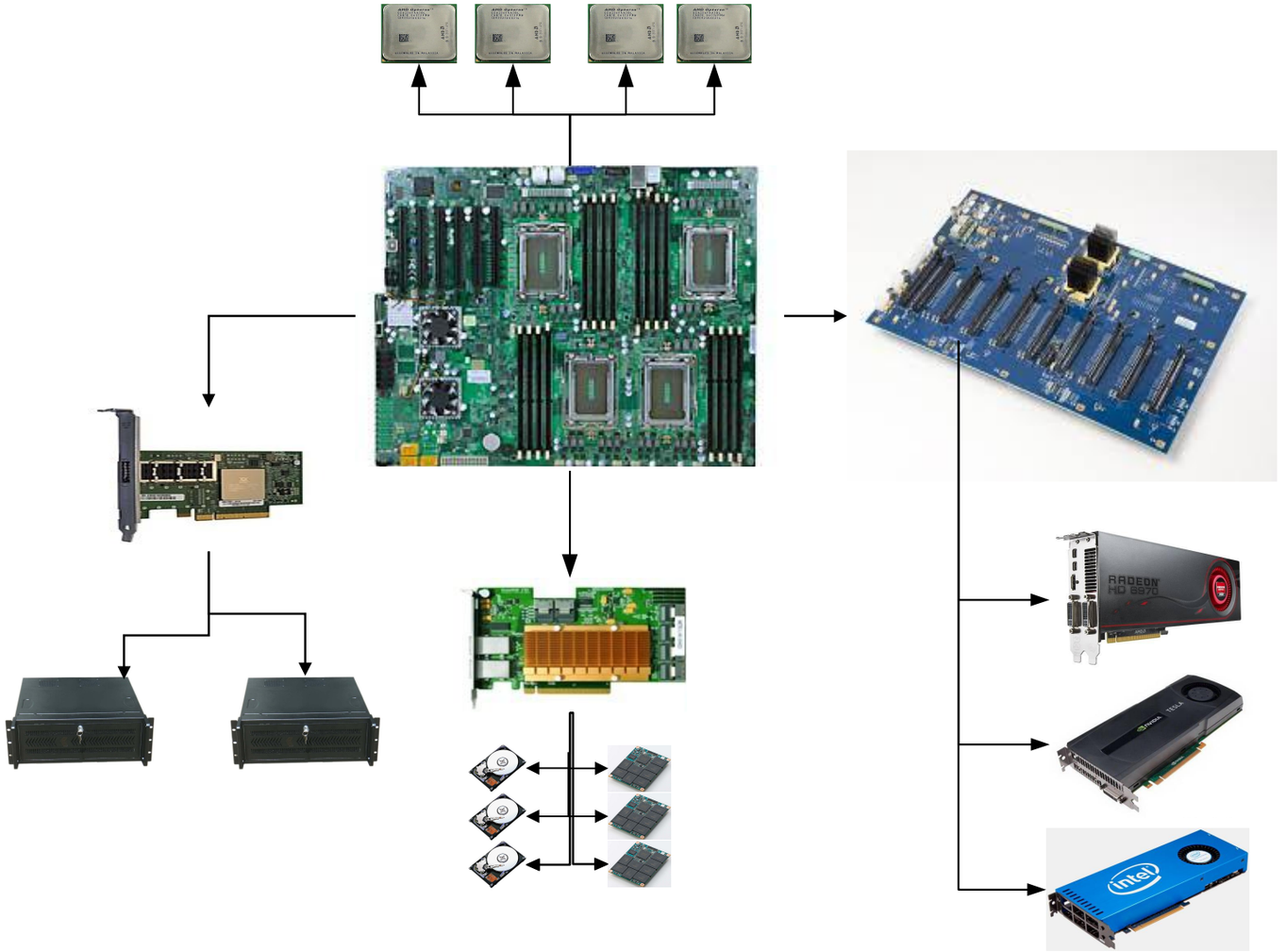
- ARM\MIPS Cores (NVIDIA\AMD)
- DDR4 multi-chanel controller (Intel\AMD)
- Forced massive-parallel\GPU computing block (AMD) with HBM
- FPGA\ASIC computing block (Intel)
- Infiniband\RapidIO interconnect (NVLink?)
- Multi socket configuration (AMD?)

Hardware

Scheme of 20TFlops DP node

- 4 CPU, 72 cores
- 1 TB RAM
- PCIe slots\expansion 8 GPU
- SAS RAID → 50 TB
- Interconnect

Per node:
- TFlops ~ 21 = 8*2500+4*250 DP
- Power ~ 2.7 kW
- Price ~ $50K



Equipment

Reference CPU:

- Intel Xeon Nehalem, Sandy\Ivy Bridge, Haswell

GPU devices:

- nVidia GeForce GTX480

164 Gflops, 177 Gb/s

- nVidia Tesla C2050

515 Gflops, 144 Gb/s

- nVidia GeForce TITAN

1560 Gflops, 288 Gb/s

- AMD Radeon 6970

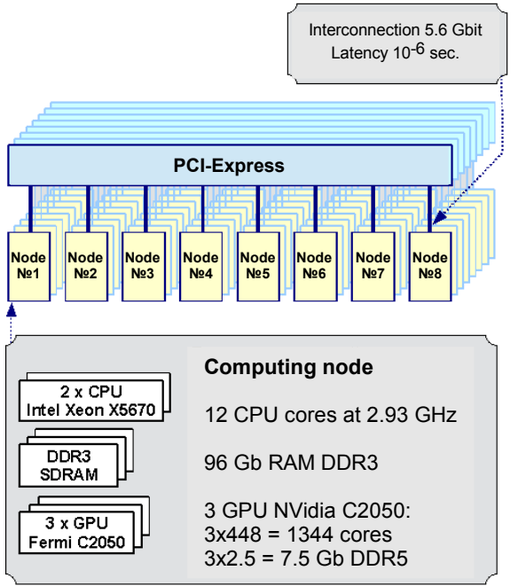
683 Gflops, 176 Gb/s

- AMD Radeon 7970

974 Gflops, 264 Gb/s

- AMD Firepro W9100

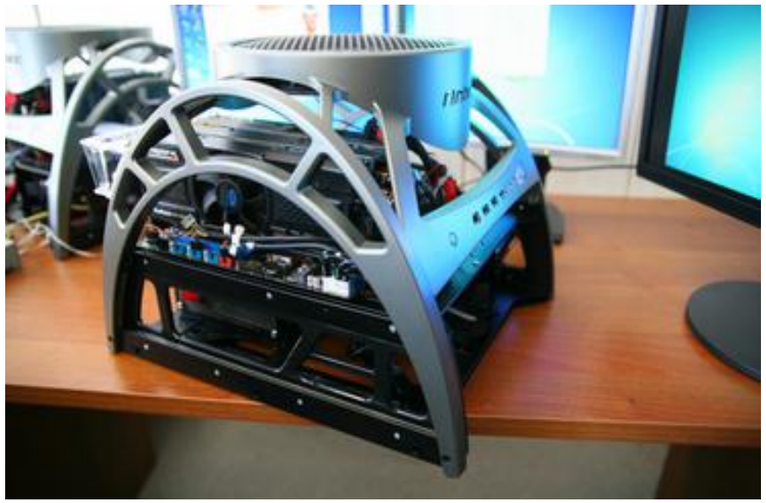
2560 Gflops, 320 Gb/s

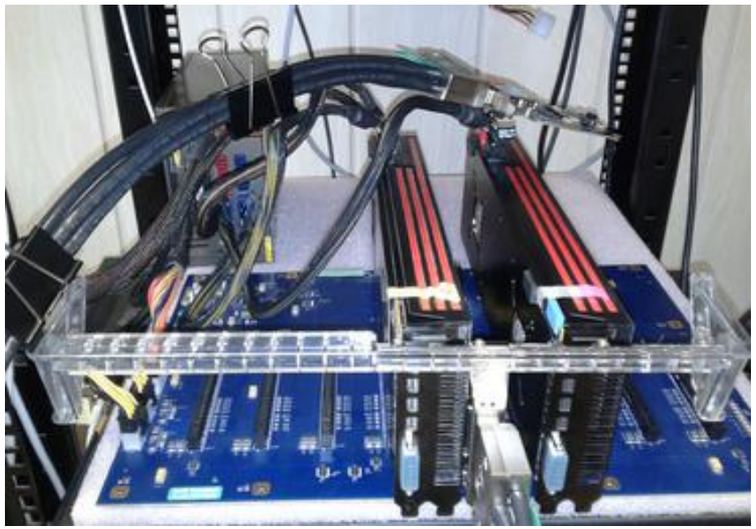


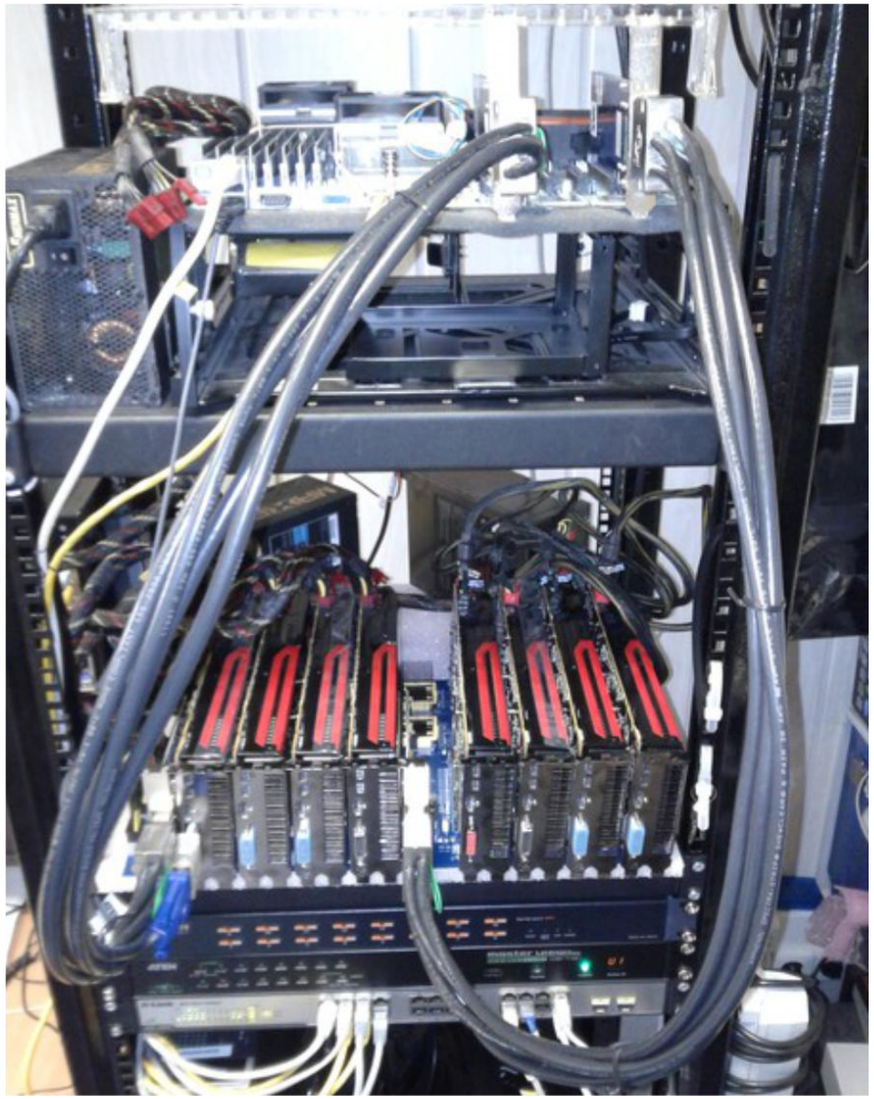
- Supercomputer K100 (www.kiam.ru)

Test platforms 2011-2013

PCI Express Expansion Systems – Cyclone







Test platforms 2013-2015

Platform TYAN

in 4U form factor:
2 x Xeon Ivy-Bridge + 8 x ACCs ≈ **12÷20 TFlops DP**
PowerC ≈ 2.5 kW Cost ≈ 20k÷50k $

Platform SuperServer Supermicro

in 4U form factor (4 nodes):
8 x Xeon Ivy-Bridge +
12 x ACCs ≈ **18÷30 TFlops DP**
+ FDR Infiniband
PowerC ≈ 4.2 kW Cost ≈ 55k÷80k $

Platforms options

Platform	ACC Type	ACC Num	kWatt	Cost $K	Peak Perf DP TFLOPS
TYAN 	NV TITAN ~ K40X 	8	2.6 kW	$20K	12 TFlop/s
	AMD FP W8100 	8	2.0 kW	$20K	16 TFlop/s
	AMD FP W9100 	8	2.6 kW	$45K	20 TFlop/s
Supermicro SuperServer 	NV TITAN ~ K40X 	8	3.6 kW	$38K	13 TFlop/s
	AMD FP W8100 	8	3.0 kW	$38K	17 TFlop/s
	AMD FP W9100 	8	3.6 kW	$62K	21 TFlop/s
	NV TITAN ~ K40X 	12	4.6 kW	$42K	20 TFlop/s
	AMD FP W8100 	12	4.0 kW	$42K	25 TFlop/s
	AMD FP W9100 	12	4.6 kW	$78K	31 TFlop/s
	4 x InfiniBand FDR adapter 			$4K	
	SX6005/SX6012 12-port 56Gb/s FDR InfiniBand/VPI Switch System 			$5K	

The main problems of software development for modern supercomputers

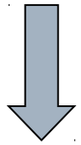
Who will win: SMP CPU or HYBRID?

Target architecture? How to develop the software?

What does the HPC industry do for software developers?

Low-level models:

Classic scheme: MPI/OpenMP



Hybrid node: MPI/OpenMP

CUDA vs **OpenCL**

- Proprietary model
- Works only with nVidia GPUs

- Open standard
- Supports a wide class of devices: CPU, GPU, MIC, FPGA, CELL and more
- Supported by Khronos group. AMD, NVidia, Intel, IBM and else provide OpenCL drivers

What does the HPC industry do for software developers?

Hi-level models:

Oppan pragma style:

- **OpenACC** - programming standard for parallel computing (Cray, CAPS, Nvidia(PGI))
- **OpenHMPP** - Hybrid Multicore Parallel Programming (INRIA, CNRS, University of Rennes, INSA of Rennes)
- **OmpSs** - Barcelona Supercomputing Center
- **OpenMP 4.0** – different device execution support
- **OpenCL SPIR** – Standard Portable Intermediate Representation

---

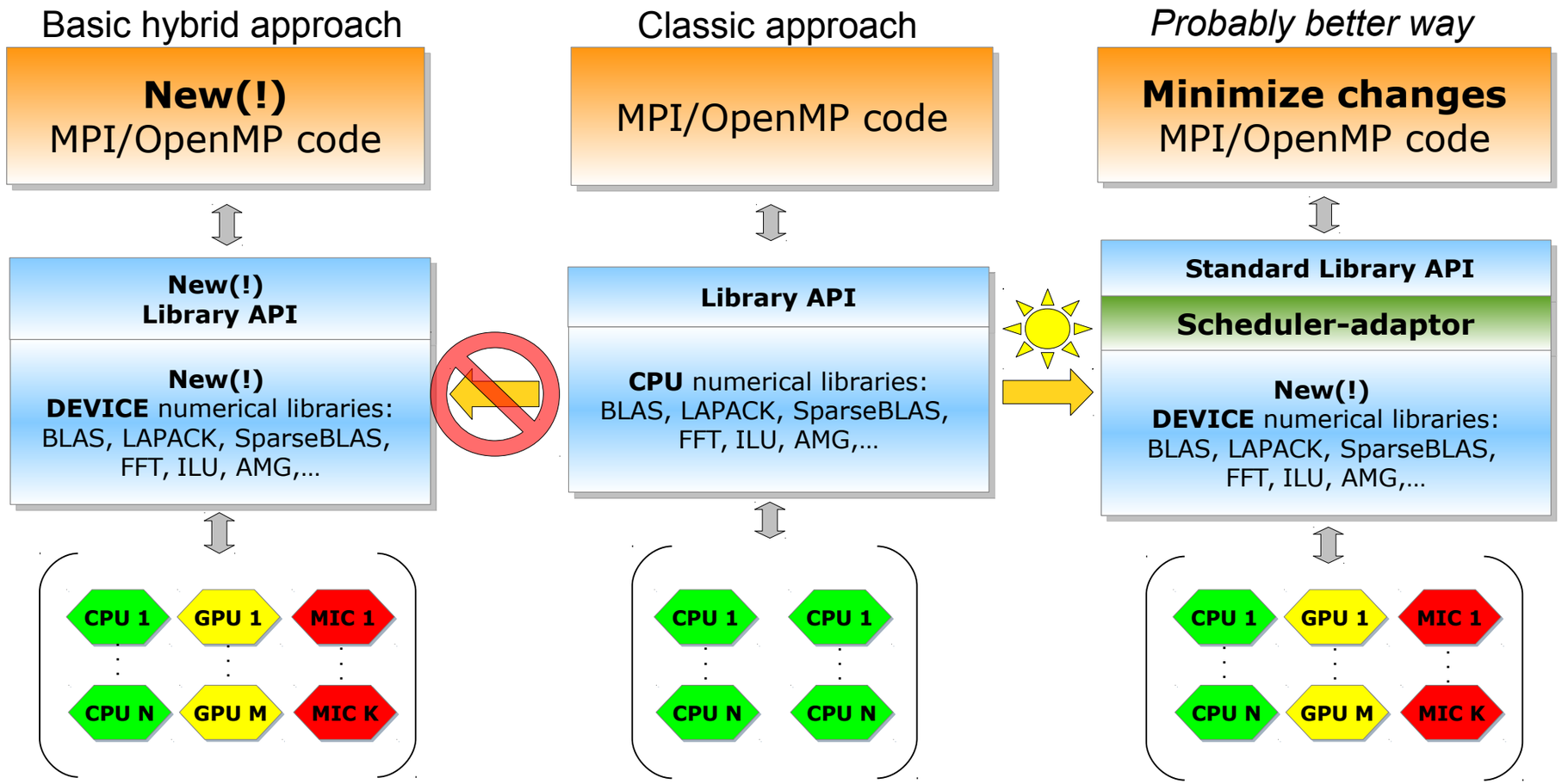
Wrappers:

- **C++ OpenCL, SYCL, Bolt++, Kalmar, C++ AMP, VexCL, Boost.compute, SkelCL, xxxCL,** …
- **Aparapi, PyOpenCL,** …
- **V(irtual)CL, SnuCL,** ...

Task scheduling:

- **StarPU** - task programming library for hybrid architectures (INRIA, France)
- **Distri-GPUs Framework** (University of Tennessee, USA)
- **CGCM** - CPU-GPU Communication Manager (Princeton University, Princeton, NJ, USA)

Legacy codes problem



Main problems

- ✔ Does it make sense to start developing?
- ✔ Unified development model
- ✔ Legacy codes

and another problem...

Programming infrastructure of heterogeneous computing

Infrastructure: main ideas

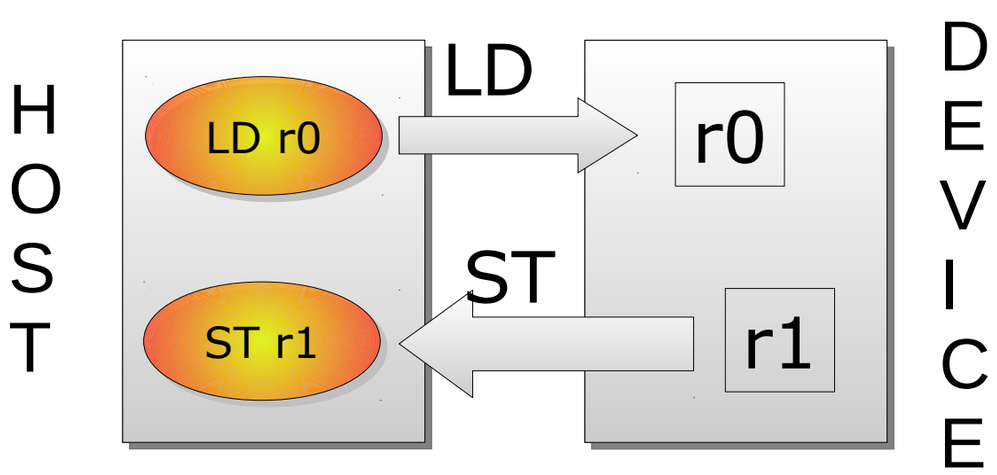
- Provide unified infrastructure to develop programs for hybrid clusters:
  - simple wrapper for **OpenCL** commands and queues
  - static scheduler
- Priorities:
  - Simple logical model
  - Scalability
  - Minimal code changes
  - Automatic transfer overlap when possible

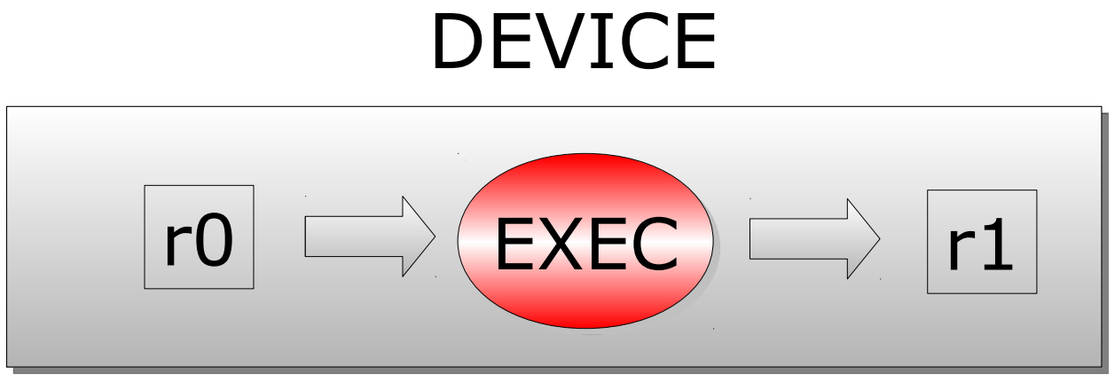
Infrastructure: concepts

- **Register** – a region in device global memory
- **Instruction** – compute kernel for the device
- Logically, a **device is** represented as **a coprocessor running instructions on registers**

Wrapper: memory commands

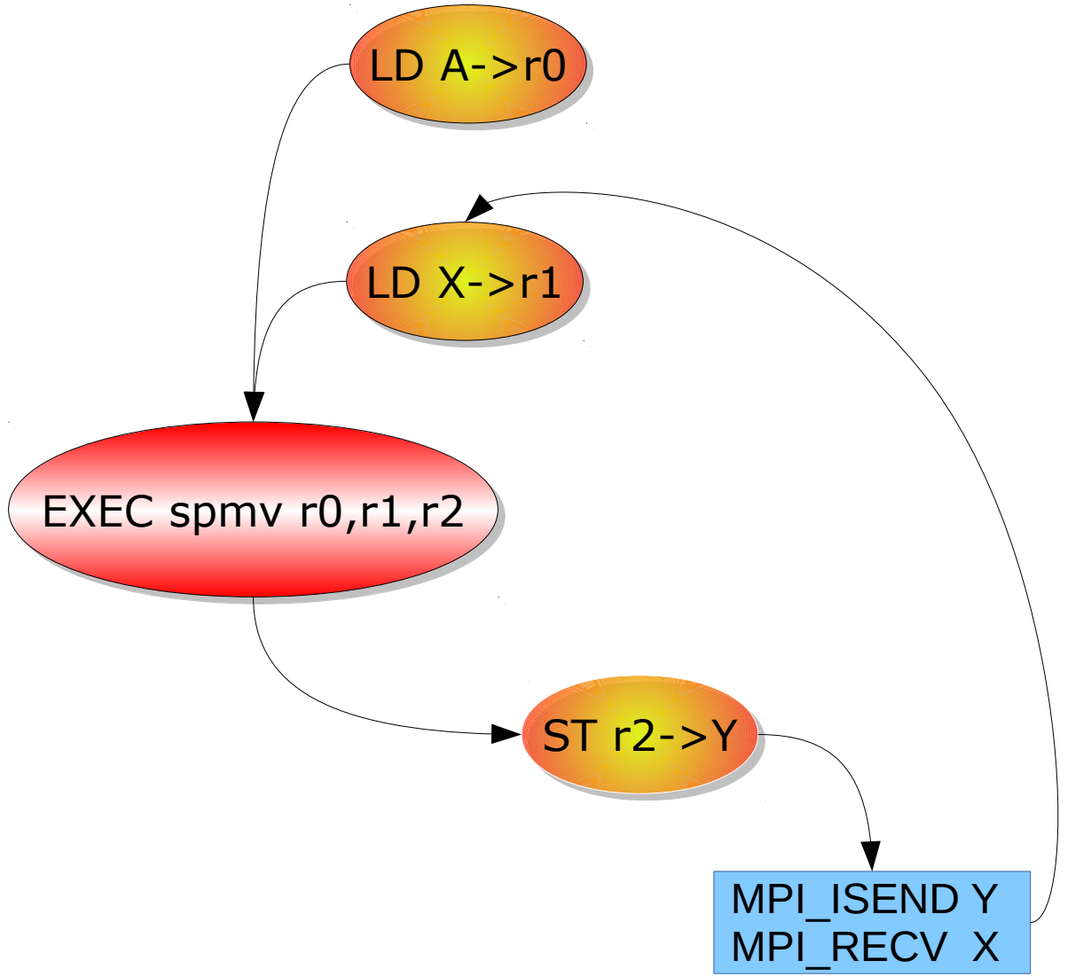
- **LD**: load a n-dimensional region from host memory to a register on device
- **ST**: store a region from device to host
- **EXEC** command launches an arbitrary kernel on a device
- Arguments – registers and scalar values





Scheduler: program

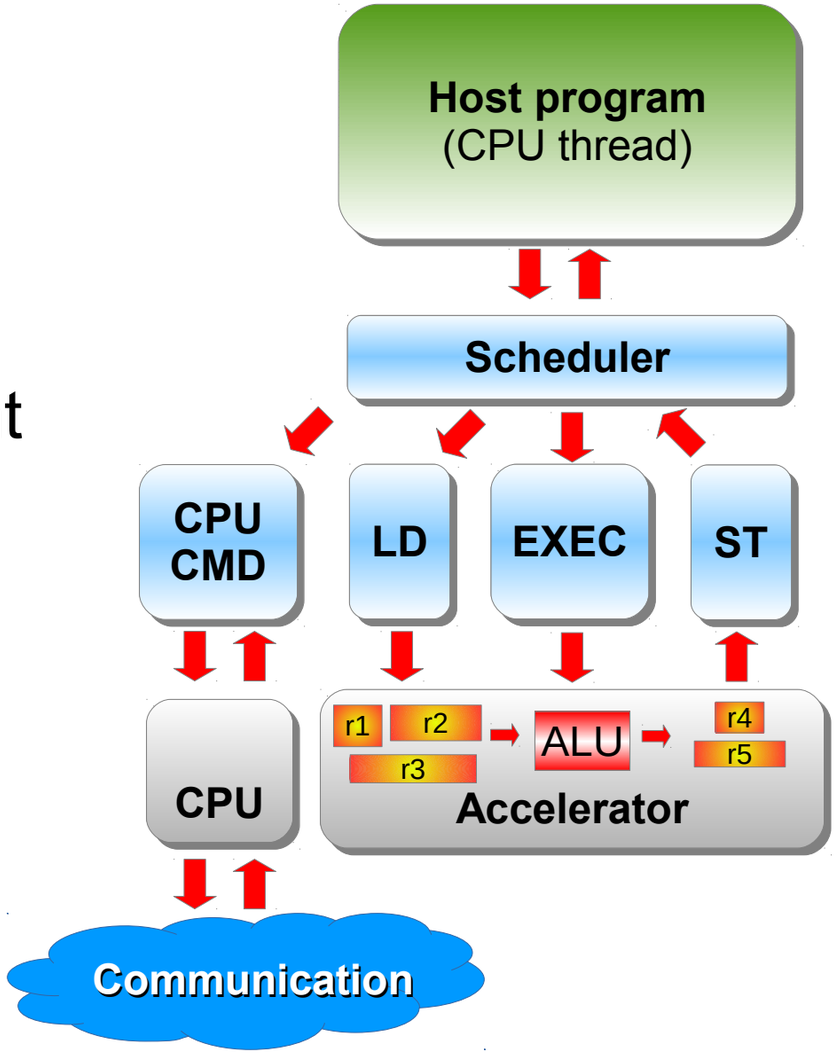
- **A program** for a scheduler **is a graph of dependencies between commands**
- Command is ready when all commands it depends on are ready



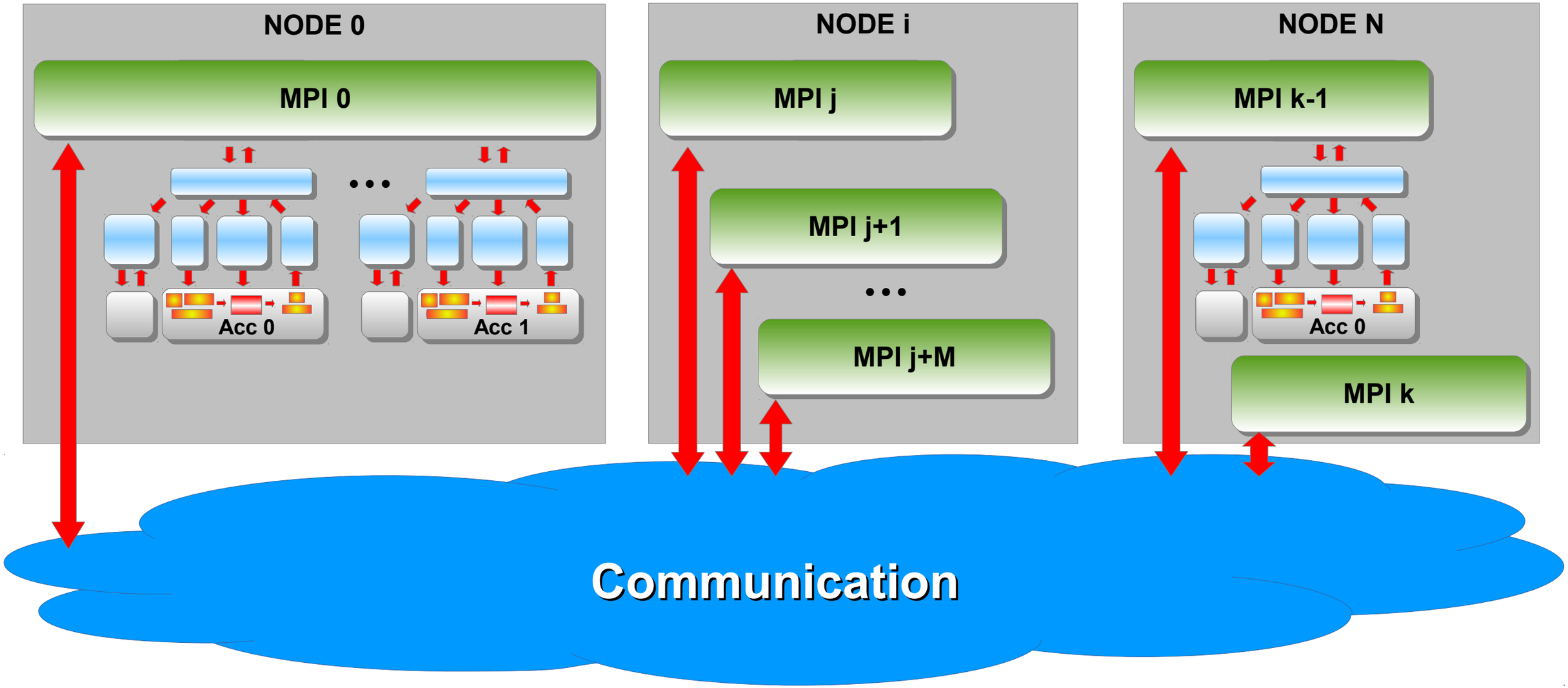
Scheduler: queues

- Scheduler has 4 queues for LD, ST, EXEC and CPU commands
- Independent commands from different queues can be executed in parallel
- EXEC and CPU run in-order

  LD and ST – asynchronously



Infrastructure : usage



Infrastructure: usage

1. Write compute kernels

2. Arrange dependencies and fill queues

3. Start execution

Classical benchmarks

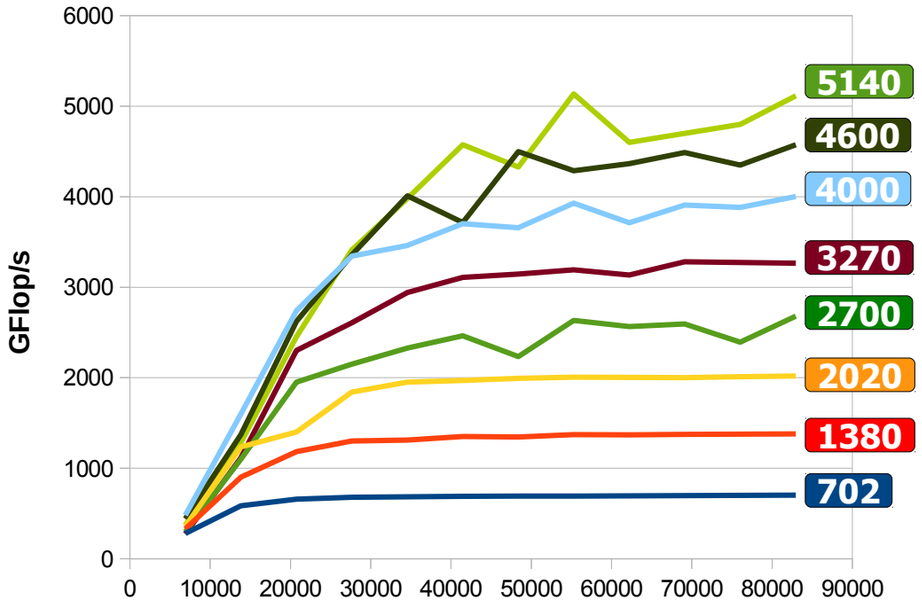
BLAS 2012

Basic Linear Algebra Subprograms. Level 3: matrix-matrix operations.

ISR RAS implementation: scalability on **AMD Radeon 7970 (Firepro 9000)**

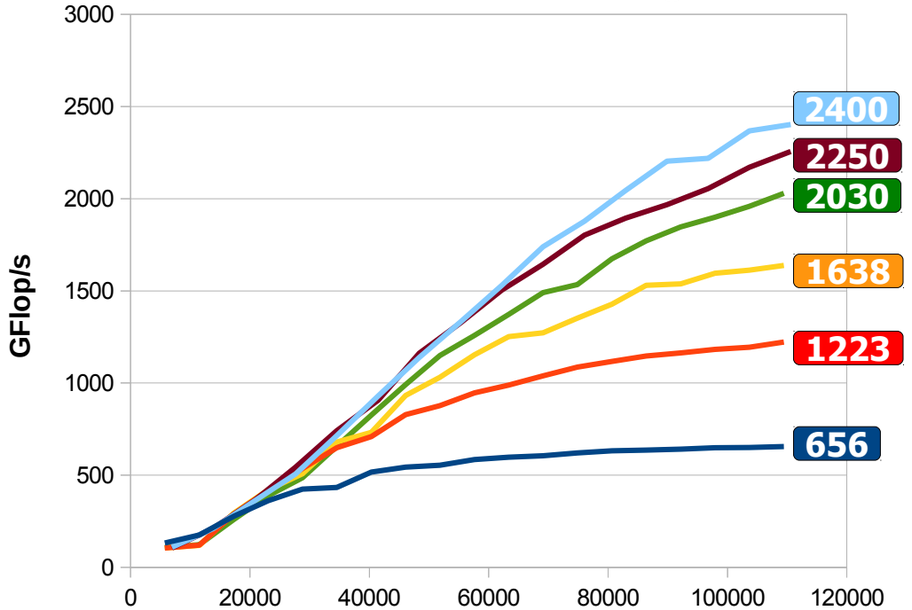
$$PEAK = 8 \times 1,034 \approx 8,3 \text{ TFLOP/s}$$

DGEMM – matrix multiplication



EFF ~ 65 % *(incl. logging)*

DGETRF – LU decomposition
(includes ***DTRSM*** – triangular SLAE solution)



EFF ~ 45 % *(incl. logging)*

Details at `http://devgurus.amd.com/thread/159457`

ISR RAS vs MAGMA

Jack Dongarra

THE UNIVERSITY of TENNESSEE

Collaborating Partners:
- University of California, Berkeley
- University of Colorado, Denver
- INRIA Bordeaux - Sud Ouest
- INRIA Saclay/University Paris-Sud
- KAUST

MAGMA

Matrix Algebra on GPU and Multicore Architectures

- `MAGMA for CUDA`
- `clMAGMA for OpenCL`
- `MAGMA MIC for Intel Xeon Phi`

	PEAK 1 DEV Tflop/s DP	DGEMM Tflop/s DP	DGEMM Eff %	DGETRF Eff %	SCL
MAGMA-MIC (Xeon Phi)	1.04	0.83	84	60	4
MAGMA-CUDA (Kepler K20X)	1.32	1.2	92	73	3
ISR RAS (Radeon HD 7970 Ghz Ed)	**1.03**	**0.7**	**68**	**50**	**8**

High-performance LINPACK (HPL) OpenCL 2013

A performance benchmark for distributed parallel computing systems;
basic test for the Top500 list.
Includes computationally intensive routines with lots of data exchanges.
Uses 10 BLAS routines (DP),
~95% of the running time – DGEMM and DTRSM.

The ISR heterogeneous implementation:

HPL + MPI library + heterogeneous BLAS library (hBLAS)

Test platform	PEAK (GFLOP/s)	PERF (GFLOP/s)	EFF (%) incl. logging
2 x Xeon E5-2670 + 2 x Radeon 7970	2300	887	38
9 x K100 node [2 x Xeon X5670 + 3 x Tesla M2050]	9 x 1800 = 16200	3119	20

Details at `http://devgurus.amd.com/thread/159457`

A hybrid system model

A methodology for algorithm performance estimation

Our approach to algorithm performance study:

- a model of the target computing system
- parameter set
- a computation scheme for the model
  - kernel
  - a sequence of loads/stores and kernel calls
- formulas for performance estimates
- implementation?

The considered computing system

- Central Processing Unit;
- A number of coprocessors it controls;
- System (host) memory.

All coprocessors share one channel to access the host memory.

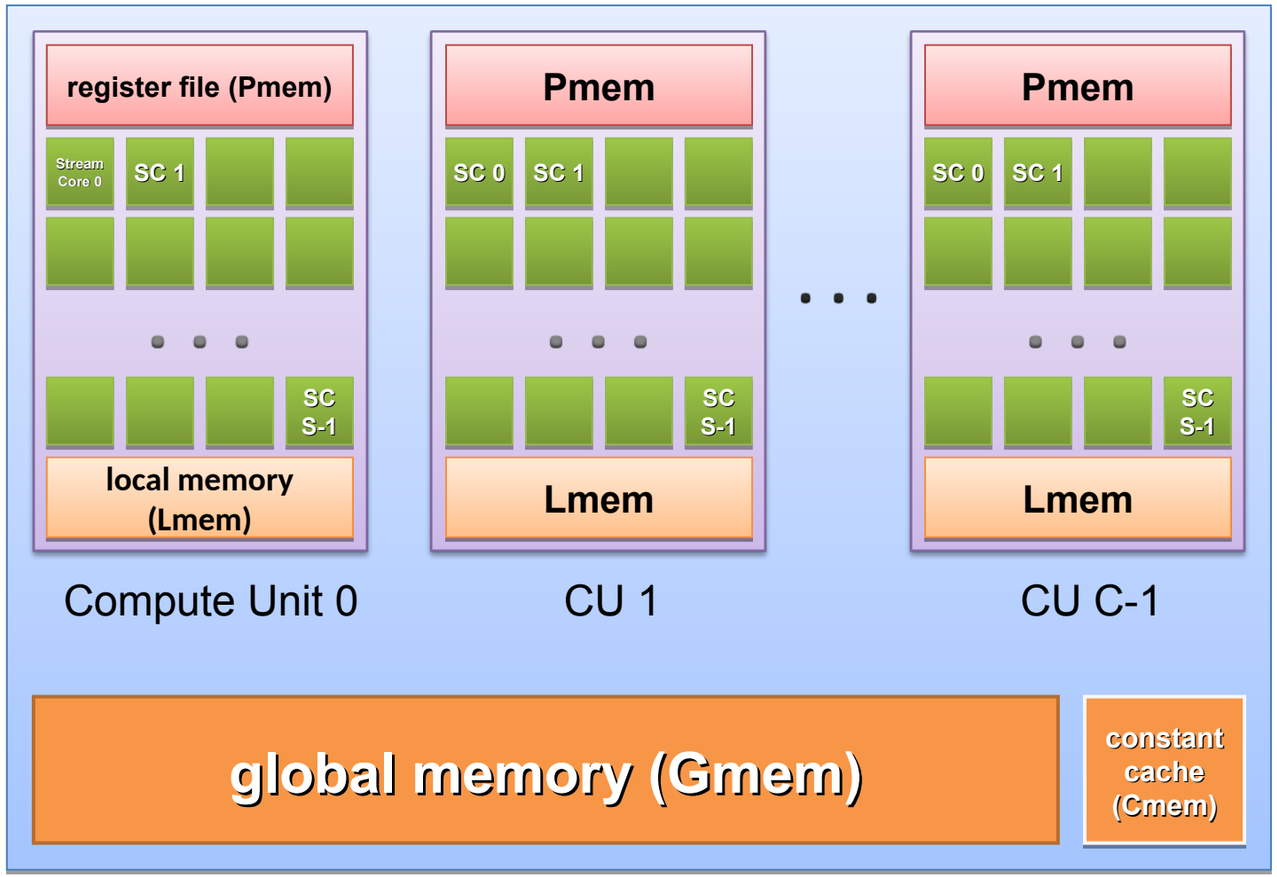
Data exchanges between coprocessors – through the host memory.

Typical coprocessors: GPGPUs.

Another example: **Intel Xeon Phi**.

Coprocessor model

(generalized GPU)



Key parameters:

$$Gmem$$

$$bw_g$$

$$Lmem$$

$$Pmem$$

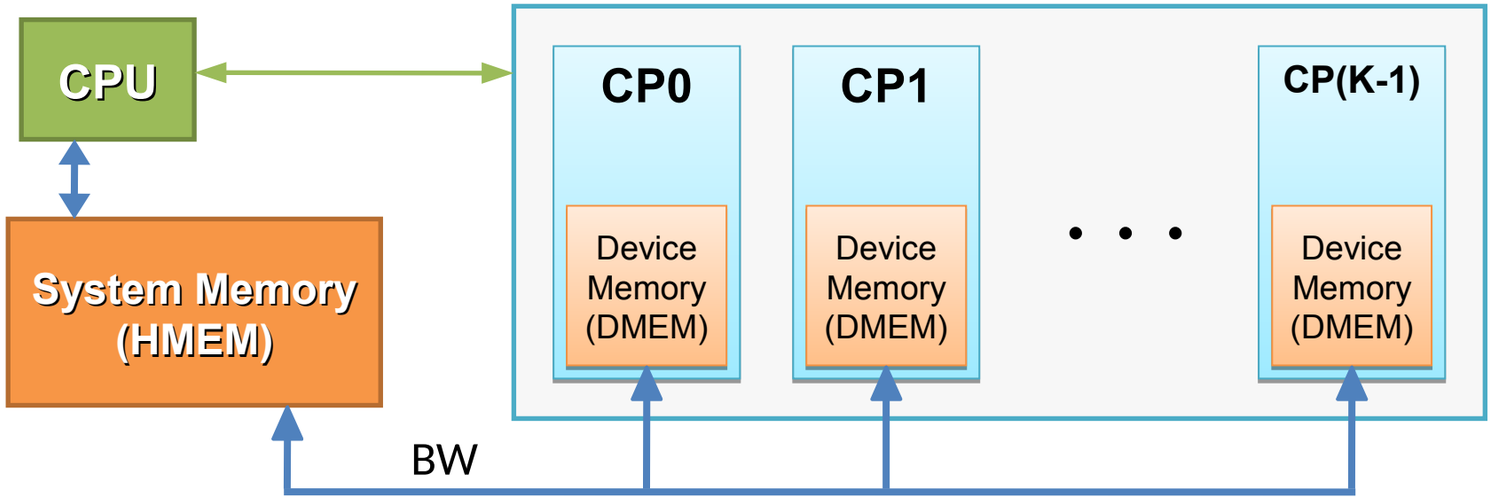
$$C$$

$$PEAK$$

$$Perf_{Kern}$$

– *kernel performance*

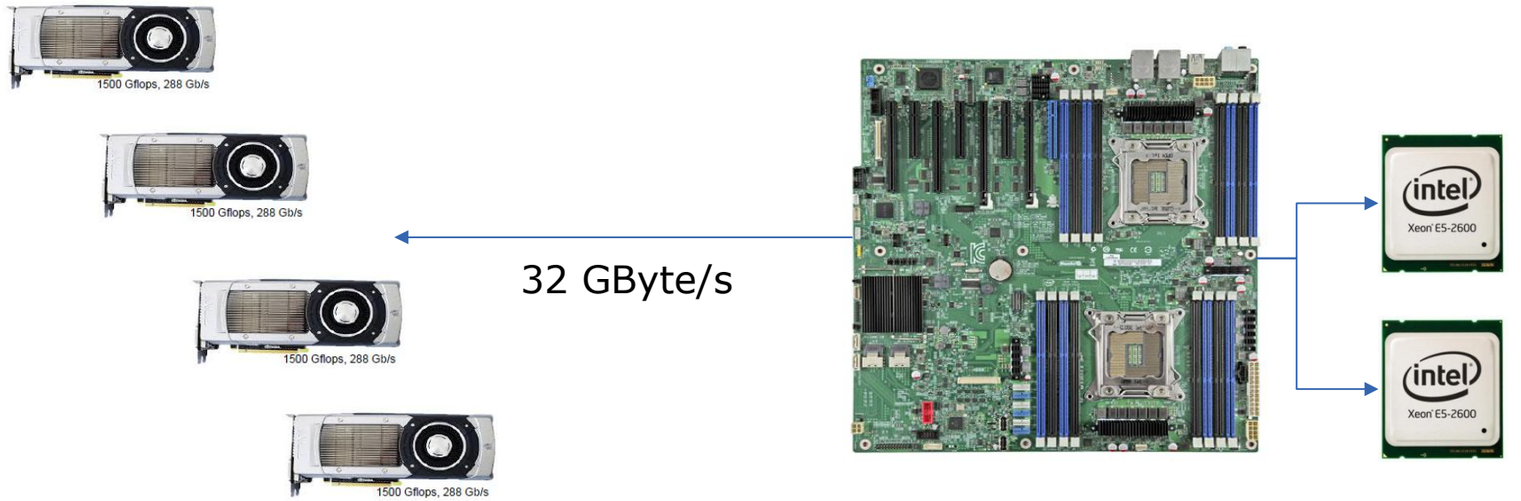
The whole hybrid system model



Parameters:

- *K* — the number of coprocessors
- *DMEM* — one coprocessor device memory size
- *DPEAK* — one coprocessor peak performance
- *HMEM* — host memory size
- *BW* — the channel bandwidth between the host memory and the coprocessors

Example 1: one of the ISR test configurations – single hybrid computing node



2 CPU, Total 16 cores

Intel Sandy\Ivy Bridge

128 GBytes Memory

Performance (double-precision, DP):

2xXeon E5-2670 = **0.330 Tflops**

NVIDIA GeForce GTX Titan = **1.5 Tflops**

4 x [NVIDIA Titan] = **6.0 TFlops**

Total: ~6.33 TFlops DP (~19 TFlops SP)

PowerC: ~ 1.6kW

K = 4

HMEM = 128 GB

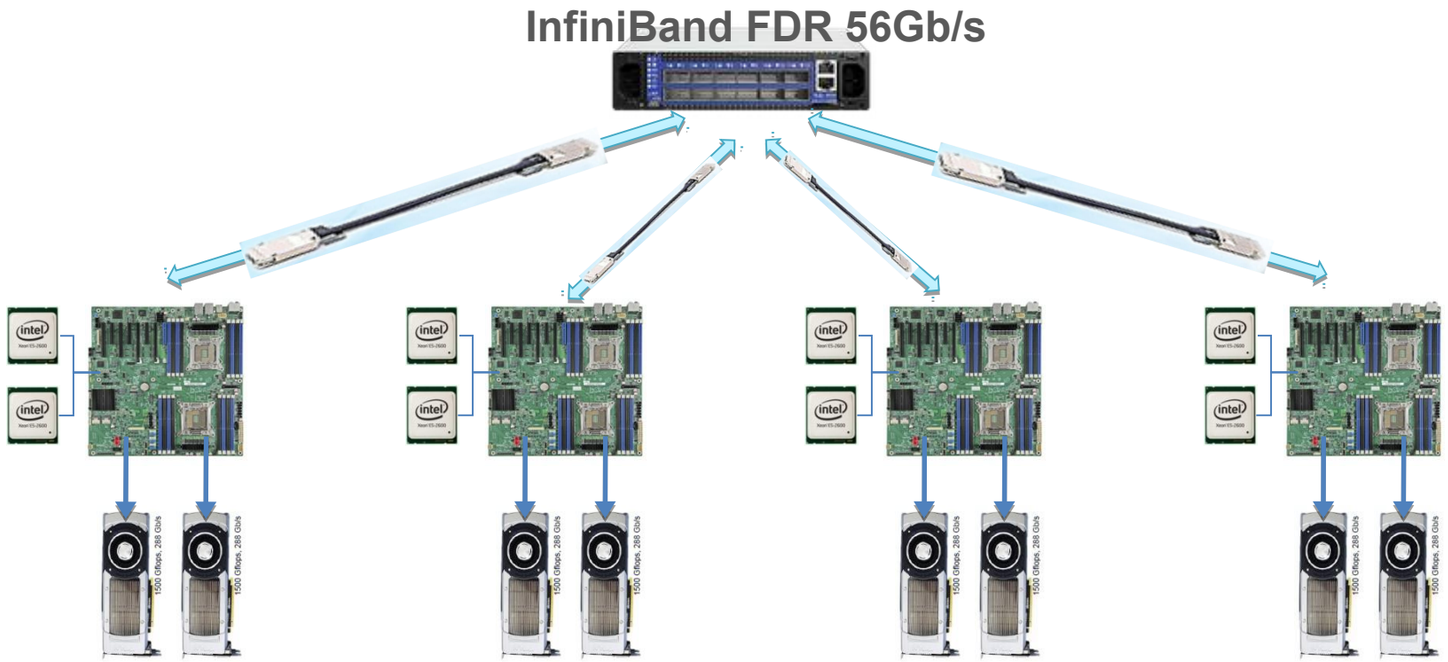
BW = 32 GB/s

DMEM = 6 GB

DPEAK = 1500 GFLOP/s (DP)

bw = 288 GB/s

Example 2: the ISR hybrid computing cluster of 4 nodes



K = 8
DMEM = 6 GB
DPEAK = 1500 GFLOP/s (DP)
bw = 288 GB/s

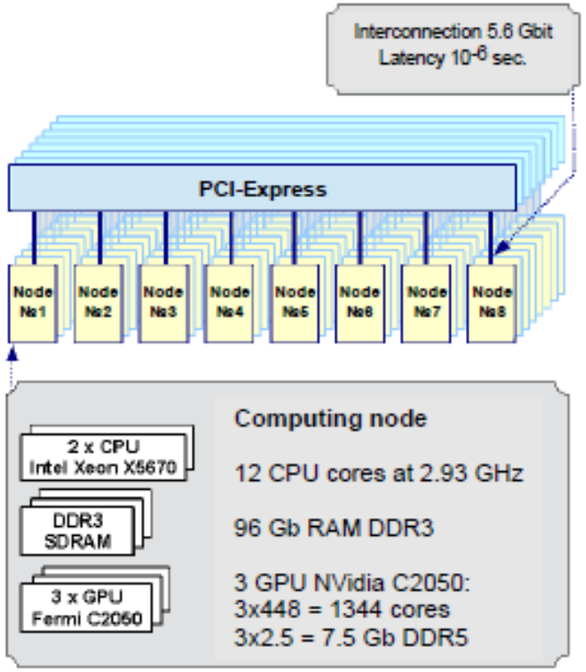
BW = 6 GB/s

HMEM = 512 GB
HPEAK = 13200 GFLOP/s



Example 3: the K100 supercomputer (Keldysh Institiute of Applied Mathematics, Russian Academy of Sciences)

www.kiam.ru



64 nodes

For each node:

K = 3

HMEM = 96 GB

BW = 16 GB/s

DMEM = 2,5 GB

DPEAK = 515 GFLOP/s (DP)

bw = 144 GB/s

- nVidia Tesla C2050

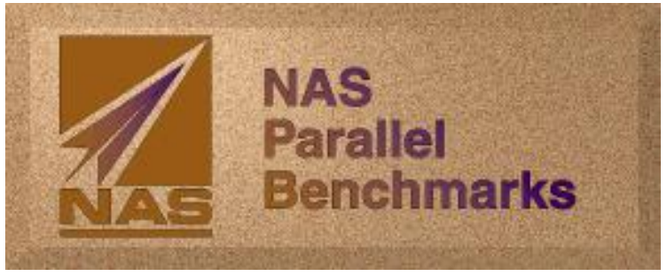
515 Gflops, 144 Gb/s

NAS Parallel Benchmarks

The NAS Parallel Benchmarks

Numerical Aerodynamics Simulation Parallel Benchmarks (NPB)

https://www.nas.nasa.gov/publications/npb.html



A set of benchmarks for the performance evaluation of highly parallel supercomputers.
Developed at NASA, derived from computational fluid dynamics (CFD) applications.
1994 – 2012 – …

Key features:

- «Paper-and-pencil» algorithm specification;
- Restrictions on programming languages and tools;
- Given input data and reference results;
- Double-precision floating-point arithmetic.

NAS PB contents

The «kernel» benchmarks:

1) IS – large integer sort;
2) EP – «embarrassingly parallel» kernel;
3) CG – conjugate gradient method;
4) MG – multigrid method;
5) FT – Fast Fourier Transform.

Pseudo-applications: a simplified CFD application – numerical solution of a system of 5 nonlinear partial differential equations, using the:

1) BT – block tri-diagonal solver;
2) SP – scalar penta-diagonal solver;
3) LU – Lower-Upper Gauss-Seidel solver.

Later extensions:

- Multi-zone applications;
- Unstructured computations, parallel I/O, data movement;
- Computational grid tests.

NAS PB task classes

S – small tasks, for quick test purposes;
W – «workstation» size (90's);
A, *B*, *C* – standard test problems;
D, *E*, *F* – large test problems.

Test	Parameter	Class S	Class W	Class A	Class B	Class C	Class D	Class E
CG	no. of rows	1400	7000	14000	75000	150000	1500000	9000000
	data volume	0.7 MB / 10 KB	4 MB / 55 KB	15 MB / 110 KB	112 MB / 0.6 MB	293 MB / 1.2 MB	5.4 GB / 11 MB	49 GB / 69 MB
	no. of iterations	15	15	15	75	75	100	100
MG	grid size	32 x 32 x 32	128 x 128 x 128	256 x 256 x 256	256 x 256 x 256	512 x 512 x 512	1024 x 1024 x 1024	2048 x 2048 x 2048
	data volume	0.25 MB	16 MB	128 MB	128 MB	1 GB	8 GB	64 GB
	no. of iterations	4	4	4	20	20	50	50

NAS PB sample codes

Reference implementations of the benchmarks, to aid the programmer.

Versions:

- FORTRAN (mostly), C
- Serial, MPI, OpenMP
- SMP, HPF, Java (earlier releases)

Available at: `https://www.nas.nasa.gov/cgi-bin/software/start`
Latest release: **NPB 3.3.1**.

Basic optimization methods for a hybrid system

Necessary (but not sufficient!) steps to develop an effective computing procedure for the discussed hybrid system:

1)Develop effective computing kernels for the coprocessor.

2)Minimize data transfers to/from coprocessors.

3)Hide the remaining transfers behind computations on the coprocessors, where possible.

Any specific complex algorithm needs some research of possibilities for its efficient implementation, prior to actual programming.

For simplicity, one or several base operation can be chosen.

NAS PB FT: Fourier Transform

The NAS PB Fourier Transform (FT): statement of problem

Numerically solve the partial differential equation using the Discrete Fourier Transform.

$$\frac{\partial u(x,t)}{\partial t} = \alpha \nabla^2 u(x,t) \quad x \in \Re^3$$

The essence of many «spectral» codes. Tests long-distance communication performance.
Base operation: 3D complex FFT.

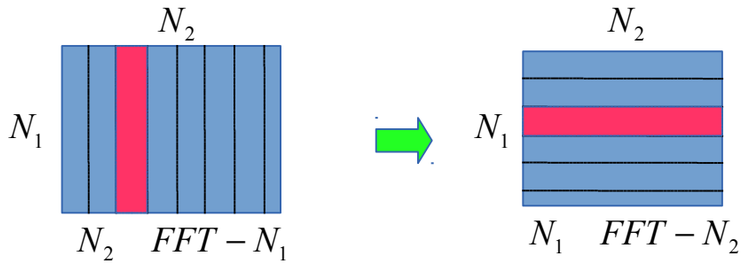
$N = 2^n$

$y = FFT(x)$:

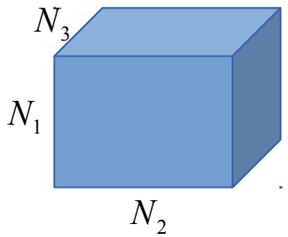
$$y_k = \sum_{j=0}^{N-1} x_j e^{-\frac{2\pi kj}{N} i}, \quad k = \overline{0, N-1}$$

– the Cooley-Tukey algorithm

$N_1 \times N_2$



$N_1 \times N_2 \times N_3$

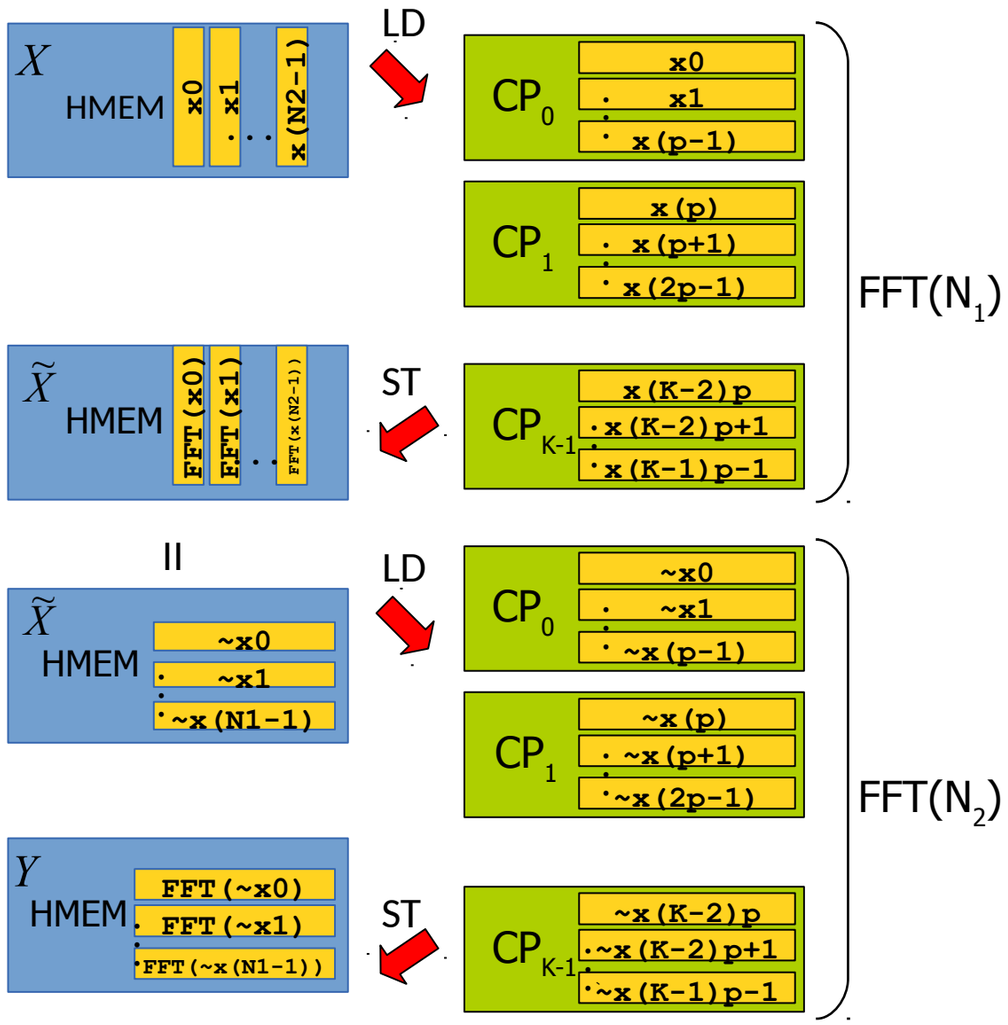


$N_2 \times N_3$ $FFT - N_1$

$N_1 \times N_3$ $FFT - N_2$

$N_1 \times N_2$ $FFT - N_3$

NAS PB FT: 2D computation scheme for the hybrid system, with data placed in HMEM



1) $LOAD(N_1)_0$

2) $FFT(N_1)_0$ | $LOAD(N_1)_1$

(3) $STORE(N_1)_0$ | $FFT(N_1)_2$
$LOAD(N_1)_2$) *loop*

4) $FFT(N_1)_{P1-1}$ | $STORE(N_1)_{P1-2}$

5) $STORE(N_1)_{P1-1}$

---

6) $LOAD(N_2)_0$

7) $FFT(N_2)_0$ | $LOAD(N_2)_1$

(8) $STORE(N_2)_0$ | $FFT(N_2)_1$
$LOAD(N_2)_2$) *loop*

9) $FFT(N_2)_{P2-1}$ | $STORE(N_2)_{P2-2}$

10) $STORE(N_2)_{P2-1}$

NAS PB FT: theoretical performance estimates

Restrictions:

1)

HMEM, DMEM, BW, ... / `sizeof(datatype)`

I. $2N_1N_2 < HMEM$ (data in HMEM)

II. $N_1 N_2 < HMEM$, $K \cdot N_1 N_2 < K \cdot DMEM$ (data in coprocessor memory)

2) $N_1, N_2 < DMEM$

3) $Perf_{kern}$ – 1D FFT kernel performance on one coprocessor

---

$$T_{L+S} = \frac{2KN_1}{BW} \quad !! \quad >> \quad T_C = \frac{5N_1 \log N_1}{Perf_{Kern}} \qquad (OP_{CP} = 5N_1 \log N_1)$$

2D:

I. $T = \frac{4N_1N_2}{BW}$

II. $T = \frac{2N_1N_2}{BW}$

3D:

I. $T = \frac{6N_1N_2N_3}{BW}$

II. $T = \frac{4N_1N_2N_3}{BW}$

$$PERF_{FFT} = \frac{OP_{ALL}}{T} \qquad (OP_{ALL} = 5N_1N_2N_3 \log(N_1N_2N_3))$$

NAS PB FT: hypothetical heterogeneous implementation

Class C: $N_1 = N_2 = N_3 = 512$,

4x NVIDIA GeForce GTX TITAN

$Perf_{kern} \approx 0,1$ DPEAK = 150 GFLOP/s

$T_c \approx 1,536 \cdot 10^{-11}$ s << $T_{L+S} \approx 10,24 \cdot 10^{-11}$ s

I) $T \approx 0,2013$ s $PERF_{ALL} \approx$ 90,01 GFLOP/s

II) $T \approx 0,1342$ s $PERF_{ALL} \approx$ 135,02 GFLOP/s

Conclusions:

The 2D and 3D implementation on the discussed hybrid system is unreasonable!

Possibilities:

- small 1D, 2D or 3D FFT usage as a part of more computationally intensive algorithms,
- giving some part of work to coprocessors

Bottleneck: PCI Express data transfers.

NAS PB CG:
Conjugate Gradient method

The NAS PB Conjugate Gradient method (CG): statement of problem

Given a symmetric positive-definite sparse matrix A, of size $n \times n$, with a random pattern of nonzeros, find its largest eigenvalue.

Tests irregular long-distance communications and unstructured matrix-vector multiplication.

The NAS PB paper defines the matrix generation routine (in FORTRAN) and, for each task class:

- the matrix size n;
- the nz parameter of the matrix generation routine;
- eigenvalue shift λ;
- the number of algorithm iterations – IT;
- reference eigenvalue for the generated matrix;
- allowable error.

NAS PB CG: the base operation of the algorithm is SpVM

The base operation of the algorithm (the most time- and resource- consuming) is SpMV.

Sparse matrix format: Sliced ELLpack.

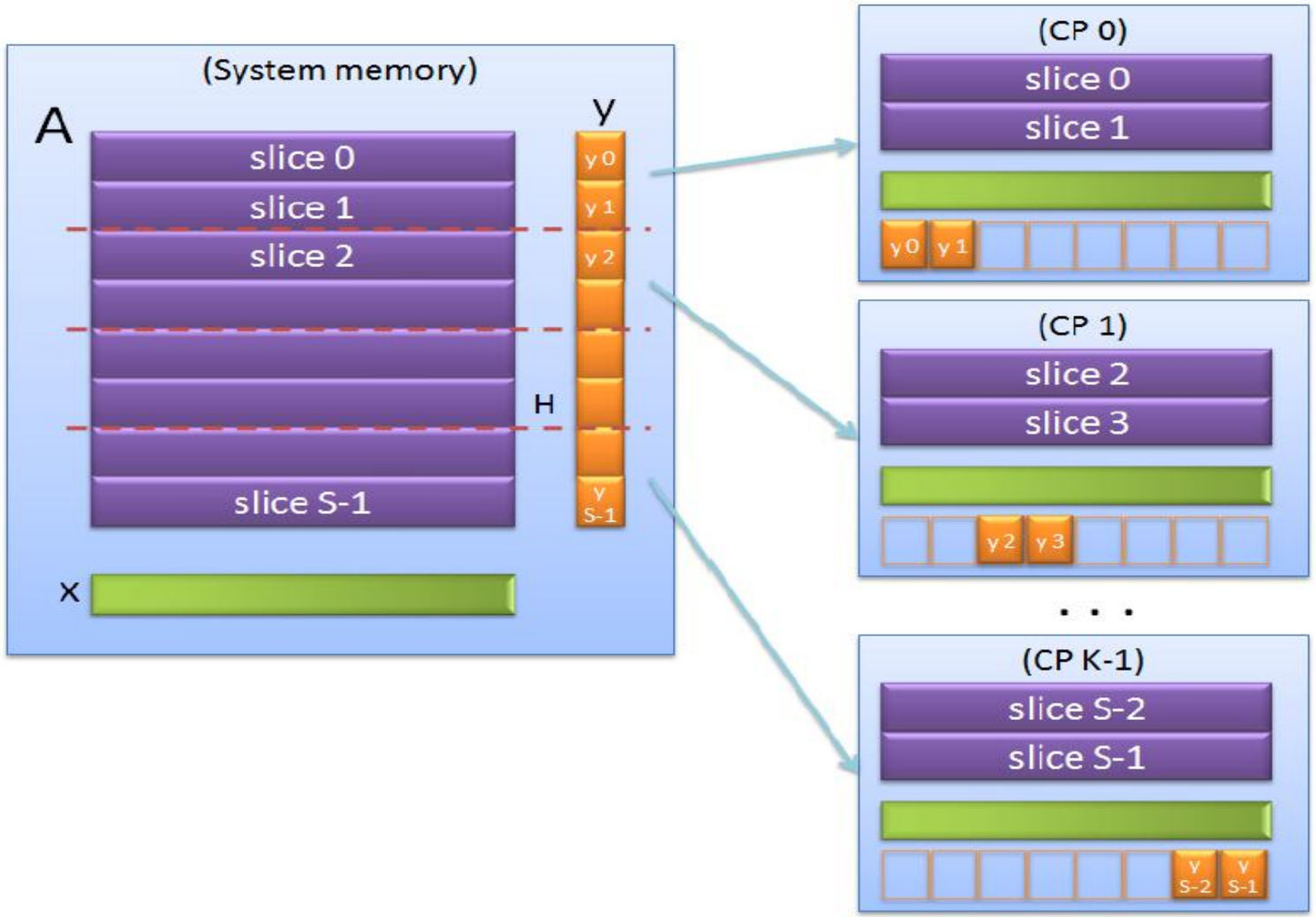
$$A \mapsto (\mathrm{val}, \mathrm{col}, \mathrm{slp})$$

The matrix slices are multiplied by the input vector independently.

Effective parallelization: give several matrix slices (layers) to each coprocessor. Slices are loaded into the coprocessor memory only once, before the computations start.

Coprocessor communications while computing a series of SpMV ($y_i = Ax_i, \quad x_{i+1} = y_i$):
send and receive corresponding output vector fragments.
Computations and data exchange overlap: e.g., process one slice, on each coprocessor, while exchanging previous result fragments.

NAS PB CG: work distribution between coprocessors in the hybrid system



NAS PB CG: preliminary performance estimates

Restriction on problem size: the matrix and all the necessary vectors must fit into total coprocessors memory.

The procedure running time is defined by the largest of two:
- kernel computation time for one slice – T_C;
- *K+1* vector parts exchange time for *y*
  (1 part – store from coprocessor to host memory,
  K parts – load to coprocessor memory) – T_{L+S}.

Notations in formulas:

NZ – the number of matrix nonzero elements, approximately equals the `val` and `col` arrays length.

$Perf_{Kern}$ – kernel performance on one coprocessor.

H – slice width (the number of rows).

S – the number of slices.

NAS PB CG: formulas for performance estimates

$$T_{L+S} \approx \frac{(K+1)H}{BW} \quad ? \quad T_C \approx \frac{2\,NZ}{S \cdot Perf_{kern}}$$

$$T_{CG} \approx K \cdot \frac{S}{K} \cdot T_{L+S} = \frac{(K+1)n}{BW}$$

$$T_{CG} \approx \frac{S}{K} \cdot T_C = \frac{2\,NZ}{K \cdot Perf_{Kern}}$$

$OP_{CG} = 2NZ$ – the number of arithmetical operations

$$PERF_{CG} \approx \frac{2NZ}{T_{CG}} \text{ FLOP/s}$$

$$PERF_{CG} \approx K \cdot Perf_{Kern} \text{ FLOP/s}$$

HMEM, DMEM, BW, ... / `sizeof(datatype)`

NAS PB CG: performance estimates for the test configuration

For our initial kernel, on NVIDIA GeForce GTX TITAN:

$$Perf_{Kern} \approx 11,57 \text{ GFLOP/s} \qquad \text{(class C)}$$

$$T_{L+S} \approx 7,45 \cdot 10^{-8} \text{ s}, T_C \approx 1,66 \cdot 10^{-3} \text{ s}$$

Task class	Theoretical estimate for the whole system (4 x TITAN)	Reference CPU code (2 x Xeon E5-2690v2)
S	46280 MFLOP/s	2260 MFLOP/s
W		13500 MFLOP/s
A		26500 MFLOP/s
B		10100 MFLOP/s
C		9900 MFLOP/s

Conclusion: the CG problem implementation on the discussed target architectures makes sense!

NAS PB CG: the SpMV kernel

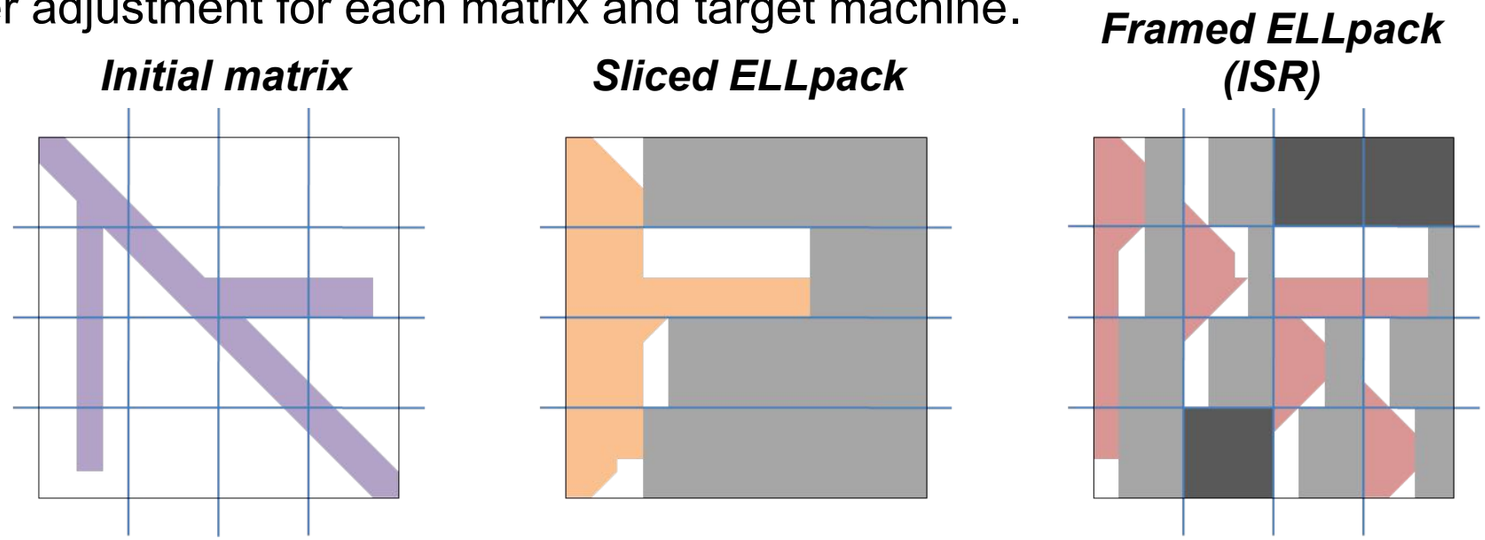
Algorithm bottleneck: random memory access in the SpMV kernel.

$y = Ax$, A – sparse matrix

Sliced ELLpack

+: Serial, aligned access to *y* and *A*.

-: Random access to *x*,
additional zeros,
parameter adjustment for each matrix and target machine.



Framed ELLpack

+: *x* – in LDS, faster random access.

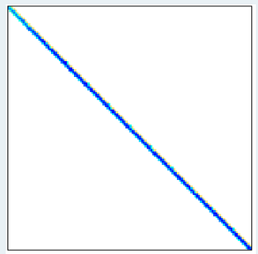
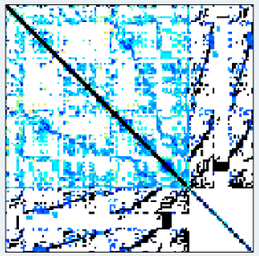
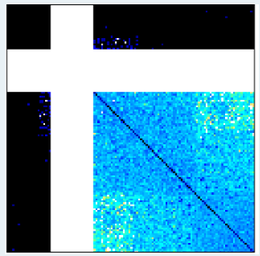
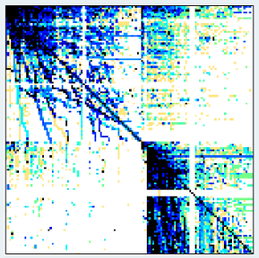
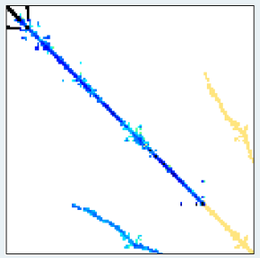
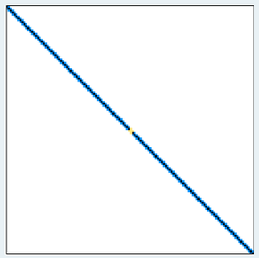
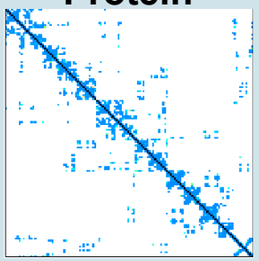
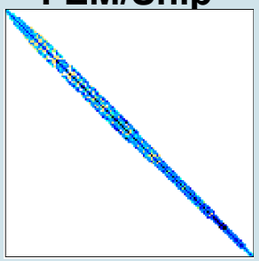
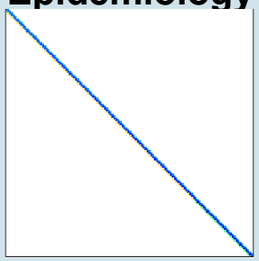
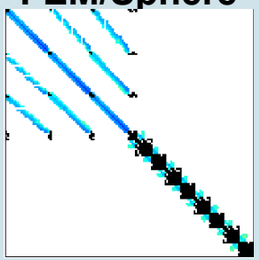
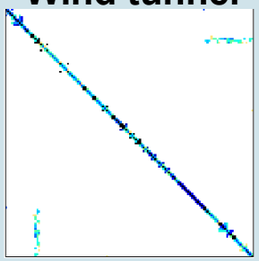
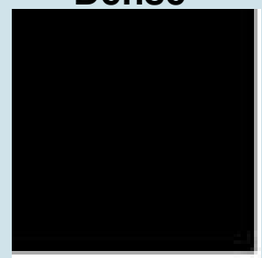
-: More additional zeros.

SpMV procedure testing

The SpMV performance strongly depends on input matrix.

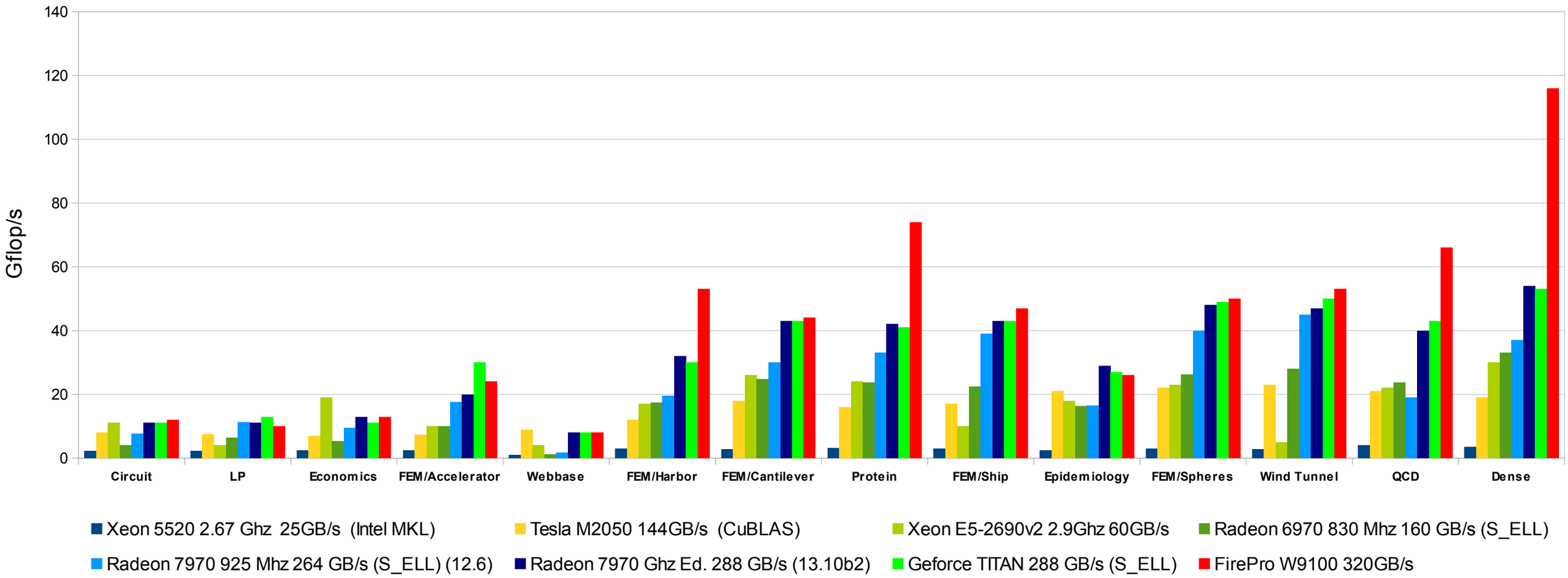
different problems ⟷ different matrix types

Standard matrix set, derived from various real applications.

Economics  *Macroeconomic model*	**LP** *Railways set cover Constraint matrix*	**Circuit**  *Motorola circuit simulation*	**FEM/Accelerator**  *Accelerator cavity design*	**Webbase**  *Web connectivity matrix*	**FEM/Harbor**  *3D CFD of Charleston harbor*	**FEM/Cantilever**  *FEM Cantilever*
Protein  *Protein data bank 1HYS*	**FEM/Ship**  *FEM ship section/detail*	**Epidemiology**  *2D Markov model of epidemic*	**FEM/Sphere**  *FEM concentric spheres*	**Wind tunnel**  *Pressurized wind tunnel*	**QCD**  *Quark propagators (QCD/LGT)*	**Dense**  *Dense matrix in sparse format*

The ISR SpMV procedure performance

The SpMV kernel in OpenCL



(Sliced ELLpack format)

NAS PB CG, ISR implementation: top-level logic, stage I

The main algorithm implementation trick: substitution of NPB reference FORTRAN code parts by our procedures.
No changes in data generation, performance measures and result verification!

Stage I – single coprocessor procedure

- A set of necessary kernels:

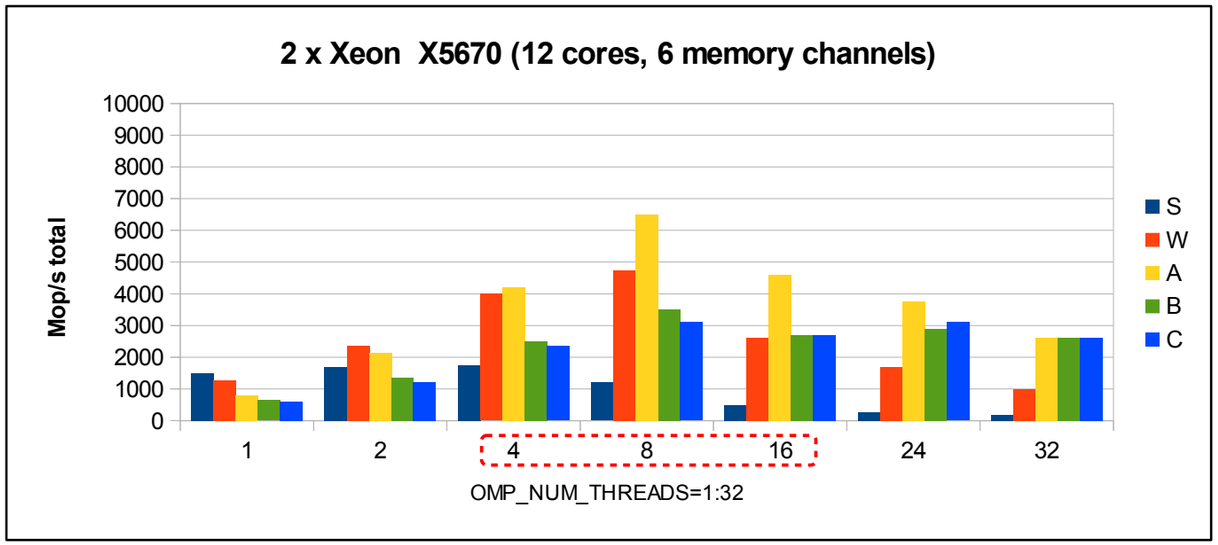
SpMV, vector dot product, addition, …

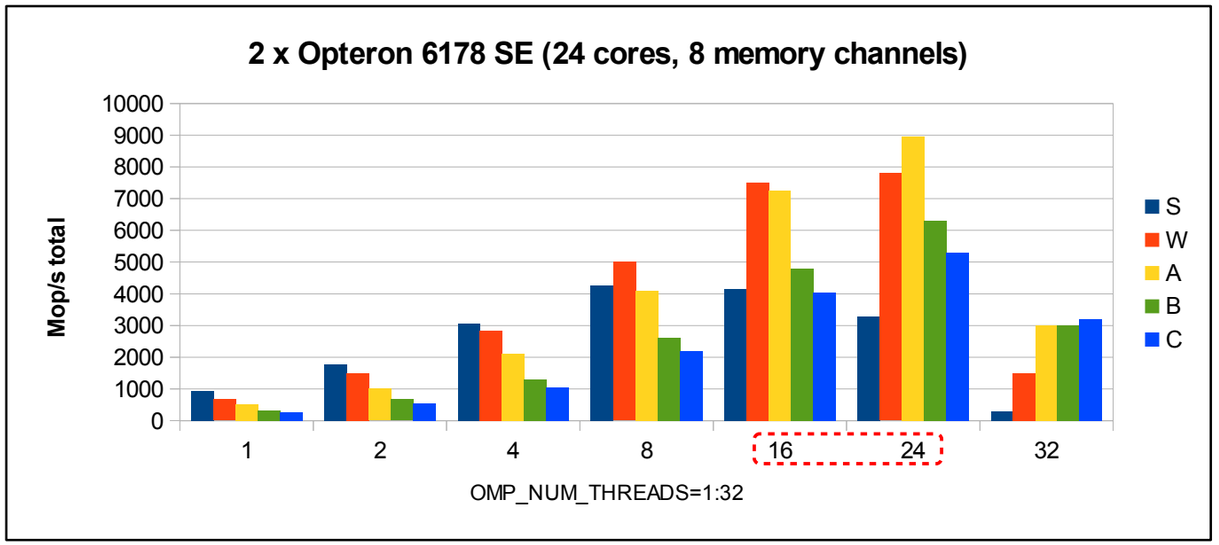
- C code using the ISR Scheduler:

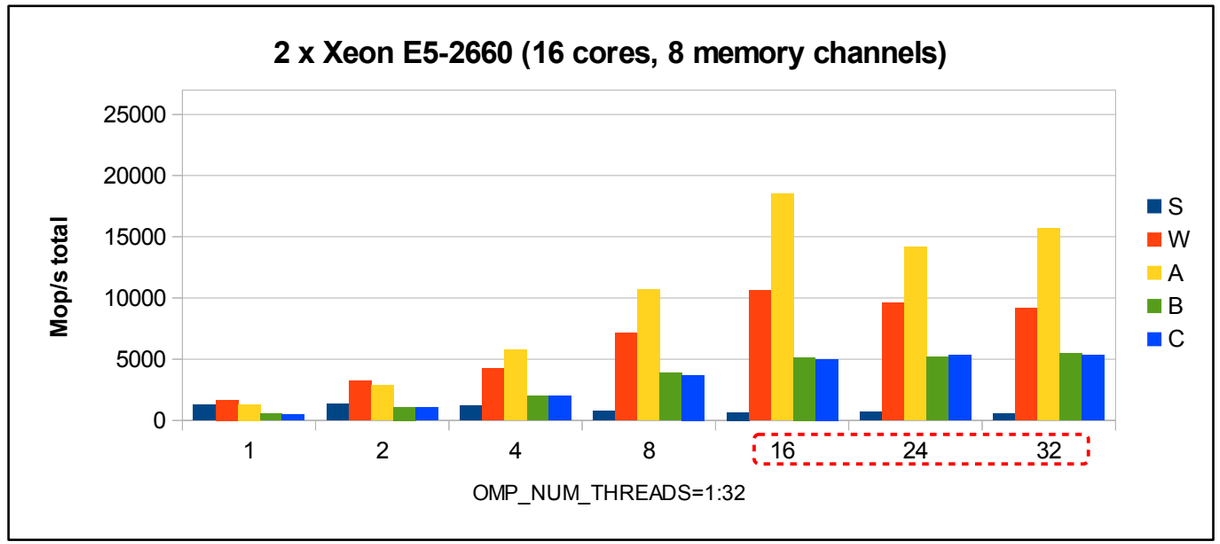
load data into coprocessor memory, call kernels in the required order, store result.

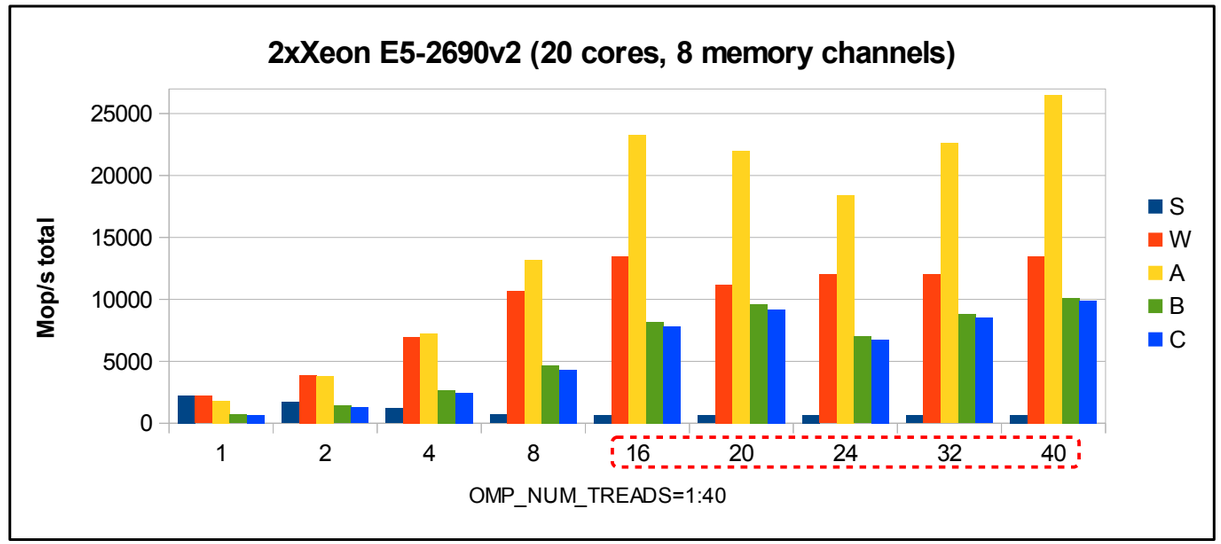
- Call the `sched_init / sched_fini` routines in the reference code.
- Call the `spmv_init` routine – includes matrix repack from CSR to Sliced (Framed) ELLpack format and load into coprocessor memory.
- Substitute the reference CG routine call by our Scheduler CG routine.

NAS PB CG: performance measures for the reference codes

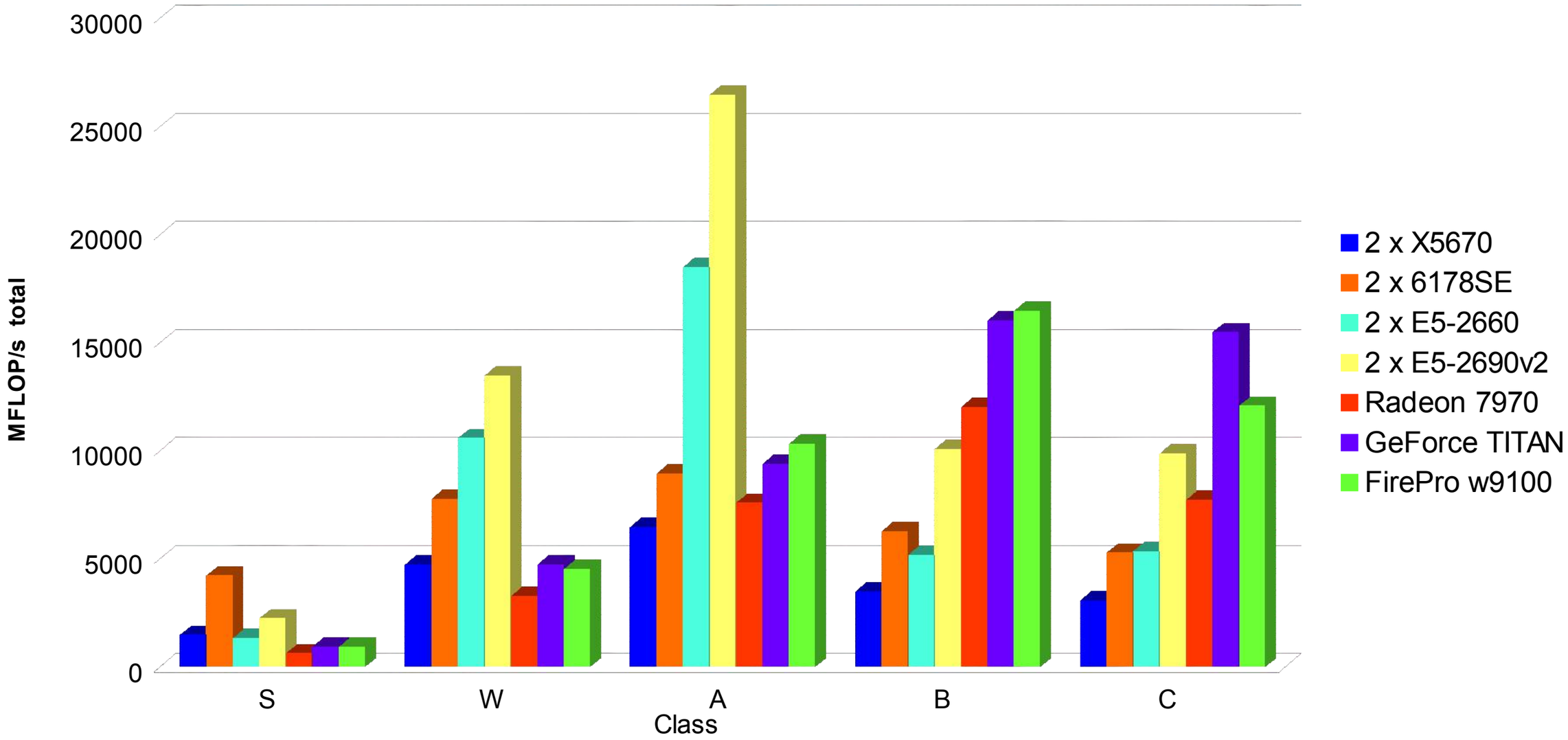








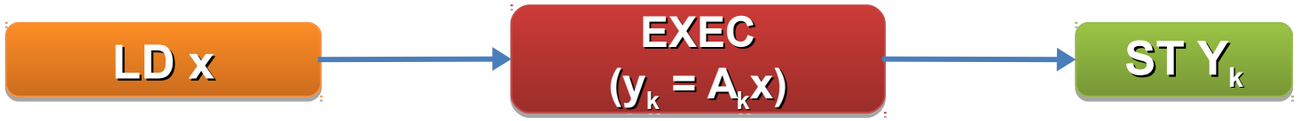
NAS PB CG: comparison of best CPU results and the ISR single-GPU procedure



NAS PB CG, ISR implementation: top-level logic, stages II-IV

Stage II – multi-coprocessor procedure

- The Scheduler procedure: matrix split, call the single-coprocessor SpMV routine for each part.
  One part is a number of matrix slices, in Sliced ELLpack format.
- Consequent SpMV procedure on one coprocessor:



- Substitute the SpMV call in reference CG code by our procedure.
- Other computations – in reference OpenMP-code, with no changes.

Stage III – heterogeneous procedure on a whole node

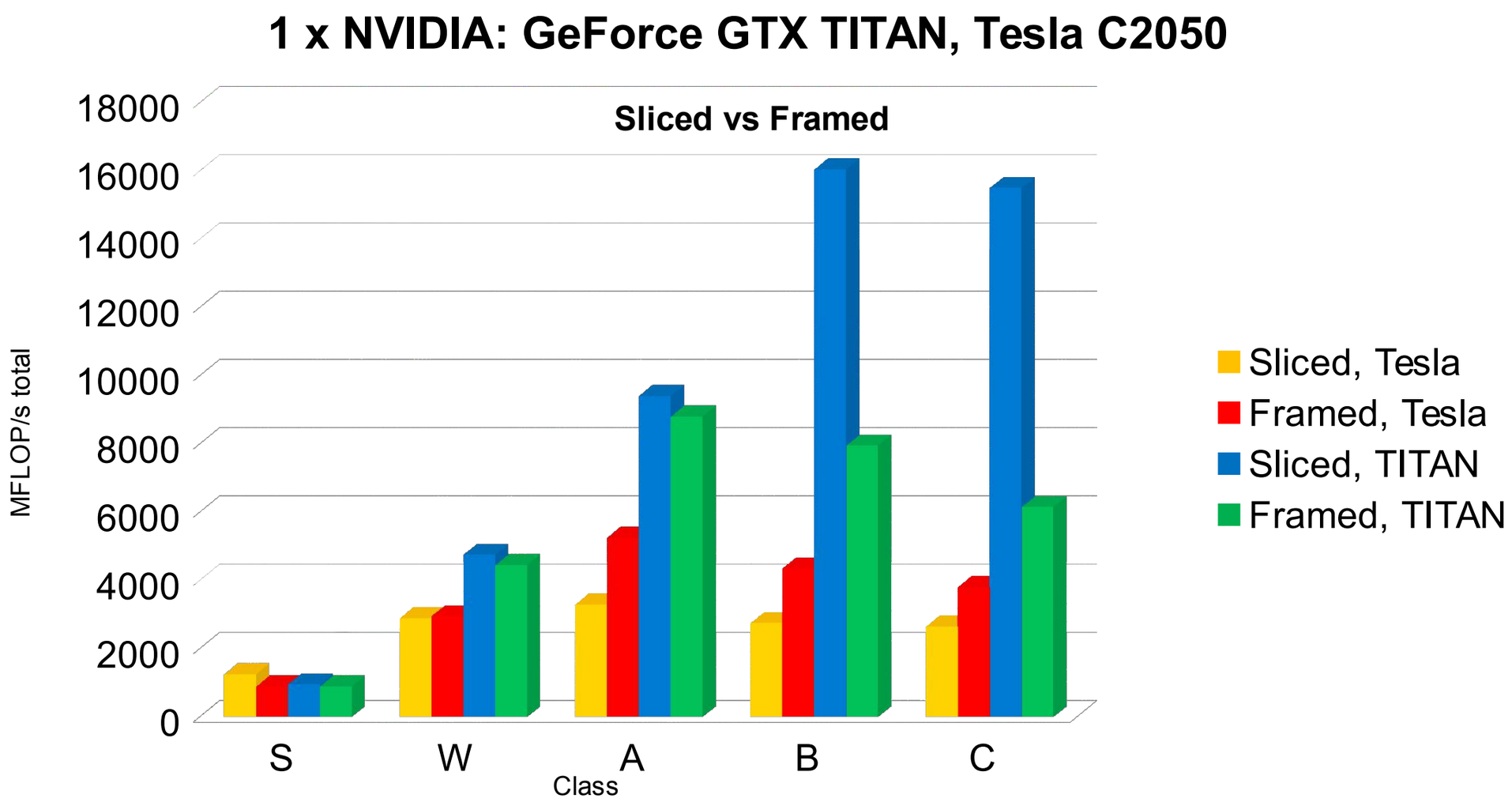
- Add the CPU SpMV routine to the Scheduler procedure.
  CSR matrix format, OpenMP parallelization.
- A new parameter: the relative size of the matrix part processed on CPU.

Stage IV – MPI + OpenMP + GPU version

- Based on the reference MPI version, with integrated OpenMP for each process.
- The Scheduler function is unchanged.

+ : parameter variation for the SpMV kernel

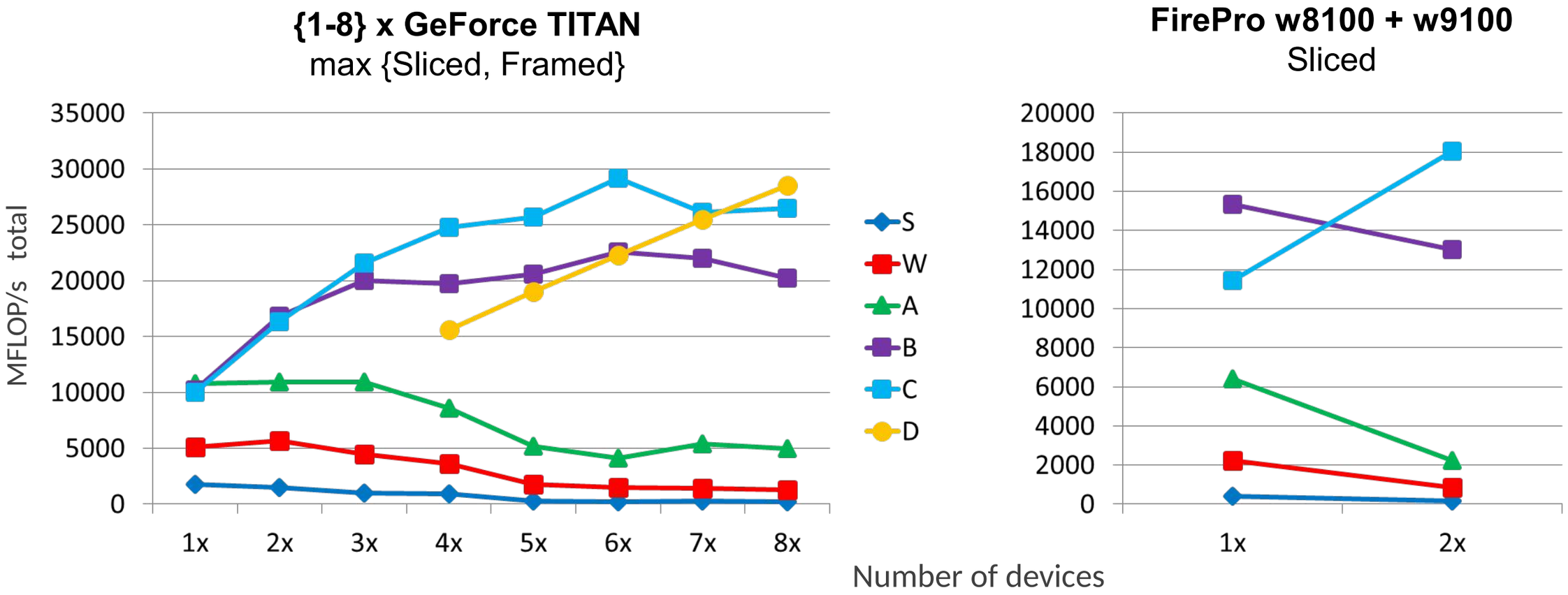
NAS PB CG: comparison of sparse matrix formats



Framed ELLpack format is useful for older GPUs.

Heterogeneous NAS PB CG routine: problem scalability

(Host: 2 x Intel Xeon E5-2690v2 CPU)

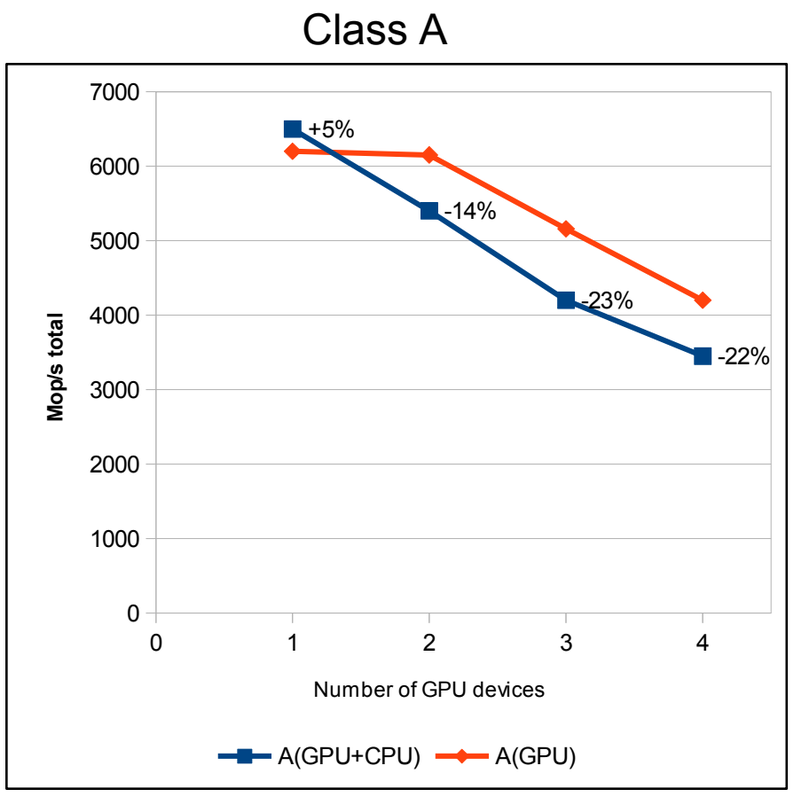


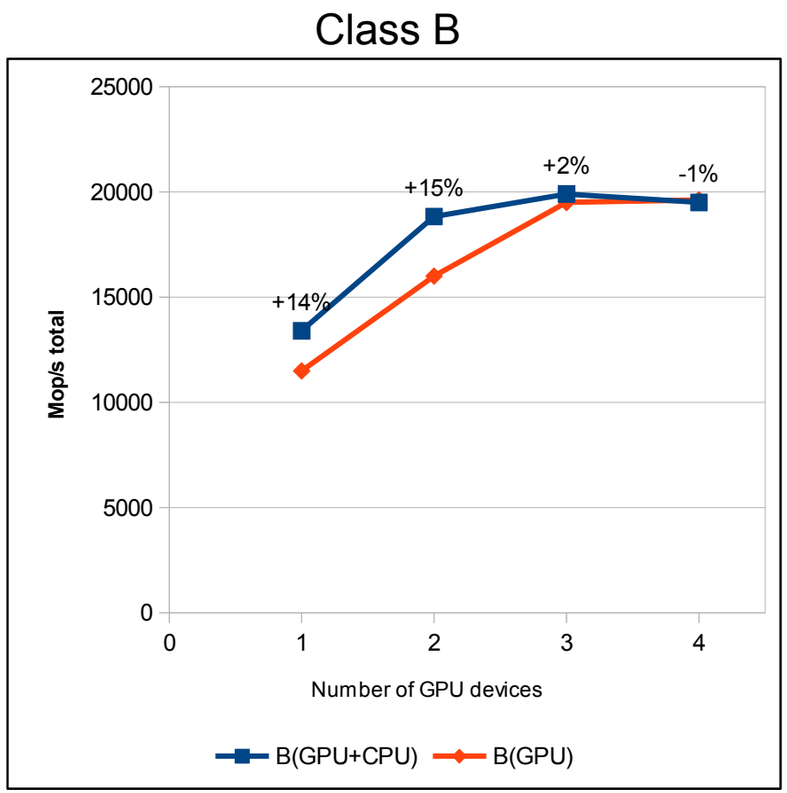
Class D: linear scalability (~ 0.8 x)

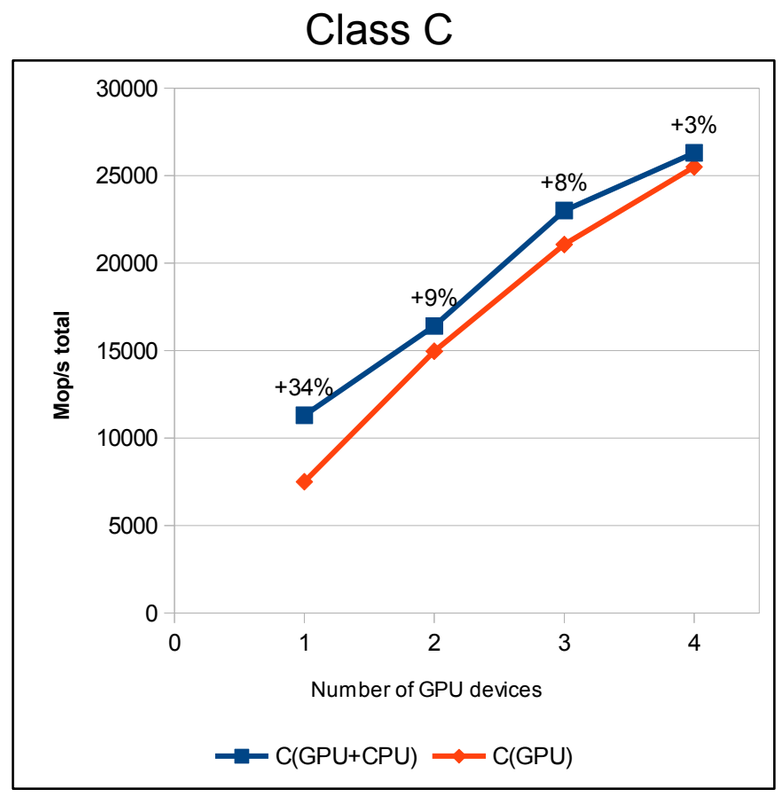
~**5400** MFLOP/s (2 x top server CPU, OpenMP) vs **28500** MFLOP/s (8 x top GPU)

Heterogeneous NAS PB CG routine: CPU+GPU testing results

CPU (2 x Xeon E5-2670) + 4 x AMD Radeon 7970 (Firepro 9000)

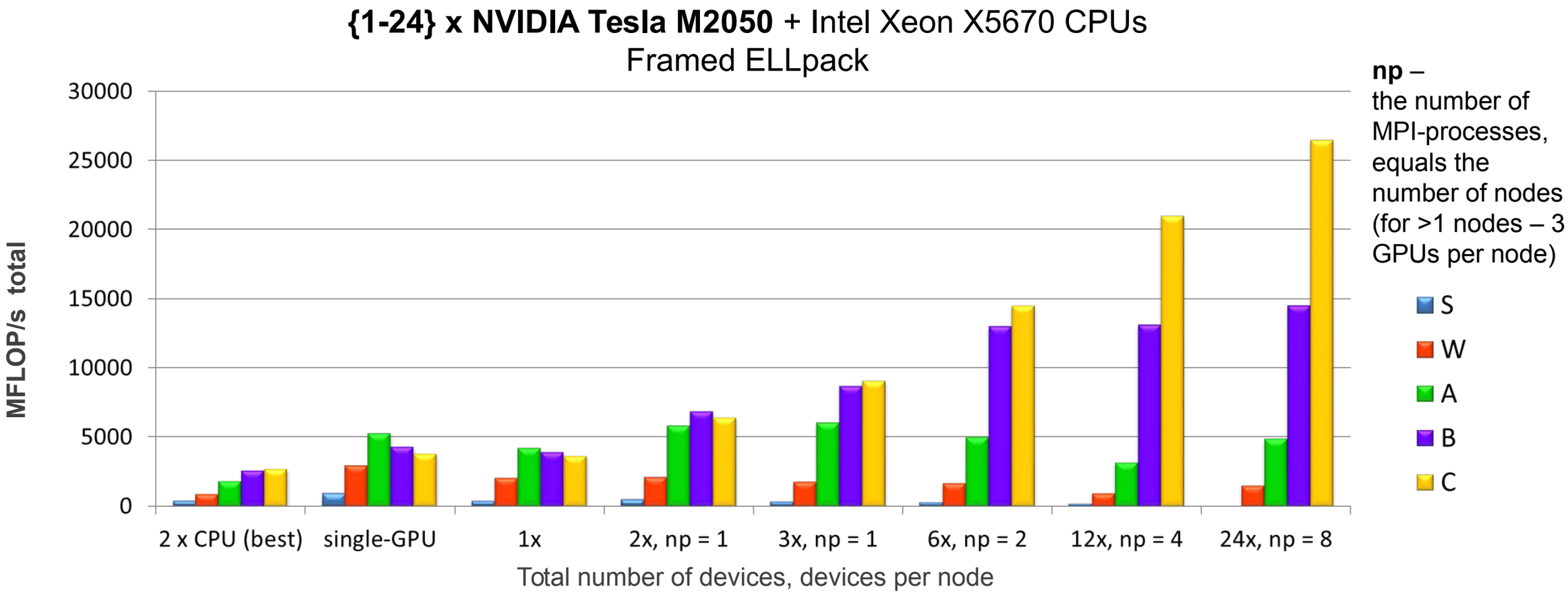






Speed advantage (up to 35%) depends on number of accelerators!

Heterogeneous NAS PB CG routine: testing results for the K100 supercomputer (Keldysh IAM, RAS)



Class C: scalability up to 24 coprocessors.

3 x Tesla – **9,1** GFLOP/s

24 x Tesla – **26,5** GFLOP/s

NAS PB CG: future work

- Computations and data transfer overlap.
- SpMV kernel optimization.
- OpenCL procedure optimization for CPU.

NAS PB MG:
MultiGrid method

The NAS PB Multigrid method (MG): statement of problem

Obtain an approximate solution to the Discrete Poisson problem

$$\nabla^2 u = v$$

on a $N \times N \times N$ grid with periodic boundary conditions..

Test structured long and short distance data communication.

The NAS PB paper defines the v function (1 or -1 in several points, 0 elsewhere), the initial approximation u_0 (zero) and, for each task class:

- the grid size – N;
- the number of algorithm iterations – IT;
- reference solution residual norm;
- allowable error.

NAS PB MG: multigrid method operators

Base operation: **Tlin** – trilinear grid operator.

Coefficients of all the operators are given, independent of the multigrid level.

A , S – keep the N x N x N grid unchanged;

P – reduces the grid: N x N x $N \mapsto N/2$ x $N/2$ x $N/2$;

Q – expands the grid: N x N x $N \mapsto 2N$ x $2N$ x $2N$.

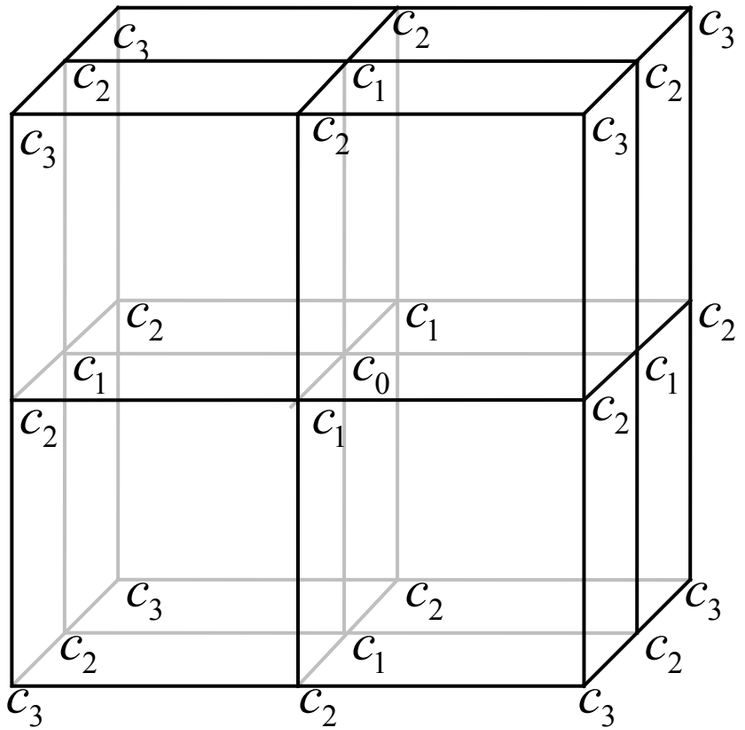
New value of each point – weighted sum of 27 old values.

$$A: \quad c_0 = -\frac{8}{3},\ c_1 = 0,\ c_2 = \frac{1}{6},\ c_3 = \frac{1}{12}$$

$$P: \quad c_0 = \frac{1}{2},\ c_1 = \frac{1}{4},\ c_2 = \frac{1}{8},\ c_3 = \frac{1}{16}$$

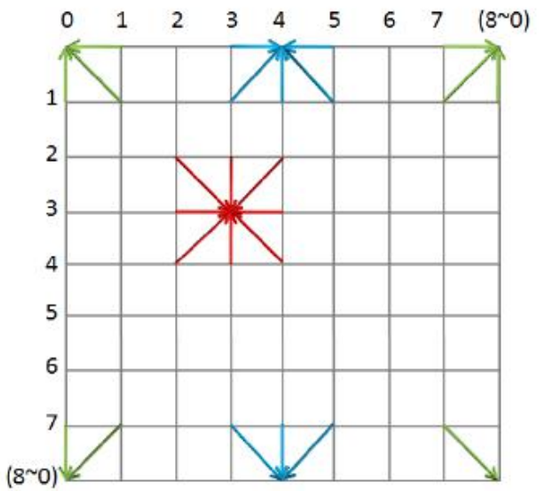
$$Q: \quad c_0 = 1,\ c_1 = \frac{1}{2},\ c_2 = \frac{1}{4},\ c_3 = \frac{1}{8}$$

$$\mathrm{S}: \quad c_0 = -\frac{3}{8},\ c_1 = \frac{1}{32},\ c_2 = -\frac{1}{64},\ c_3 = 0$$

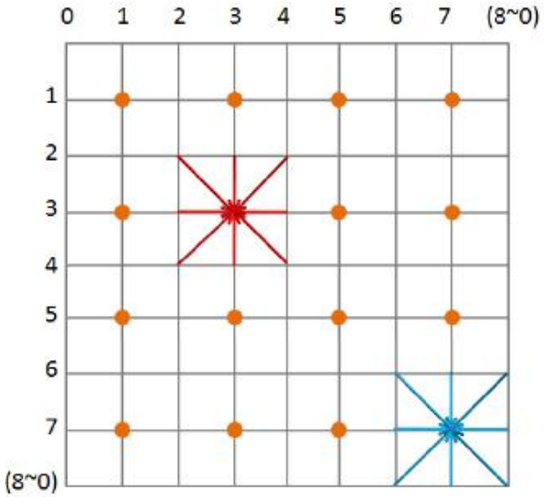


NAS PB MG: taking boundary conditions into account

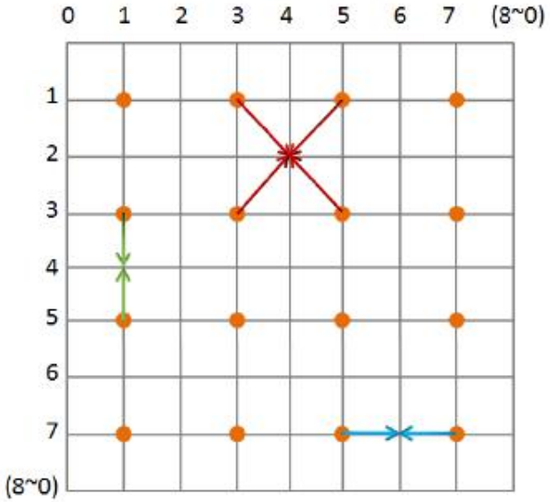
Boundary conditions: $u_N = u_1,\ u_0 = u_{N-1}$ for each dimension.



A, S



P

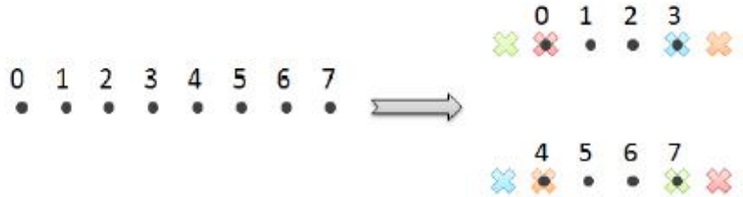


Q

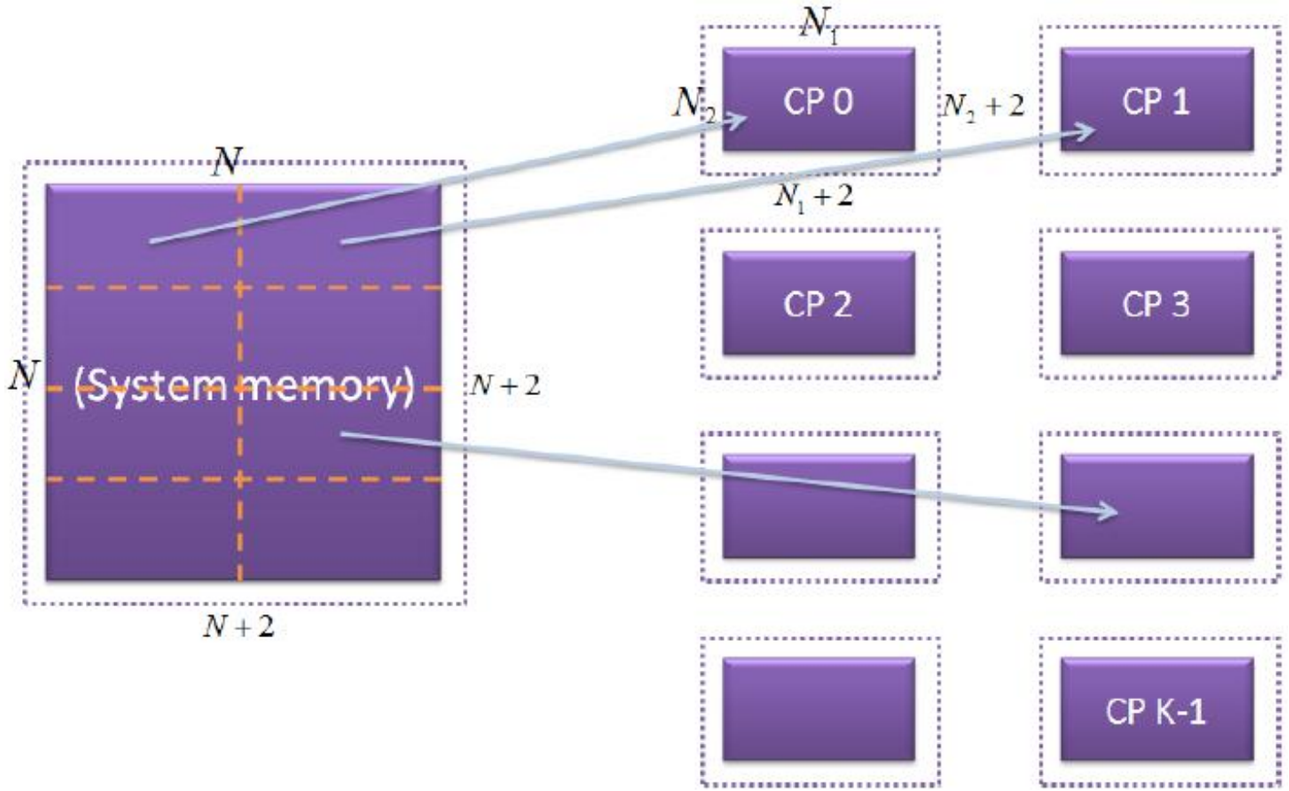
Simplified Tlin: all the coefficients are nonzero, the grid is unchanged.
Example: residual on the finest grid, takes about 90% of the total computation time.

NAS PB MG: work distribution between coprocessors on a hybrid system

Bisection, one dimension:



After each Tlin on coprocessors – boundary exchange!

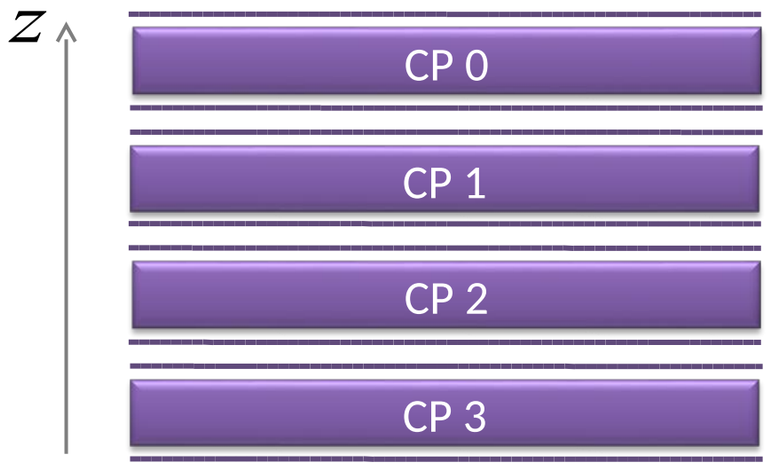


NAS PB MG: types of array split

$$N = 2^n,\ K = 2^m$$

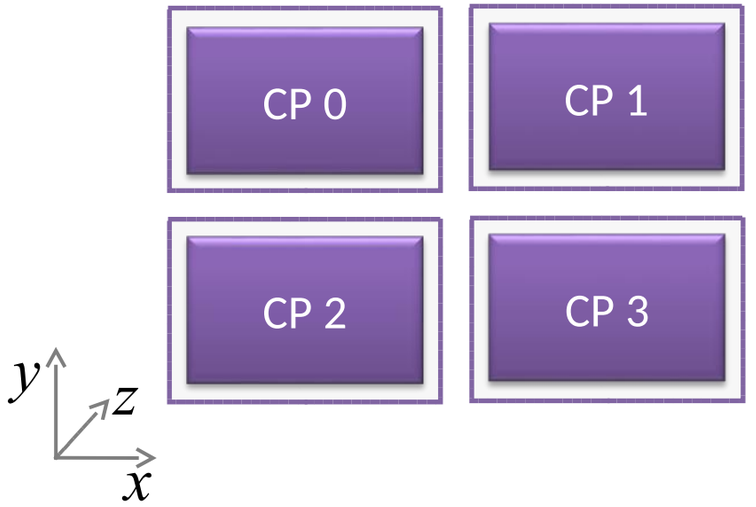
In which directions should we split the array?

I *by z*



- well suited for memory access on GPU
- simple data exchange scheme
- non-power-of-2 number of coprocessors is supported

II *by z, y, x in turns*



- minimal amount of data to exchange
- more MG levels on each CU
- implemented in reference MPI-code

NAS PB MG: preliminary performance estimates

Block size: $2^{n_1} \times 2^{n_2} \times 2^{n_3} = N_1 \times N_2 \times N_3$
The number of boundary points: $\sim 2(N_1 N_2 + N_2 N_3 + N_1 N_3)$

Problem size restriction: two blocks (input and output) with all the boundaries must fit into one coprocessor memory, at least.

The procedure running time is defined by the largest of two:
- kernel Tlin computation time, for block inner points, on each coprocessor – T_{CI} ;
- boundary exchange time, between all the coprocessors – T_{SB} .

Notations in formulas::
T_{CB} – kernel computations time for block boundary points,
$Perf_{Kern}$ – kernel performance on one coprocessor.

NAS PB MG: formulas for performance estimates

$$T_{SB} \approx \frac{4K(N_1N_2 + N_2N_3 + N_1N_3)}{BW} \quad ? \quad T_{CI} \approx \frac{30N_1N_2N_3}{Perf_{Kern}}$$

$$T_{MG} = T_{CB} + T_{SB} \approx \left(\frac{60}{Perf_{Kern}} + \frac{4K}{BW}\right) \cdot (N_1N_2 + N_2N_3 + N_1N_3)$$

$$T_{MG} \approx \frac{30N^3}{K \cdot Perf_{Kern}}$$

$OP_{MG} = 30N^3$ – the number of arithmetical operations

$$PERF_{MG} \approx \frac{30N^3}{T_{MG}} \text{ FLOP/s}$$

$$PERF_{MG} \approx K \cdot Perf_{Kern} \text{ FLOP/s}$$

HMEM, DMEM, BW, ... / `sizeof(datatype)`

NAS PB MG: performance estimates for the test configuration

For our initial kernel, on GeForce TITAN:

$$Perf_{Kern} \approx 128,42 \text{ GFLOP/s} \quad \text{(class B)}$$

$$T_{CI} \approx 9,80 \cdot 10^{-4} \text{s}, T_{SB} \approx 3,05 \cdot 10^{-4} \text{s}$$

Task class	Theoretical estimate for the whole system (4 x TITAN)	Reference CPU code (2 x Xeon E5-2660)
S	513680 MFLOP/s	4300 MFLOP/s
W		18150 MFLOP/s
A		19100 MFLOP/s
B		19500 MFLOP/s

Conclusion: the MG problem implementation on the discussed architectures makes sense!

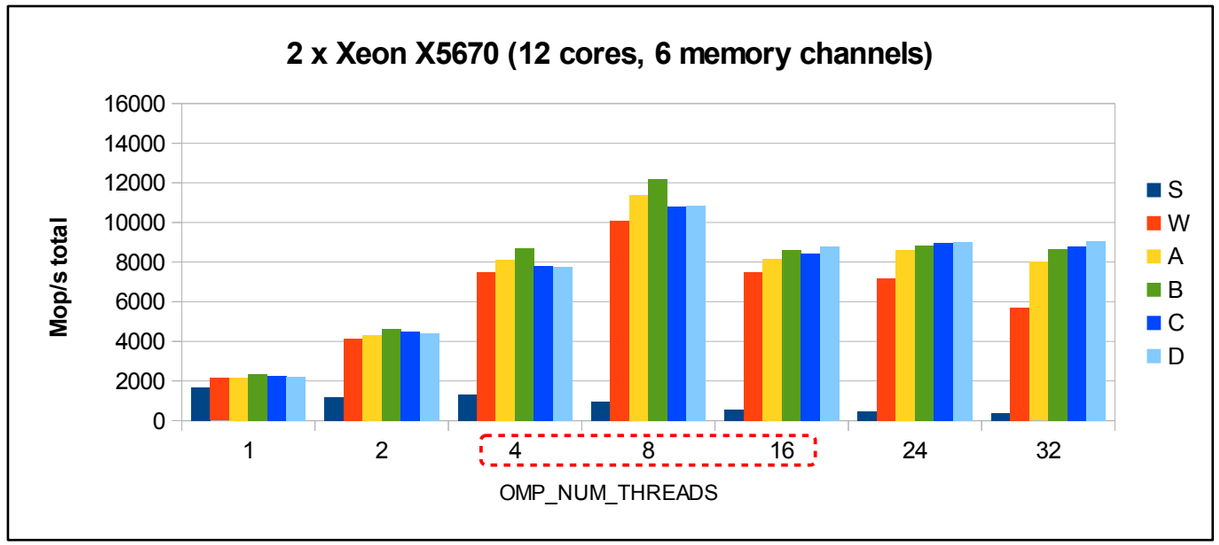
NAS PB MG, ISR implementation: top-level logic, stage I

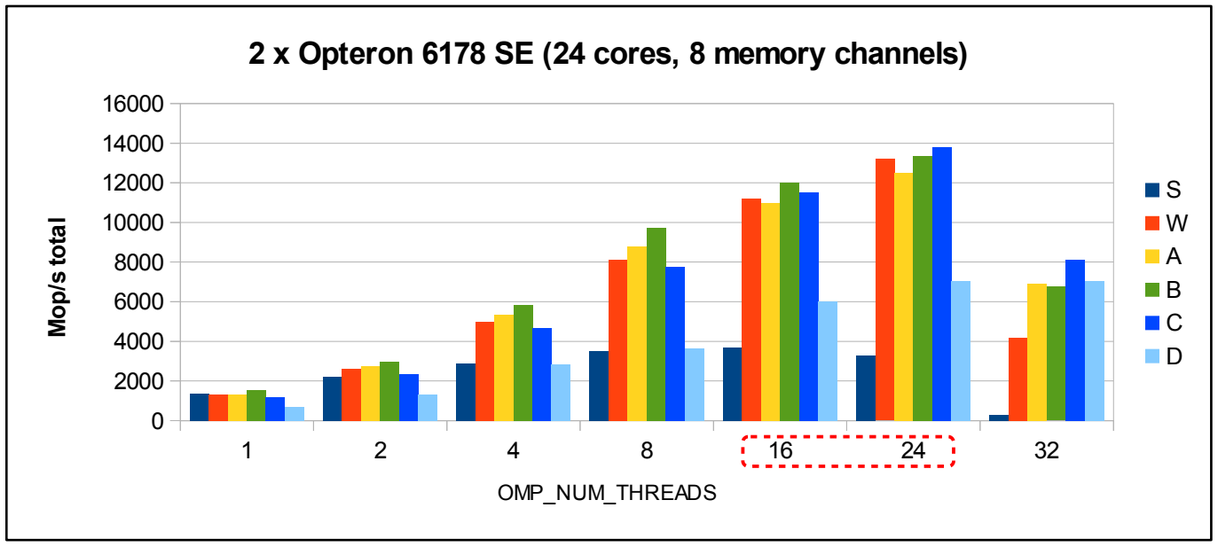
The main algorithm implementation trick: substitution of NPB reference FORTRAN code parts by our procedures.
No changes in data generation, performance measures and result verification!

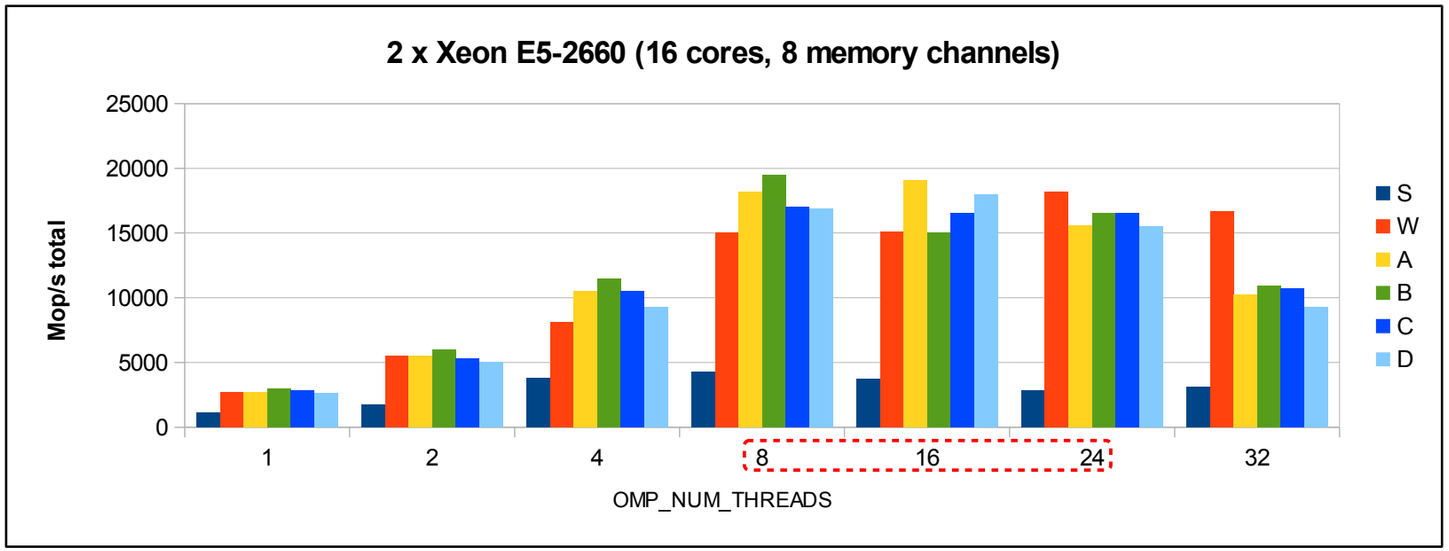
Stage I – single coprocessor procedure

- A set of necessary kernels:
  residual, restriction (projection), interpolation, smoothing, norm computation.
- Additional kernels: copy boundary planes after each Tlin
  (looking forward to array split by z).
- C code using the ISR Scheduler:
  load data into coprocessor memory, incl. additional boundary planes,
  kernel calls in the required order, result store.
- Call the `sched_init / sched_fini` routines in the reference code.
- Substitute the reference MG routine call by our Scheduler MG routine.

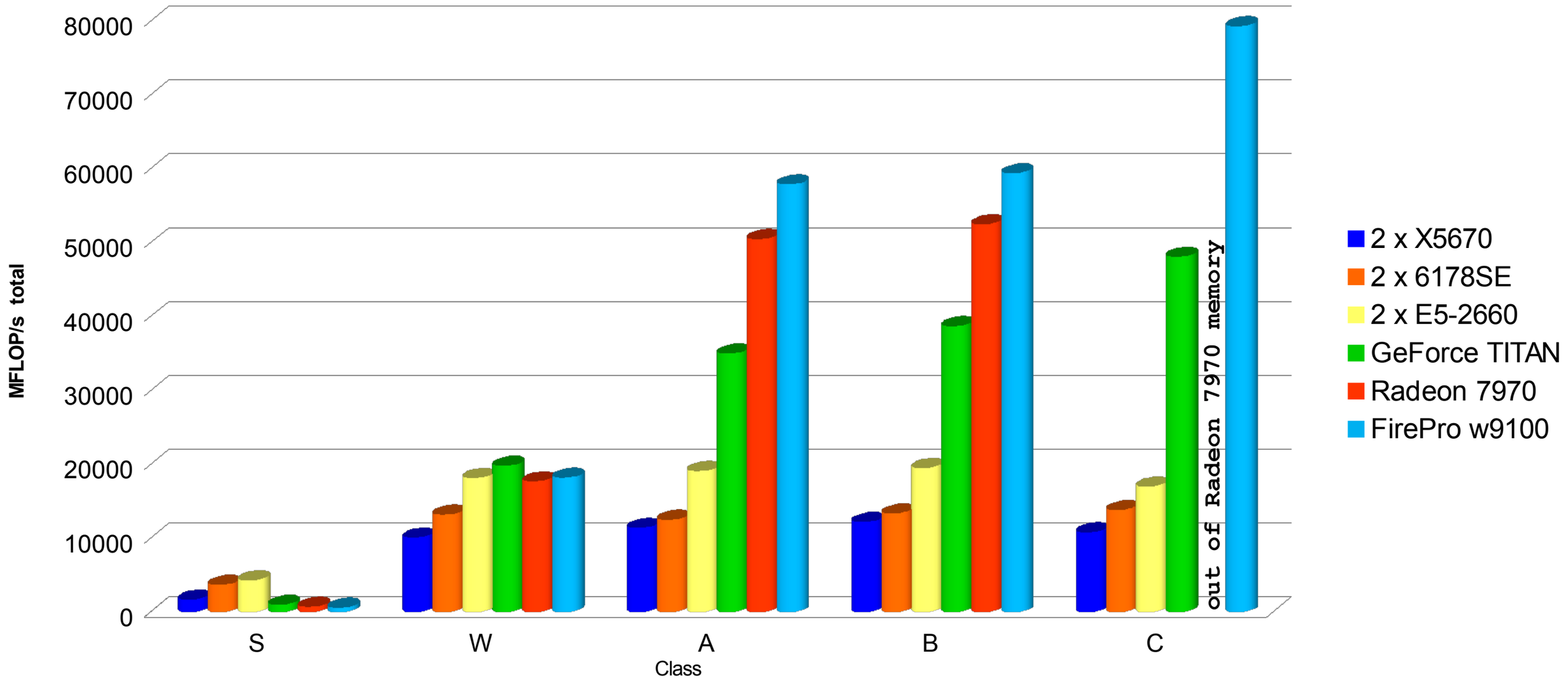
NAS PB MG: performance measures for reference codes







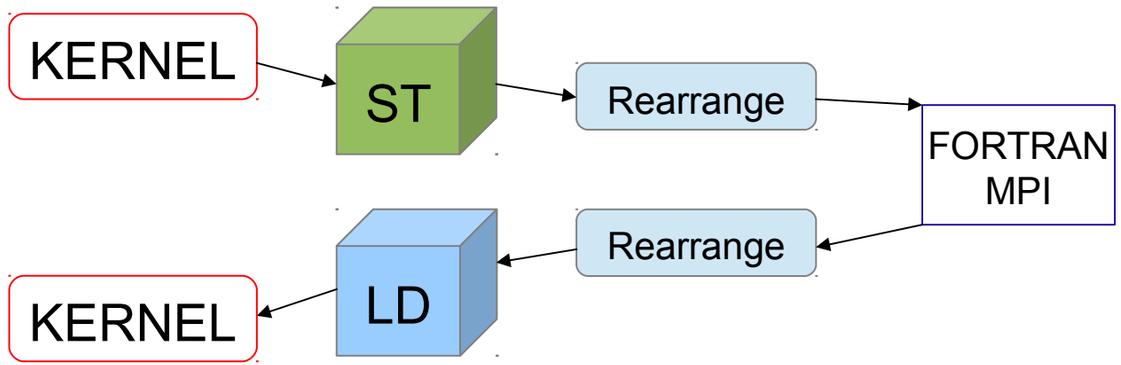
NAS PB MG: comparison of best CPU results and the ISR single-GPU procedure



NAS PB MG, ISR implementation: top-level logic, stage II

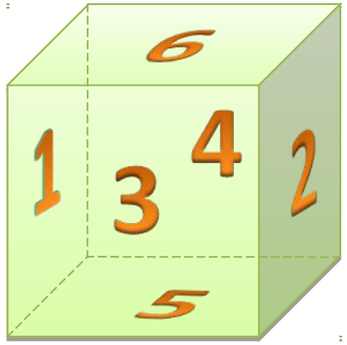
Stage II – synchronous multi-coprocessor implementation

- Base: reference MPI implementation.
- Array split: by *z, y, x*, in reference code.
  Consequence: kernel modification, 6 additional boundary planes.
- One MPI-process rules one coprocessor.
- Swap boundaries (reference implementation) after each Tlin.
- The Scheduler procedure for each MPI-process: load one array block, call a Tlin kernel, store output block.
  After store/before load: rearrange boundary data into continuous memory area, as required by the reference code.
- Substitute Tlin calls in reference MG code by our procedure.



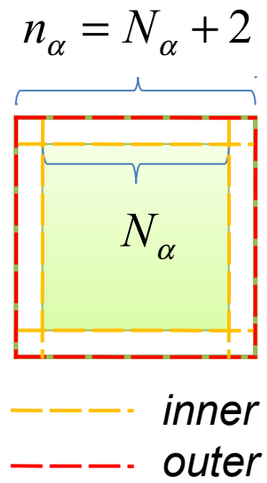
NAS PB MG, ISR implementation: boundary exchange optimization

Boundary planes numeration:



Load/store of boundaries 1, 2, 3, 4 is ineffective: separate lines or points with strides (5 and 6 – continuous in memory).
Load/store the whole block – too expensive!

Solution: 3 additional kernels (`give_x/y/z_kern`) copy pairs of boundary planes into a continuous buffer, to store them.
Kernels `take_x/y/z_kern` - copy boundaries received from other CPs into the 3D block.
Receive: «outer» boundaries ($n_\alpha \times n_\beta$).
Send: «inner» boundaries ($N_\alpha \times N_\beta$), zero-padded to form $n_\alpha \times n_\beta$ arrays – for data placement uniformity.



The running time is negligible, compared to computation and data load/store time.

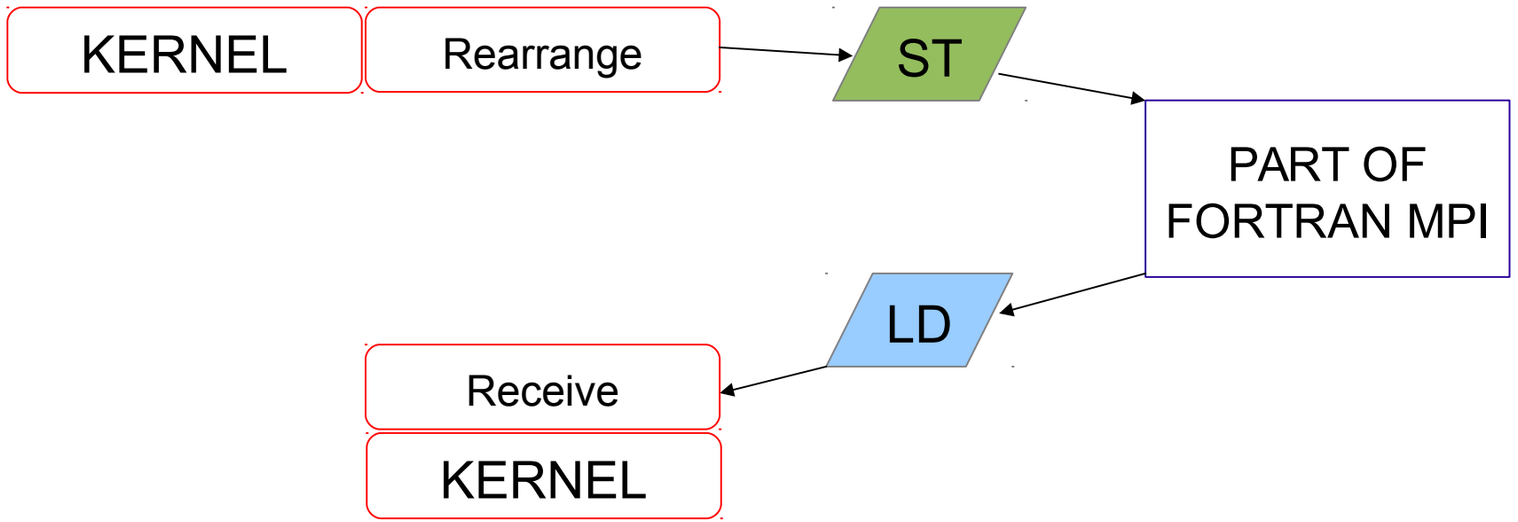
NAS PB MG, ISR implementation: top-level logic, stage III

Stage III – data transfer optimizaton

- Load and store only boundary planes.
- Before store and after load rearrange data on coprocessors.

New additional kernels.

- No need for rearrangement on CPU: call MPI-exchanges directly.
- Port MPI-exchanges to our procedure from the reference code.

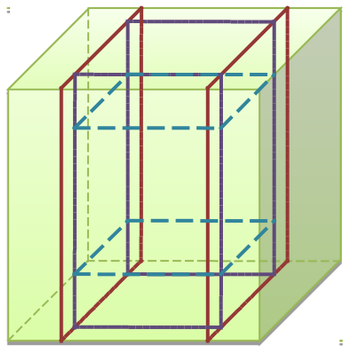


NAS PB MG, ISR implementation: multi-bufferization

Boundary data exchange and computations overlap.

$bwidth$ – a new parameter: the width of boundary blocks. Usually equals 2.

Split block into 7 parts (subblocks).
Part M – the block including the M-th boundary plane,
part 7 – inner part of the block,
part 0 – the whole block.



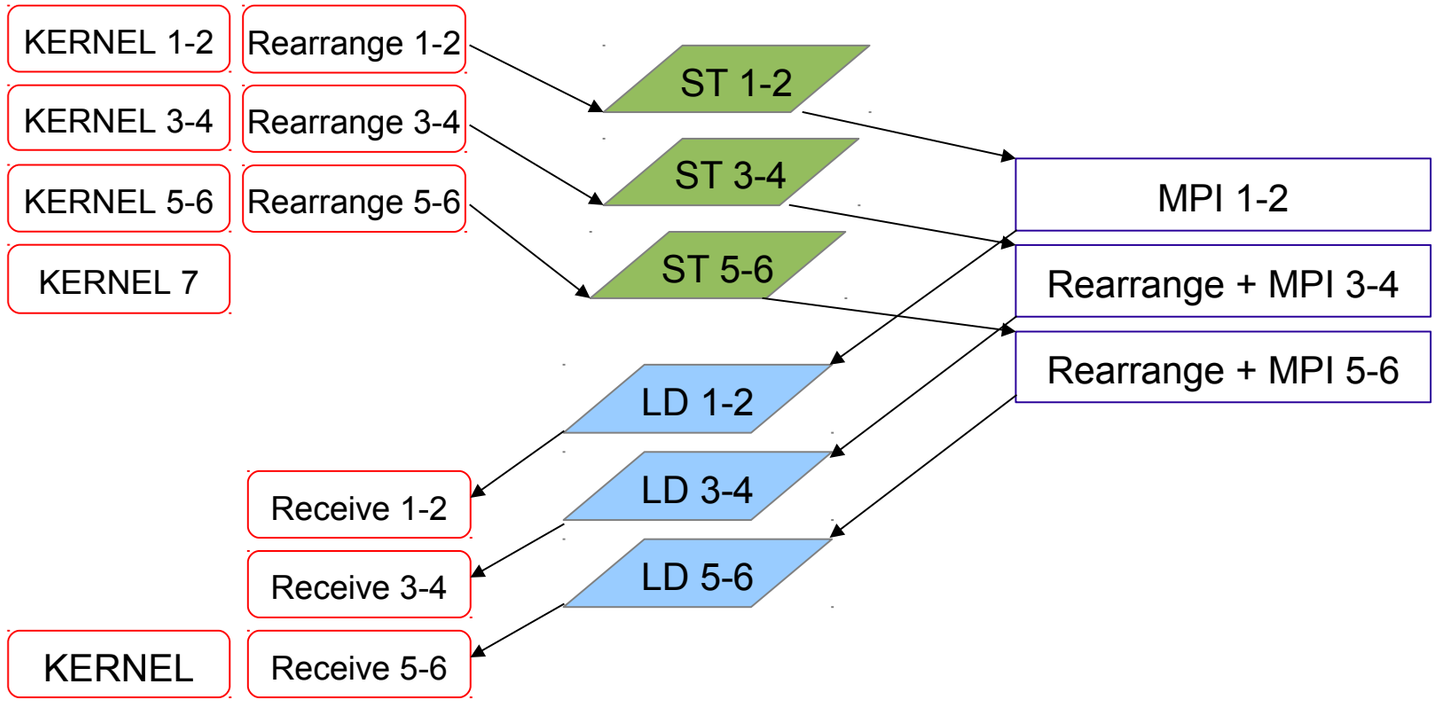
In program code a subblock is defined by its number, sizes and offsets.

Small blocks – of size $2 \times bwidth$ or less by at least one of the dimensions – are processed as a whole.

NAS PB MG, ISR implementation: top-level logic, stage IV

Stage IV – computations and data transfer overlap

- Boundary planes are computed and sent in pairs, for each dimension separately.
- Rearrangement on CPU: copy edges of the already received planes,
- Kernel execution is overlapped with data store, data store – with MPI exchanges.



NAS PB MG, ISR implementation: further optimization

Unsuccessful:

- on each node split the array by z, manually (faster memory access on GPU);
- one MPI-process rules the whole node (reduce overhead on MPI library calls).
  But! High expenses on CPU rearrangement.

Successful:

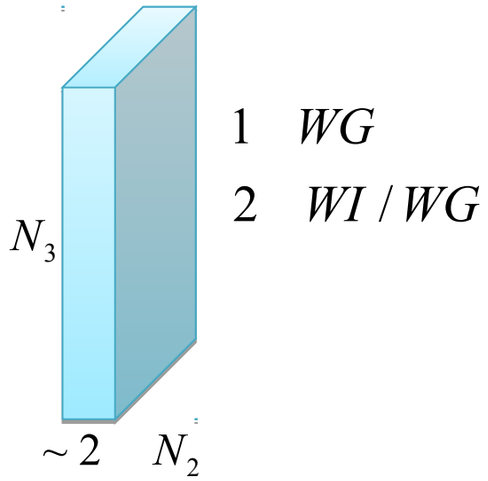
- compute the result norm on coprocessors, call MPI_Reduce in the end;
- special kernels for boundary planes computations.

+:

- for >8 GPUs: collect work on single GPU to perform computations on low MG levels, then split again;
- integrate the initial single-GPU version;
- parameter variation for each Tlin kernel and each array size.

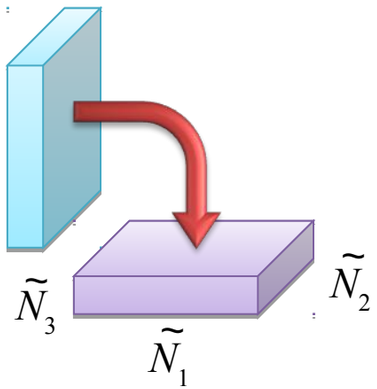
NAS PB MG, ISR implementation: boundary computations

Boundary block 1:



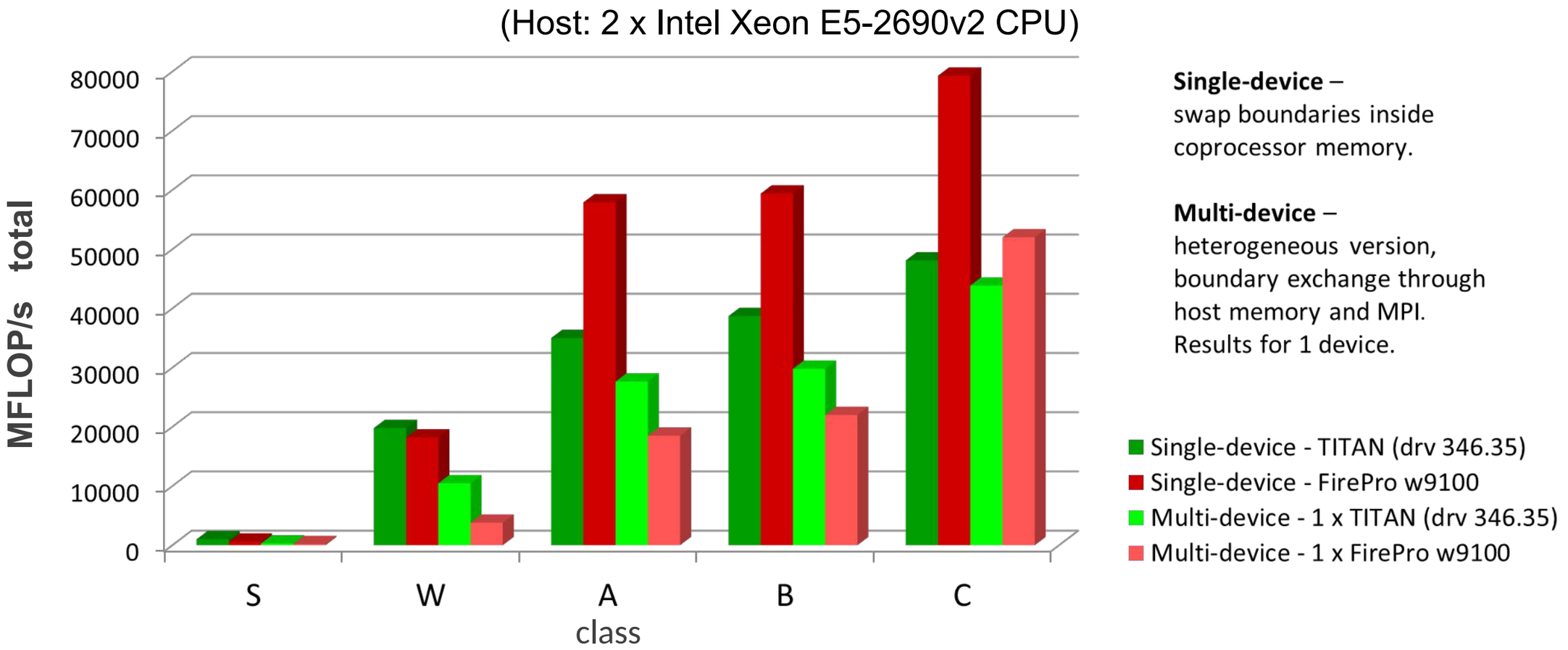
Low performance: idle GPU resources.

Solution: keep and process boundaries in special buffers, in better suited format.
$bwidth = 1$



Effective computation, similar to blocks 5 and 6.

NAS PB MG procedure: single-device and heterogeneous version

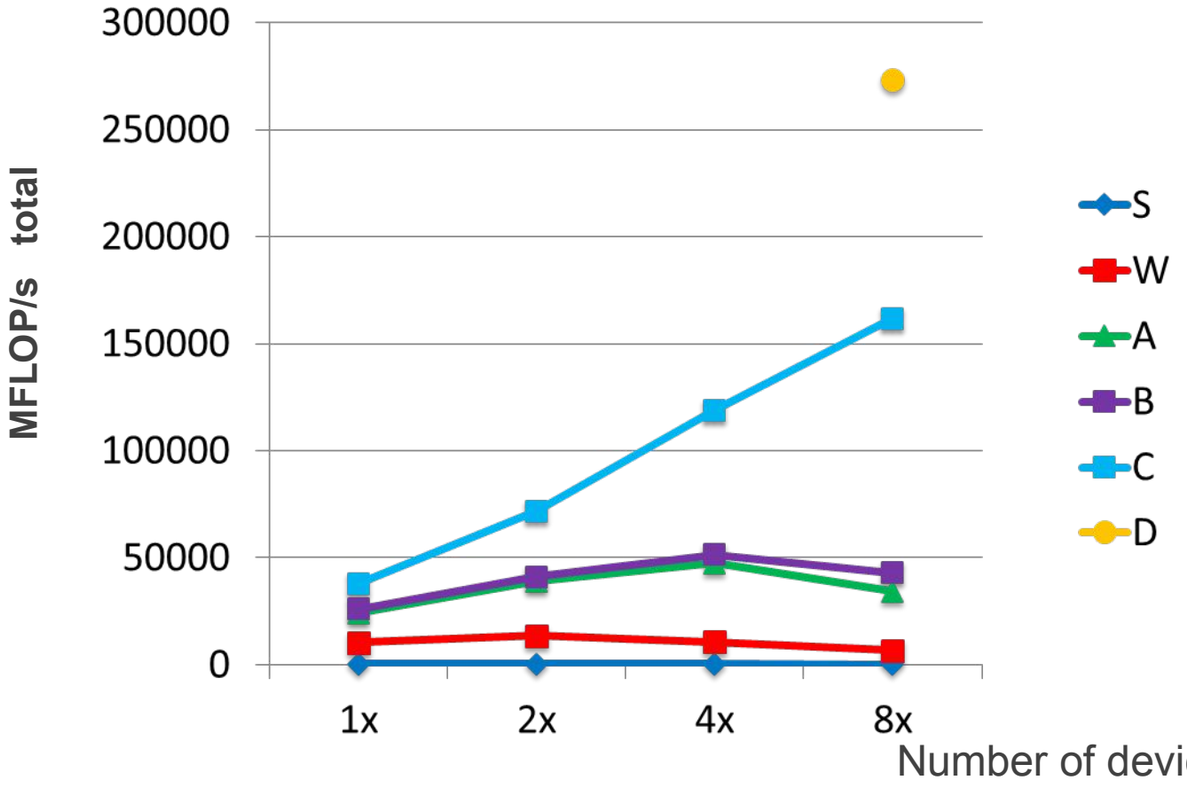


Performance loss due to host memory access and MPI:
NVIDIA GeForce Titan: **~15%**
AMD FirePro w9100: **~50% (!!!)**

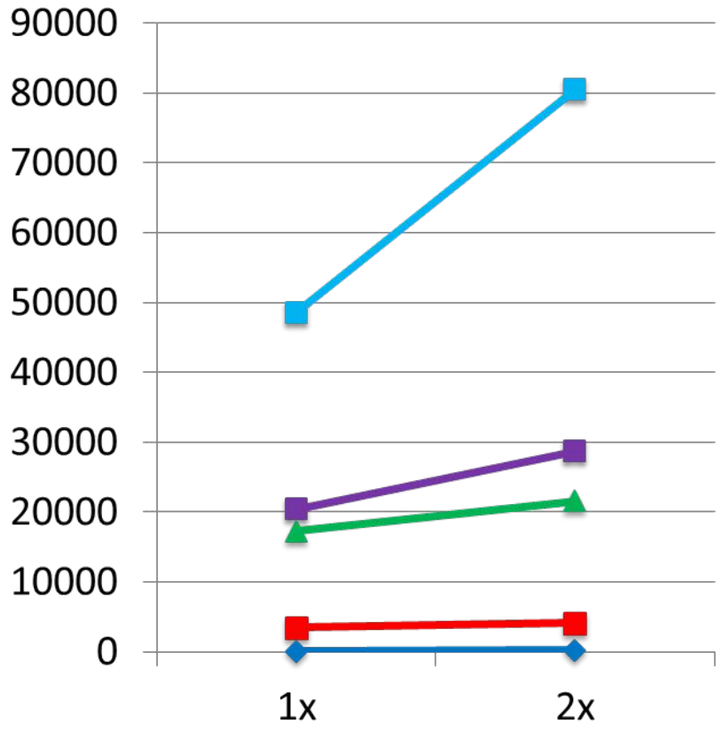
Heterogeneous NAS PB MG routine: problem scalability on one node

(Host: Intel Xeon E5-2690v2 CPU)

{1-8} x NVIDIA GeForce GTX TITAN (drv 340.32)



{1-2} x AMD FirePro w8100

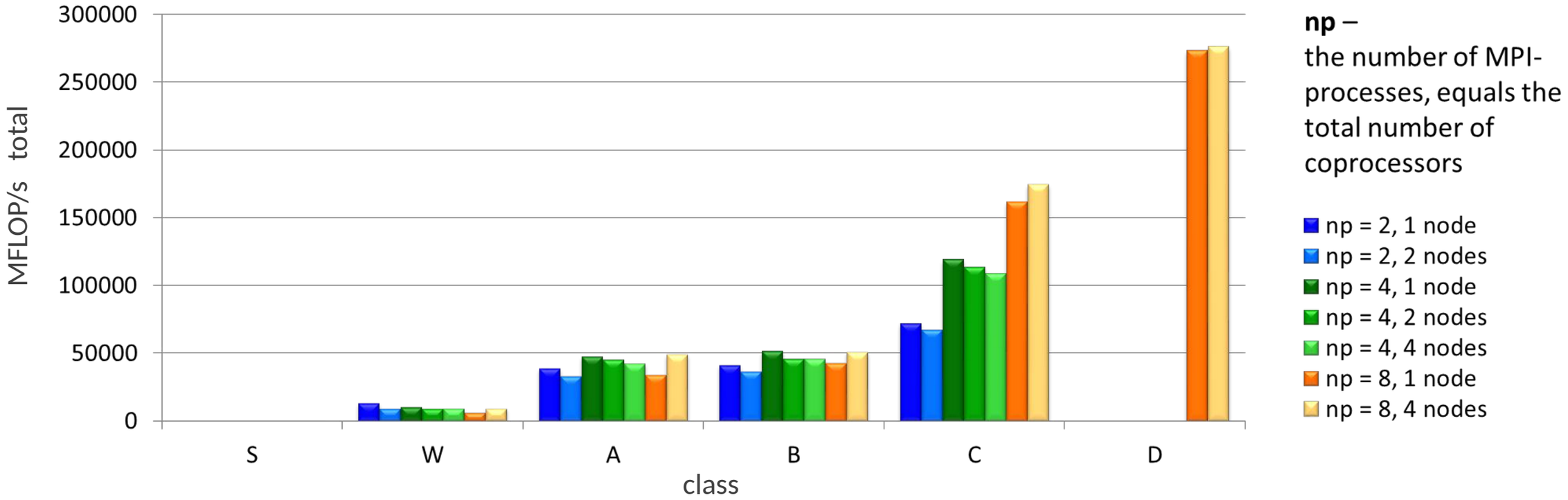


Class C: linear scalability (~ 0.5 x)

~**17000** MFLOP/s (2 x top server CPU, OpenMP) vs **161700** MFLOP/s (8 x top GPU, OpenCL)

Heterogeneous NAS PB MG routine: testing results for ISR mini-supercomputer

{2-8} x NVIDIA GeForce GTX TITAN (drv 340.32) + 2 x Intel Xeon E5-2670 CPUs



More nodes means slower data transfers, but more CPU performance for each coprocessor.

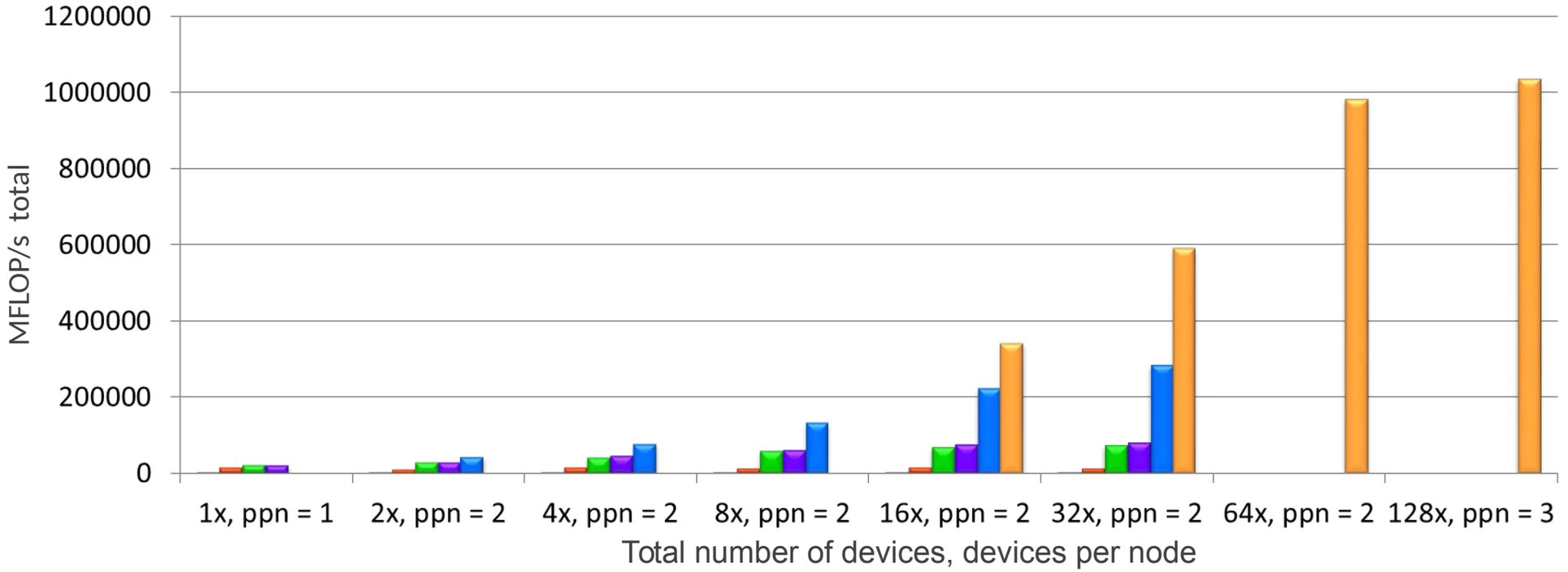
Linear scalability with data size growth;

class D – up to **276** GFLOP/s (vs ~18 GFLOP/s on 2 x top server CPU, OpenMP).

Heterogeneous NAS PB MG routine: testing results for the K100 supercomputer (Keldysh IAM, RAS)

{1-128} x NVIDIA Tesla M2050

Average results for 10 runs.
Performance depends on relative position of the used nodes.



ppn – the number of MPI-processes (equals the number of GPUs) on one node

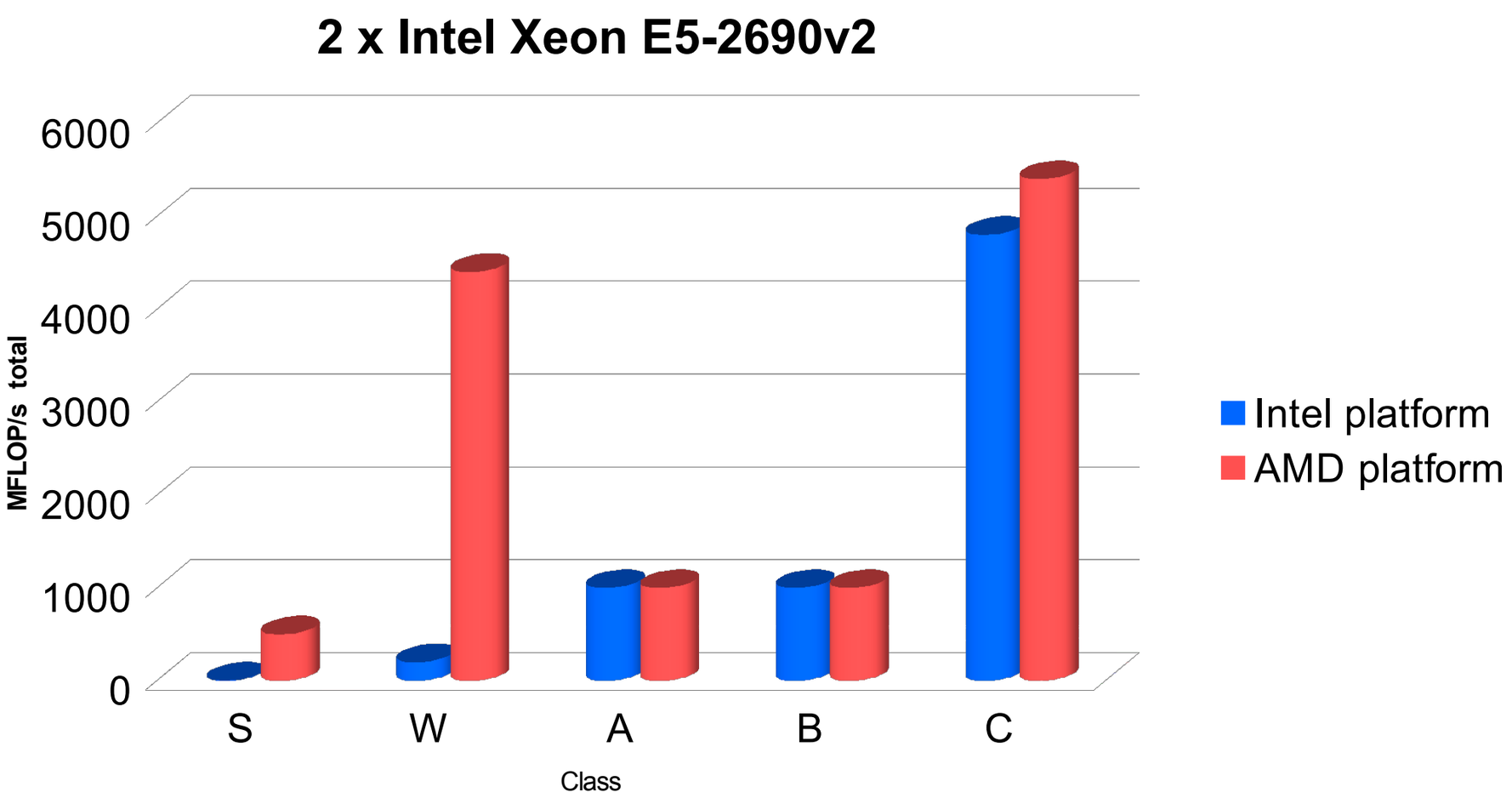
- S
- W
- A
- B
- C
- D

Class D: scalability up to 64 coprocessors!

16 x Tesla – **341,6** GFLOP/s

64 x Tesla – **1,035** TFLOP/s

Heterogeneous NAS PB MG OpenCL routine: testing results for CPU



Class C, OpenMP – ~**18000** MFLOP/s.

Portable **OpenCL** implementation **is possible**, but needs tuning!

Heterogeneous NAS PB MG: future work

- The OpenCL routine tuning for CPU.
- Heterogeneous implementation on CPU + GPU.
- Scalability up to task class E.

Conclusion

Hybrid systems applicability for scientific applications

Of the three problems we discussed:

- FT – heterogeneous implementation is ineffective.
- CG – heterogeneous implementation is reasonable, gives some advantage in performance for large problem sizes;
  bottleneck – the SpMV kernel.
- MG – heterogeneous implementation is reasonable, gives significant advantage in performance.

1. Hybrid systems are **advantageous** for a wide class of problems.
2. Personal mini-supercomputers are a **good alternative** to shared usage of large supercomputers.
3. For most of the algorithms integral performance is defined by the **memory subsystem** performance.
4. A key to high efficiency – data transfers and computations **overlap**.
5. Portable implementations still need some tuning for each specific platform.

Some of our projects

Heterogeneous numerical libraries: BLAS 2012

```
if (task size is small)
```

```
CALL cblas_dgemm()
```

```
else
```

2 x [Xeon E5-2670 8 cores 2.6Ghz]

+

8 x [AMD Radeon 7970 1010Mhz]

=

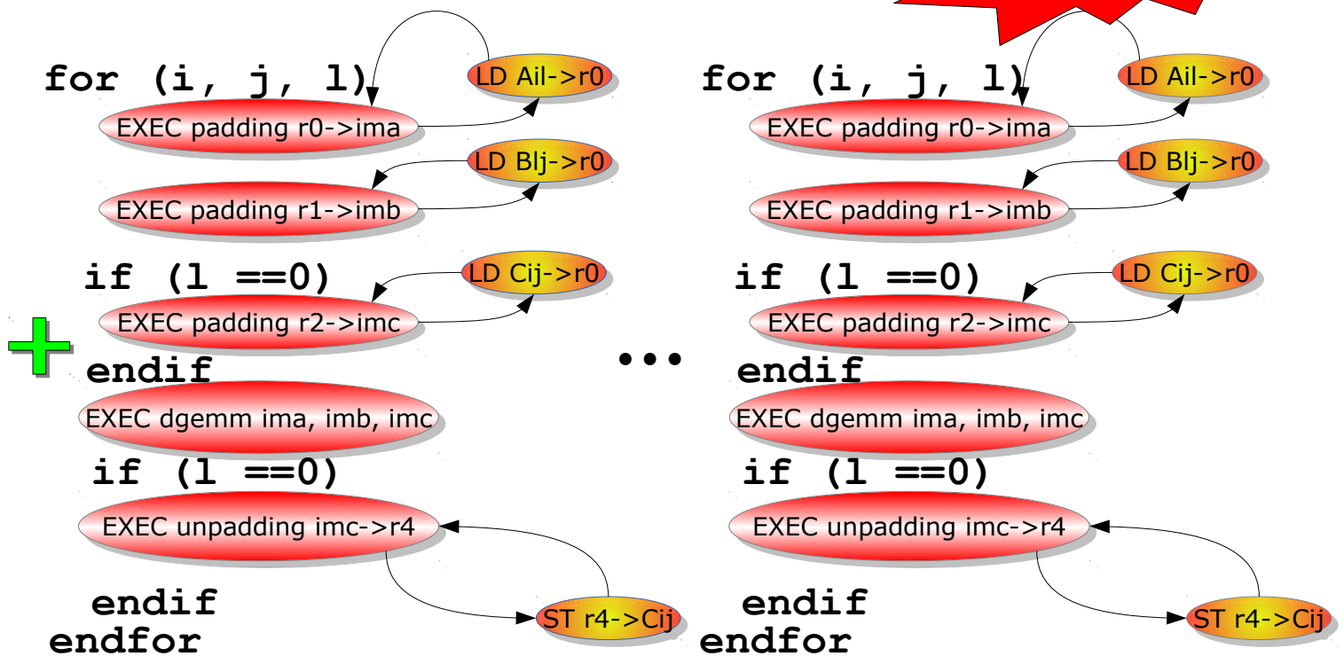
~8,6 Tflops Double Precision

First 8 GPU OpenCL DGEMM

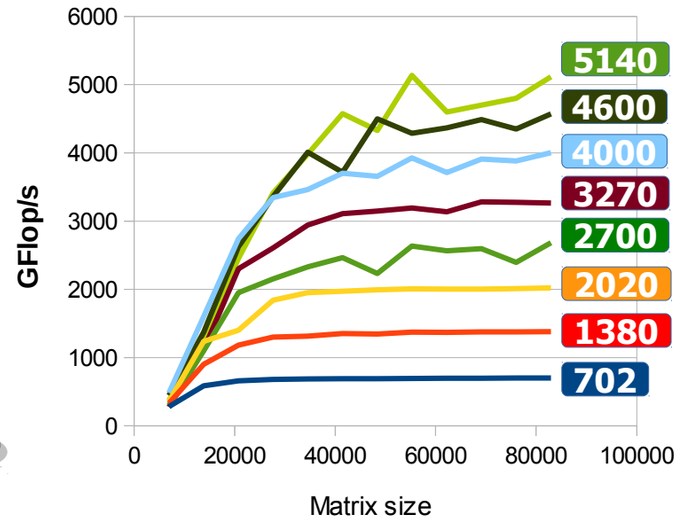
~62% efficiency On 8 GPU

nDev	Rpeak	Rmax	Scl	Eff
1	1034	702	1	68%
2	2068	1380	1,97	66%
3	3102	2020	2,88	65%
4	4136	2700	3,84	65%
5	5170	3270	4,66	63%
6	6204	4000	5,70	64%
7	7238	4600	6,55	64%
8	8272	5140	7,32	62%

```
CALL cblas_dgemm()
```



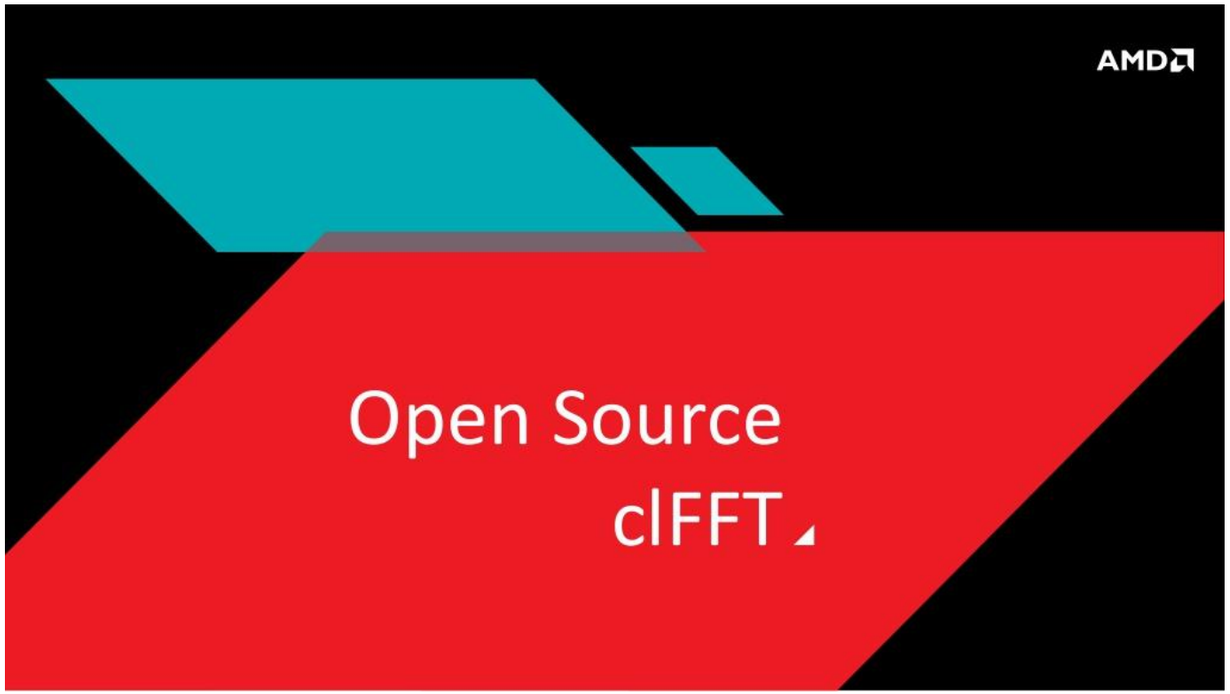
BLAS DGEMM (NxNxN)

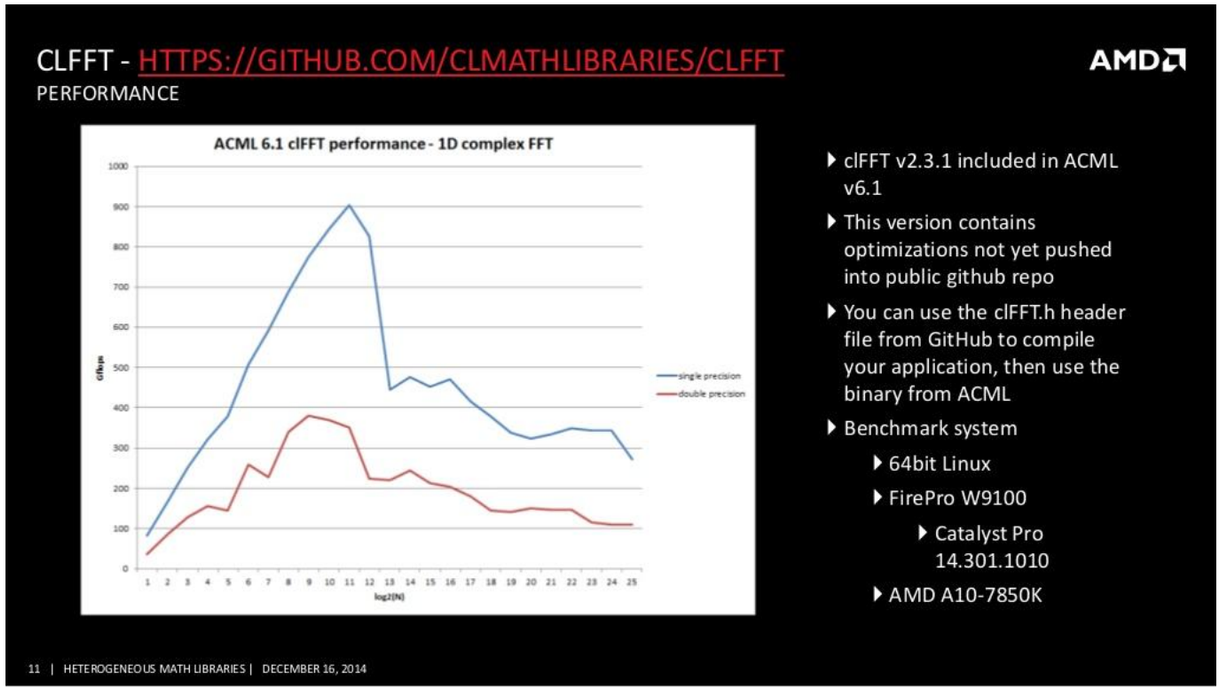



```
endif
```

http://devgurus.amd.com/thread/159457

Heterogeneous numerical libraries: clFFT improvement





Heterogeneous math libraries
http://www.slideshare.net/DevCentralAMD/leverage-the-speed-of-opencl-with-amd-math-libraries

in progress...

Canonical HPC benchmarks 2013-...

- ✔ High Performance Linpack (HPL) OpenCL
- ✔ NAS Parallel Benchmarks OpenCL
- ✔ High Performance Conjugate Gradients (HPCG) OpenCL in progress...

http://devgurus.amd.com/thread/159457

```
================================================================================
T/V                N    NB     P     Q               Time                 Gflops
--------------------------------------------------------------------------------
WR00C2L64      106496  4096     1     1             907.09              8.877e+02
HPL_pdgesv() start time Wed Jul 31 14:50:40 2013

HPL_pdgesv() end time   Wed Jul 31 15:05:54 2013

--------------------------------------------------------------------------------
||Ax-b||_oo/(eps*(||A||_oo*||x||_oo+||b||_oo)*N)=        0.0046169 ...... PASSED
================================================================================

Finished      1 tests with the following results:
              1 tests completed and passed residual checks,
              0 tests completed and failed residual checks,
              0 tests skipped because of illegal input values.
--------------------------------------------------------------------------------

End of Tests.
================================================================================
```

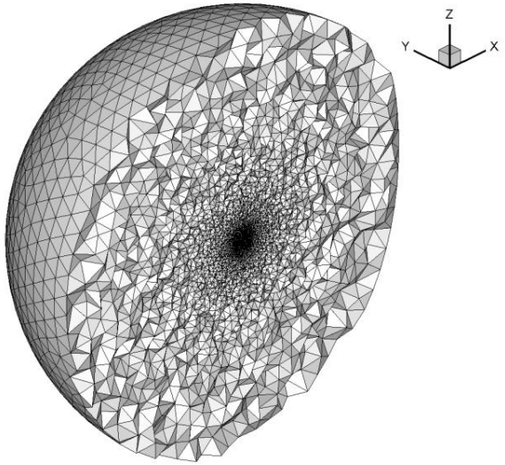
2 x Xeon E5-2670
+
2 x Radeon 7970 Ghz Ed

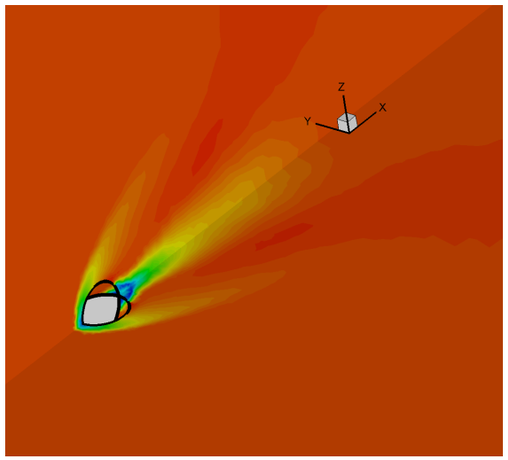
9 nodes K100:

9 x [2 x Xeon X5670 + 3 x Tesla M2050]

```
==============================================
Name                 Time     GFlop     Gflops
----------------------------------------------
cblas_dgemm       601.055  782225.4     1301.4
cblas_dtrsm        47.701   22632.0      474.5
cblas_dgemv        19.012       0.0        0.0
cblas_dswap         0.000       0.0        0.0
cblas_dscal        14.830       0.0        0.0
cblas_dger          0.000       0.0        0.0
cblas_dcopy        36.520       0.0        0.0
cblas_daxpy         0.000       0.0        0.0
cblas_dtrsv         0.429       0.0        0.0
cblas_idamax        7.148       0.0        0.0
CPU               617.829  144873.9      234.5
GPU               563.699  659983.4     1170.8
Total             726.694  804857.4     1107.6
```

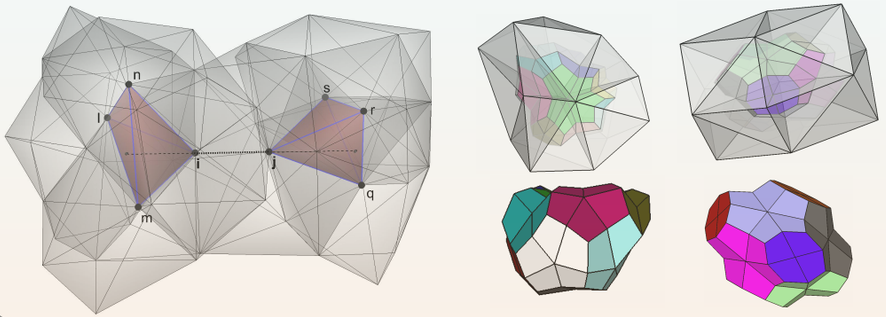
Model CFD task: sphere in a supersonic flow 2013



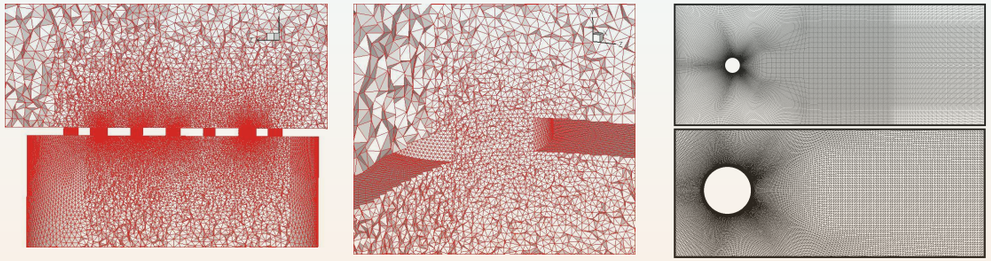


CFD is highly resource-consuming

- High-ordered schemes for accuracy

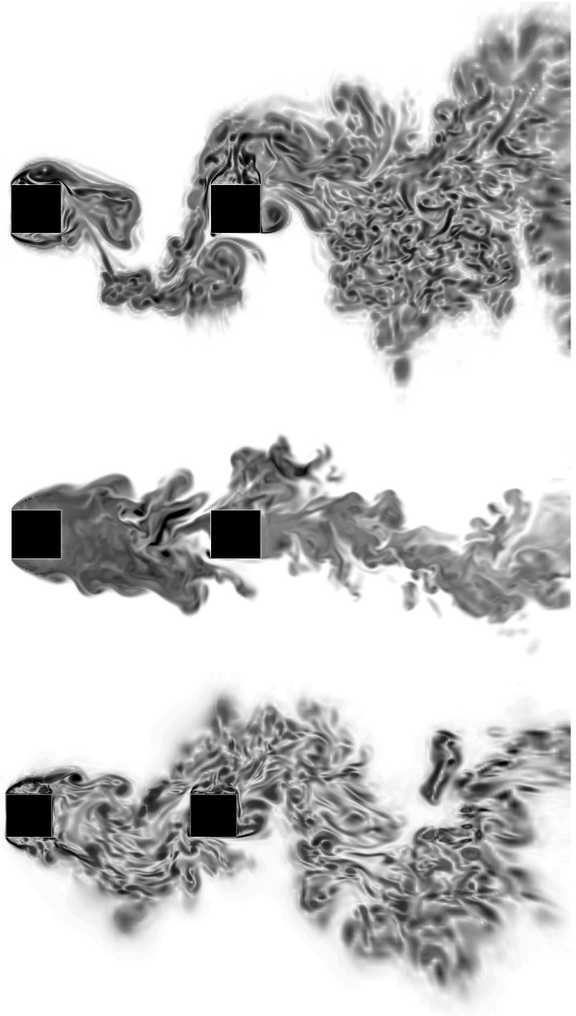


- The large size of computational domain and and high scale differential



- High space and time resolution

- Long period for time integration

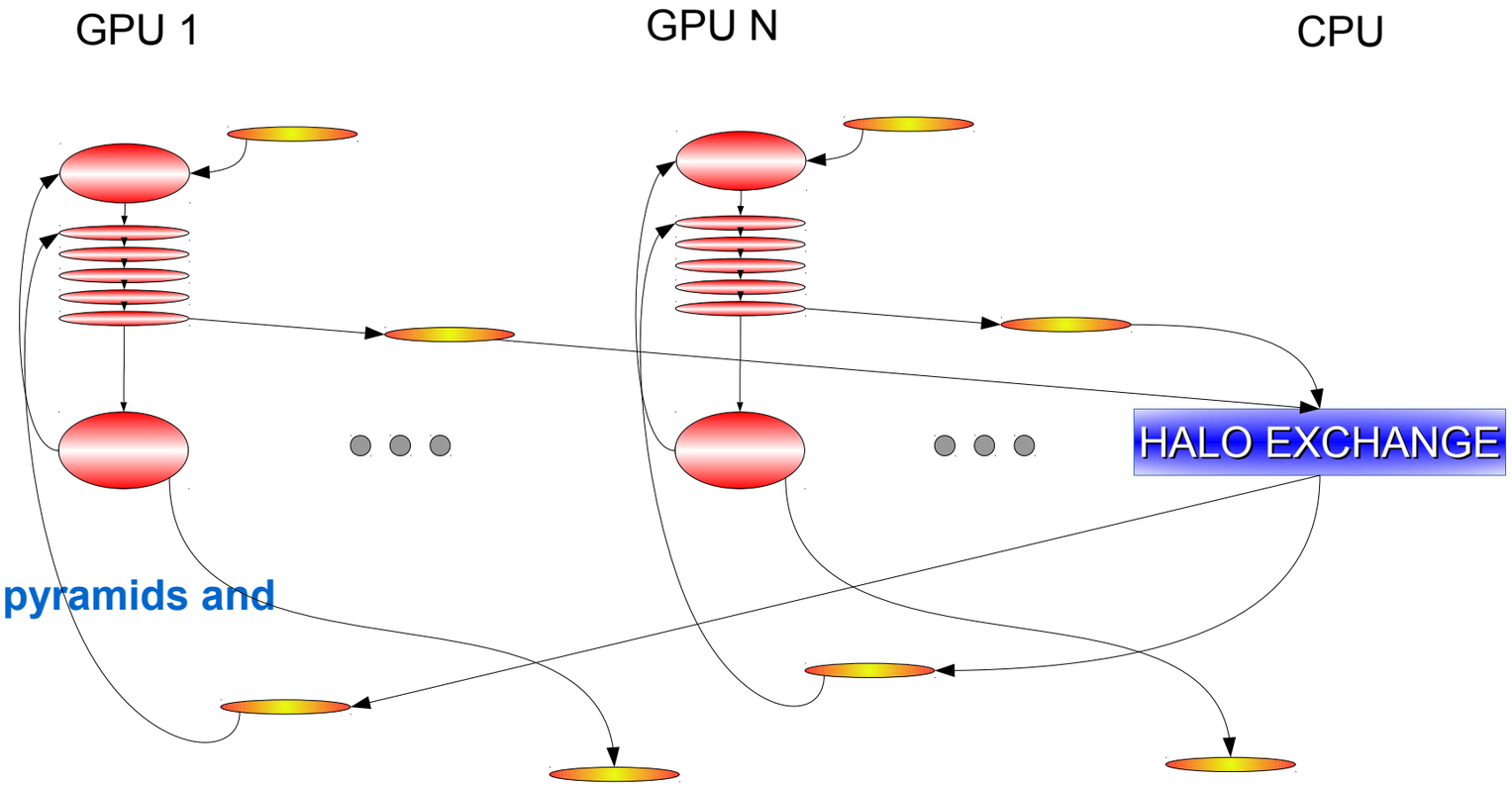


Model CFD task: sphere in a supersonic flow 2013

Model task:
sphere in supersonic flow (M = 2.75)

We consider a compressible inviscid flow:

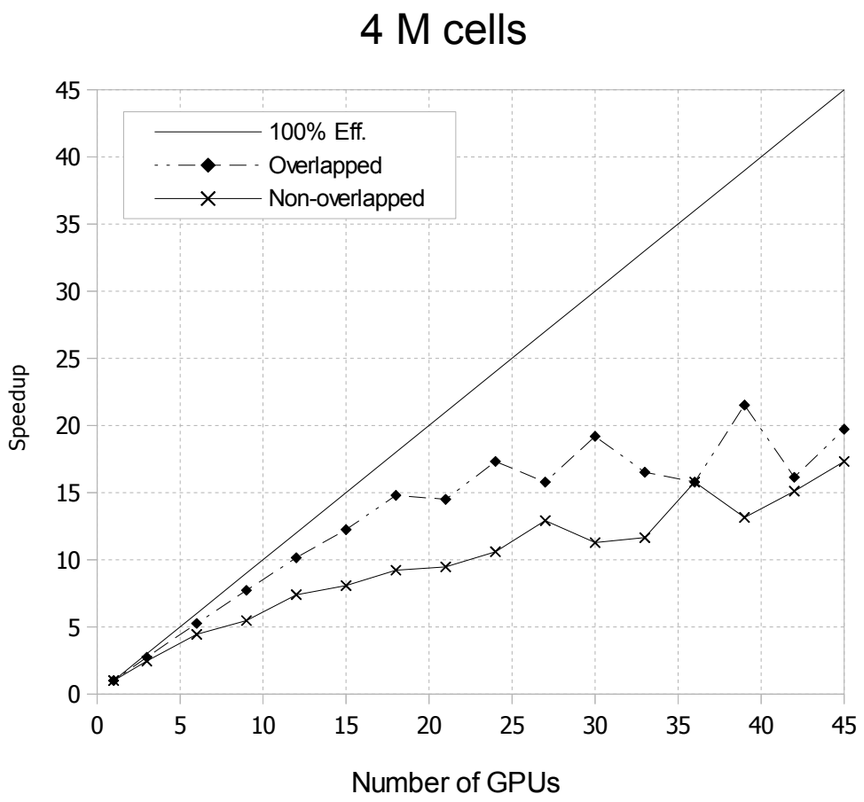
- Euler equiations
- Finite volume approach
- Unstructured hybrid mesh
  (tetrahedrons, hexahedrons, quadrangular pyramids and triangular prisms)
- Roe scheme
- The explicit Runge-Kutta scheme up to 4th order of accuracy

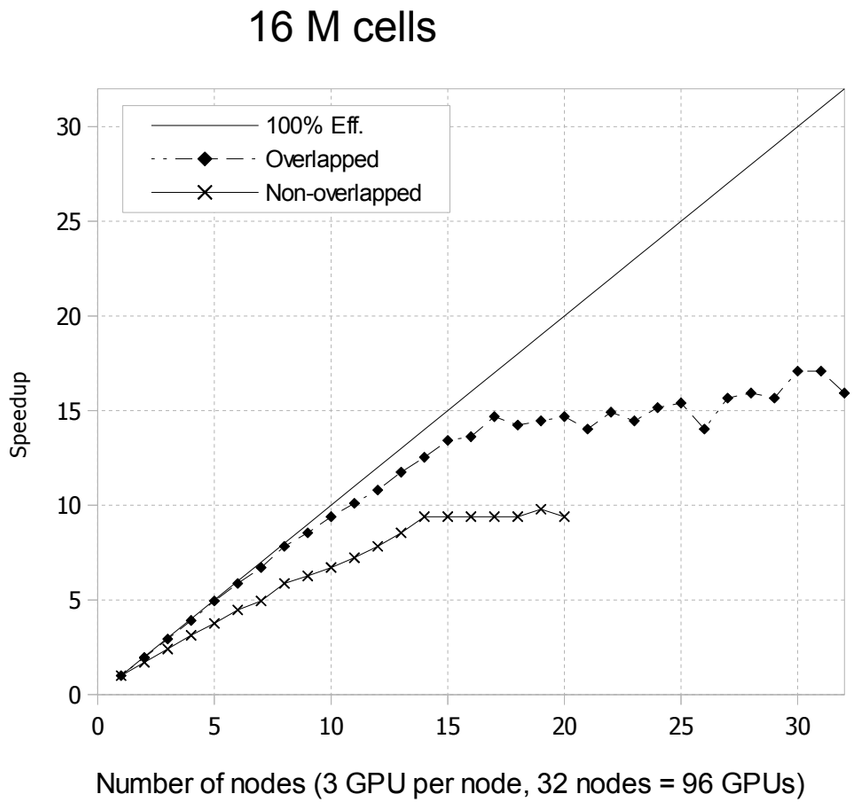


Scheduler program for N devices

P.B. Bogdanov, A.A. Efremov, A.V. Gorobets, S.A. Sukov, "Using a scheduler for efficient data exchange on hybrid supercomputers with massively-parallel accelerators", Numerical methods and programming, vol. 14 (2013), pp. 122-134.
http://www.mathnet.ru/php/archive.phtml?wshow=paper&jrnid=vmp&paperid=161&option_lang=eng

Model CFD task: sphere in a supersonic flow 2013
The efficiency of the parallelization on a supercomputer K100





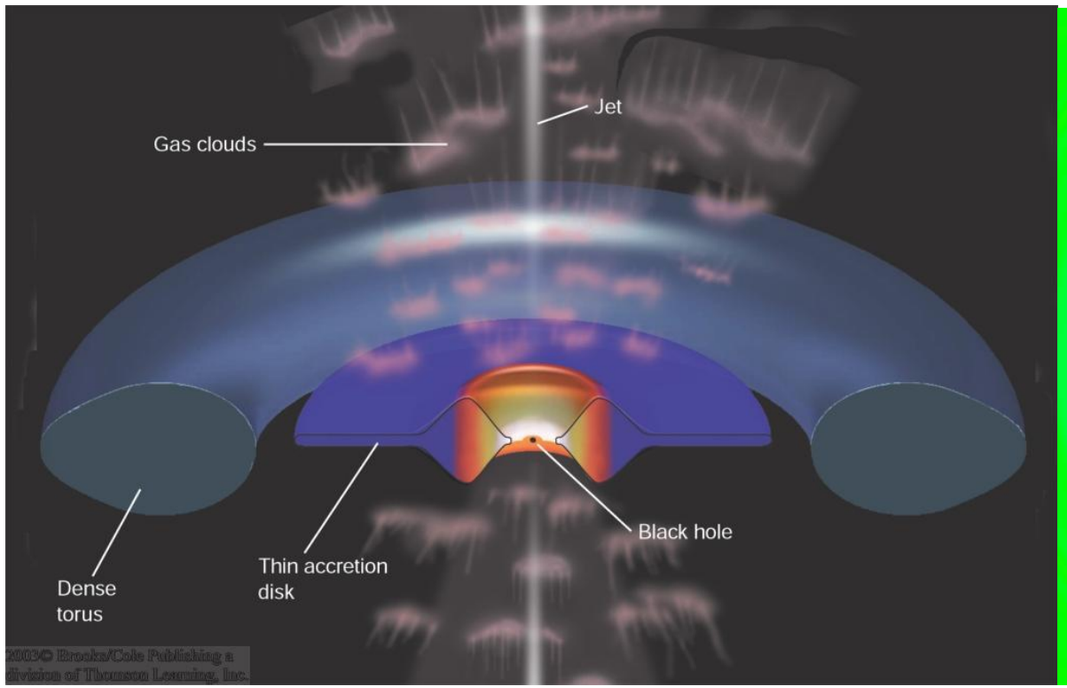
Speedups on K100 supercomputer for a mesh with 4 millions of cells (left) starting from one GPU and for a mesh with **16 millions of cells** (right) starting from one computing node.

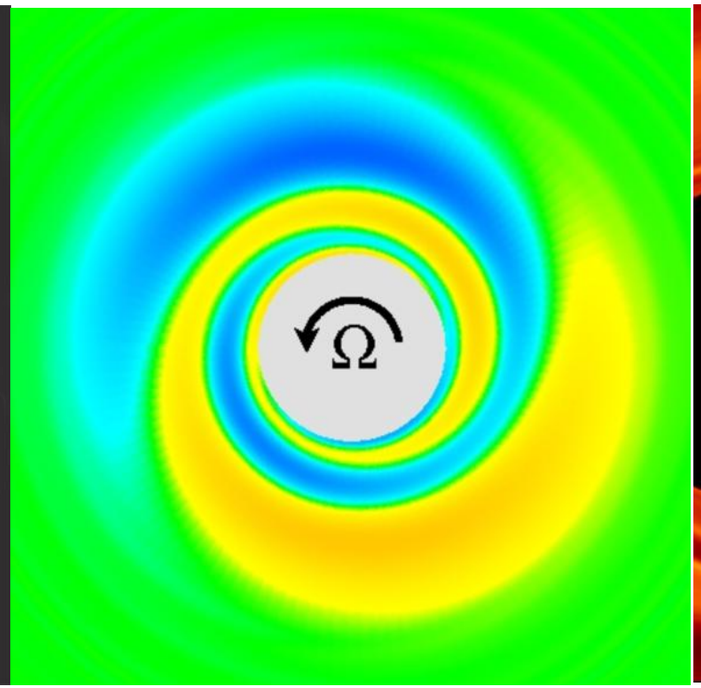
K100 = **64 nodes** X [12 CPU cores Xeon X5670, 96 Gb RAM DDR3, 3 GPU NVidia C2050]

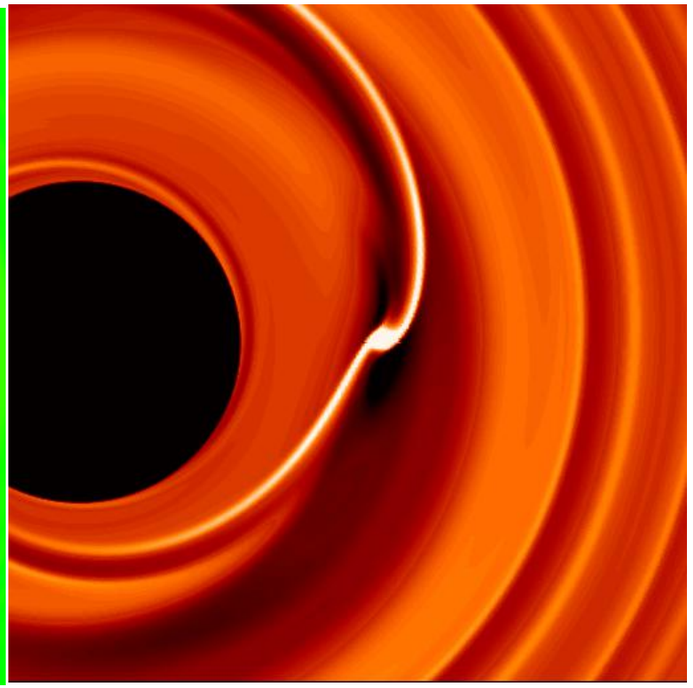
Astrophysics

SAI MSU

3D MODELING OF TRANSIENT DYNAMICS OF PERTURBATIONS IN ASTROPHYSICAL DISC FLOWS

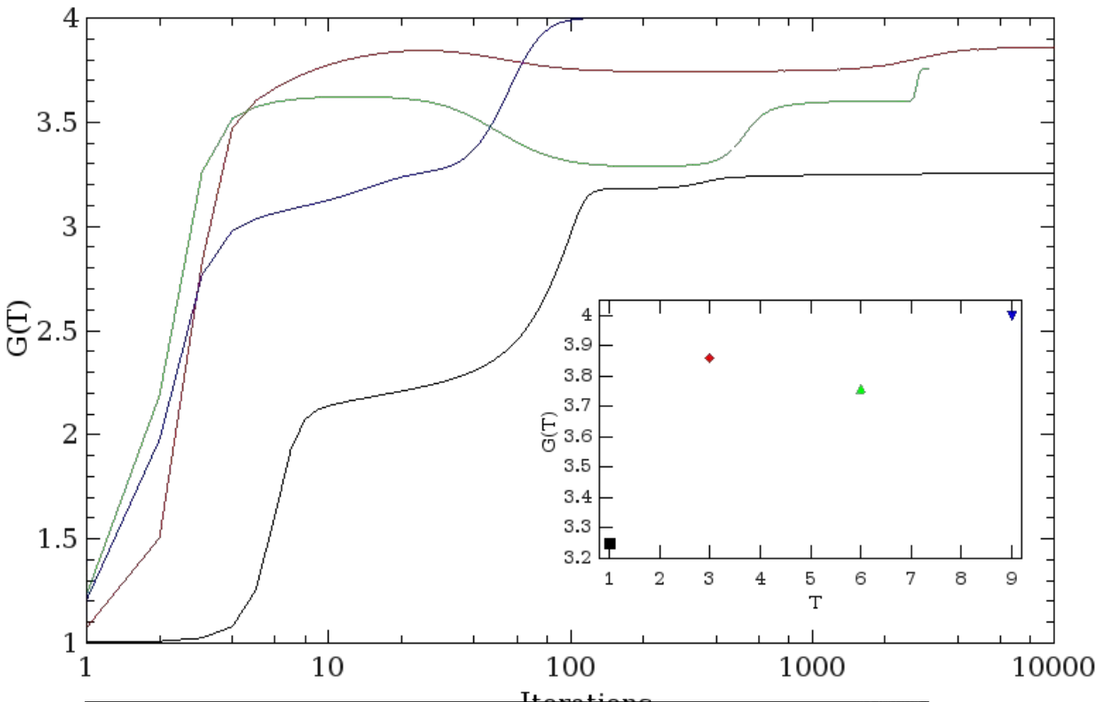


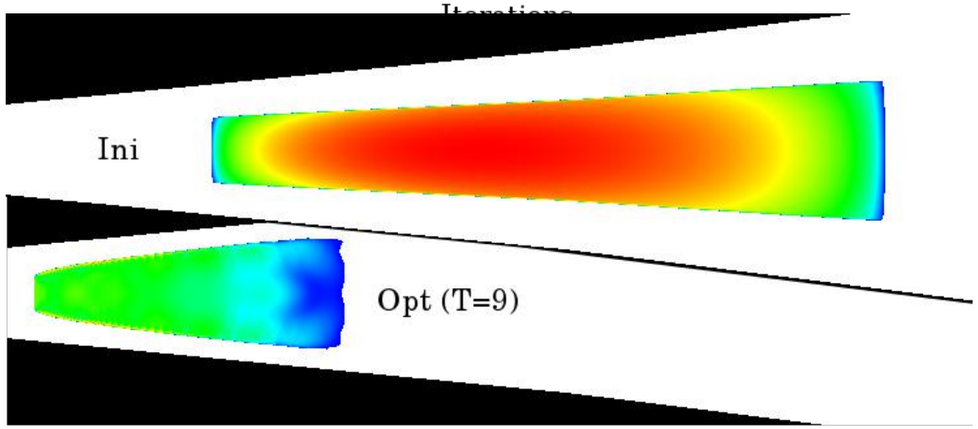


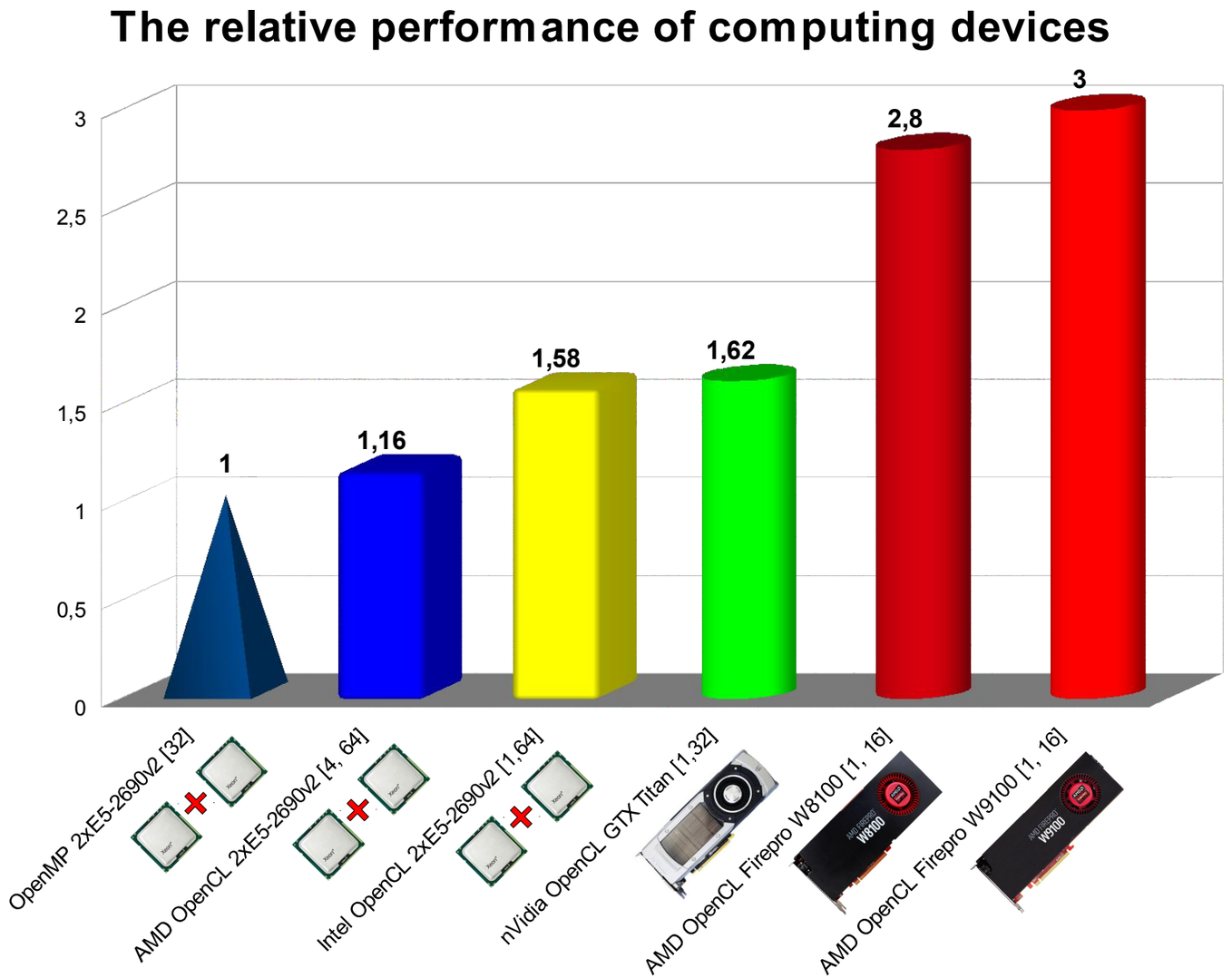


D.A. Badjin, P.B. Bogdanov, V.V. Zhuravlev, 3D modeling of transient dynamics of perturbations in astrophysical disc flows, SCCS, Sarov, Russia, 13-17 October 2014

Device performance







Sparse solvers


Krylov subspace methods (BiCGSTAB, GMRES) MPI-implementation


ILUT, BIILU2 preconditioners MPI-implementation


Heterogeneous ILU-preconditioned sparse solvers with OpenCL triangular solving of L\U


Algebraic multigrid method (AMG)


Constrained Pressure Residual (CPR\ILU) preconditioner for multiphase porous media flows


HPCG implementation


….

Conference

2011: Parallel Computational Fluid Dynamics (ParCFD), Barcelona, Spain

A. V. Gorobets, S. A. Soukov, P. B. Bogdanov, A. O. Zheleznyakov and B. N. Chetverushkin, Extension with OpenCL of the two-level MPI+OpenMP parallelization for large-scale CFD simulations on heterogeneous systems, Parallel CFD 2011, Barcelona, Spain, 16-20 May 2011

2012: Parallel Computational Fluid Dynamics (ParCFD), Atlanta, USA

A.A. Efremov, P. B. Bogdanov, OpenCL mathematical software infrastructure for heterogeneous computing, Parallel CFD 2012, Atlanta, USA, 20-24 May 2012

2013: GPU technology conference (GTC), March 18-21, San Jose, California, USA

A.A. Efremov, P. B. Bogdanov, Programming Infrastructure for Heterogeneous Computing Based on OpenCL and its Applications

2013: Exascale Application and Software Conference (EASC), April 9-11 , Edinburg, Scotland, UK

A.A. Efremov, P. B. Bogdanov, Programming infrastructure for heterogeneous computing based on OpenCL and its applications

2013: Parallel Computational Fluid Dynamics (ParCFD), May 20-24, Changsha, Hunan, China

P.B. Bogdanov, A.A. Efremov, S.A. Soukov, A.V. Gorobets, OpenCL extension of a parallel finite-volume CFD algorithm on unstructured meshes using a scheduler for overlapped communications

2013: High Performance Computing 2013 (HPC-UA), October 7-11, Kyiv, Ukraine

O.U. Sudareva, A.A. Efremov, P.B. Bogdanov, Heterogeneous programming methodology based on OpenCL framework

Conference

2014: Super-Computation and Computer Simulation (SCCS), Sarov, Russia
A.V. Gorobets, S.A. Soukov, F.X. Trias, P.B. Bogdanov, Finite-volume algorithms for modeling of turbulent flows on supercomputers, SCCS, Sarov, Russia, 13-17 October 2014

2014: Super-Computation and Computer Simulation (SCCS), Sarov, Russia
D.A. Badjin, P.B. Bogdanov, V.V. Zhuravlev, 3D modeling of transient dynamics of perturbations in astrophysical disc flows, *SCCS, Sarov, Russia, 13-17 October 2014*

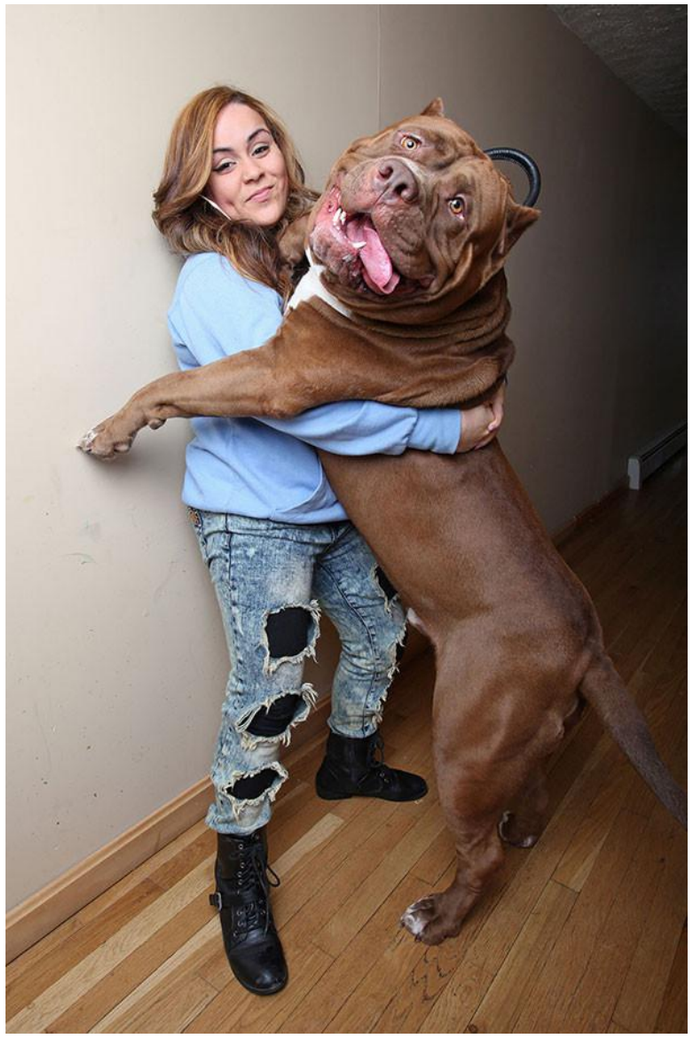
2014: Super-Computation and Computer Simulation (SCCS), Sarov, Russia
O.Y. Sudareva, P.B. Bogdanov, Heterogeneous programming on opencl, *SCCS, Sarov, Russia, 13-17 October 2014*

2014: Super-Computation and Computer Simulation (SCCS), Sarov, Russia
F.S. Zaitsev, E.P. Suchkov, P.B. Bogdanov, Parallel algorithms for toroidal plasma boundary control using the epsilon-net technique, *SCCS, Sarov, Russia, 13-17 October 2014*

2014: Super-Computation and Computer Simulation (SCCS), Sarov, Russia
F.S. Zaitsev, F.A. Anikeev, P.B. Bogdanov, High-speed algorithms for solving of multidimensional kinetic equations WITH smoothed particle method, *SCCS, Sarov, Russia, 13-17 October 2014*

We thank AMD company!

AMD



ИИСИ РАН

ISR RAS

Thank you for watching!