Intel MIC Programming Workshop @ IT4I
February 7-8, 201

Exploring the impact of Intel MIC and Intel CPU architectures on accelerating scientific applications

Łukasz Szustak
lszustak@icis.pcz.p

Roman Wyrzykowski
roman@icis.pcz.pl

Institute of Computer and Information Science
Czestochowa University of Technology, Poland

Motivation of our research

- Emerging computing platforms become increasingly complex, hierarchical and heterogeneous
- The efficient usage of all opportunities offered by modern computing systems represents a global challenge
- The research background of our team implies a profound and efficient exploration of emerging multi-/many-core architectures
- Our research includes two scientific applications:
  - Multidimensional Positive Definite Advection Transport Algorithm
    - Low-level approach
  - Numerical model of solidification
    - High-level approach

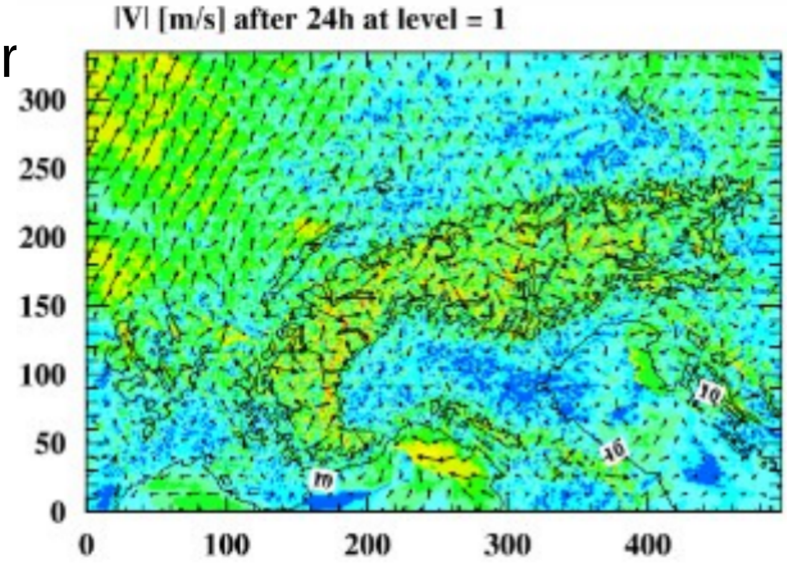
MPDATA

Łukasz Szustak
lszustak@icis.pcz.p

Roman Wyrzykowski
roman@icis.pcz.pl

MPDATA

- Multidimensional Positive Definite Advection Transport Algorithm is one of the main parts of the EULAG model
- MPDATA is a real-life CFD application
- EULAG is an established computational model developed by the group headed by Piotr K. Smolarkiewicz for simulating thermo-fluid flows across a wide range of scales and physical scenarios
- One of the most interesting applications of the EULAG model is numerical weather prediction (NWP)
- In our research, we propose to rewrite the main parts of EULAG and replace standard HPC systems by emerging computing cluster



MPDATA

- MPDATA belongs to the class of the forward-in-time algorithms which assume iterative execution of multiple time steps
- The number of required time steps depends on a type of simulated physical phenomenon, and can exceed even few millions
- The whole MPDATA computations in each time step are decomposed into a set of 17 heterogeneous stencils
- The stages depend on each other
- A single MPDATA time step requires 5 input and 1 output matrices
- MPDATA, as a part of EULAG, is interleaved with other important computation in each time step
- We focus on simulations using 3D grid

Basic parallel version of MPDTA

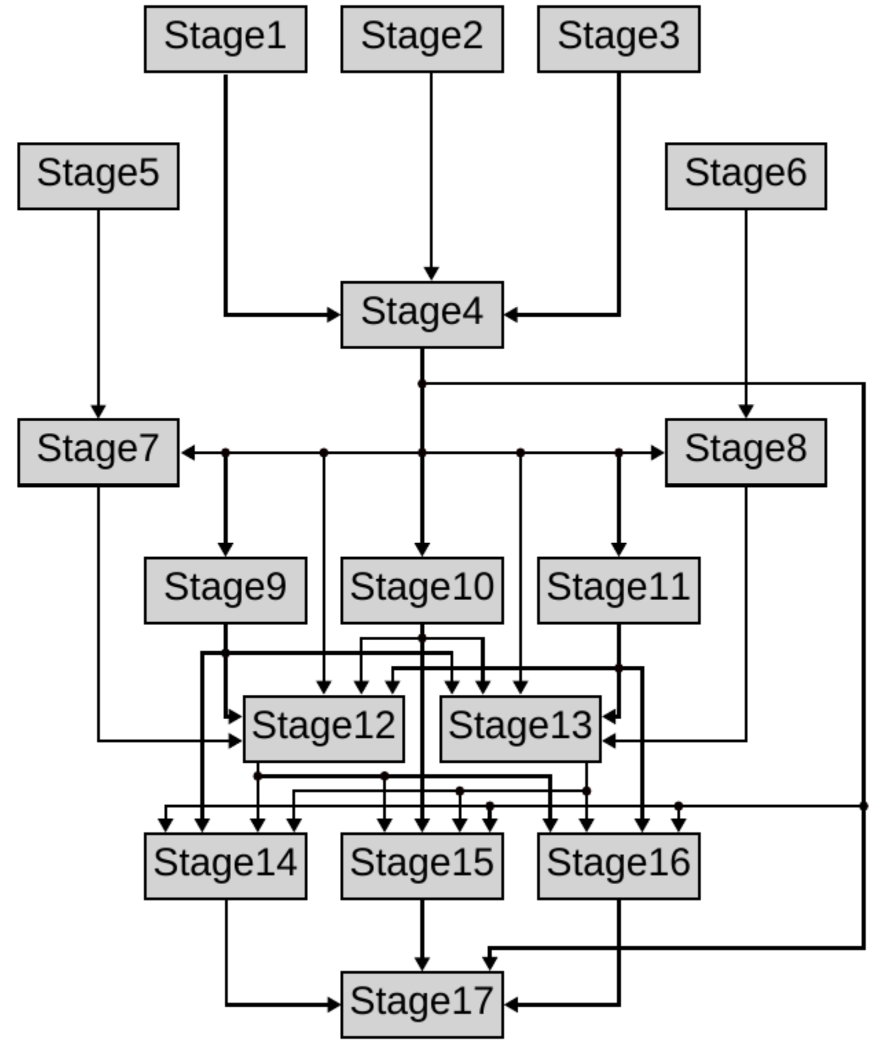
```
#pragma omp for
  for(i=0; i<n;++i)
    for(j=0;j<m;++j)
      for(k=0;k<l;++k)
        f1(i,j,k) = ...

#pragma omp for
  for(i=0; i<n;++i)
    for(j=0;j<m;++j)
      for(k=0;k<l;++k)
         f2(i,j,k) = ...
/*
....
*/
#pragma omp for
  for(i=0; i<n;++i)
    for(j=0;j<m;++j)
      for(k=0;k<l;++k)
         x(i,j,k) = ...
```

Stage 1

Stage 2

Stage 17

Data dependencies



MPDATA is a memory-bounded algorithm

MPDATA on a single node (shared-memory version)

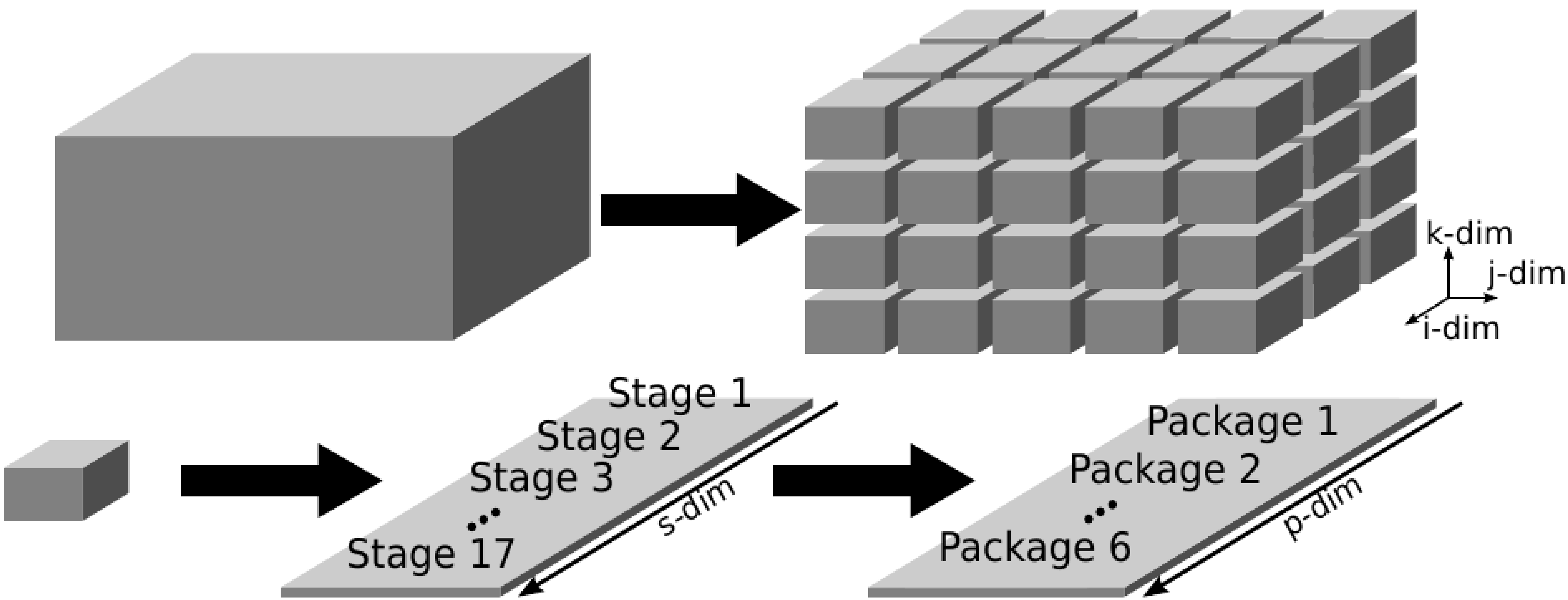
- To improve performance we reorganized computation inside each time step:
  - (3+1)D decomposition of MPDATA
  - Improving efficiency of this decomposition
  - New strategies for workload distribution and data parallelism
    - Proprietary scheduler with affinity-aware threads/cores placement
  - Providing 2 levels of threads/cores grouping:
    - Work teams of **cores** – better use of LLC
    - Work groups of **threads** – better use of L1
  - New strategies for synchronization (dedicated to MPDATA)
  - Providing appropriate data layout and hints for auto-vectorization
  - Providing configurability of the code for application portability

(3+1)D decomposition of MPDATA

- The main goal is to eliminate main memory data transfers associated with all intermediate computation/matrices
- Therefore, intermediate outcomes of computation should be kept in cache only - **without transferring them to the main memory**
- The main memory traffic will be generated only to transfer the required input and output data for each MPDATA time step
- To reach this goal, we proposed (3+1)D decomposition of MPDATA computation that is based on a combination of **loop fusion** and **loop tiling** techniques

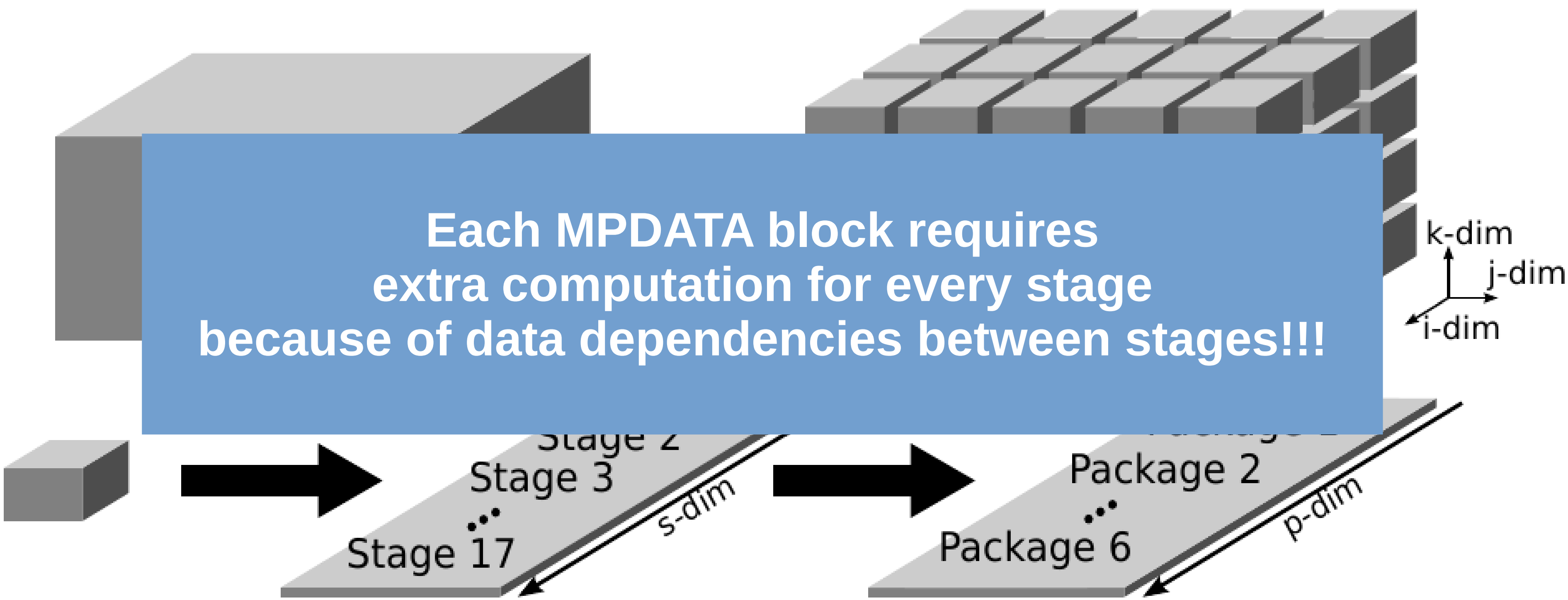
(3+1)D decomposition of MPDATA

- As a results, the MPDATA domain is decomposed into a set of blocks,
- Blocks are processed **independently**, and each block is computed in **parallel** by all cores (within each time steps)



(3+1)D decomposition of MPDATA

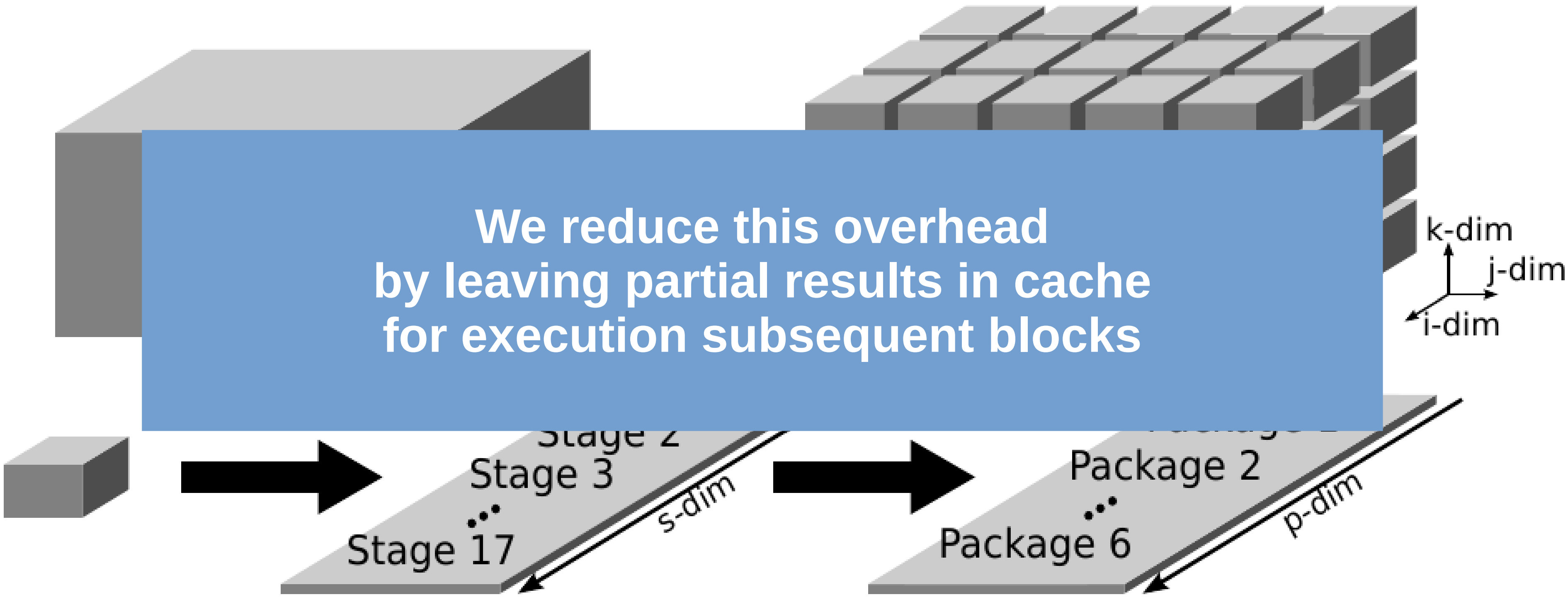
- As a results, the MPDATA domain is decomposed into a set of blocks,
- Blocks are processed **independently**, and each block is computed in **parallel** by all cores (within each time steps)



Each MPDATA block requires extra computation for every stage because of data dependencies between stages!!!

(3+1)D decomposition of MPDATA

- As a results, the MPDATA domain is decomposed into a set of blocks,
- Blocks are processed **independently**, and each block is computed in **parallel** by all cores (within each time steps)

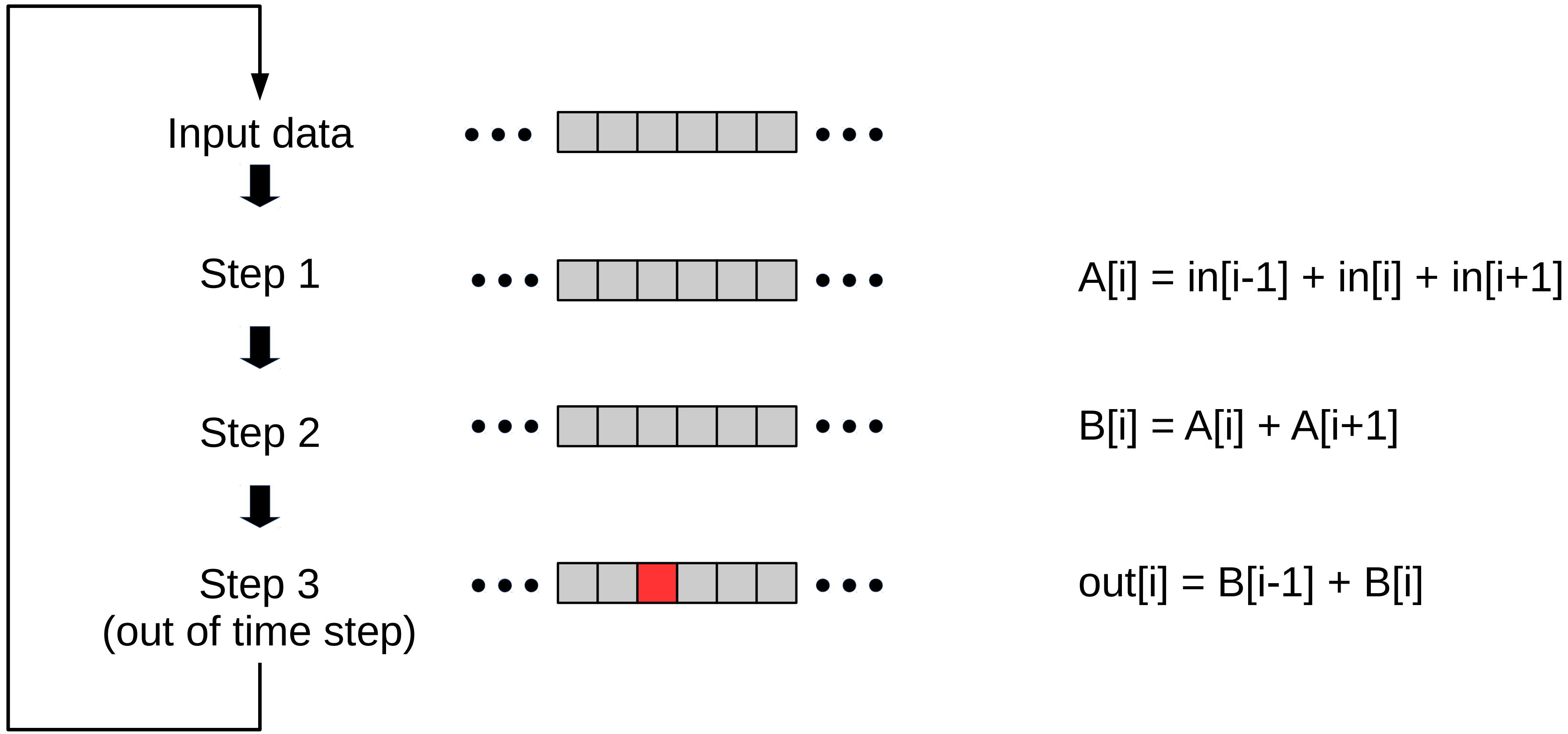


We reduce this overhead by leaving partial results in cache for execution subsequent blocks

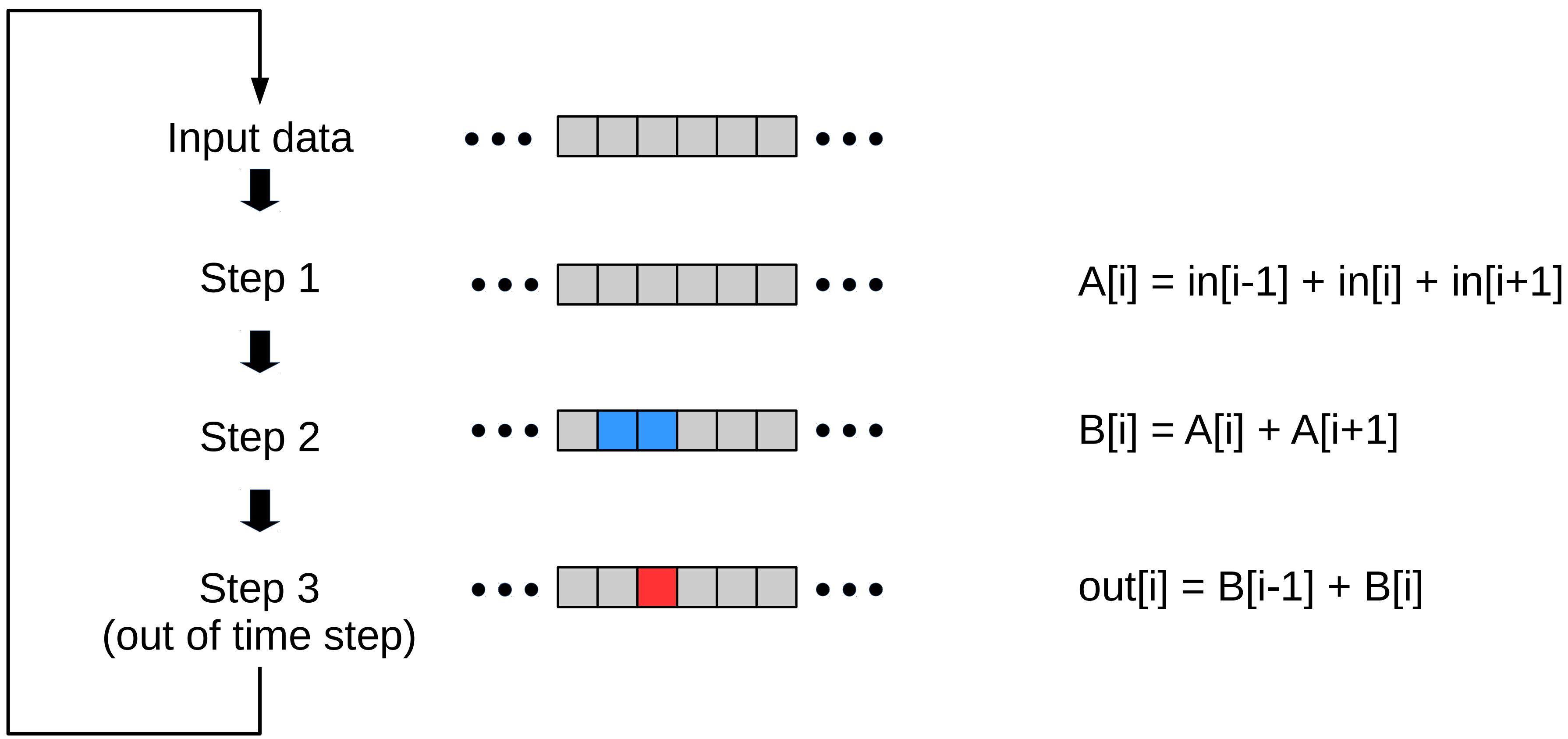
Idea of islands of cores

- The (3+1)D decomposition shifts the data traffic from the main memory to the cache hierarchy
- In consequence, a lot of intra cache communication between threads is generated
- New strategy of MPDATA decomposition requires also much more points of synchronizations than the basic one
- It is particularly significant when more than 200 Intel MIC threads cooperates
- To solve this issue we propose to create islands of cores that:
  - share all input data,
  - provide independent computations according to 17 stages,
  - and return common output data for every MPDATA time step

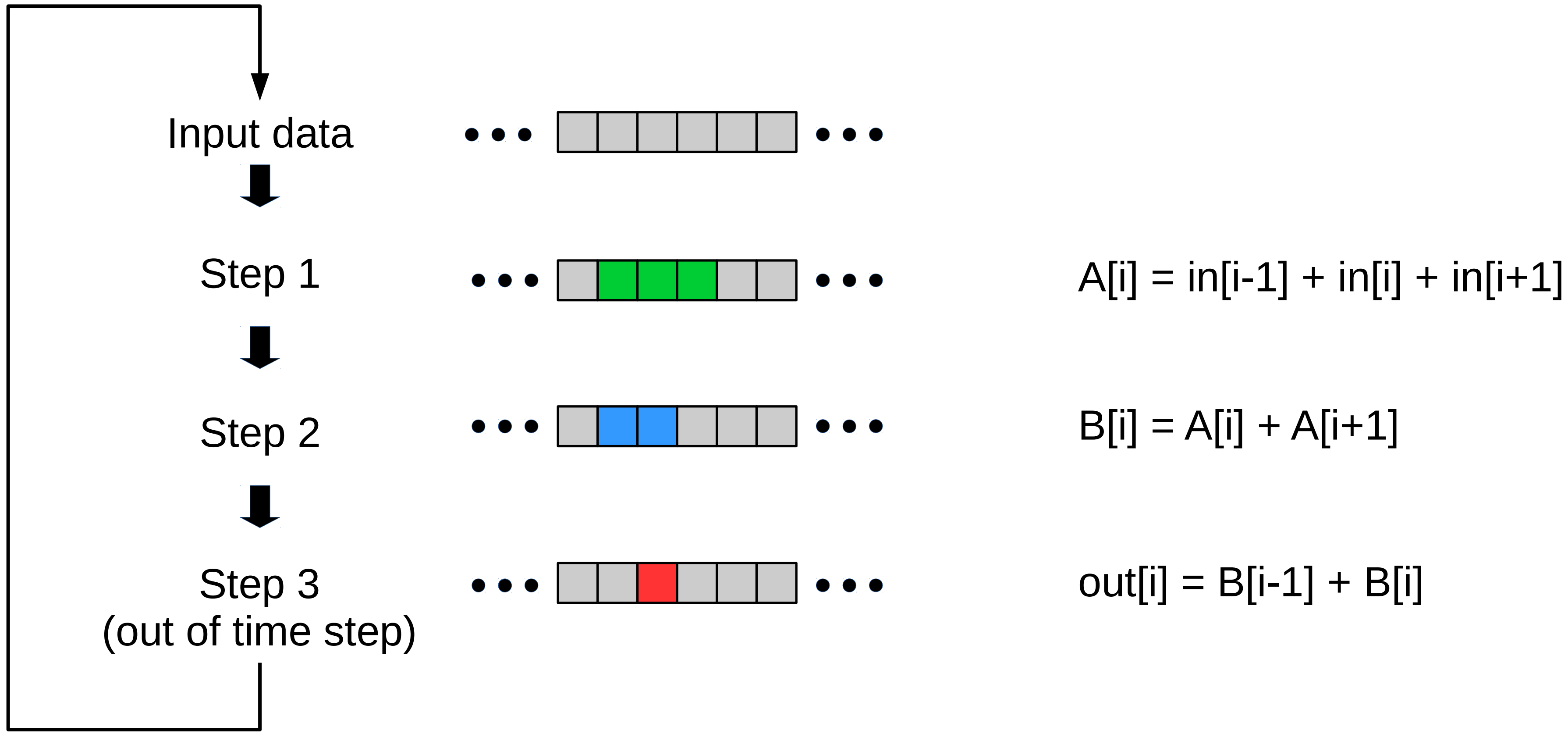
Trade-off between computation and communication for stencils



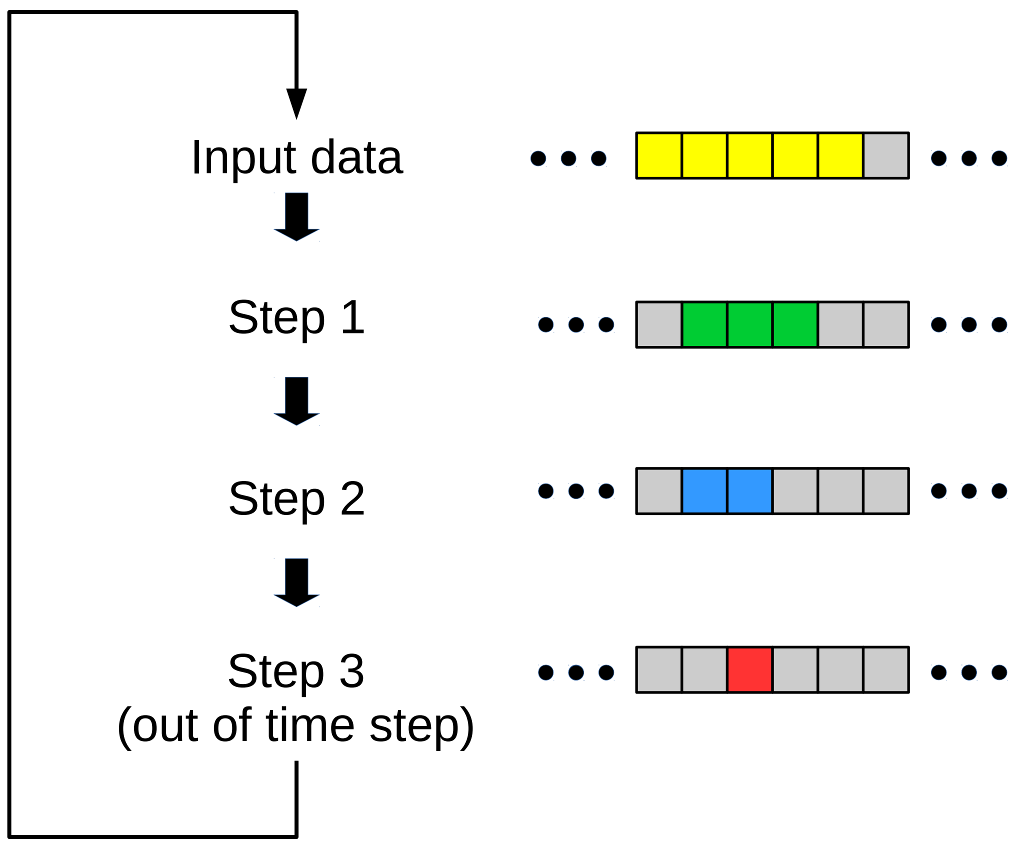
Trade-off between computation and communication for stencils



Trade-off between computation and communication for <u>stencils</u>



Trade-off between computation and communication for <u>stencils</u>

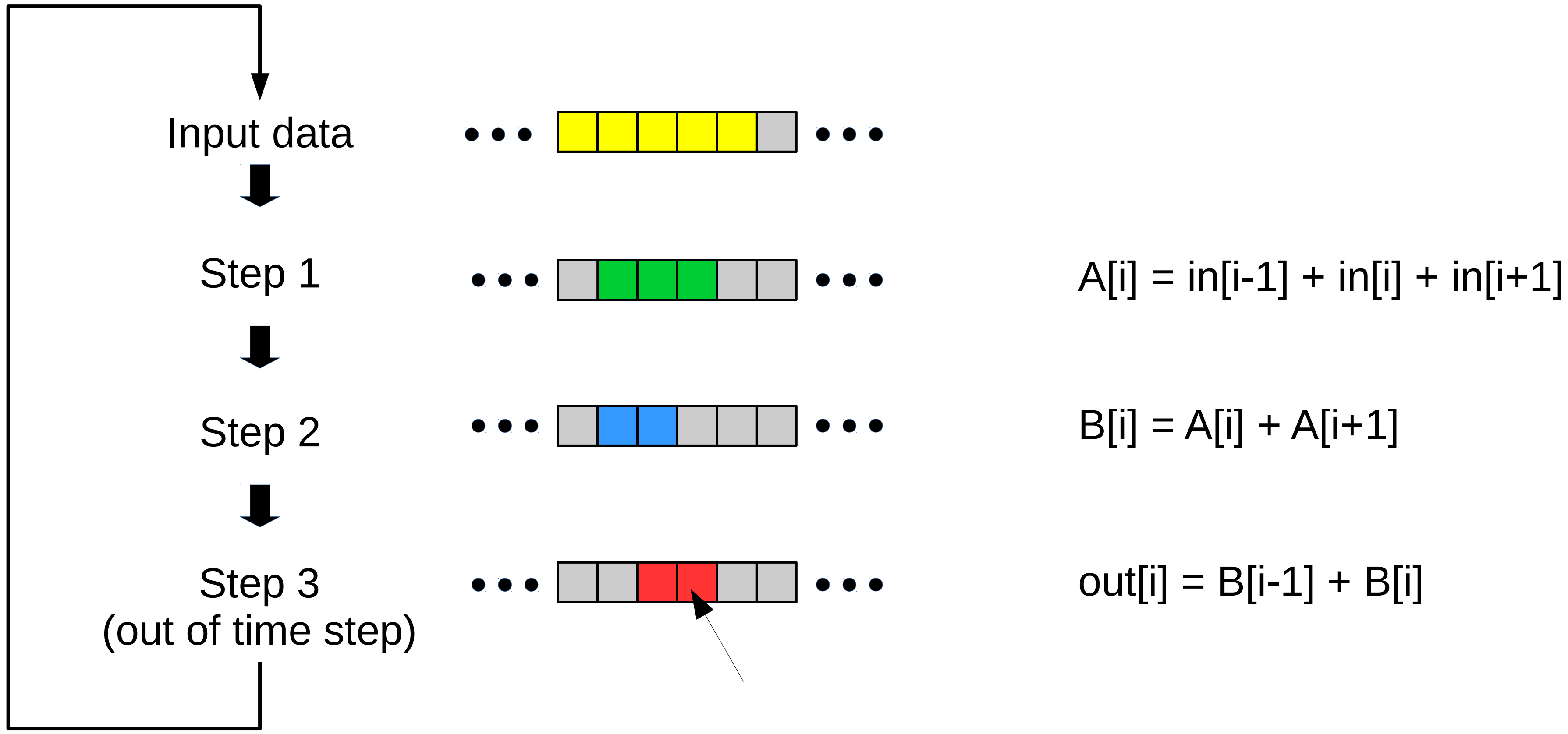


A[i] = in[i-1] + in[i] + in[i+1]

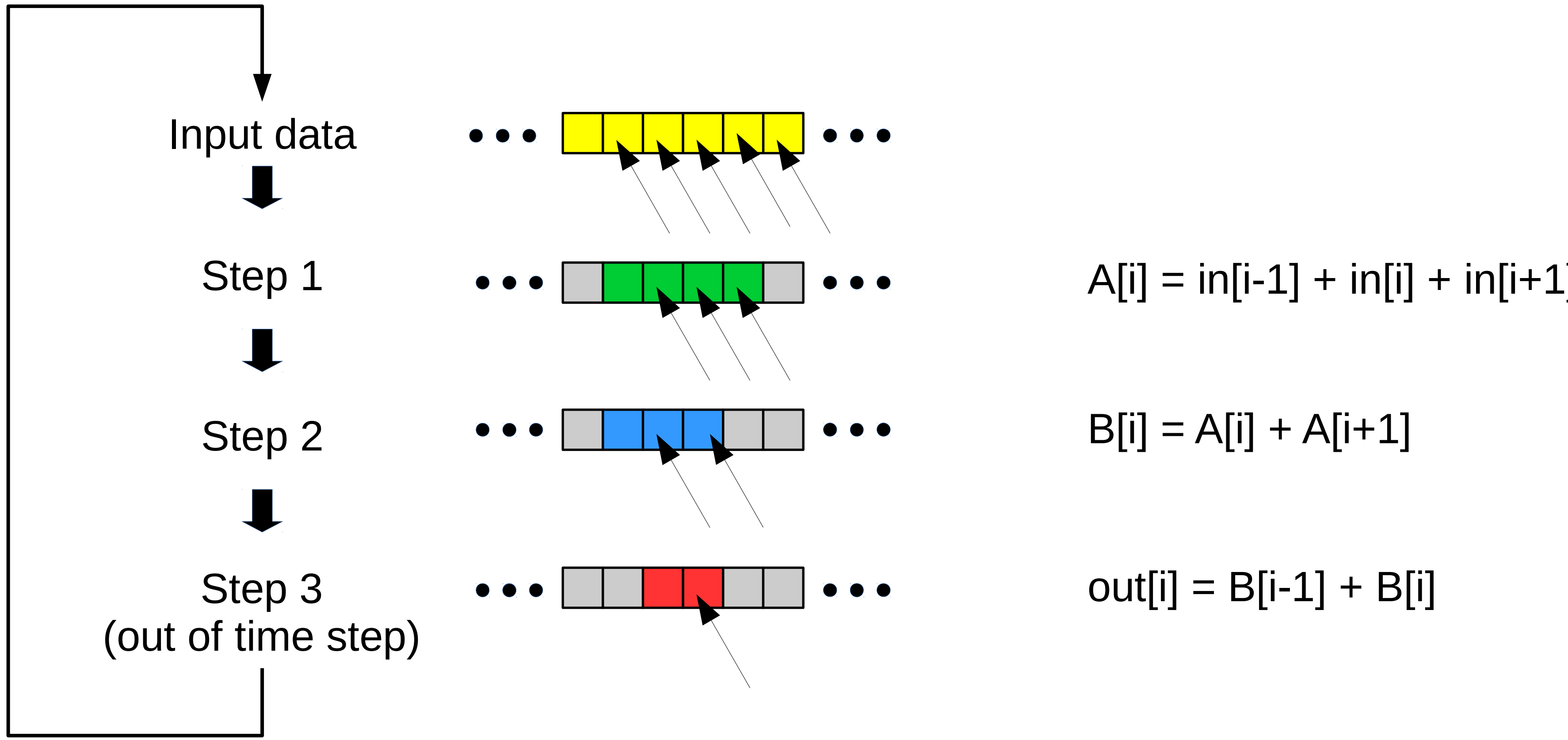
B[i] = A[i] + A[i+1]

out[i] = B[i-1] + B[i]

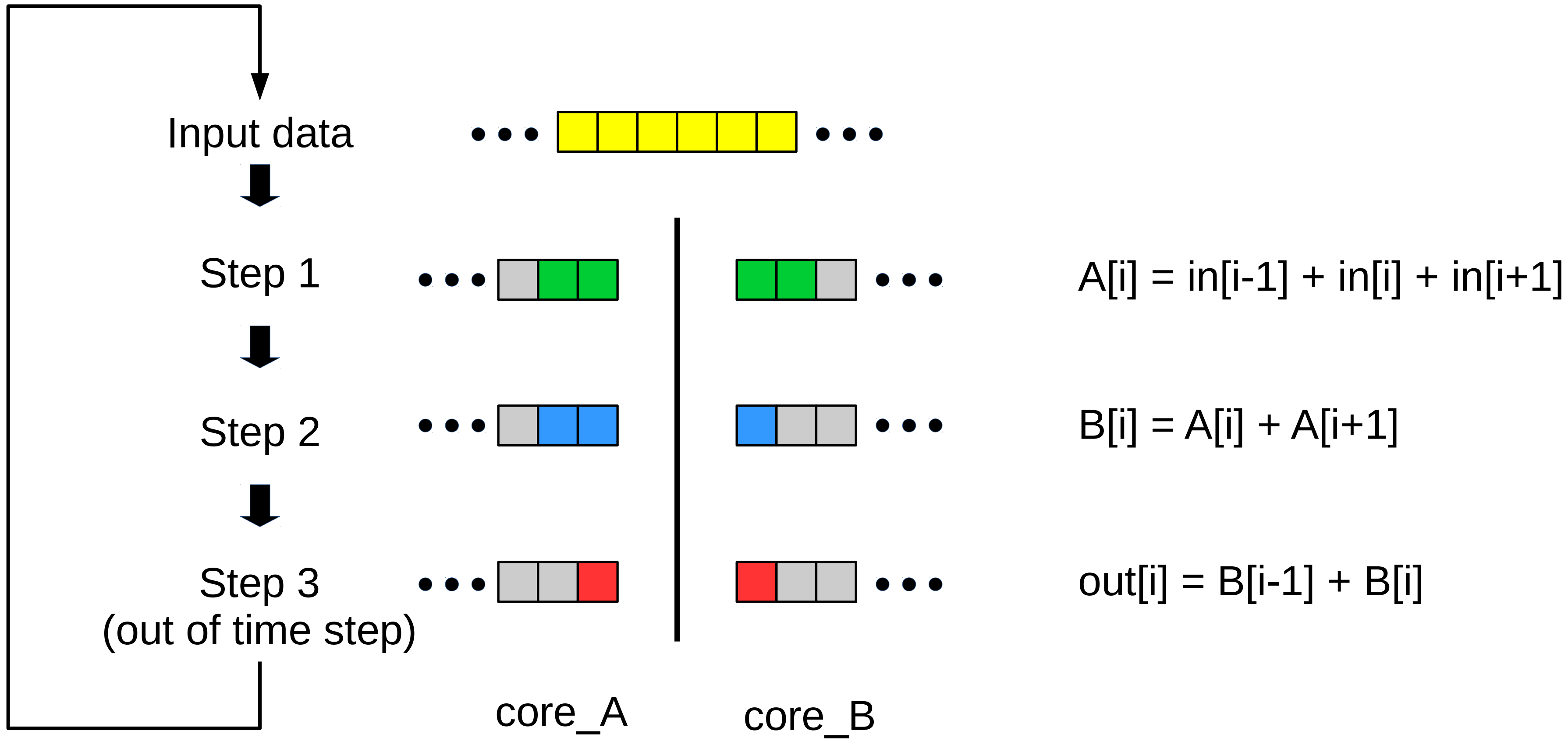
Trade-off between computation and communication for stencils



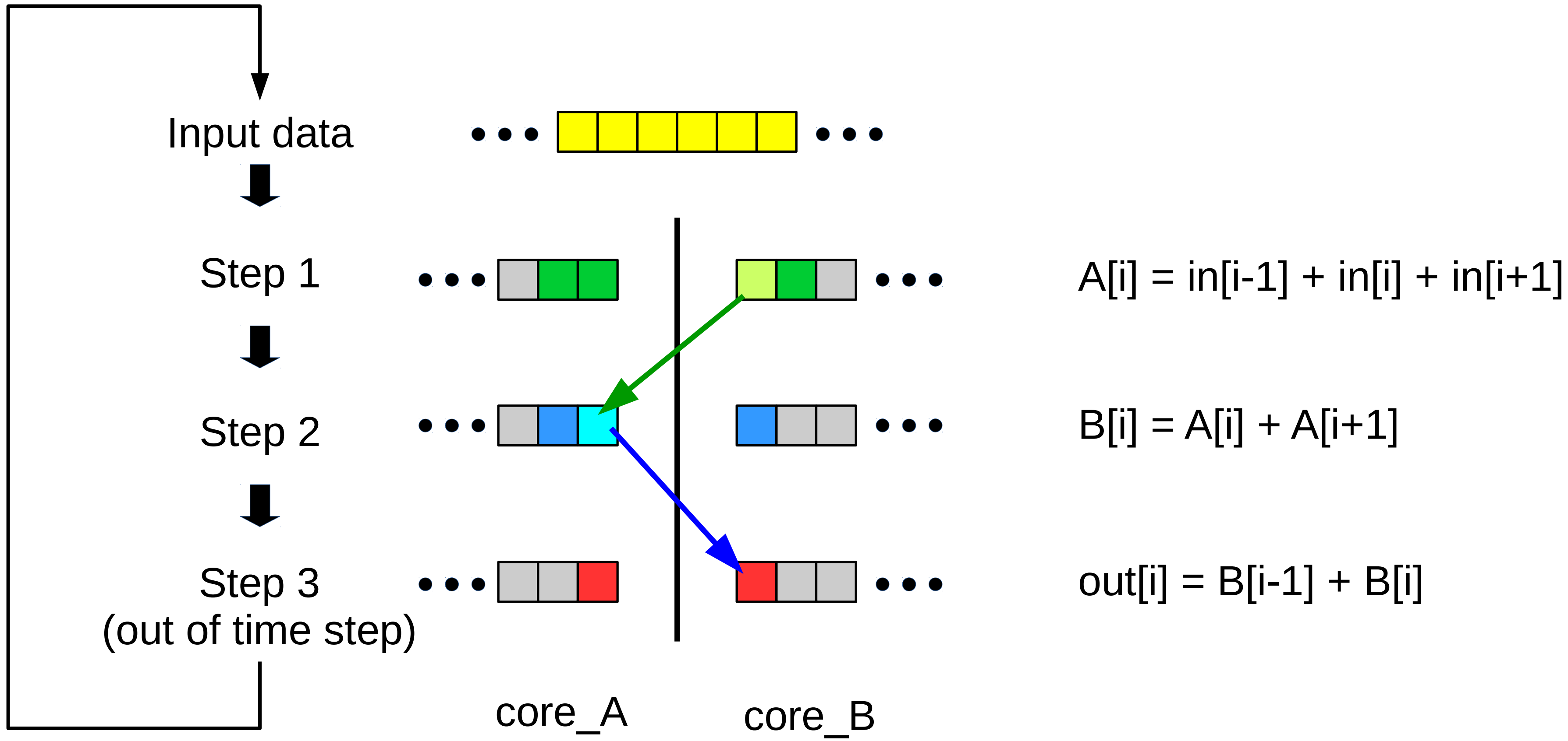
Trade-off between computation and communication for <u>stencils</u>



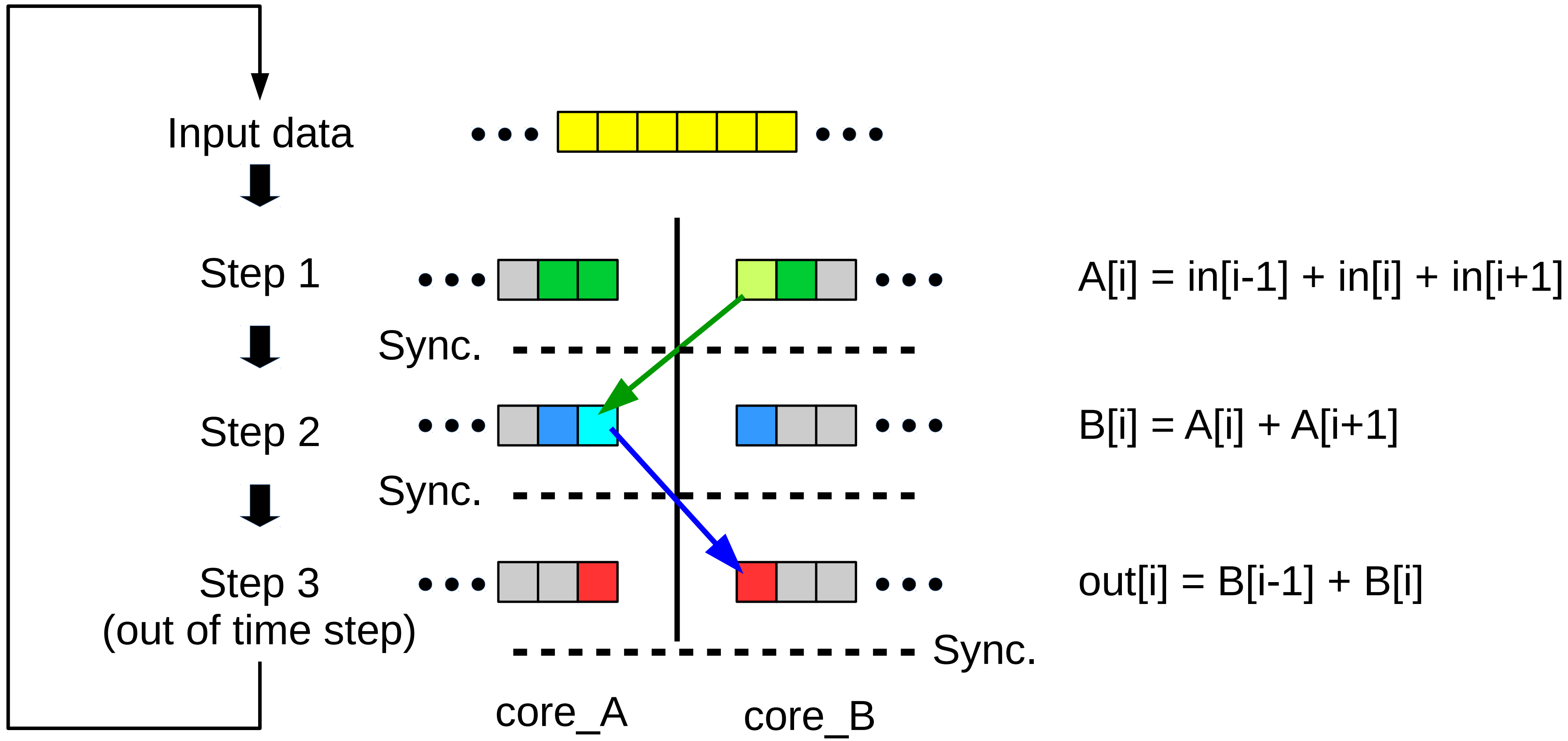
Trade-off between computation and communication for <u>stencils</u>



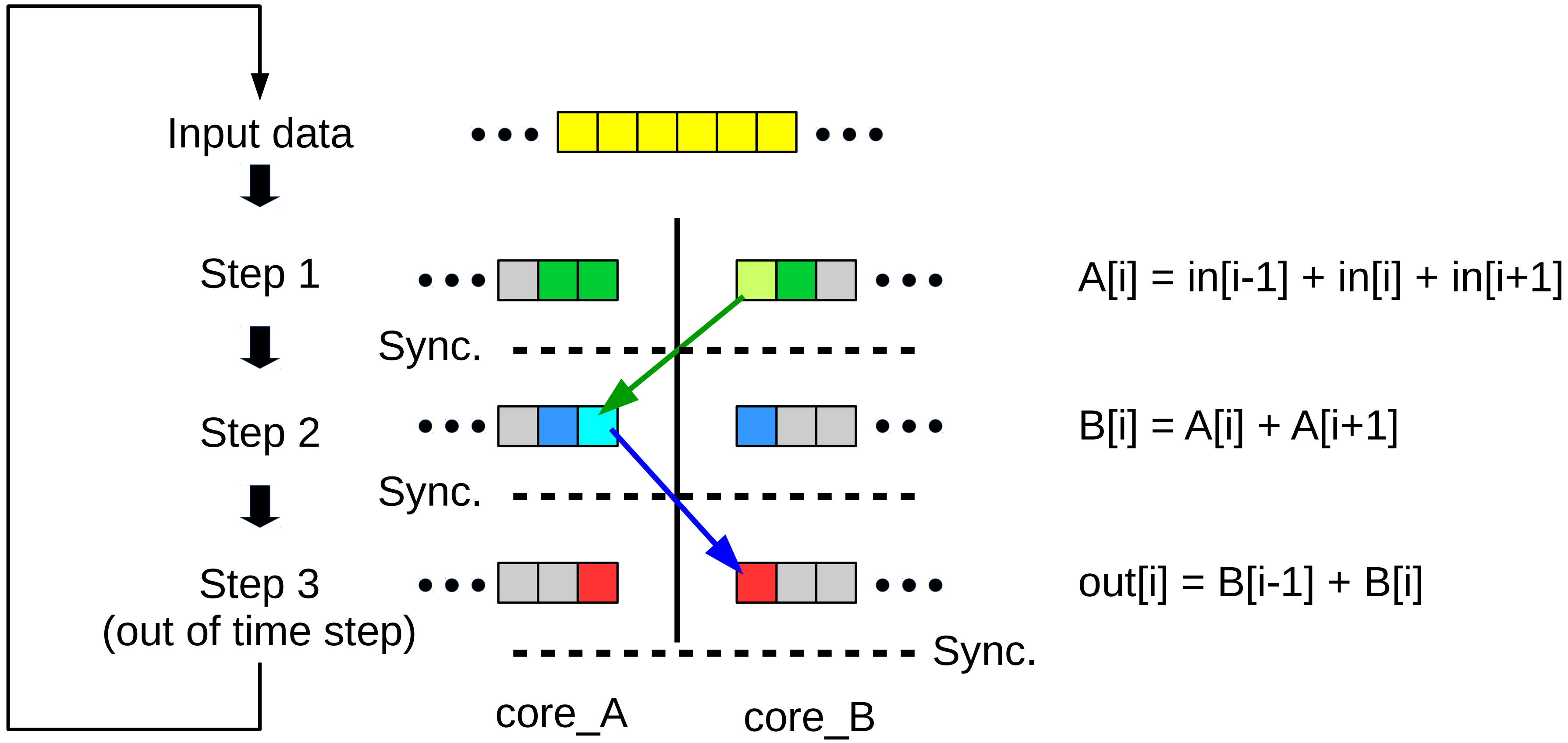
Trade-off between computation and communication for <u>stencils</u>



Trade-off between computation and communication for stencils

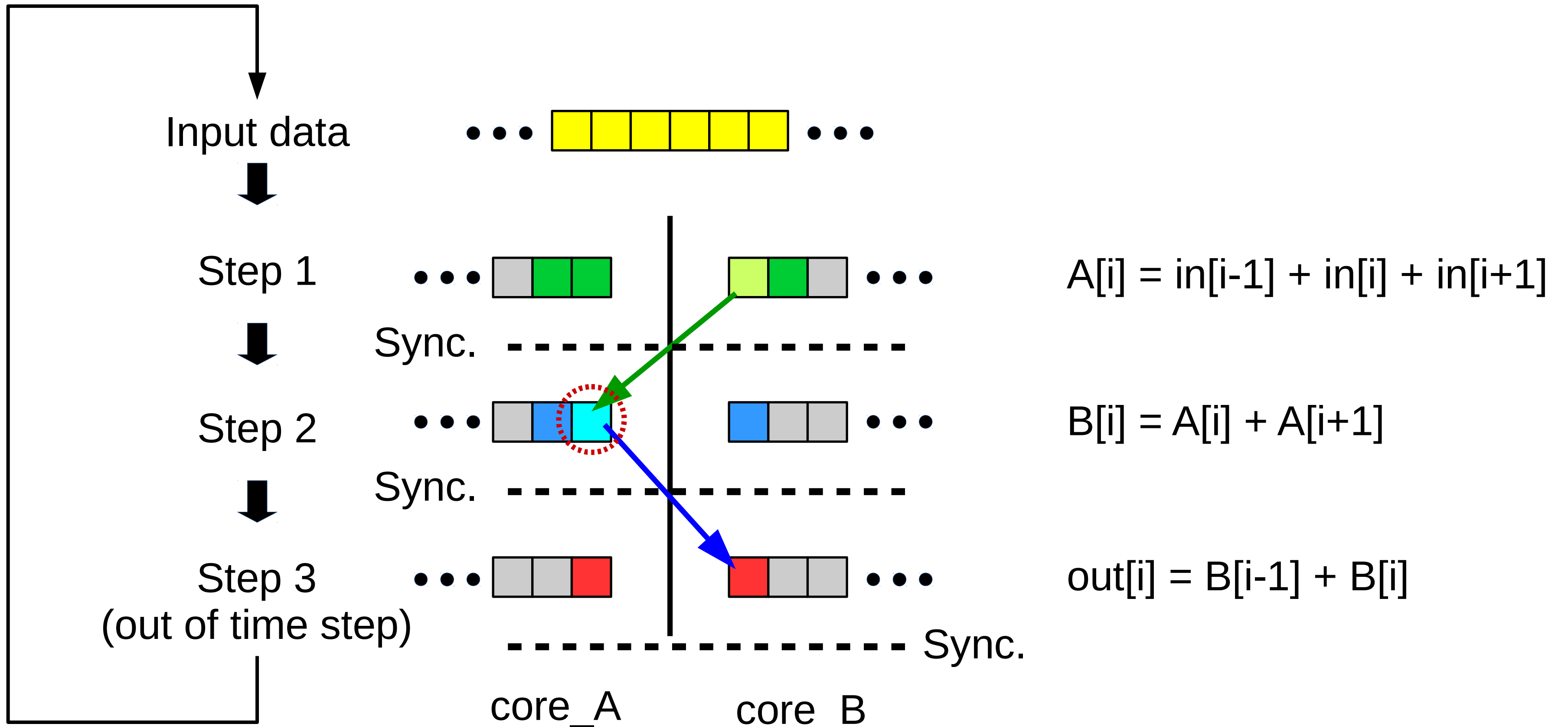


Trade-off between computation and communication for <u>stencils</u>



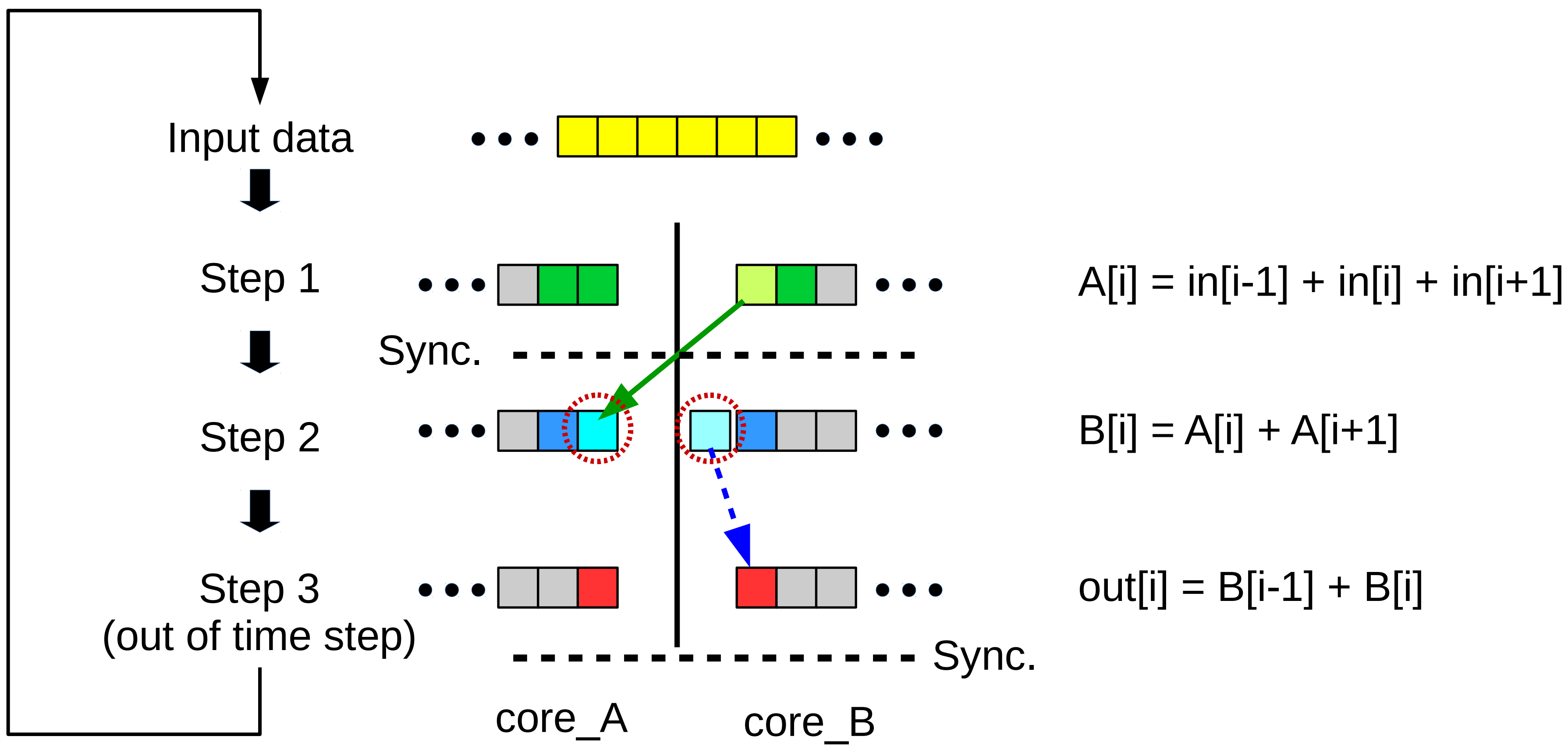
- 3 synchronization points
- 2 paths for implicit transfers of data between cores

Trade-off between computation and communication for <u>stencils</u>



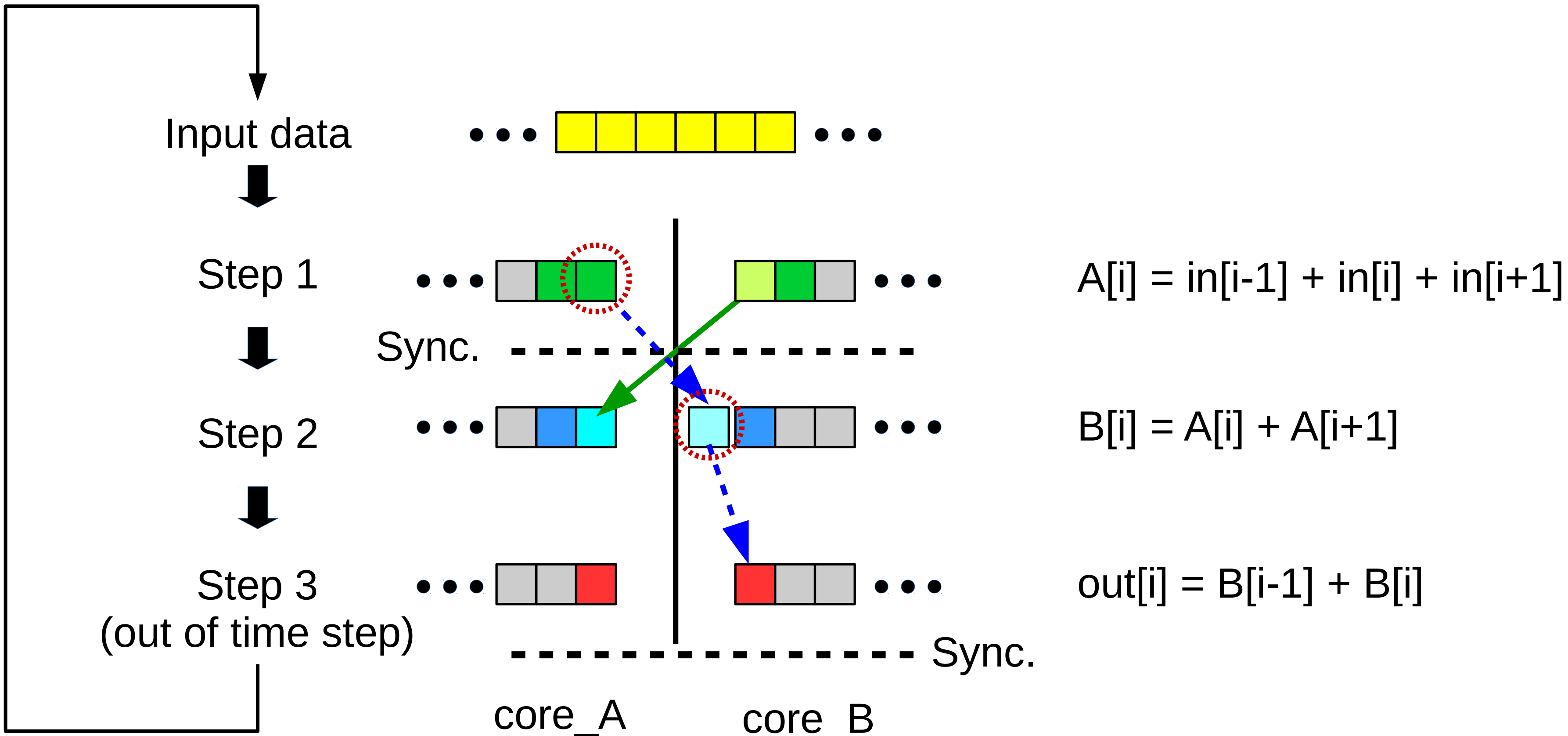
- 3 synchronization points
- 2 paths for implicit transfers of data between cores

Trade-off between computation and communication for <u>stencils</u>



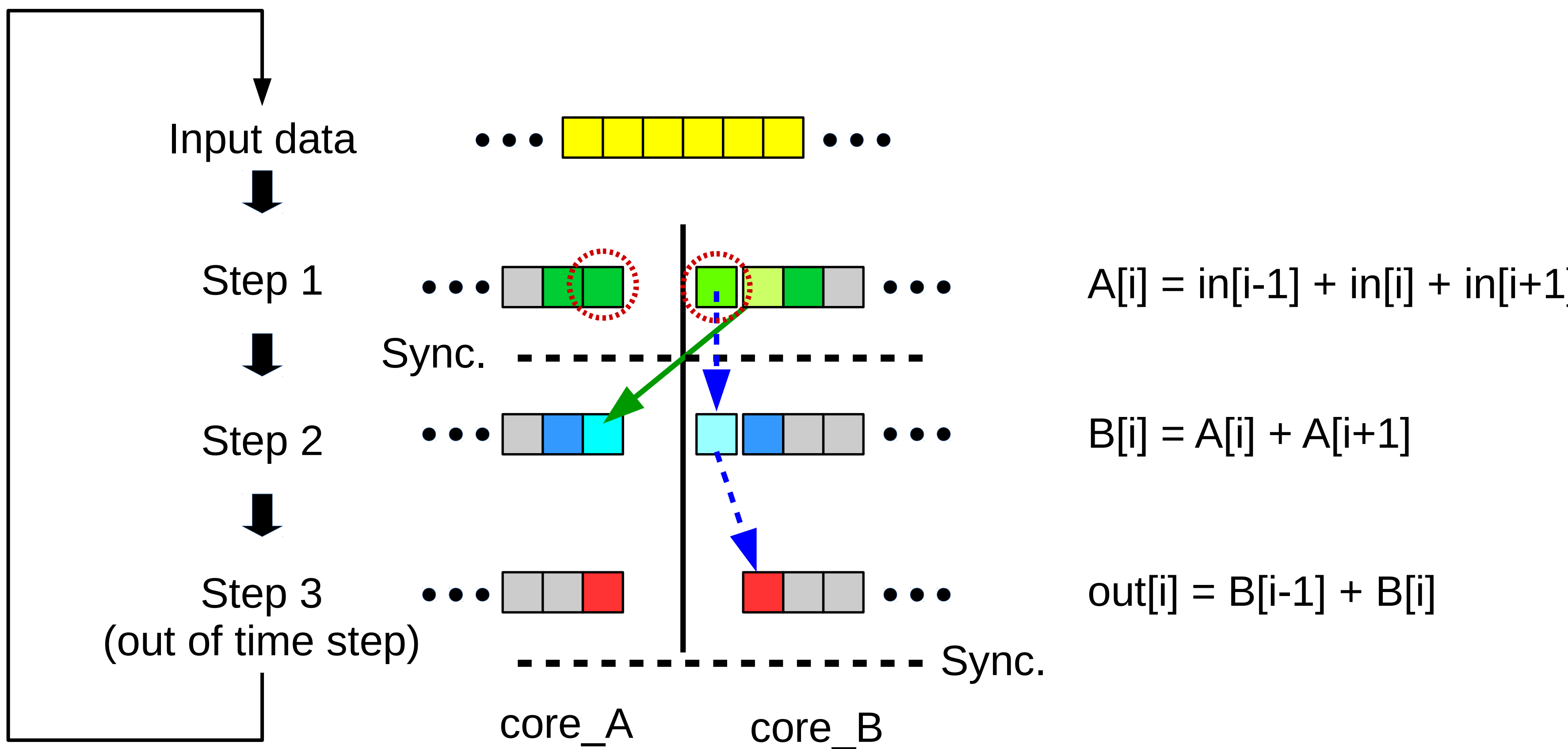
- ~~3~~ 2 synchronization points
- ~~2~~ 1 path for implicit transfers of data between cores
- 1 extra cells necessary for computing

Trade-off between computation and communication for stencils



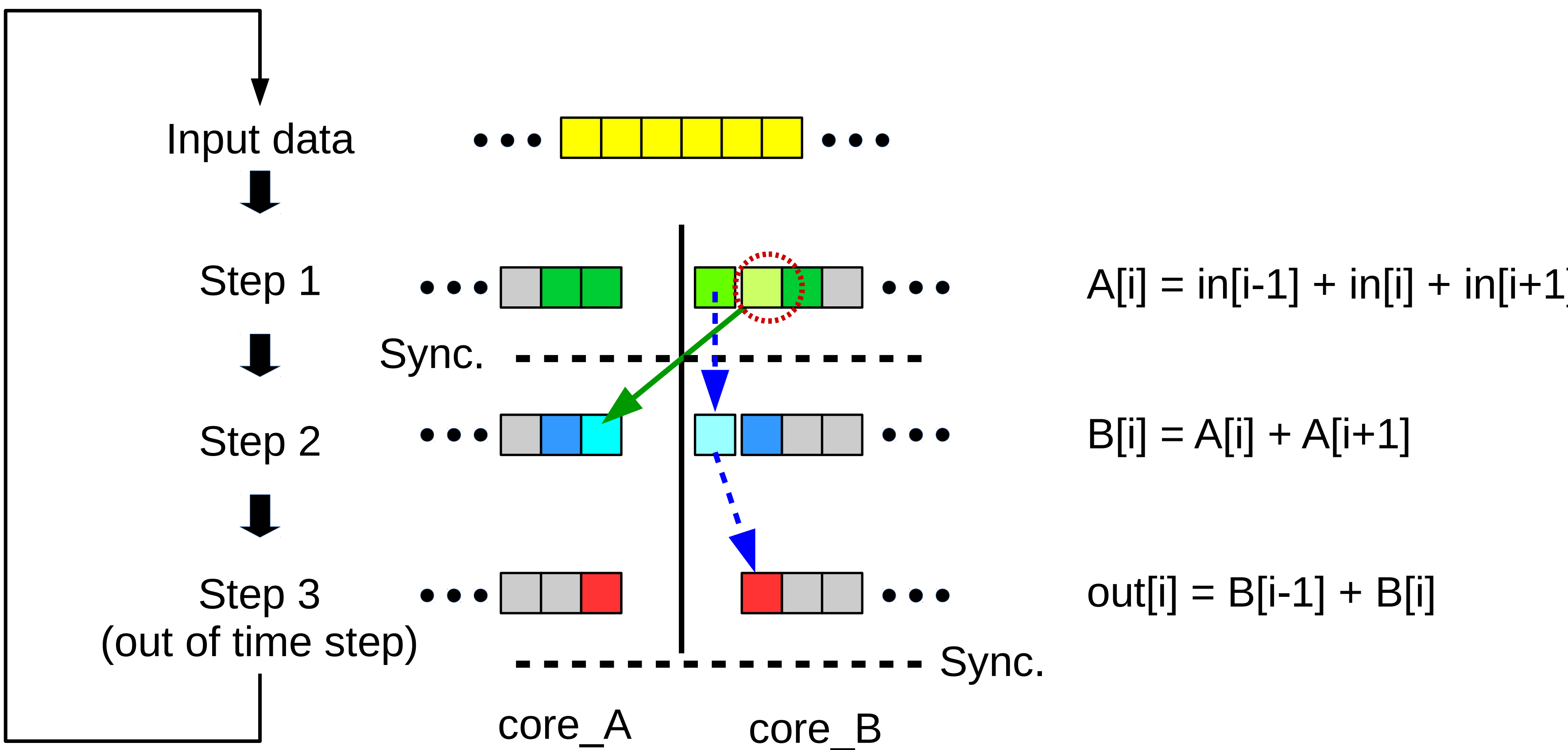
- ~~3~~ 2 synchronization points
- ~~2~~ 1 path for implicit transfers of data between cores
- 1 extra cells necessary for computing

Trade-off between computation and communication for stencils



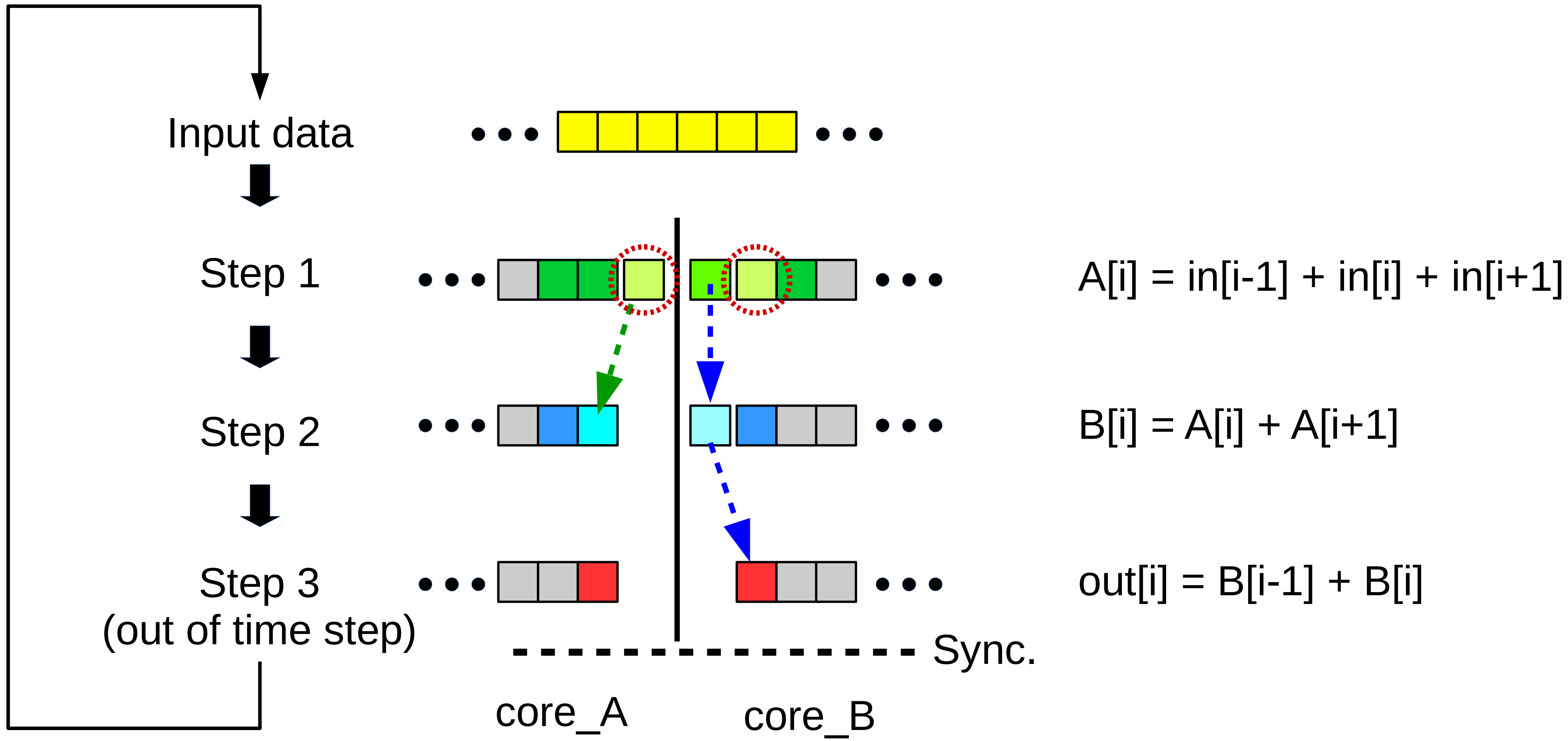
- ~~3~~ 2 synchronization points
- ~~2~~ 1 path for implicit transfers of data between cores
- 2 extra cells necessary for computing

Trade-off between computation and communication for <u>stencils</u>



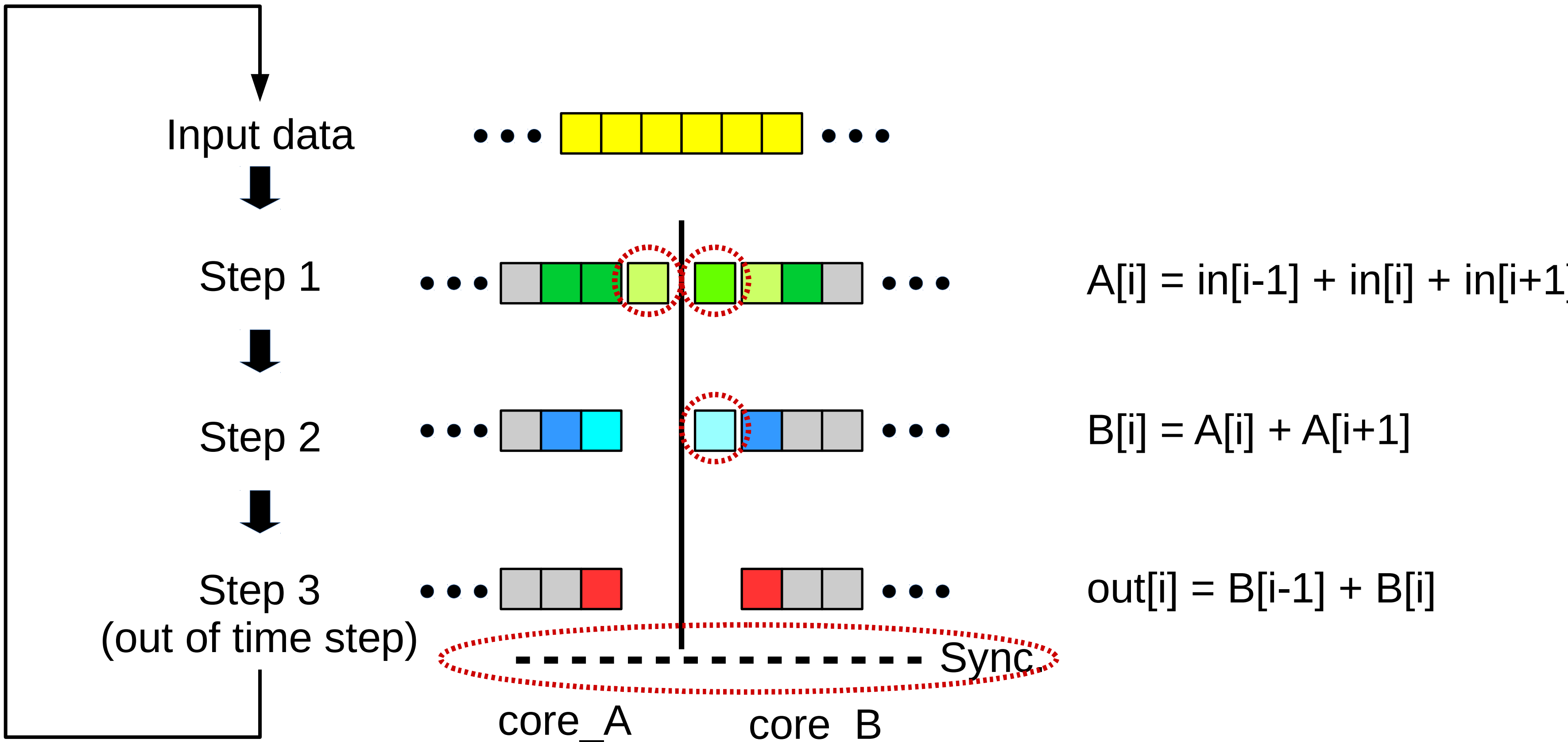
- ~~3~~ 2 synchronization points
- ~~2~~ 1 path for implicit transfers of data between cores
- 2 extra cells necessary for computing

Trade-off between computation and communication for <u>stencils</u>



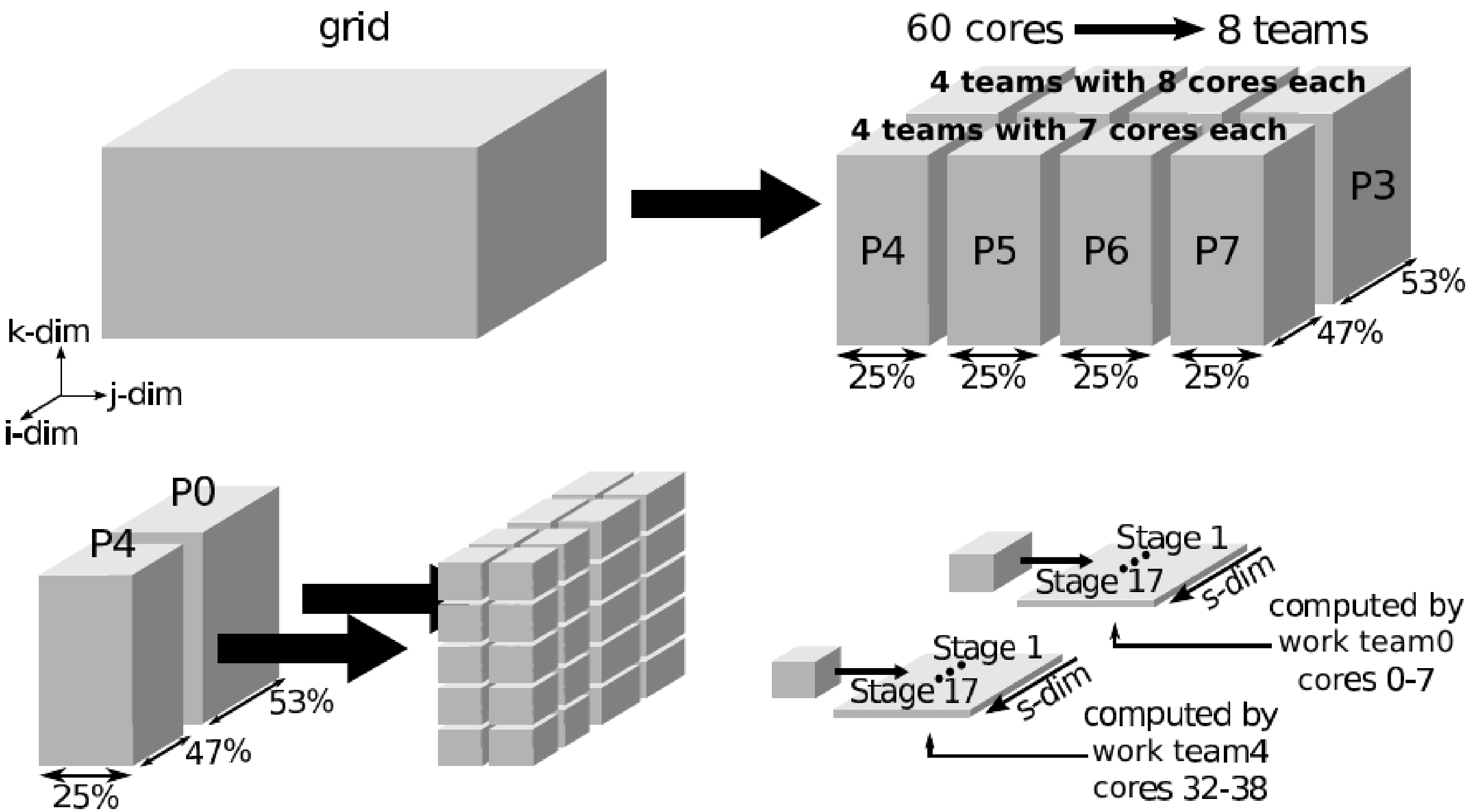
- ~~3~~ 1 synchronization points
- ~~2~~ 0 path for implicit transfers of data between cores
- 3 extra cells necessary for computing

Trade-off between computation and communication for stencils

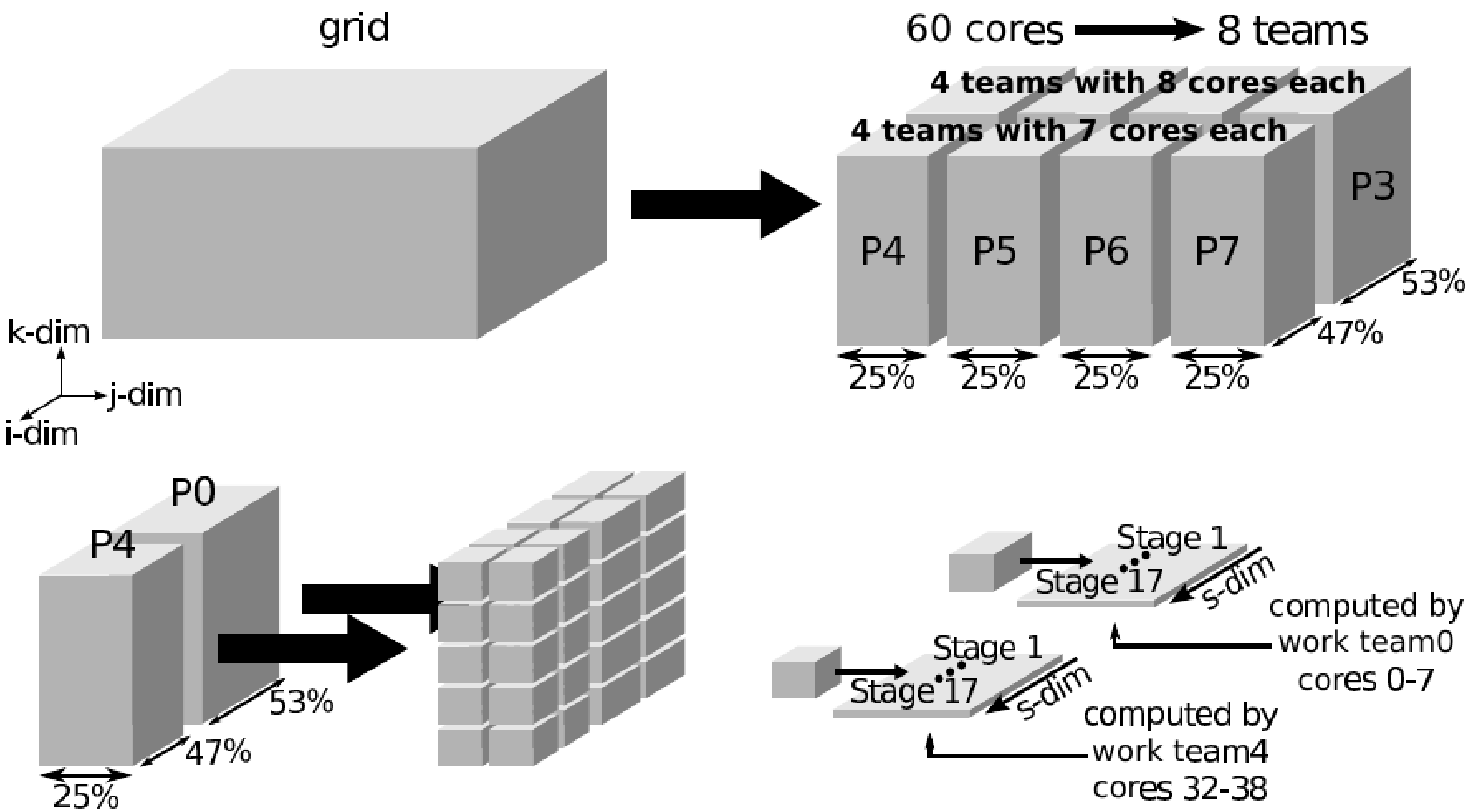


- ~~3~~ 1 synchronization points
- ~~2~~ 0 path for implicit transfers of data between cores
- 3 extra cells necessary for computing

Idea of islands of cores



Idea of islands of cores



- This methodology is also well suitable for **ccNUMA** and **SMP/NUMA** architectures
  - In this case, it allows reducing communication paths between CPUs (islands of CPUs)

Target Platforms

- 2 x Intel Xeon E5-2699v3*:* 2.30GHz, 2x18 cores
- Intel Xeon Phi 7120P (*Knight Corner*): 1.24GHz, 61 cores
- Intel Xeon Phi 7210 (*Knight Landing*): 1.30GHz, 64 cores
- SMP/NUMA SGI UV 2000 server, 14x Intel Xeon E5-4627v2 (*Ivy Bridge EP*): 3.30GHz, 8 cores

- All performance results are obtained for the **double precision**
  - Accuracy of computation plays key role for MPDATA
- Compilation flags for Intel compiler:

  -O3 -xavx/-xMIC-AVX512 -qopt-streaming-stores always

  -**fp-model precise -fp-model source**

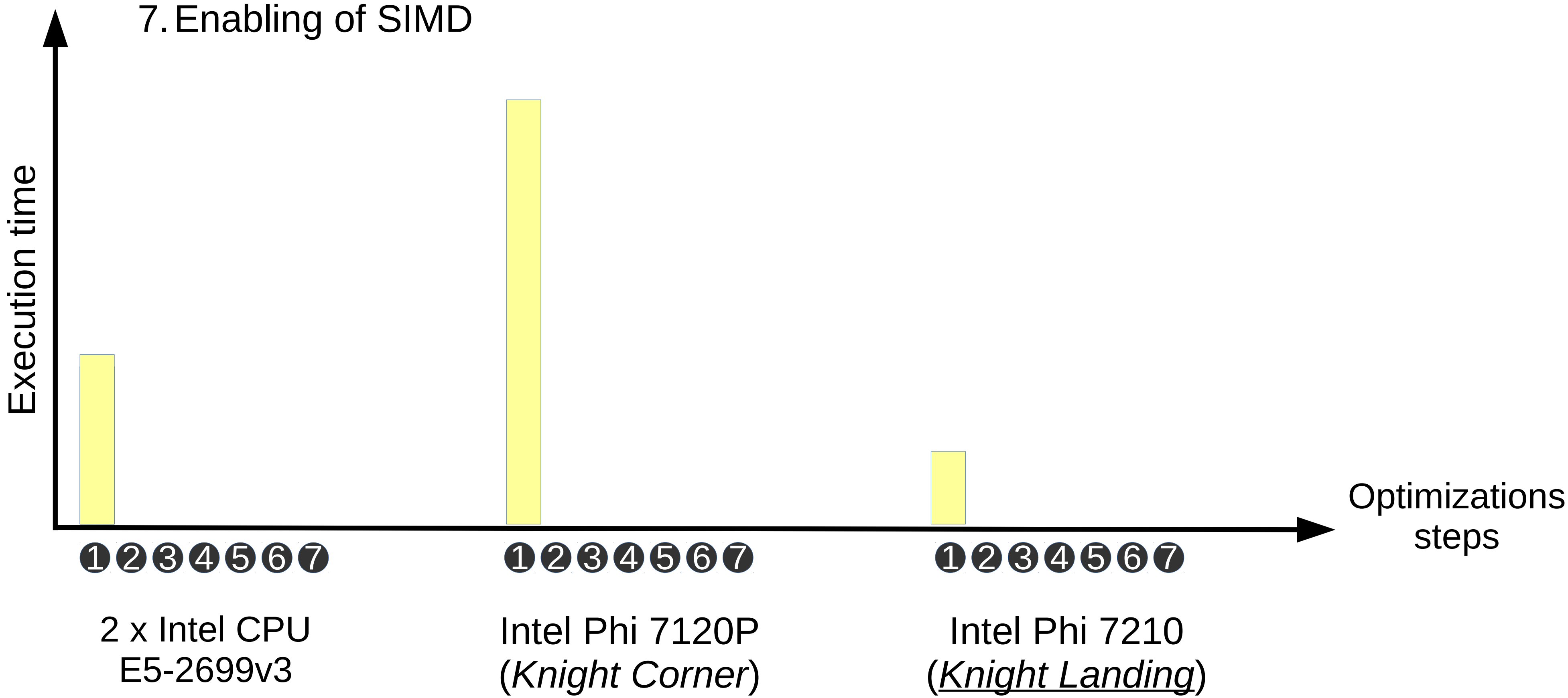
  -no-vec

Performance results

Optimizations steps:

1. **Basic version – start point of our work**
2. Parallel initialization of data
3. (3+1)D Decomposition
4. Islands of cores: workTeams
5. New strategy for synchronization
6. Work groups of threads
7. Enabling of SIMD

Important:
Auto vectorization is enable

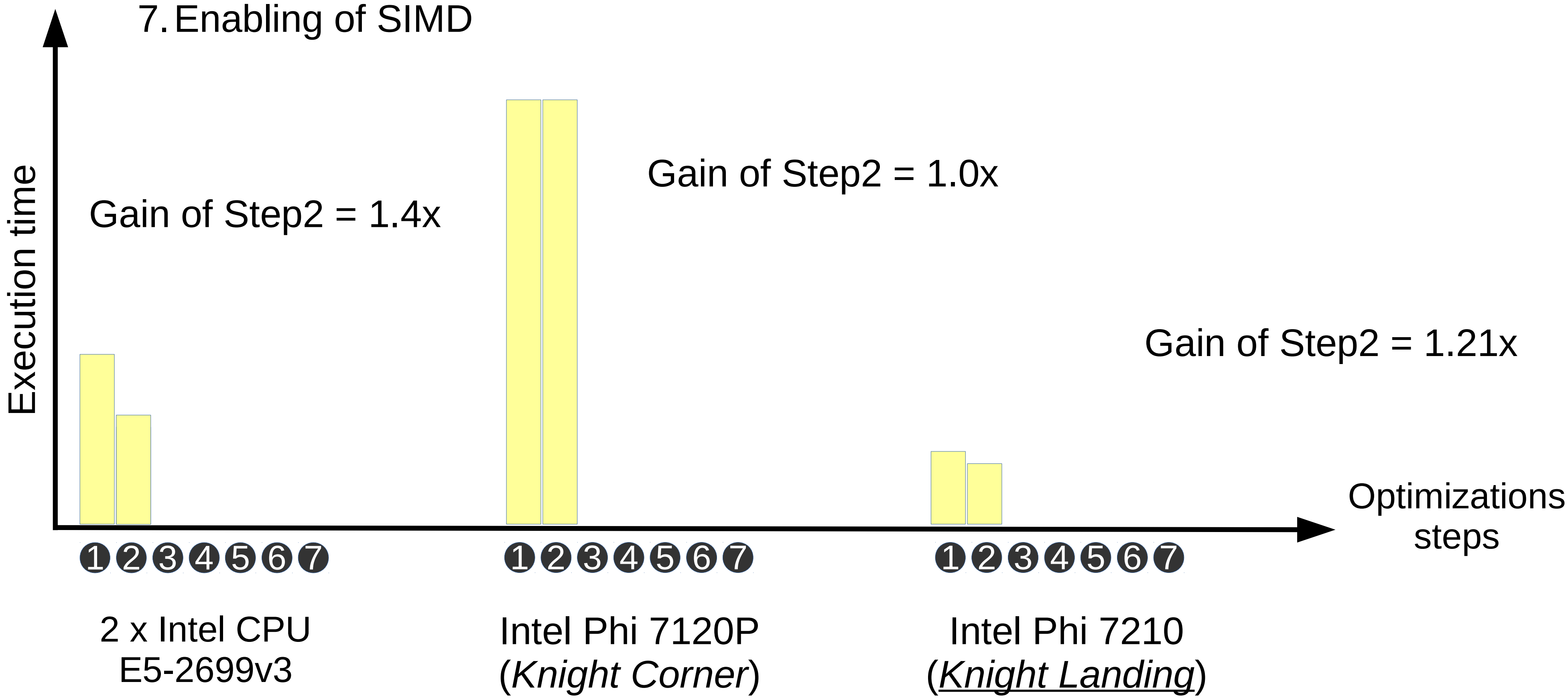


Performance results

Optimizations steps:

1. Basic version – start point of our work
2. **Parallel initialization of data**
3. (3+1)D Decomposition
4. Islands of cores: workTeams
5. New strategy for synchronization
6. Work groups of threads
7. Enabling of SIMD

Important:
Auto vectorization is enable

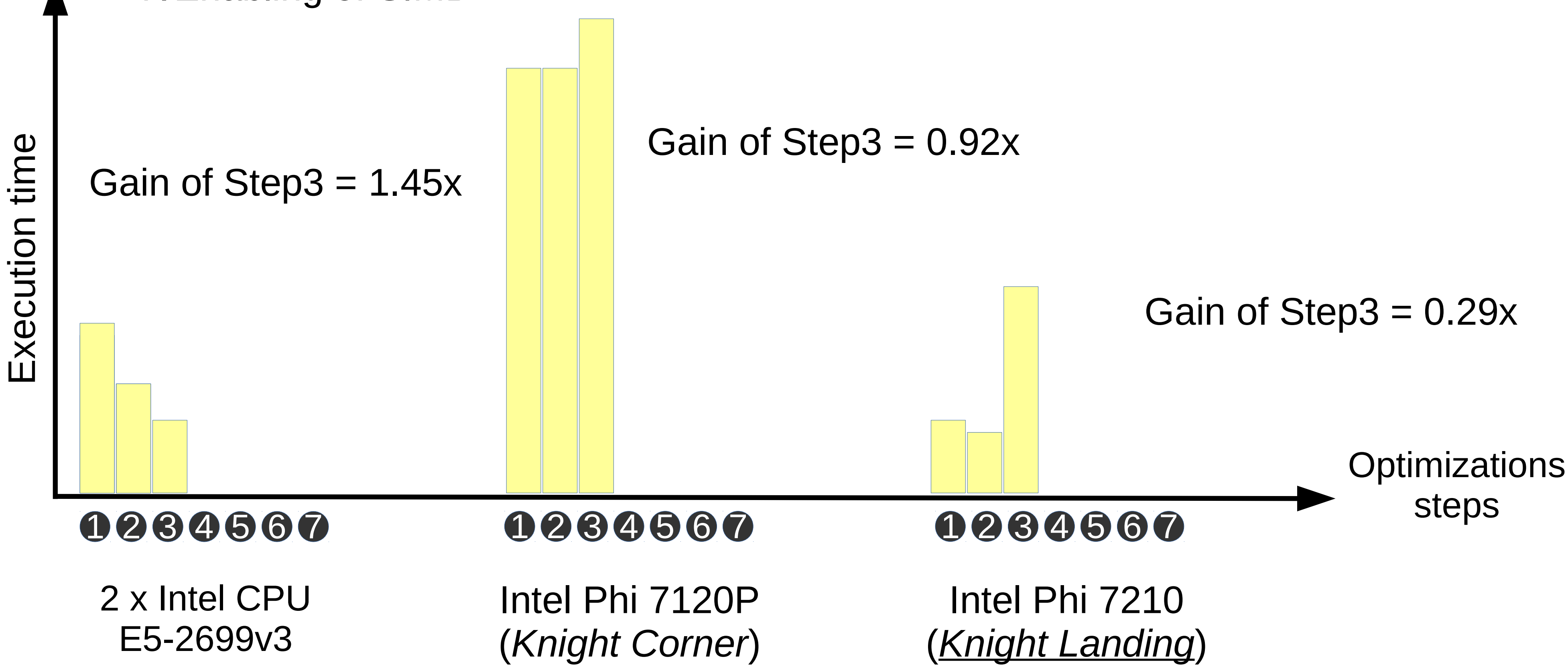


Performance results

Optimizations steps:

1. Basic version – start point of our work
2. Parallel initialization of data
3. **(3+1)D Decomposition**
4. Islands of cores: workTeams
5. New strategy for synchronization
6. Work groups of threads
7. Enabling of SIMD

Important:
Vectorization is disabled

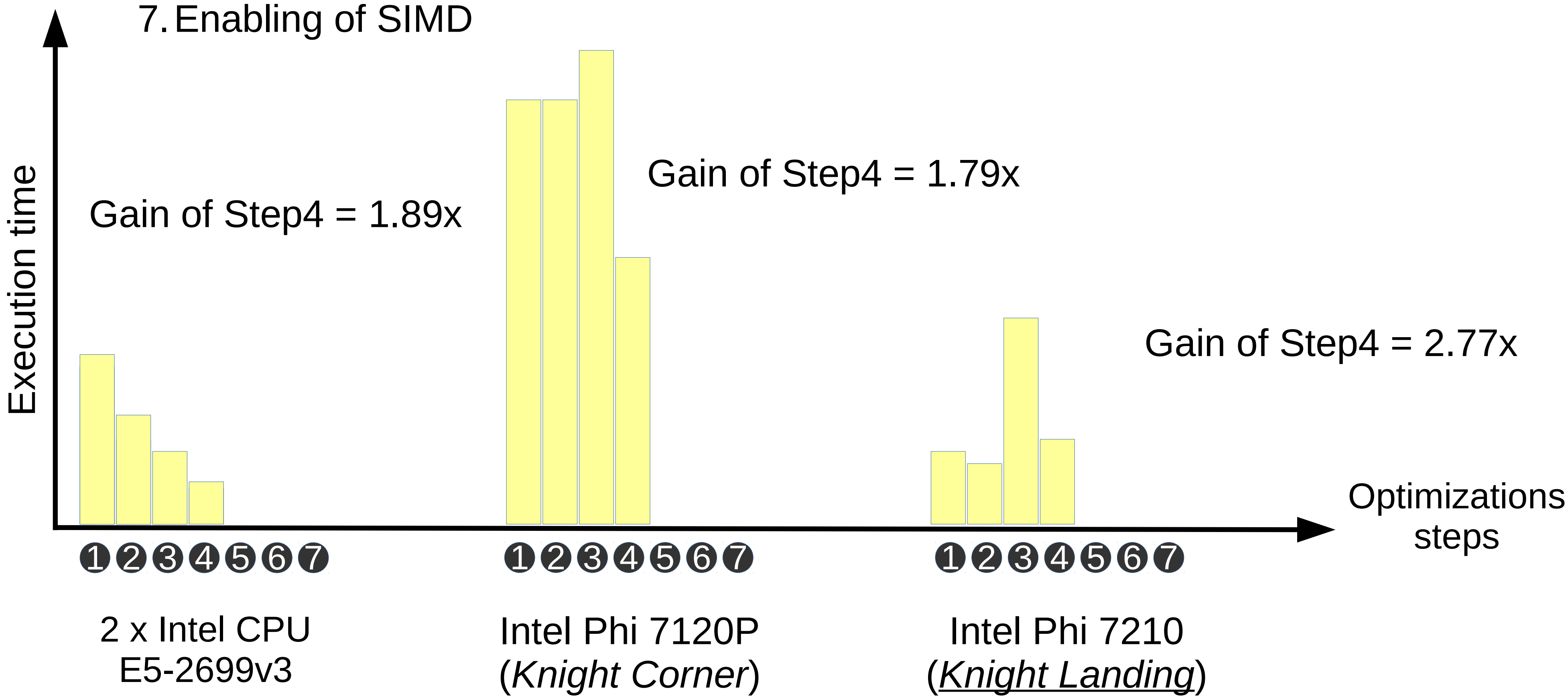


Performance results

Optimizations steps:

1. Basic version – start point of our work
2. Parallel initialization of data
3. (3+1)D Decomposition
4. **Islands of cores: workTeams**
5. New strategy for synchronization
6. Work groups of threads
7. Enabling of SIMD

Important:
Vectorization is disabled

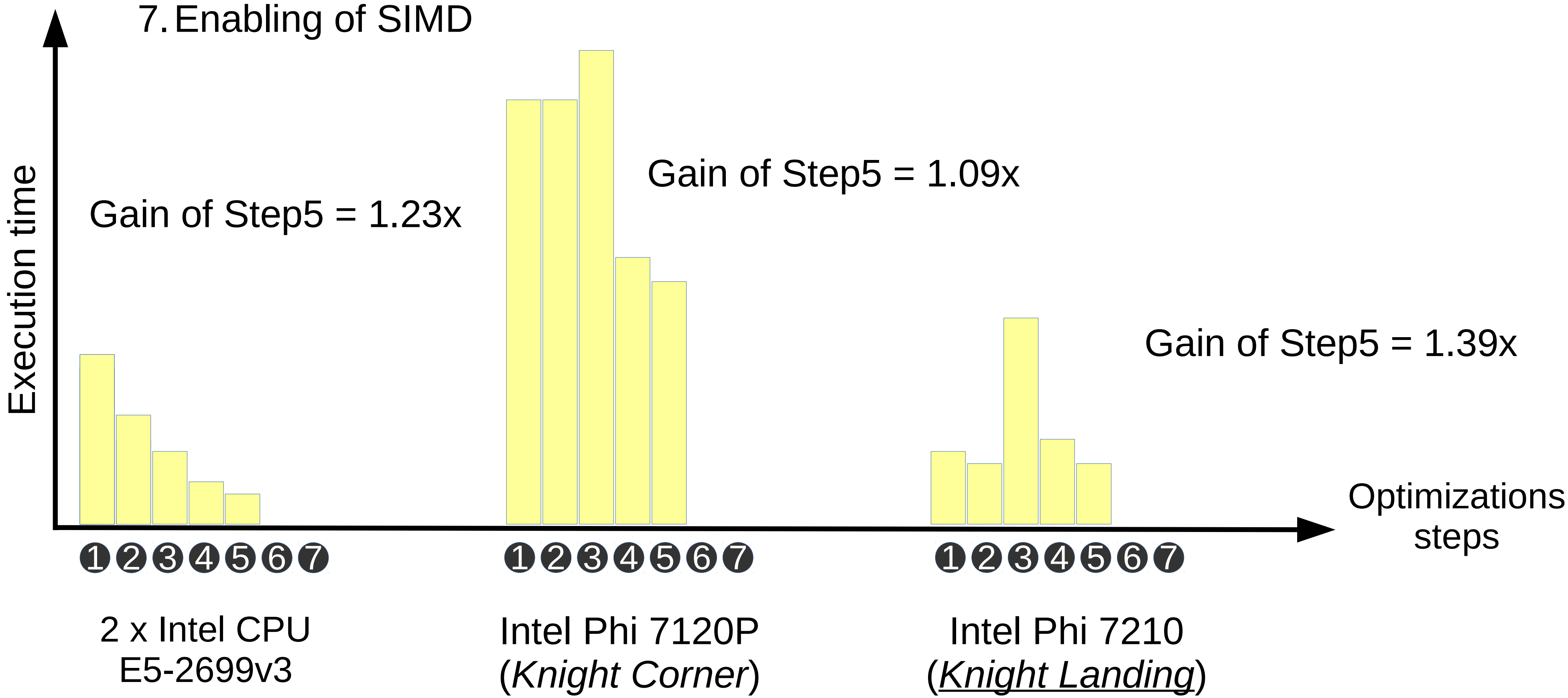


Performance results

Optimizations steps:

1. Basic version – start point of our work
2. Parallel initialization of data
3. (3+1)D Decomposition
4. Islands of cores: workTeams
5. **New strategy for synchronization**
6. Work groups of threads
7. Enabling of SIMD

Important:
Vectorization is disabled

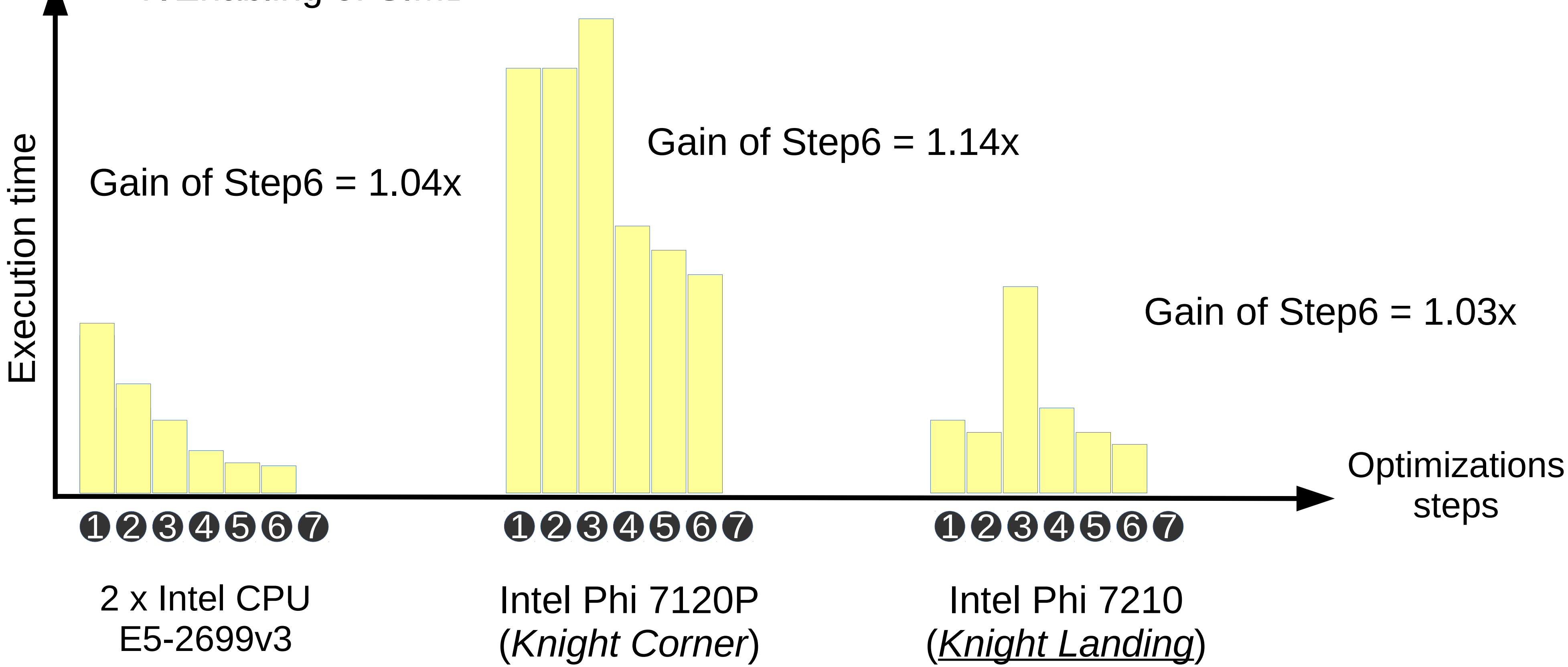


Performance results

Optimizations steps:

1. Basic version – start point of our work
2. Parallel initialization of data
3. (3+1)D Decomposition
4. Islands of cores: workTeams
5. New strategy for synchronization
6. **Work groups of threads**
7. Enabling of SIMD

Important:
Vectorization is disabled

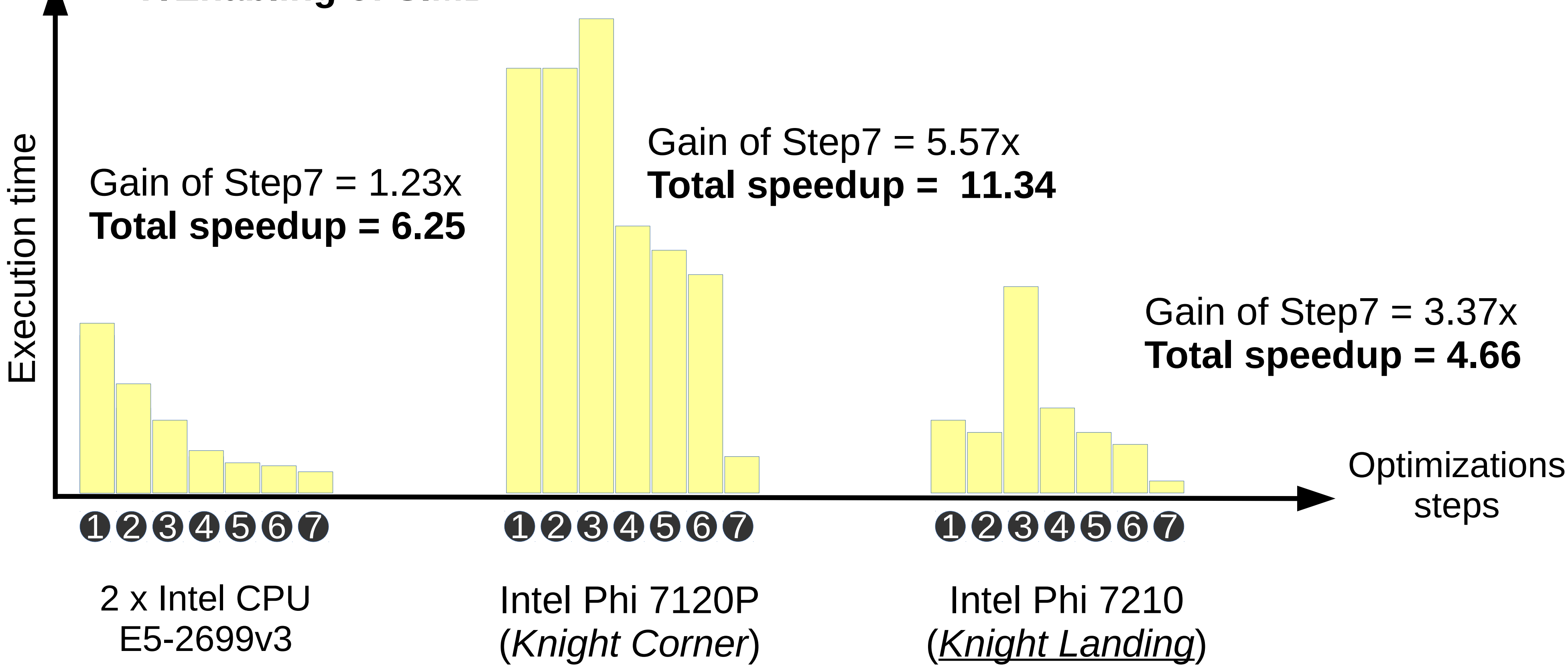


Performance results

Optimizations steps:

1. Basic version – start point of our work
2. Parallel initialization of data
3. (3+1)D Decomposition
4. Islands of cores: workTeams
5. New strategy for synchronization
6. Work groups of threads
7. **Enabling of SIMD**

Important:
Vectorization is enable



Performance results SMP/NUMA SGI UV 2000 server 14x Intel Xeon E5-4627v2

- Execution times [seconds] for the double precision, 500 time steps, and the grid of size 1024×512×64

#CPUs	#cores	Basic version	New version (1 island)	New version (x islands)	Strong scaling	Speedup against the best basic
1	**8**	**300**	89.9	85.3	100%	3
2	**16**	445	82.0	43.3	98%	6
4	**32**	688	79.8	22.4	95%	13
8	**64**	737	76.9	11.7	91%	25
10	**80**	777	145.0	9.4	91%	32
14	**112**	822	155.0	**7.4**	82%	40

Comparison of CPU, MIC, and GPU

- Performance results are obtained for the double precision, 500 time steps, and the grid of size 1024×512×64

Devices	Time [s]
14 x Intel Xeon E5-4627v2 SMP/NUMA	**7**
2 x NVIDIA Tesla K80 (Kepler)	10
NVIDIA Tesla P100 (Pascal)	12
Intel Xeon Phi 7210 (KNL)	**17**
NVIDIA Tesla K80 (Kepler)	19
2 x Intel Xeon CPU E5-2699 v3 (Haswell)	**28**
2 x Intel Xeon CPU E5-2695 v2 (Ivy Bridge)	**34**
Intel Xeon Phi 7120P (KNC)	**36**
NVIDIA Tesla K20 X (Kepler)	38

Numerical model of solidifaction

<u>Łukasz Szustak</u>
lszustak@icis.pcz.p

Roman Wyrzykowski
roman@icis.pcz.pl

Kamil Halbiniak
khalbiniak@icis.pcz.pl

Motivation of our research

- The main goal of our current research is the suitability assessment of three heretogeneous programming models in practice:
  - Intel Offload Interface,
  - OpenMP 4.0 Accelerator Model,
  - Hetero Streams Library.
- We focus on porting an application which implements the numerical model of alloy solidification to hybrid CPU-MIC platforms
- The prime assumption is developing new parallel version without <u>significant modifications of the application code</u>

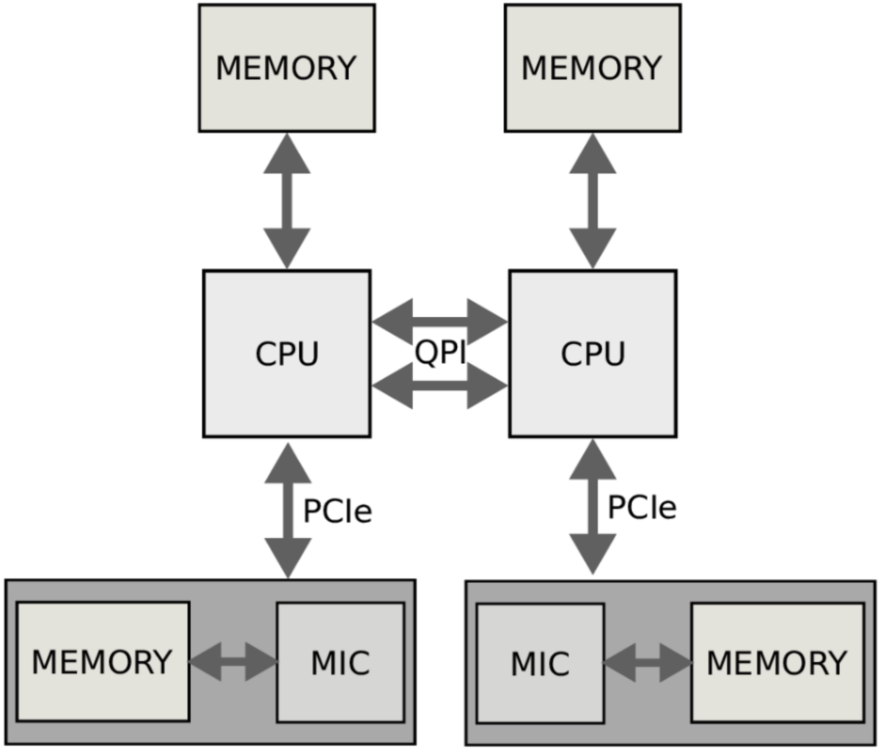
Motivation of our research

- OpenMP Accelerator Model
  - It is available from version 4.0
  - It assumes that a computing platform is equipped with multiple target devices connected to the host
  - This extension provides an unified directive-based programming model
- Hetero Streams Library
  - This is open source project supported by Intel
  - hStreams allows for stream programming in heterogeneous environments
  - It is based on the heterogeneous asynchronous multitasking model
  - It uses two key definitions:
    - **source** - place where work is enqueued (only one source)
    - **sink -** place where jobs defined as streams are executed (multiple sinks)
  - Stream is FIFO queues which resides in selected device or part of it

Target platforms

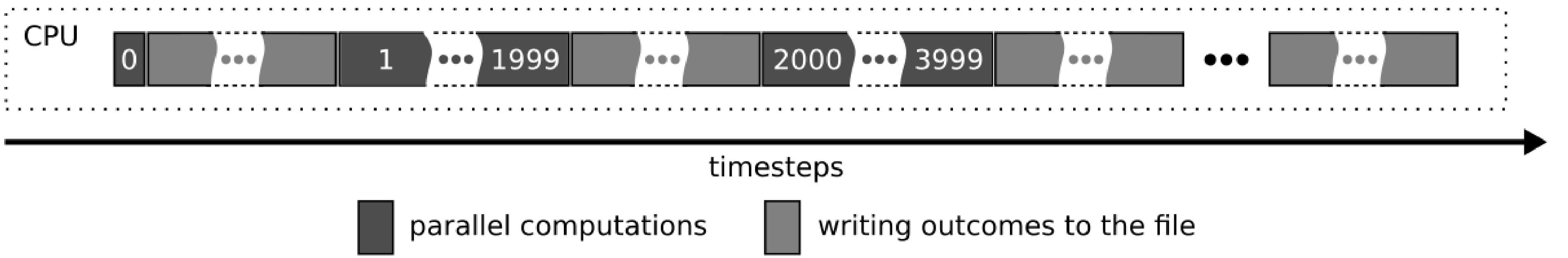
- In our study, we use the following CPU-MIC platform:

	Intel Xeon E5-2699 v3	Intel Xeon Phi 7120P
Number of components	**2**	**2**
Number of cores	**18**	**61**
Number of threads	**36**	**244**
Frequency [GHz]	**2.3**	**1.2**
SIMD [bit]	**256**	**512**
LLC [MB]	**45**	**30.5**
Main memory [GB]	**256**	**16**
Peak performance for DP [Gflop/s]	**662.4**	**1208.3**



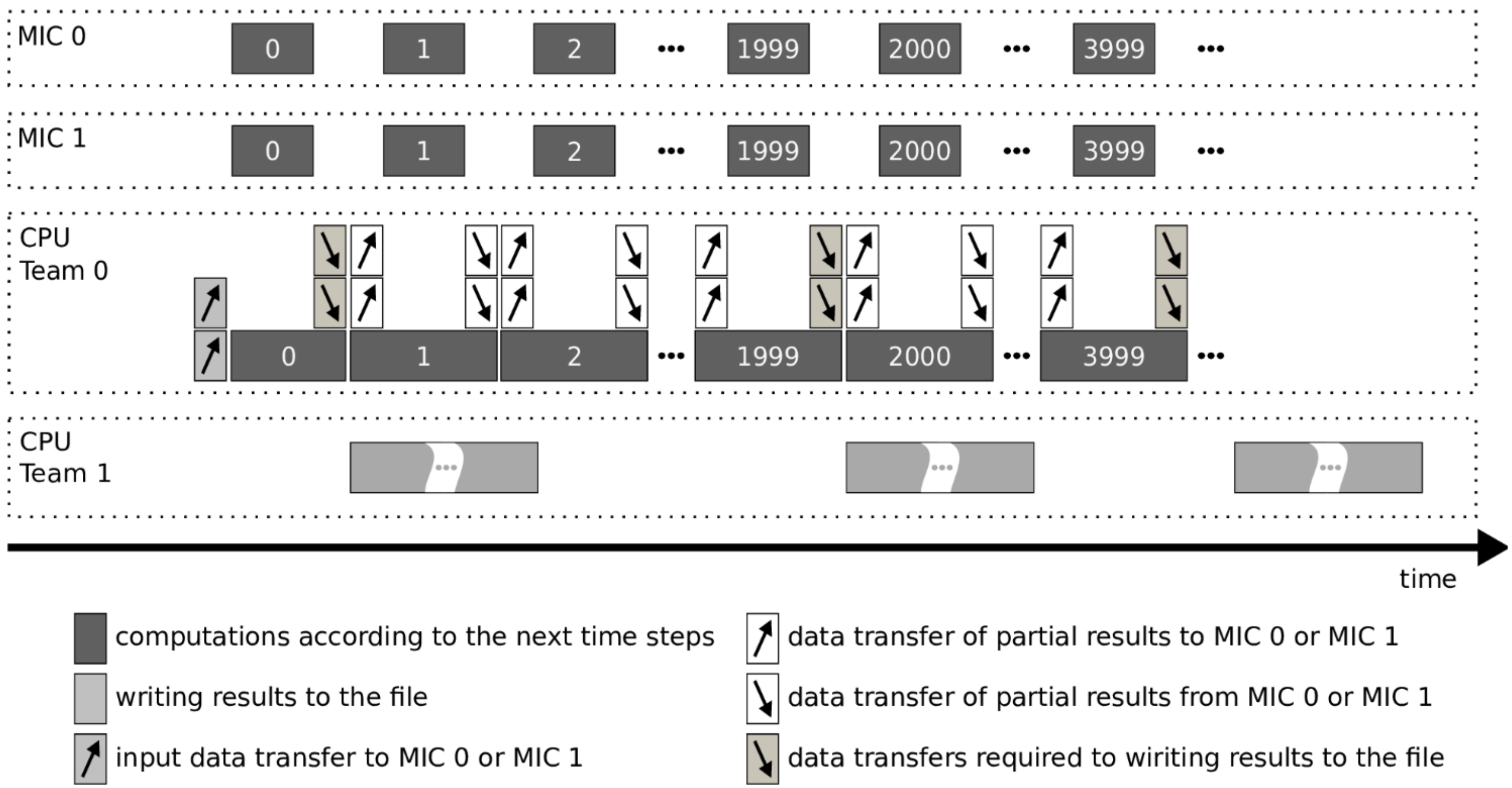
Idea of adaptation

- Original version of the solidifcation application:



Porting application with Intel Offload

- New version of the application:

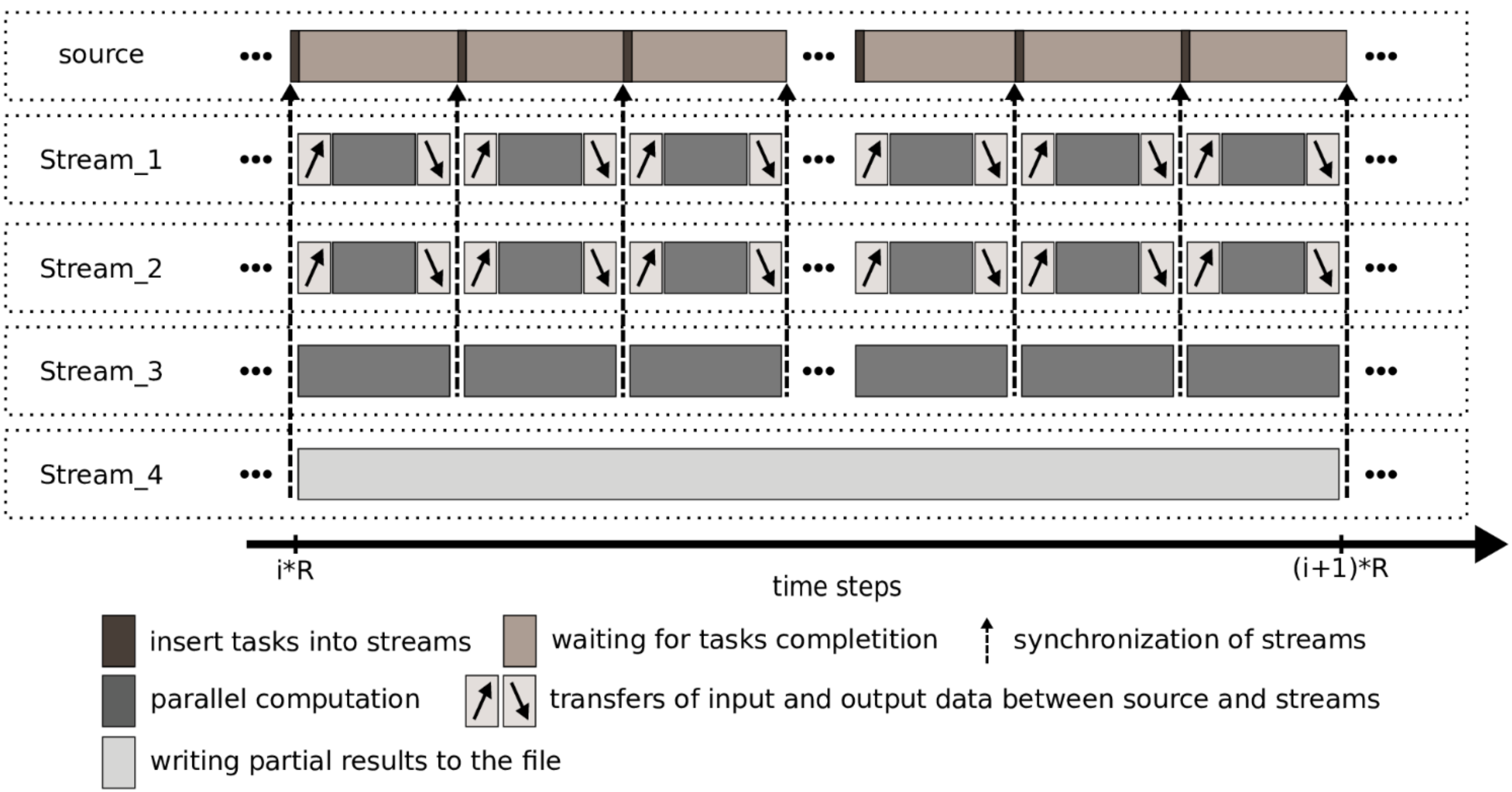


Porting application with Intel Offload

- Porting the solification application to hybrid CPU-MIC platform requires:
  - Employing all available resources of CPUs and MICs to joint problem solving
  - Providing appropriate workload distribution between processors and two MICs, with partitioning CPU threads into two work teams
  - Applying adequate data partitioning between devices
    - Arrays used in computations are partitioned into three parts
    - We assign the first part of computations to MIC0, the second one to CPUs, and the last part to MIC1
    - It allows us to decrease communication paths
  - Optimization of data movements
    - Reduce the total size of transferred data (only the necessary data are transferred)
    - Reduce the number of memory allocations (data are allocated only once at the beginning of computations)
  - Provide a double buffering technique to overlap computations and writing data to file
  - Utilization of vector processing units of coprocessors as well as CPUs

Mapping application workload onto hStreams

- Idea of mapping application workload onto Hetero Streams

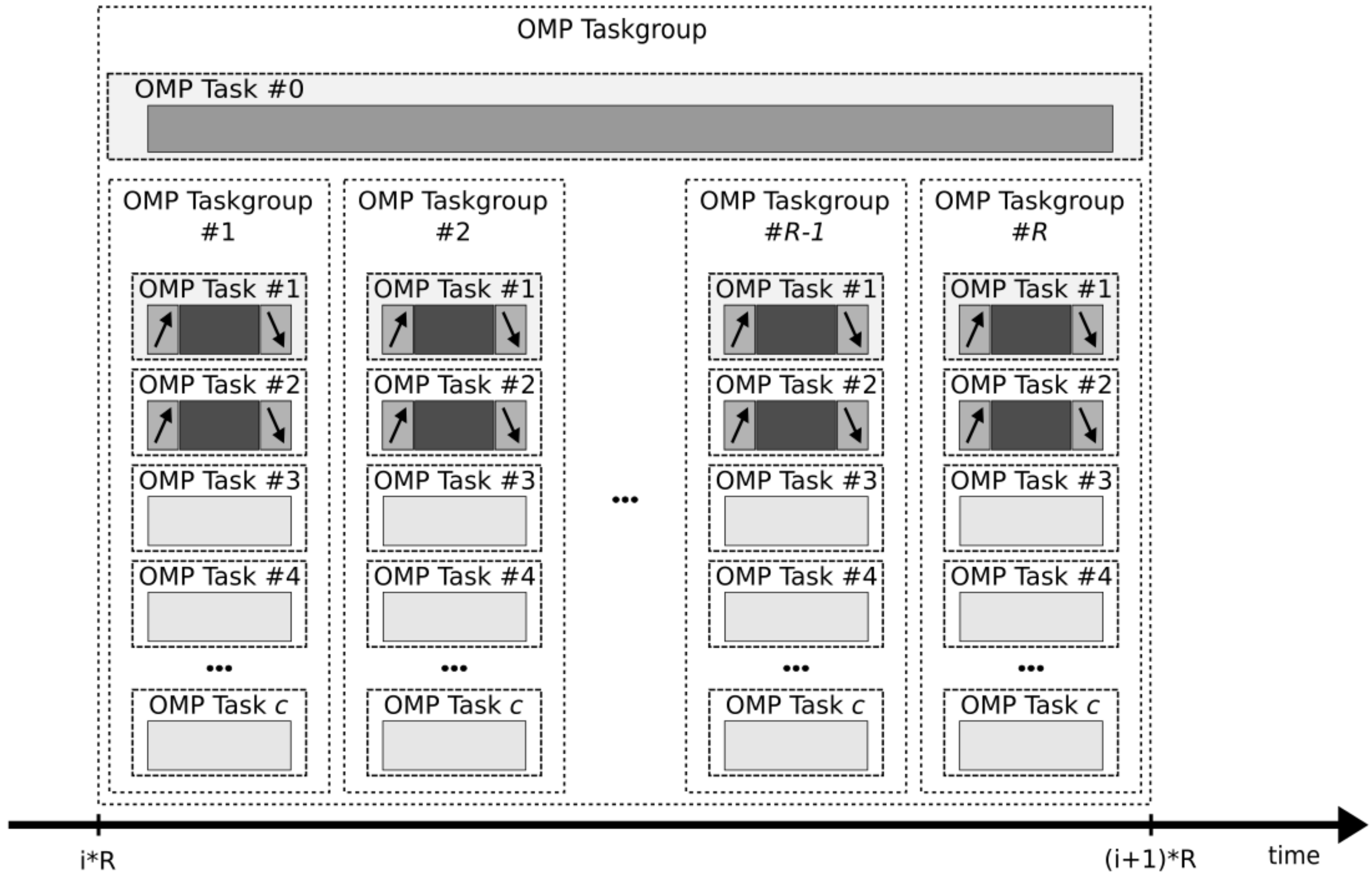


Mapping application workload onto hStreams

Idea of mapping application workload onto Hetero Streams:

- Creation of four logical streams in three logical domains:
  - Three streams used for parallel computations of packages with 2000 time steps (MICs + CPUs)
  - One stream responsible for writing partial result to the file (CPU)
- Use of hStreams Core API to manually initialize the streams on CPU responsible for parallel computations and data writing, respectively
- Applying appropriate methods for stream synchronization:
  - An active synchronization when source process is stopped during execution of streams
  - The proposed aproach requires applying two scenerio of synchronization
  - The first one used for synchronization streams responsible for parallel computations
  - The second scenerio used to synchronize all the streams after completion both computations within packages and data writing to the file
- Utilization of aliased buffers mechanism:
  - Source process and CPU streams share the same memory regions
  - It gives us a strong possibility for optimizing data exchanges
- Applying double-buffering technique

Porting application with OpenMP 4.0 Accelerator Model



OMP Task #0 - writing partial results to the file

OMP Task #1 and #2 - asynchronous offloading data and computations between CPU to coprocessors

OMP Task #3 ... OMP Task *c* - computations performed by CPU threads

Porting application with OpenMP 4.0 Accelerator Model

Idea of porting application with OpenMP 4.0 Accelerator Model:

- Utilize two major construct which allows offloading computation and data:
  - target data construct which create data device enviroment
  - target construct used to trasnfers computations to the device
  - target update construct to transfer data from host to coprocessors and vice versa
- Applying OpenMP task parallelism for:
  - Implementation asynchronous offloads to all coprocessors
  - Parallel computations performed by CPUs
  - Writing partial results to the file
- Select an appropriate method for task synchronization:
  - To ensure adequate and efficient synchronization, taskgroup construct is used
  - Proposed aproach requires applying two scenerio of synchronization
  - The first one used for synchronization only tasks responsible for management of processors and parallel computations performed by the CPU
  - The second scenerio used to synchronization of all the tasks after completion both computations for package of 2000 time steps and data writing to the file
- Utilize double-buffering technique

Performance comparison

- All performance results are obtained for double-precision format
- The tests are performed for 110 000 time steps and grid containing 4 000 000 nodes
- Numerical accuracy of simulation results is verified experimentally

Version	Computing resources	Time	Speedup
Basic	**2 x CPU**	**645 min 43 sec**	**-**
hStreams-based	**1 x MIC**	**182 min 13 sec**	**3.54x**
OpenMP-based	**1 x MIC**	**189 min 55 sec**	**3.40x**
offload-based	**1 x MIC**	**180 min 56 sec**	**3.56x**
hStreams-based			
OpenMP-based			
offload-based			
hStreams-based			
OpenMP-based			
offload-based			

Performance comparison

- All performance results are obtained for double-precision format
- The tests are performed for 110 000 time steps and grid containing 4 000 000 nodes
- Numerical accuracy of simulation results is verified experimentally

Version	Computing resources	Time	Speedup
Basic	2 x CPU	645 min 43 sec	-
hStreams-based	1 x MIC	182 min 13 sec	3.54x
OpenMP-based	1 x MIC	189 min 55 sec	3.40x
offload-based	1 x MIC	180 min 56 sec	3.56x
hStreams-based	**2 x MIC**	**97 min 31 sec**	**6.62x**
OpenMP-based	**2 x MIC**	**104 min 51 sec**	**6.16x**
offload-based	**2 x MIC**	**91 min 17 sec**	**7.07x**
hStreams-based			
OpenMP-based			
offload-based			

Performance comparison

- All performance results are obtained for double-precision format
- The tests are performed for 110 000 time steps and grid containing 4 000 000 nodes
- Numerical accuracy of simulation results is verified experimentally

Version	Computing resources	Time	Speedup
Basic	2 x CPU	645 min 43 sec	-
hStreams-based	1 x MIC	182 min 13 sec	3.54x
OpenMP-based	1 x MIC	189 min 55 sec	3.40x
offload-based	1 x MIC	180 min 56 sec	3.56x
hStreams-based	2 x MIC	97 min 31 sec	6.62x
OpenMP-based	2 x MIC	104 min 51 sec	6.16x
offload-based	2 x MIC	91 min 17 sec	7.07x
hStreams-based	**2 x CPU + 2 x MIC**	**61 min 35 sec**	**10.48x**
OpenMP-based	**2 x CPU + 2 x MIC**	**Too slow !**	**Too slow !**
offload-based	**2 x CPU + 2 x MIC**	**59 min 23 sec**	**10.87x**

Conclusions: part I (MPDATA!)

- Our work shows that it is a challenging task to fully utilize emerging computing system for real-life scientific applications which like the MPDATA algorithm are memory bound, and have a complex structure of data dependencies
- Novel HPC platforms require often to develop a new programming abstraction for better utilization of computing resources, including <u>threads/cores and Vector Processing Unit</u>
  - The data locality plays a key role to reach this goal
- With the quick development of computing platforms and software environments, application developers are forced to contend with a variety of parallel architecture
  - In consequence, providing the performance portability for parallel codes becomes the key issue for creators of applications

Conclusions: part II

- Using of hStreams and Intel Offload programming models give us the performance gain on desired level of efficiency
  - While OpenMP provides an unified directive-based programming model, the current stable version of this standard is not efficient to fully utilize all hybrid component
- The proposed method of adapting the solidification application allows hiding more than 99% of transfers behind computations
- hStreams addresses key programming productivity issues by allowing a separation of concerns between:
  - the expression of functional semantics and disclosure of task parallelism (important for creators of scientific algorithms who are generally not computer scientists)
  - the performance tuning and control over mapping tasks onto a platform (responsibility of code tuners and runtime developers)

Thank You