3rd International EULAG Workshop on Eulerian/Lagrangian methods for fluids

Parallelization of MPDATA on Multicore Architectures with GPU Accelerators Using Load Balancing and Autotuning Techniques

Krzysztof Rojek
Łukasz Szustak
Roman Wyrzykowski
[krojek, lszustak, roman]@icis.pcz.pl

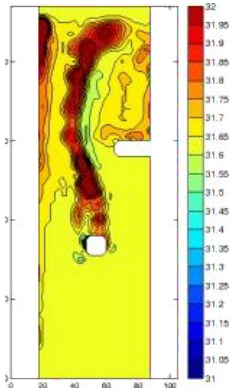
Czestochowa University of Technology

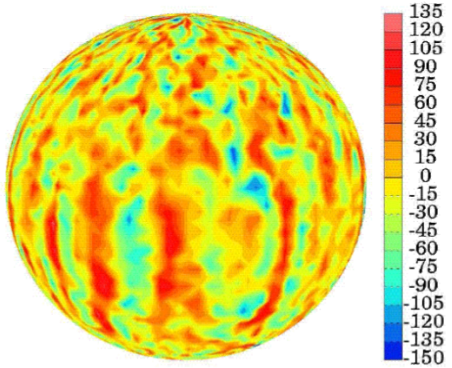
Agenda

- MPDATA overview
- GPU architecture overview
- GPU parallelization of MPDATA
- Optimization of MPDATA using autotuning technique
- CPU-GPU architecture overview
- OpenMP and OpenCL hybrid programming model
- CPU parallelization of MPDATA
- MPDATA parallelization using CPU-GPU architecture
- Conclusions and future work

MPDATA overview

- Our research includes Multidimensional Positive Definite Advection Transport Algorithm (MPDATA)[1], which is one of the main part of the EULAG model
- EULAG can be used to simulate:
  - weather prediction
  - ocean currents
  - areas of turbulence
  - urban flows
  - gravity wave dynamics
  - micrometeorology
  - cloud microphysics and dynamics





[1] J. Prusa, P. K. Smolarkiewicz, A. Wyszogrodzki: EULAG, a computational model for multiscale flows

Comparison of NVIDIA Tesla M2070-Q and ATI Radeon HD 5870

NVIDIA

- 14 compute units
- **448 processing elements**
- 1.147 GHz of clock frequency
- 6 GB of global memory
- 148.0 GB/s of global memory bandwidth
- It gives: 448 * 1.147 * 2MADD = 1.03 **Tflops**

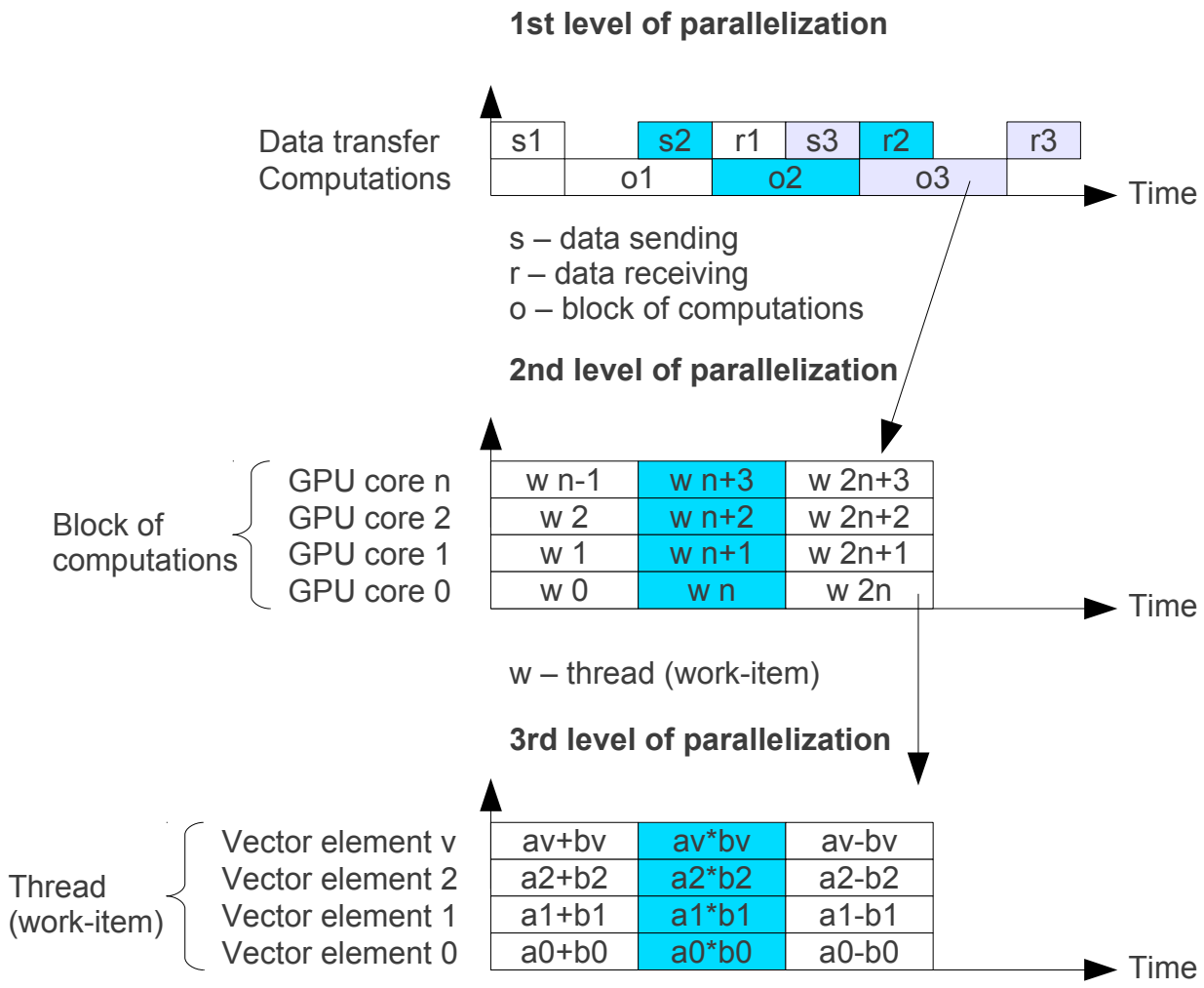
ATI Radeon

- 20 compute units
- **1600 stream processors**
- 850 MHz of clock frequency
- 1 GB of global memory
- 153.6 GB/s of global memory bandwidth
- It gives: 1600 * 0.850 * 2MADD = 2.72 **Tflops**

GPU parallelization of MPDATA

- Idea of GPU parallelization:
  - MPDATA is decomposed according to data blocks
  - each data block is executed by one GPU task
  - one task is computed by a sequence of 13 kernels
  - kernels are executed by GPU cores
- We distinguish the following levels of GPU parallelization:
  - overlapping of data transfer with computations
  - computations on GPU
  - vectorization of GPU threads

Three levels of GPU parallel hierarchy

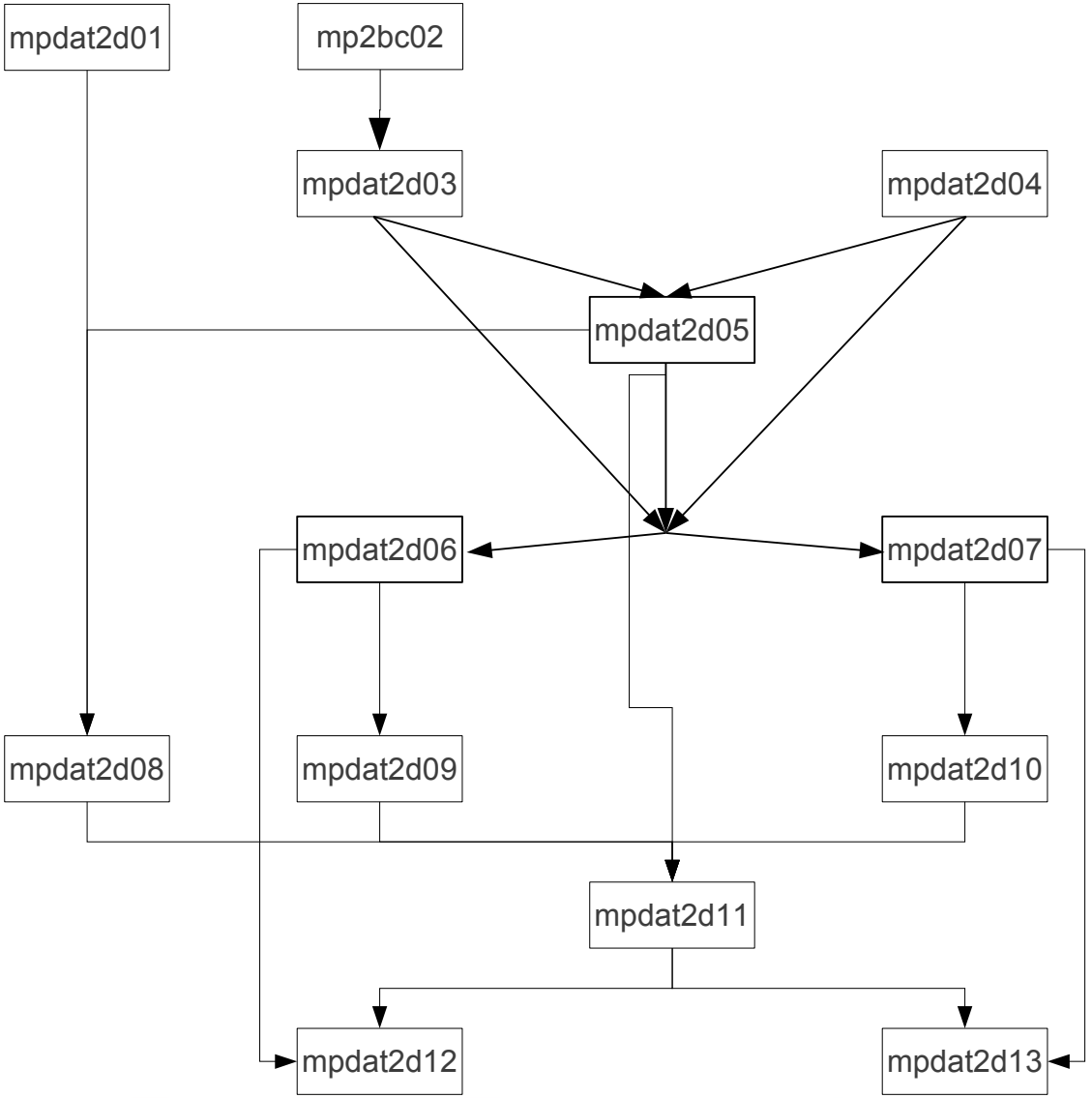


MPDATA task decomposition

- Each MPDATA task is decomposed into 13 kernels based on data dependencies and synchronization points
- Each kernel computes a different part of MPDATA
- Kernels operate on data blocks which are received from the host memory
- Each kernel is configured in an individual way considering:
  - number of global work-items (GPU threads)
  - number of local work-items
  - number of dimensions of work-group
  - size of vector

Dependency tree

- Kernels are executed in a FIFO order corresponding to the dependency tree expressing data dependencies between kernels

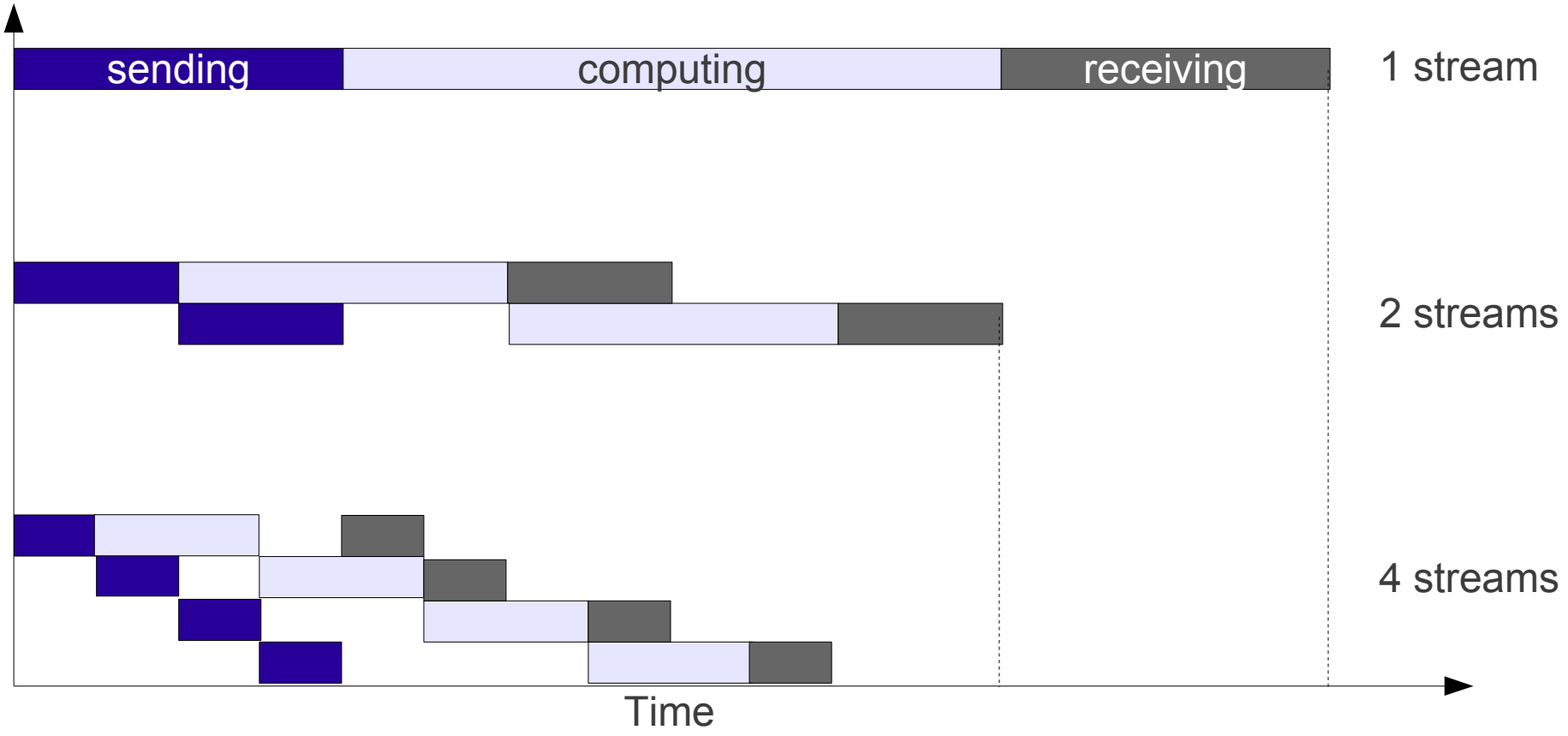


Overlapping of data transfer with computations on GPU (1/2)

- The algorithm is processed by streams
- One stream operates on a collection of data blocks
- Each stream consists of a sequence of following instructions:
  - sending data blocks from host memory to GPU global memory
  - computations performed by kernels
  - receiving data blocks from GPU global memory to host memory
- Number of streams depends on GPU architecture and size of matrices

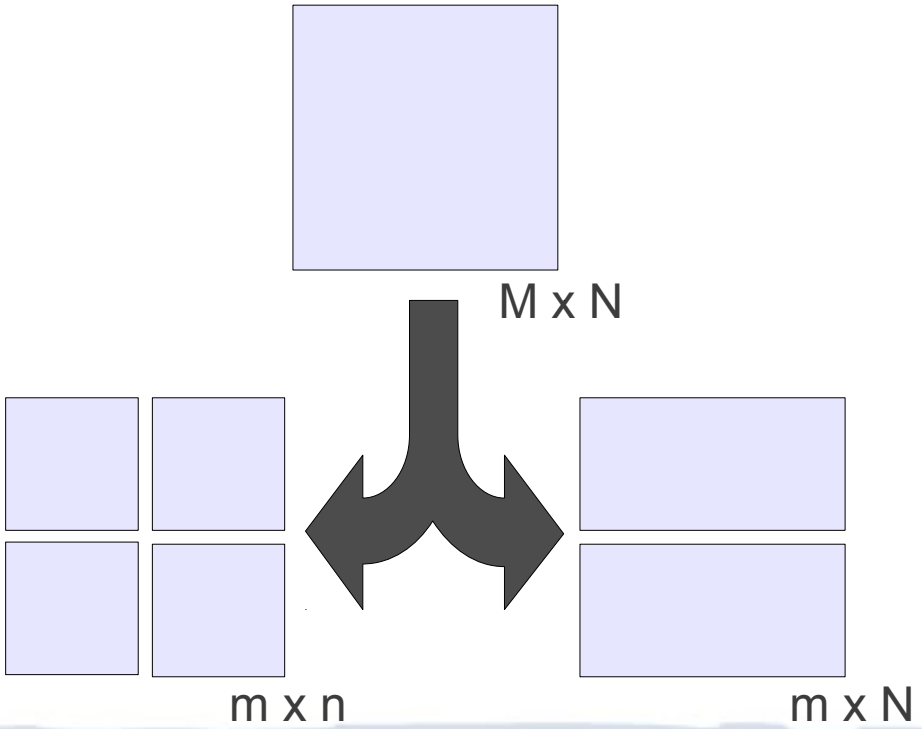
Overlapping of data transfer with computations on GPU (2/2)

- An example of stream processing on GPU that supports overlapping of data transfer with computations



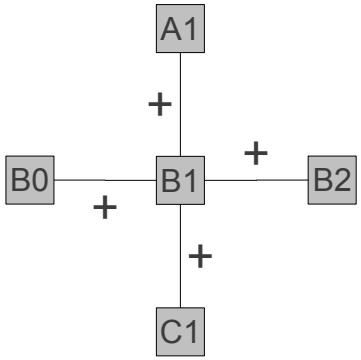
Computations on GPU

- MPDATA is executed by work-items that are grouped in work-groups
- There are 1- or 2-dimensional work-groups
- No synchronization between work-groups, they are independent
- One work-group is executed by a single compute unit



Vectorization of GPU threads

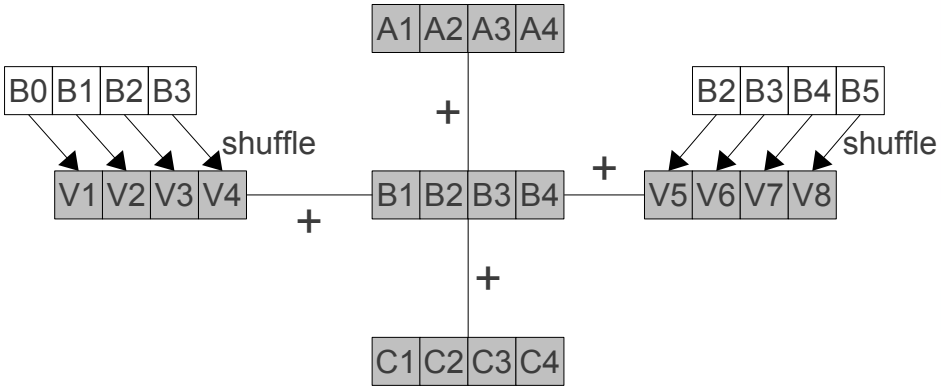
Sequential version



$e_1 = A1 + B0 + B1 + B2 + C1$

4 ”+” → 1
4n → n

Vectorized version



$V_{1,2,3,4} = shuffle(B_{-3,-2,-1,0}, B_{1,2,3,4}, ...)$

$V_{5,6,7,8} = shuffle(B_{1,2,3,4}, B_{5,6,7,8}, ...)$

$e_{1,2,3,4} = A_{1,2,3,4} + V_{1,2,3,4} + B_{1,2,3,4} + V_{5,6,7,8} + C_{1,2,3,4}$

Size of vector	# of seq. instructions	# of SIMD instructions
1	4*n	6*n
2	4*n	3*n
4	4*n	1.5*n
8	4*n	0.75*n

4 ”+” i 2 ”shuffle” → vs
6n/vs → n

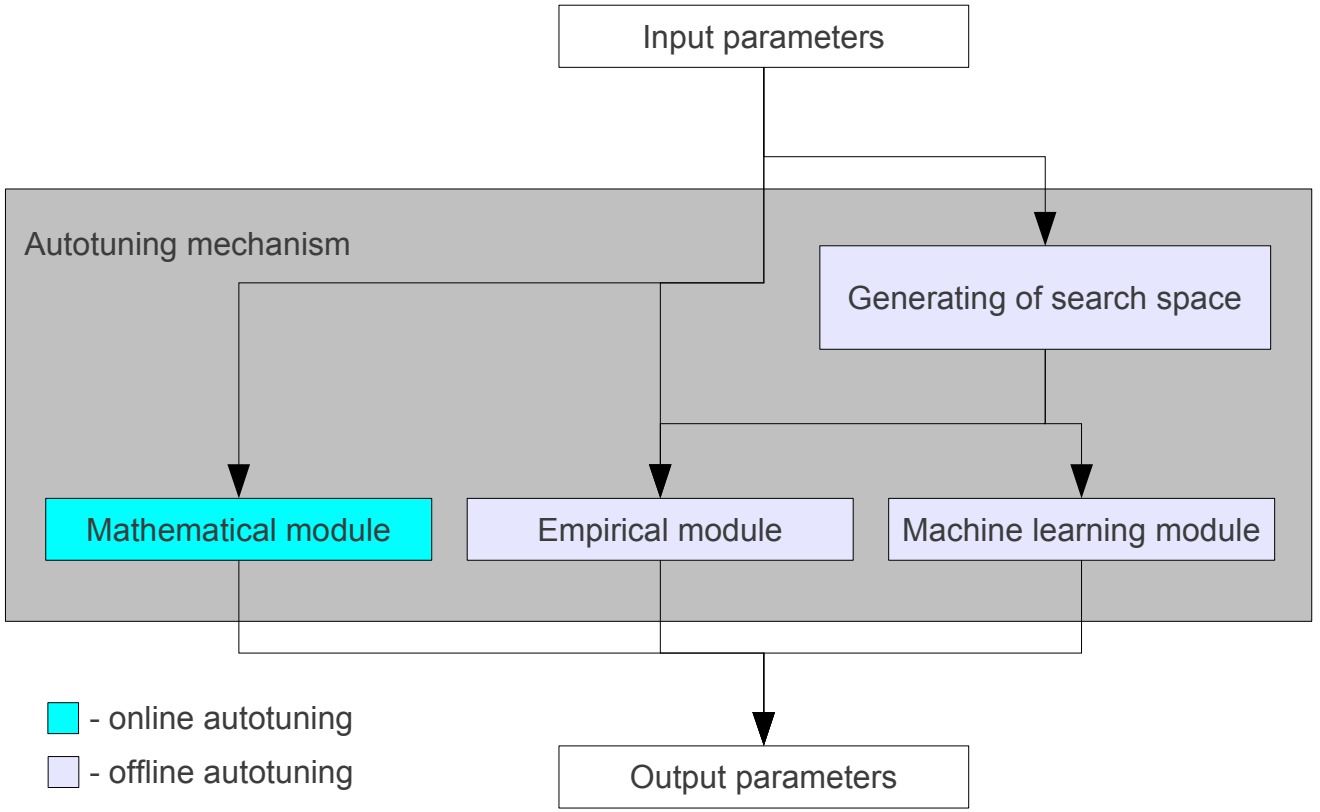
vs – size of vector

GPU task scheduler

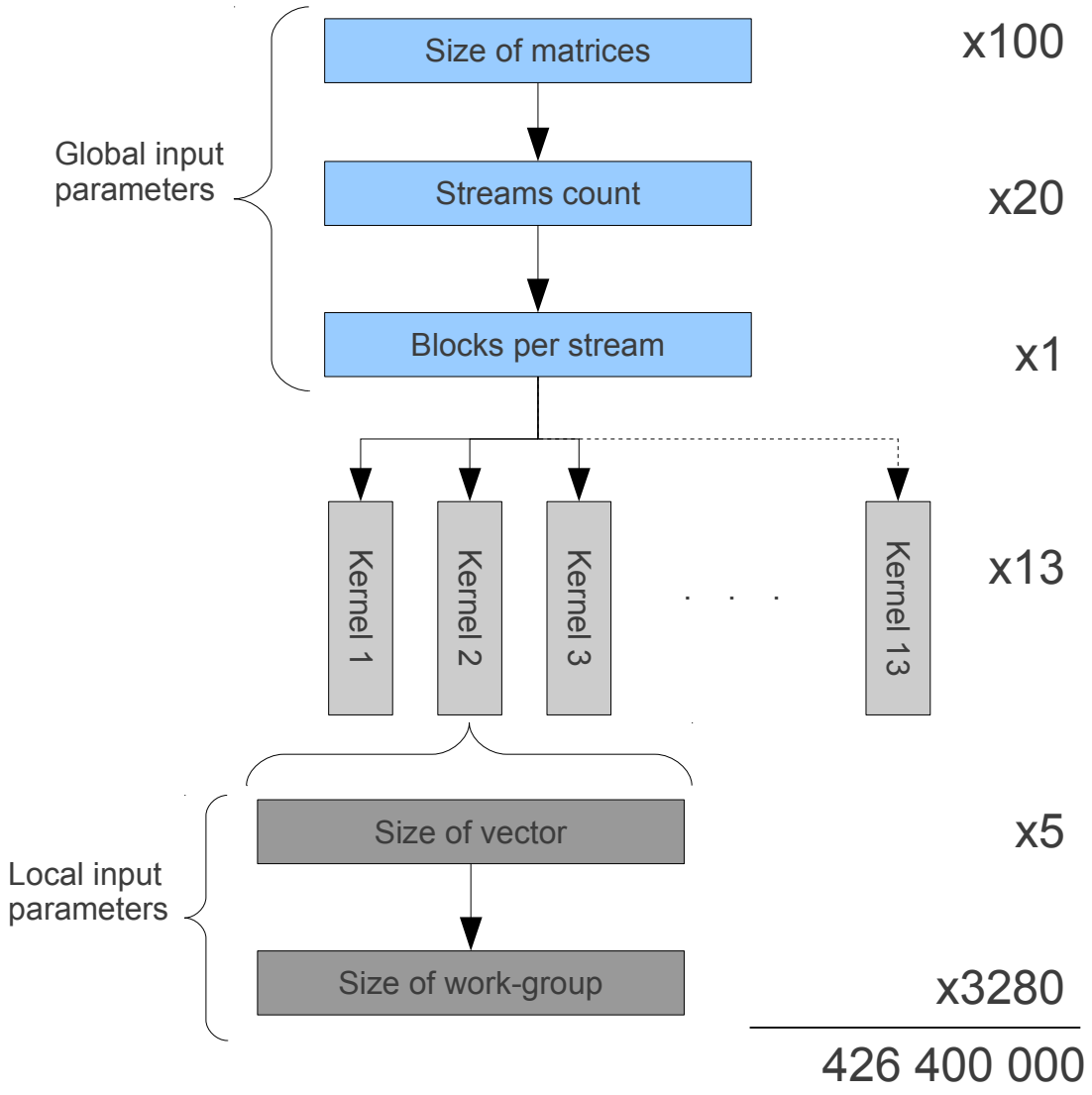
- GPU task scheduler is responsible for:
  - creating streams in a very elastic way
  - dividing streams into tasks, where one task operates on a single block of every matrix of MPDATA
  - running tasks in accordance to dependency tree
  - creating work-groups for each kernel

Autotuning approach

- Software automatic tuning (autotuning) is an optimization technique, which provides performance portability across a variety of hardware platforms

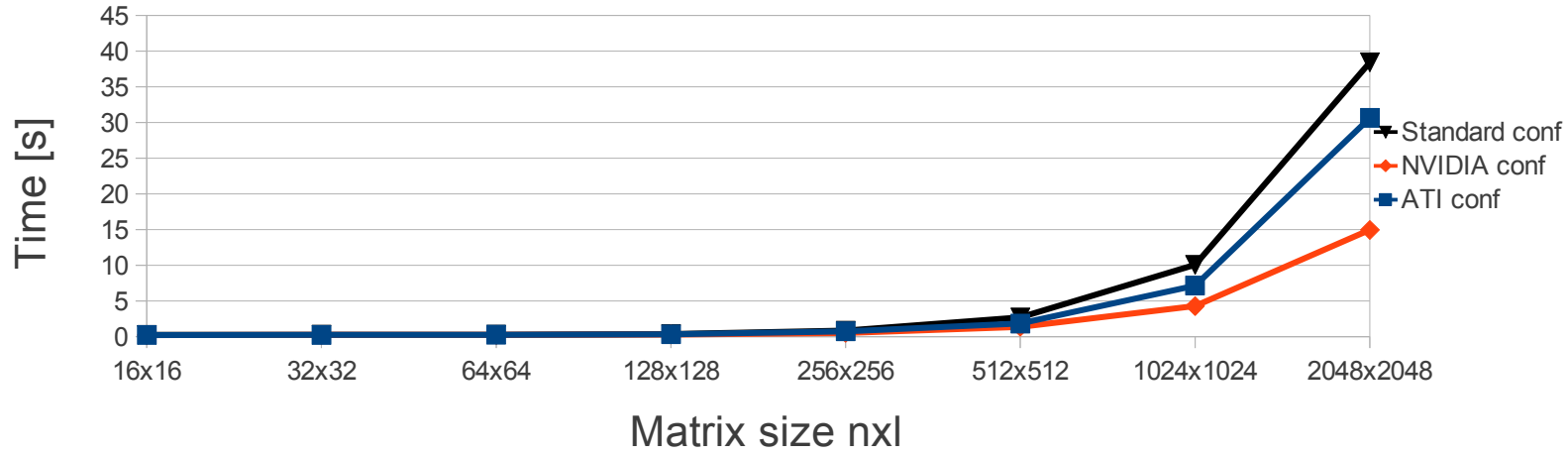


Autotuning: Space of possible solutions

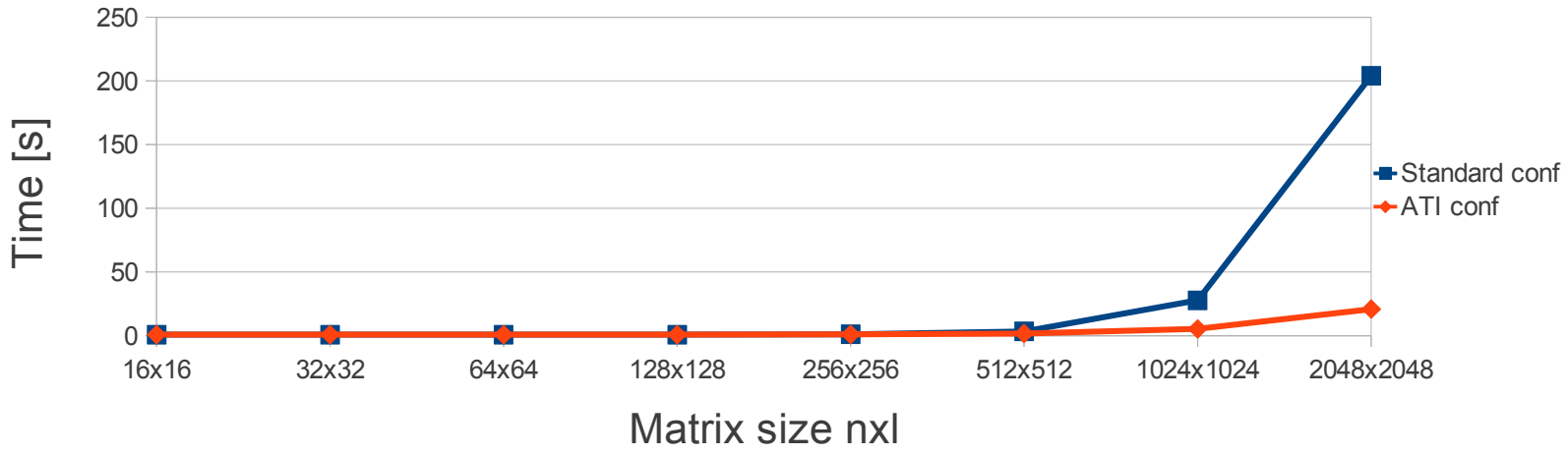


Performance analysis of autotuned adaptation

- NVIDIA Tesla M2070-Q – MPDATA with autotuned configuration is 2.6 times faster than MPDATA with standard configuration

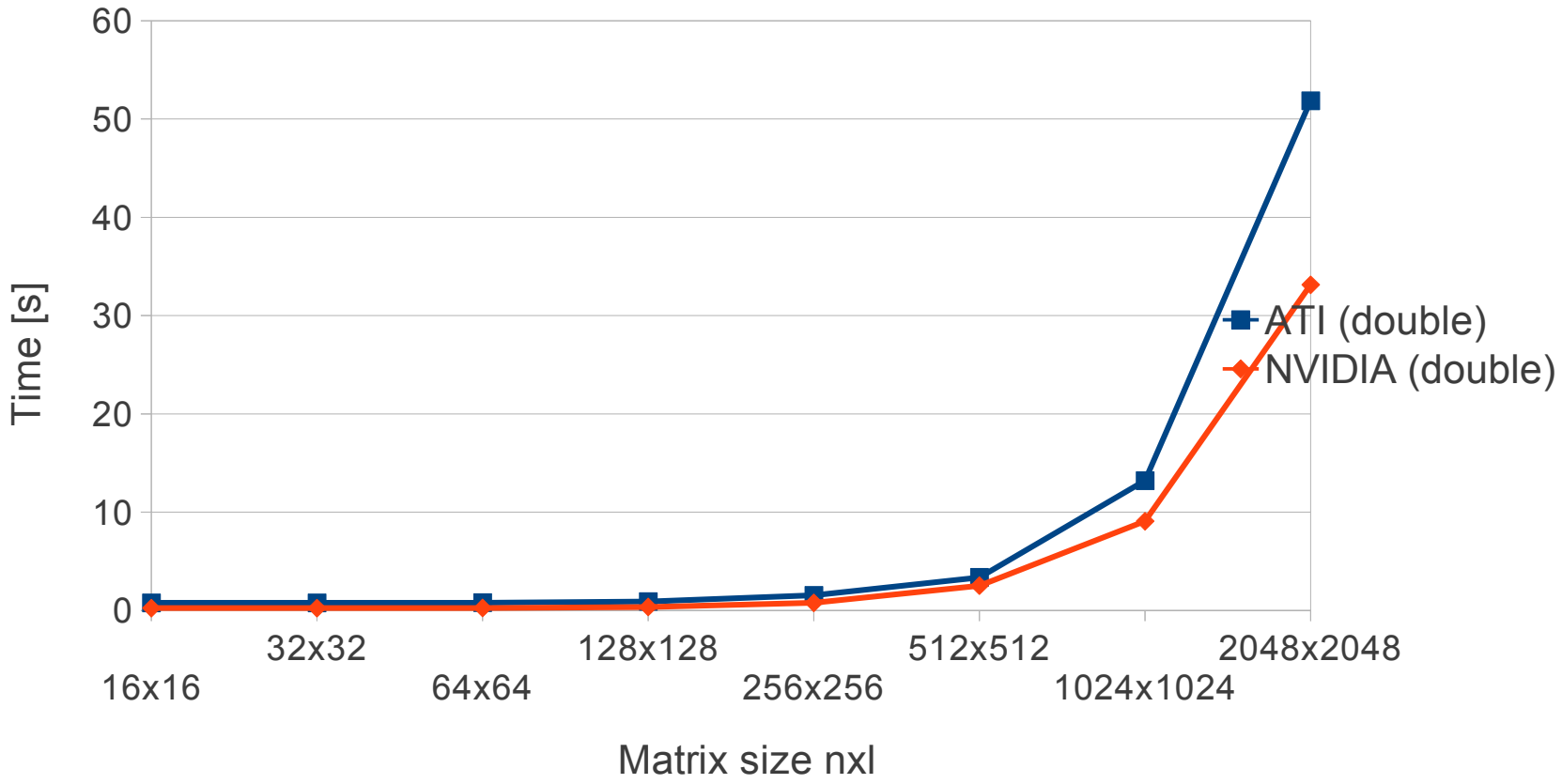


- ATI Radeon HD 5870 – MPDATA with autotuned configuration is 9.8 times faster than MPDATA with standard configuration



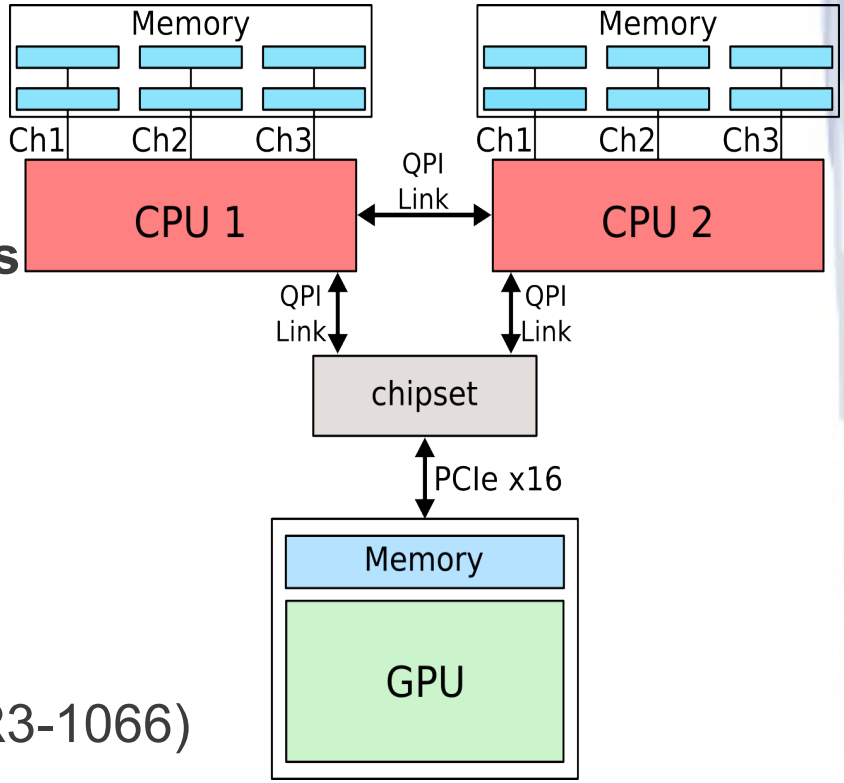
Comparison of MPDATA execution time between ATI Radeon and NVIDIA

- ATI Radeon HD 5870 vs. NVIDIA Tesla M2070Q
- Execution time for 1000 time steps

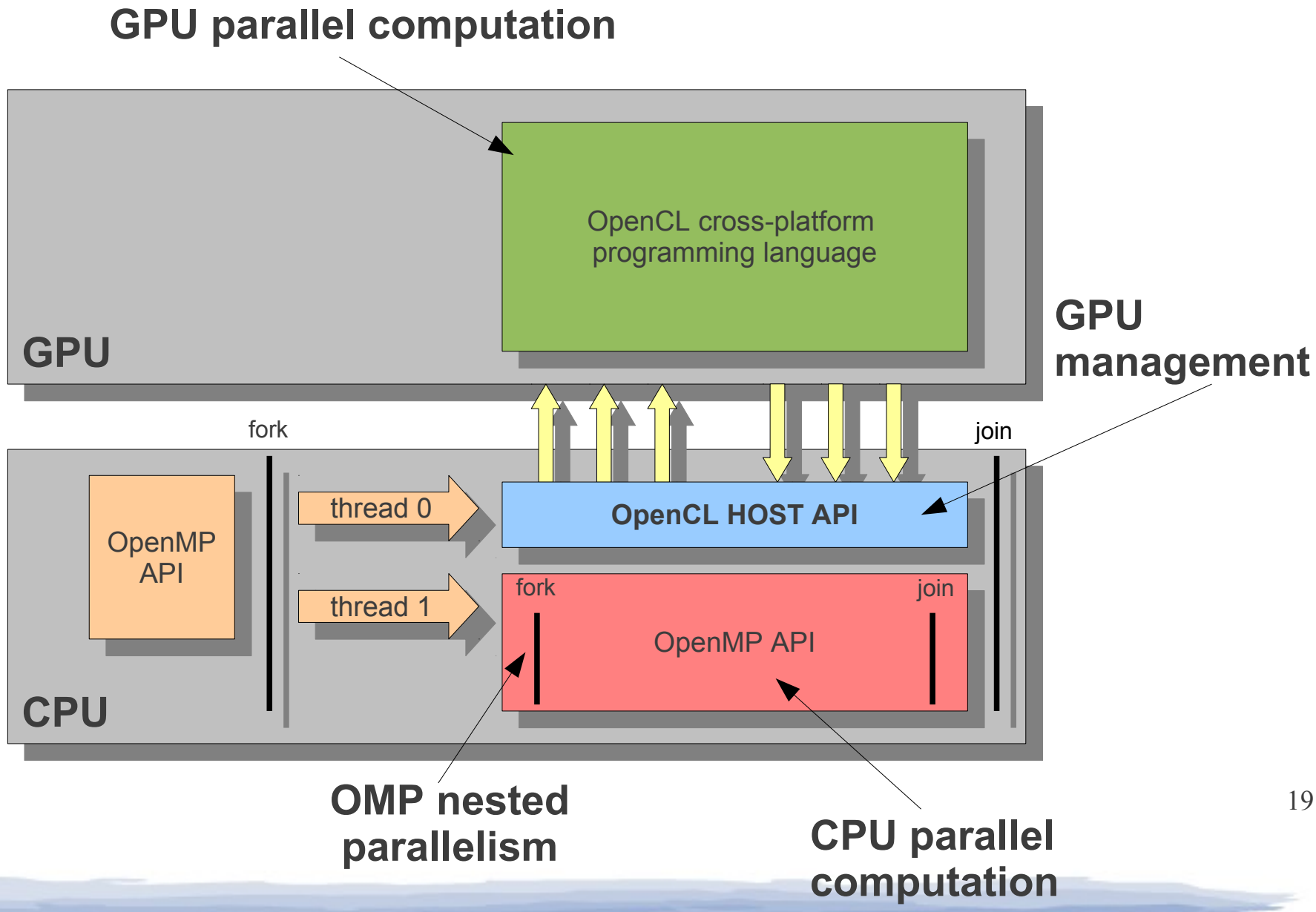


CPU-GPU architecture overview

- Our IBM BladeCenter HS22 includes:
  - 2 x Intel Xeon E5649: **0.24 Tflop/s**
  - 12 x 4GB DDR3-1066
  - NVIDIA Tesla M2070Q: **1.03 Tflop/s**
- Intel Xeon E5649 (Westmere):
  - 6 cores
  - 2.53 GHz
  - SSE4.2
  - memory bandwidth 25.6 GB/s (DDR3-1066)
  - QPI 23.44 GB/s in two directions



OpenMP and OpenCL hybrid programming model

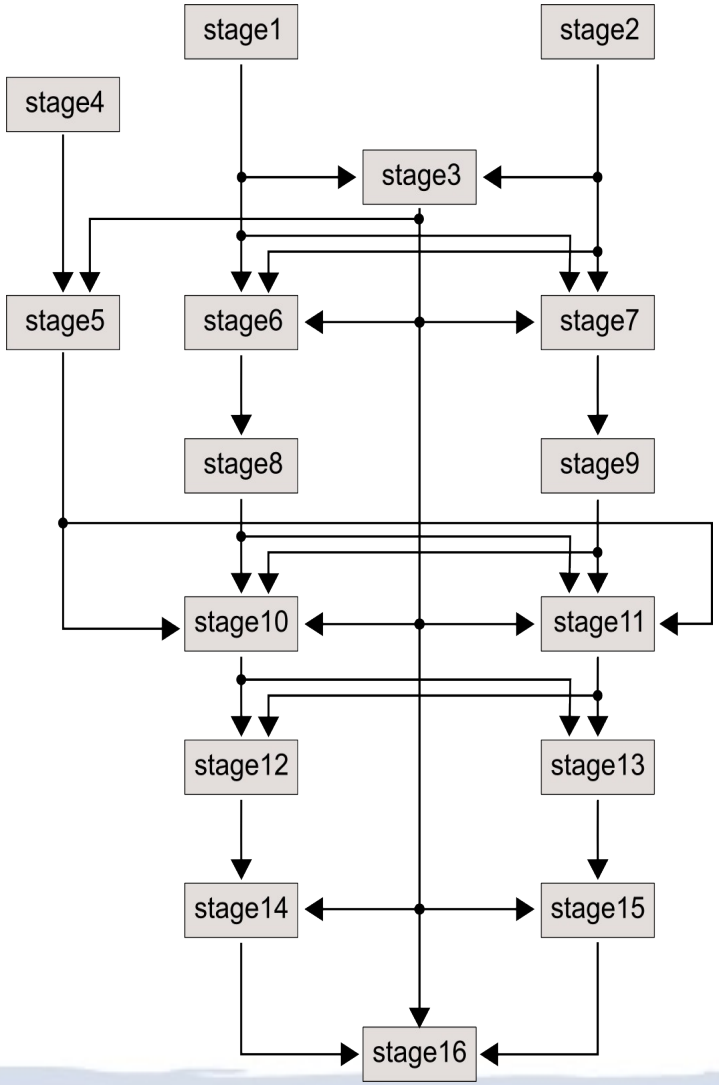


CPU parallelization of MPDATA algorithm

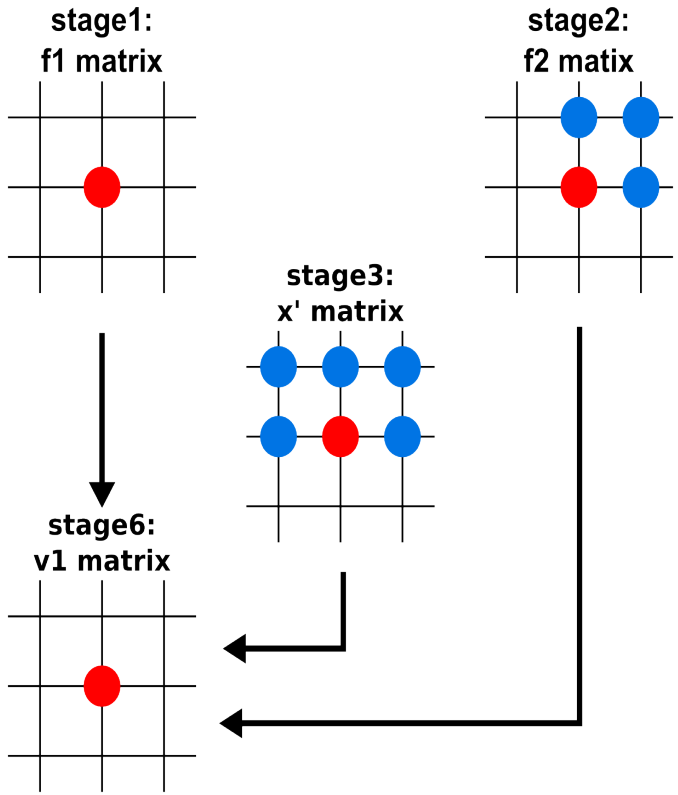
- In this approach, we distinguish the following aspects of the CPU parallelization:
  - multicore processing
  - cache reusing
  - SIMD vectorization
- The main SIMD processing challenges:
  - matching MPDATA algorithm to SIMD processing
  - replacing scalar operations by SSE intrinsics whenever is possible
  - suitable alignment of data: each row of matrices is aligned to at least 16 bytes

Data dependencies for MPDATA

- Data dependency tree



- Examples of data dependencies between stages:



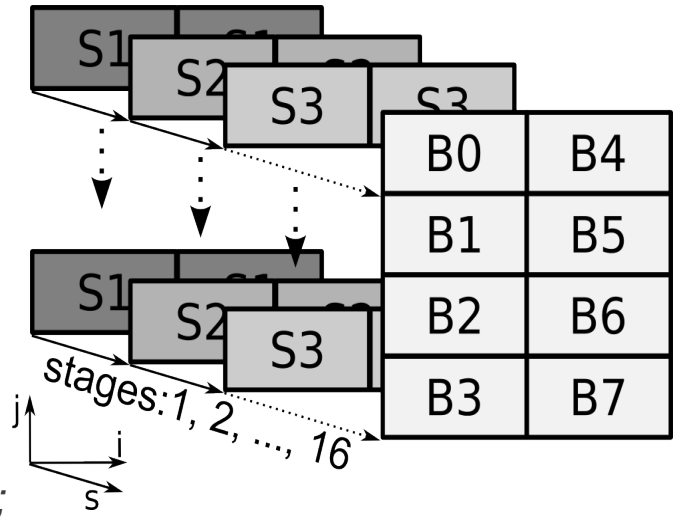
Block decomposition of MPDATA for CPU (1/4)

- The decomposition of MPDATA is based on the loop tiling technique:

for $\frac{n}{nBlcokSize}$ *tiles // i – dimension*

for $\frac{l}{lBlcokSize}$ *tiles // j – dimension*

```
MPDATA_block(...) {
    loading data from main memory to cache;
    stage1: parallel computations;
    saving partial results in cache;
    stage2: parallel computations
    saving partial results in cache;
    (…)
    saving final results in main memory for each block;
}
```



- Each *MPDATA_block* should be stored in cache, that allows for efficient cache reusing
- This approach reduces memory traffic

Block decomposition of MPDATA for CPU (2/4)

- Each *MPDATA_block* requires additional calculations for every stage because of data dependencies between each stage
- Additional computations of each stage correspond to hallo areas of four sides of block: ihT, ihB, ihL, and ihR (top, bottom, left, and right)
- Each stage has its own values of ihT, ihB, ihL and ihR, e.g.

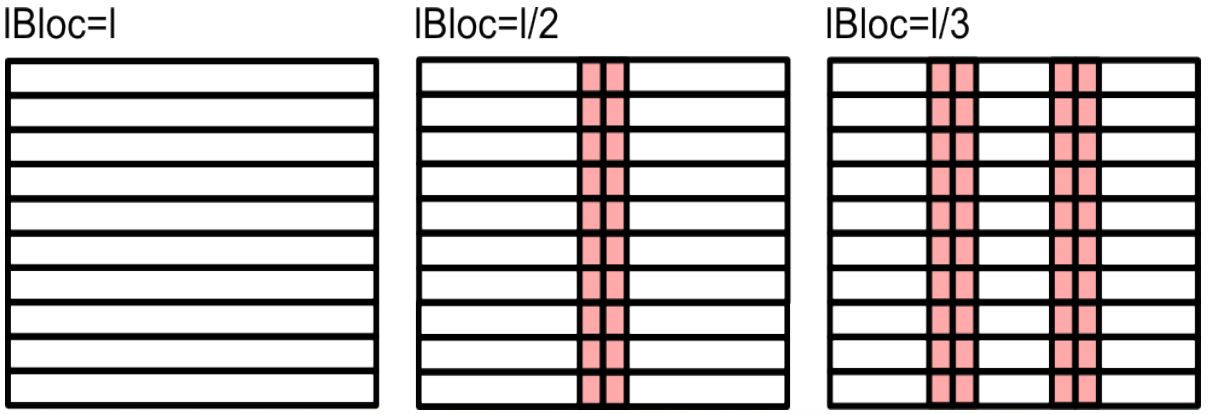
  stage1: ihT=2; ihB=3; ihL=2; ihR=2;

  stage3: ihT=2; ihB=2; ihL=2; ihR=2;
- Values of ihL and ihR are matched (increased) to vector size in order to utilize the SIMD processing: ihL and ihR are 2 or 4 for SSE extensions
- The smaller blocks size (larger number of blocks) the larger hallo areas (**more additional calculations**)
- Performance analysis for mesh of size 1024 x 1024:

block size	# of additional elements	cache consumption
16 x 16	55.7 %	74 KB
64 x 512	3.6 %	5484 KB
512 x 512	0.8 %	41402 KB

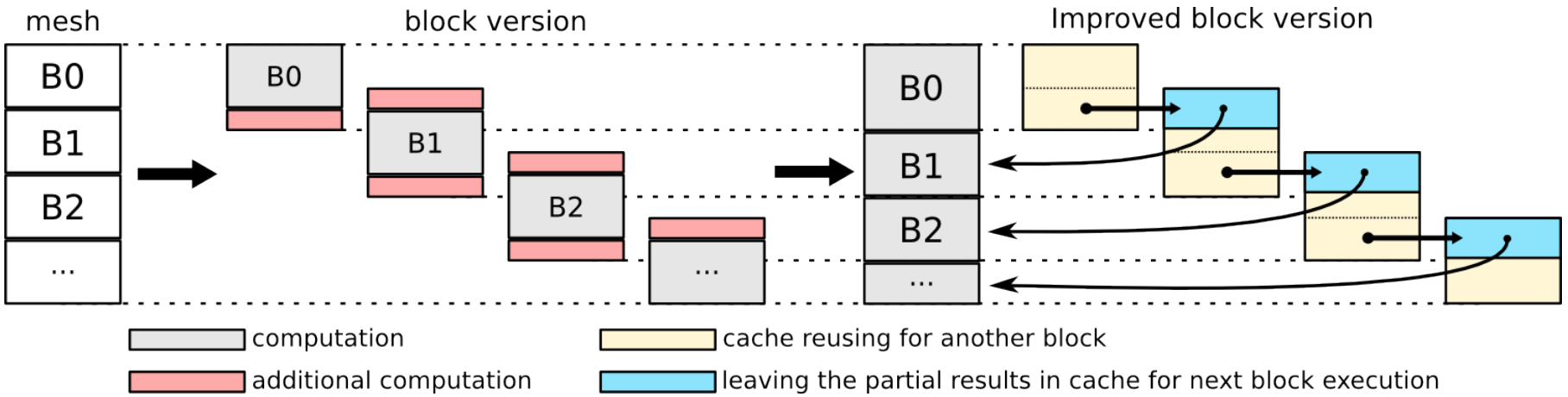
Block decomposition of MPDATA for CPU (3/4)

- We would like to develop a method that allows us to reduce or even avoid additional computations
- There are two groups of additional calculations: in vertical areas (ihL and ihR) and in horizontal areas (ihT and ihB)
- Additional calculations in vertical areas can be avoided if **lBlock=l**, but the size of nBlock must be small enough to save all blocks in cache
- If the size of cache is not large enough to save all blocks, the size of lBlock should be **lBlock=l/2**, which allows us to reduce additional operations



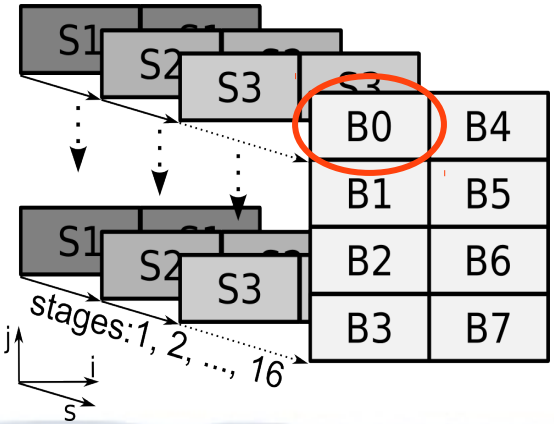
Block decomposition of MPDATA for CPU (4/4)

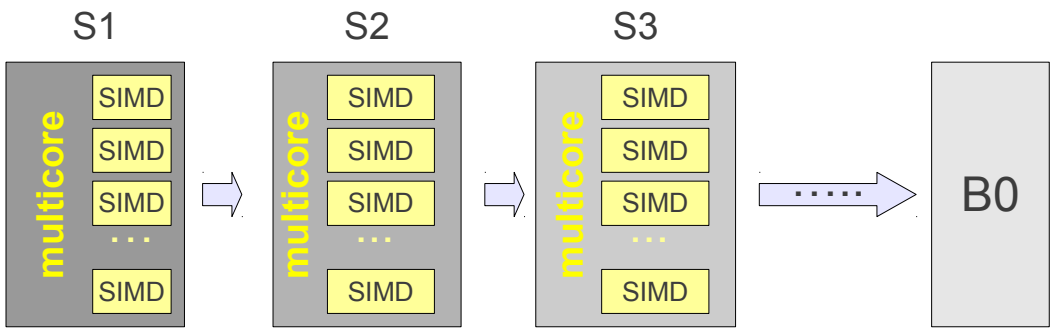
- Because of dependencies, each block requires additional calculations which are repeated by other blocks
- Independent of nBlock size, additional calculations in horizontal areas can by avoided by leaving partial results in cache:
  - the order of blocks execution is important
  - it requires additional offline and online management of computations for all stages, as well as smart mapping of partial results onto cache space for computations within neighbour blocks



CPU parallelization

- Blocks are executed sequentially in the following order:
  B0, B1, B2, … - only this order of execution allows us to reduce or even avoid additional computations
- For each block, a sequence of stages S1, S2, S3,… is executed, satisfying the dependency tree
- The calculations within a stage are divided between the available threads separately (each stage has its own operational intensity), according to the hallo areas and dependencies between stages
- SIMD processing is applied inside each thread





CPU parallelization: performance results

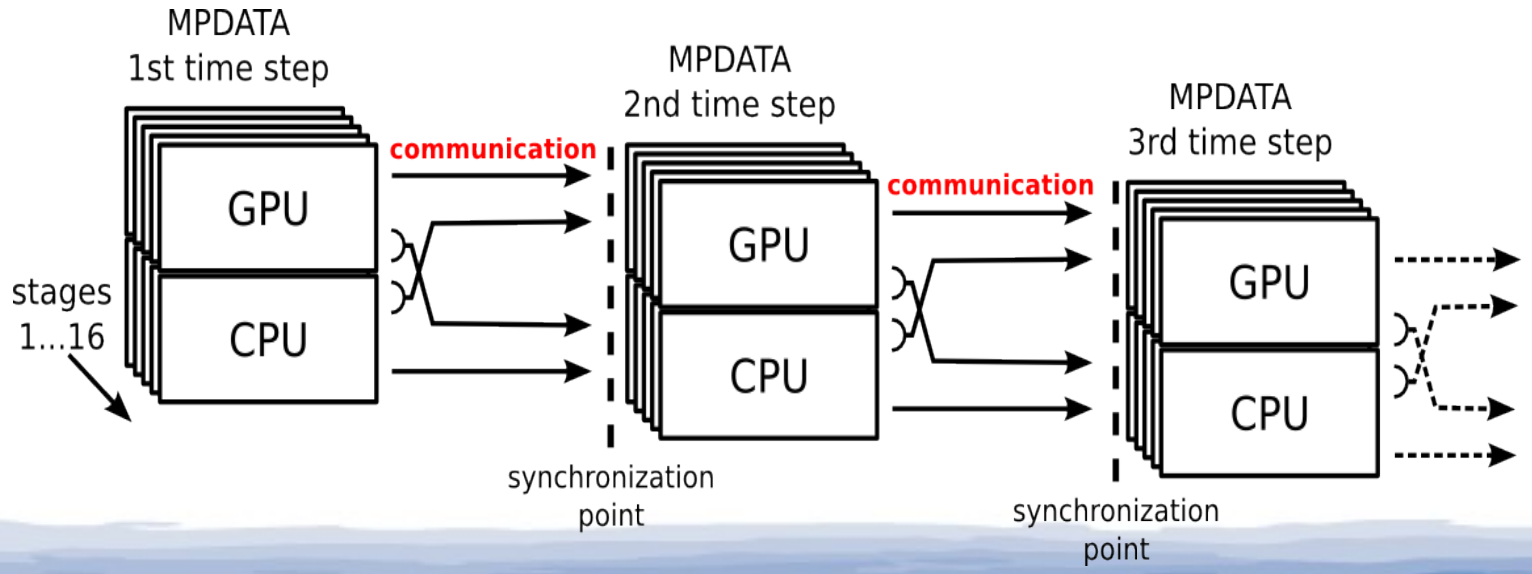
- Performance results of double precision MPDATA with mesh of size 1024x1024 for 500 time steps, using Intel Xeon E5649 CPU

Version	Parallel mode	Time [s]	Speedup
Serial	1 core	55.34	1
Parallel	6 cores	18.35	3.02
Parallel	6 cores and SSE	17.48	3.17
Block (64×512)	6 cores and SSE	13.37	4.13
Improved block (16×1024)	6 cores and SSE	10.95	**5.05**

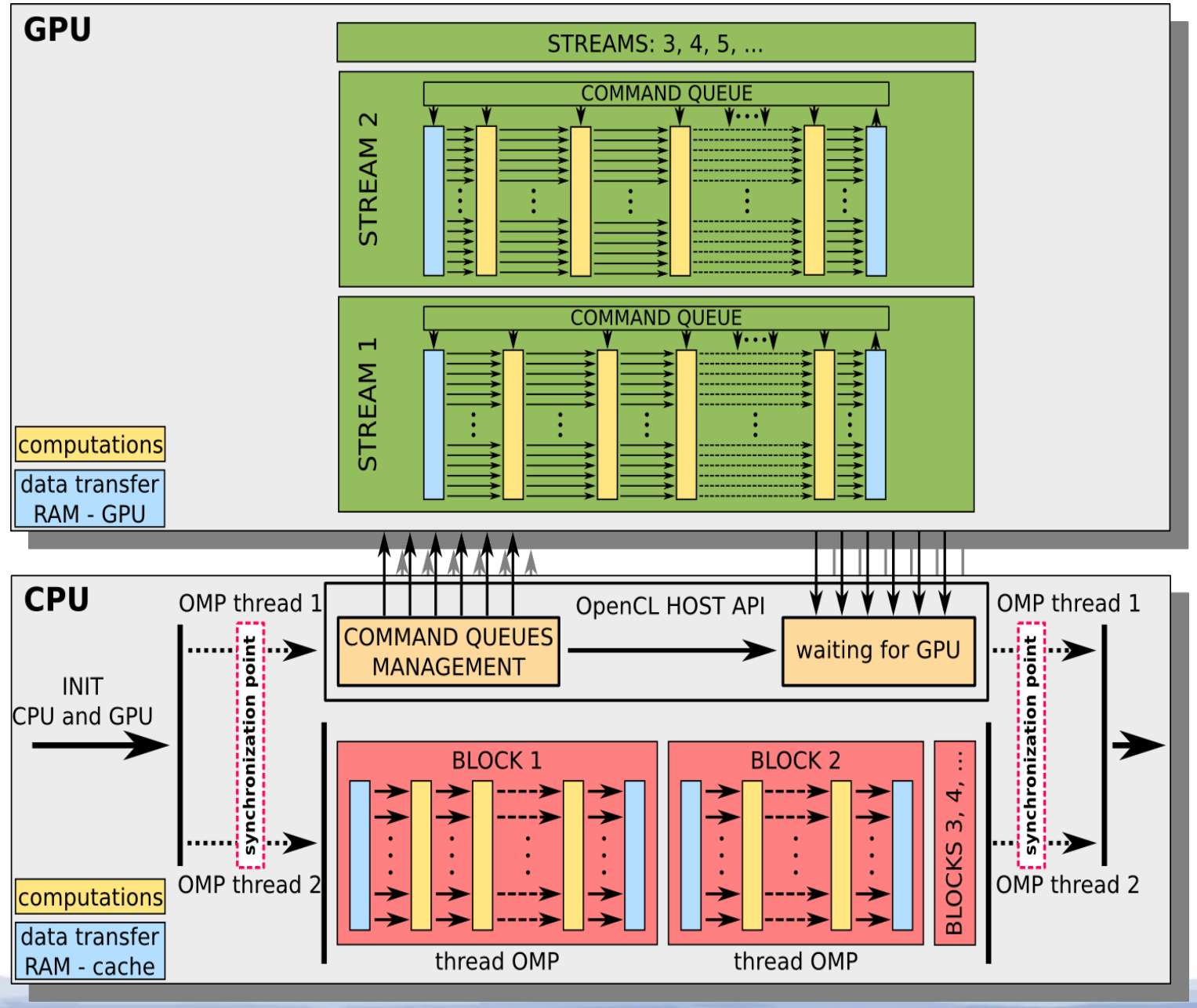
- Improved block version achieves speedup of **1.59** over parallel version
- Improved block version achieves speedup of **1.22** over original block version (3.6% of additional calculated elements)

CPU-GPU parallelization

- MPDATA is divided into two parts: GPU part and CPU part
- Each part is responsible for computing all 16 stages
- CPU and GPU parts require additional calculations according to data dependencies
- CPU and GPU communication is required between time steps
- Currently, hybrid CPU-GPU version is based on **static load balancing:** 50% of the algorithm is executed on CPU and 50% on GPU



Management of CPU-GPU resources



CPU-GPU: preliminary performance results (1/2)

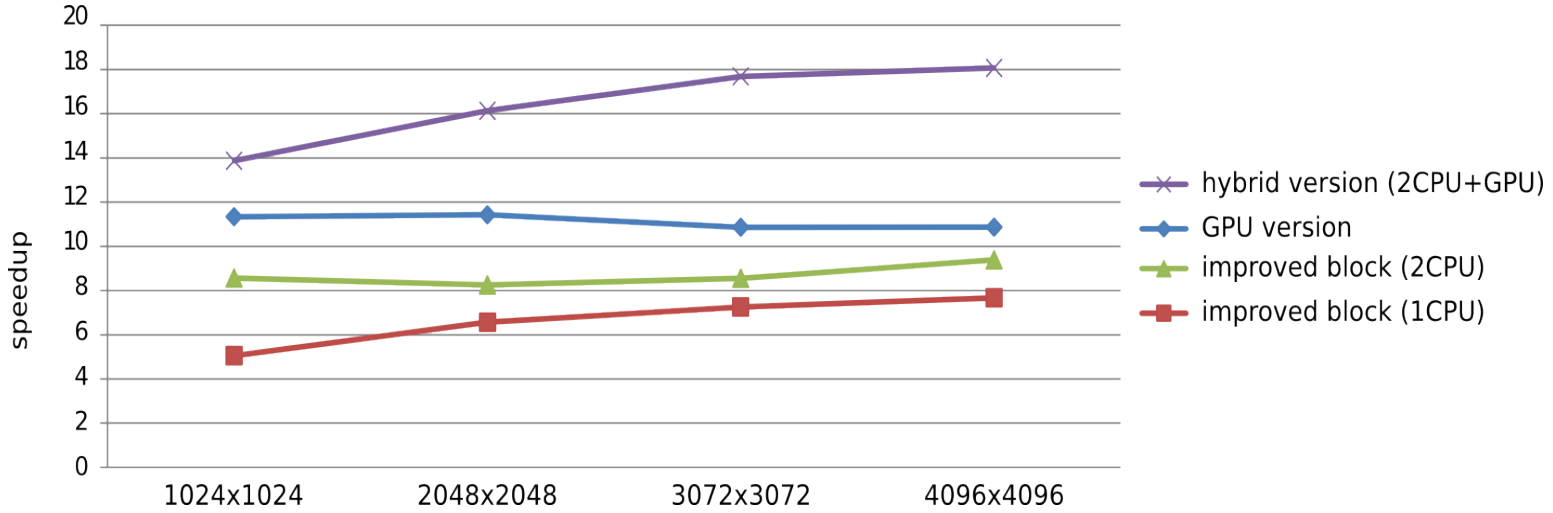
- Performance results of double precision MPDATA with mesh of size 1024×1024 for 500 time steps

Version	Device	Time [s]	Speedup
Serial	1 core of Intel Xeon E5649	55.34	1
Block	1 x Intel Xeon E5649 with SSE	10.95	5.05
Block	2 x Intel Xeon E5649 with SSE	6.56	8.44
GPU	NVIDIA Tesla M2070Q	4.88	11.34
Hybrid	2 x Intel Xeon E5649 with SSE NVIDIA Tesla M2070Q	3.99	**13.87**

- GPU version achieves speedup of **1.34** against two-CPUs block version
- Hybrid version (2CPU+GPU) achieves speedup of **1.2** over GPU version
- The reason for a relatively small performance advantage of hybrid version over GPU version is overhead associated with creating nested OMP threads, which takes about **0.6** seconds for 500 time steps

CPU-GPU: preliminary performance results (2/2)

- Speedup for double precision parallel MPDATA against sequential version



- Hybrid version (2CPU+GPU) gives the best results for all mesh sizes. For example, for mesh of size 4096×4096:
  - execution time for serial version is 840.34 seconds, while for hybrid version is only 46.43 seconds, which gives speedup of **18.09**
  - hybrid version achieves speedup of **1.66** over GPU, and **1.92** over two CPUs
  - GPU version achieves speedup of **1.41** against one CPU, and **1.16** over two CPUs
  - due to a load unbalance in hybrid version, there is about 5-7% loss of performance

Conclusions and future work (1/2)

- GPGPU computing is a promising approach for increasing performance of numerical simulations of geophysical flows using the EULAG model
- Adaptation of this model (in particular MPDATA algorithm) to GPU architecture is based on the hierarchical approach
  - MPDATA task decomposition allows for avoiding dependencies between work-groups
  - Stream processing allows for overlapping data transfer with computations
- Automatic adaptation of MPDATA to GPU architecture is a very complex problem, which requires to apply a miscellaneous optimization techniques for different parameters of configuration
- Autotuning mechanism allows for achieving speedup of about **10** for ATI Radeon HD 5870 over standard configuration, and 2.6 in case of NVIDIA

Conclusions and future work (2/2)

- Using both OpenMP and OpenCL for hybrid model of parallel programing allows us to take advantage of CPU-GPU architecture
- Two separate adaptations of MPDATA algorithm to CPU-GPU hybrid architecture are required, in order to better utilize features of hybrid architecture
- New strategies for memory and computing resources management allow us to ease memory bounds, and better exploit the theoretical floating point efficiency of hybrid architecture
- The hybrid version gives the best results for all mesh sizes
- The future work will be focus on:
  - parallelization of MPDATA in 3D based on 3D grid decomposition
  - adaptation to other architectures – clusters with CPU-GPU nodes, Intel MIC accelerator architecture, …
  - taking into consideration not only performance but also power consumption

3rd International EULAG Workshop on Eulerian/Lagrangian methods for fluids

Thank YOU for your attention!