Renato Vacondio
University of Parma

A multi-GPU implementation of a 2D shallow water equations solver with variable resolution

Flooding Events in Europe (2017)

Spain, March

Russia, May

UK, September

Cortina, August

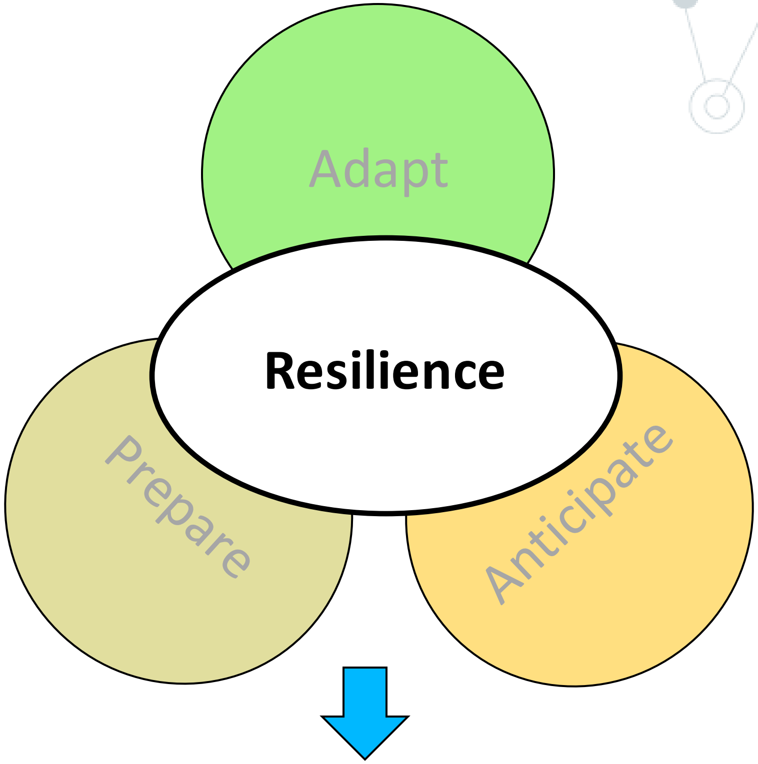
Livorno, September

Reggio E., December

How can we fix it?

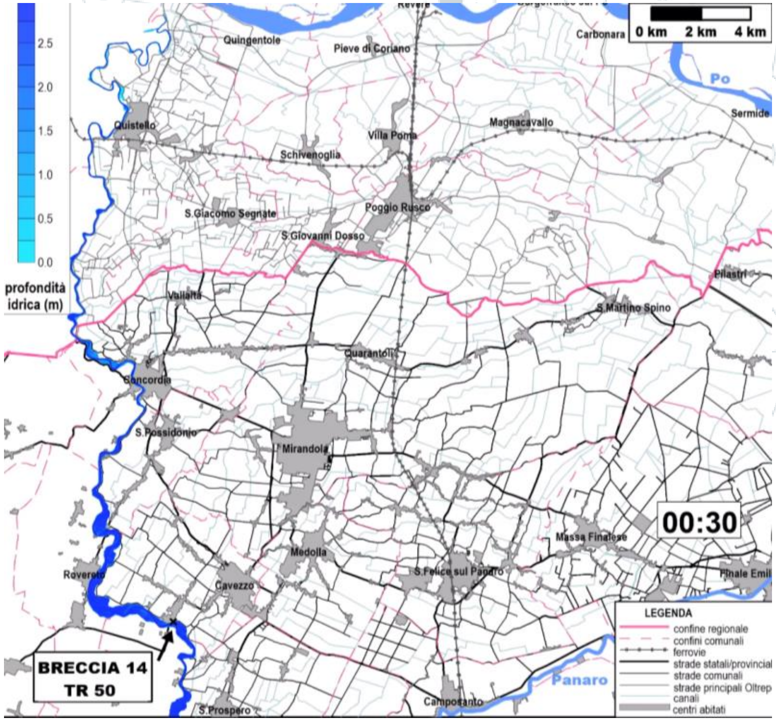
Structural defense systems?

Low Flexibility



Require fast and accurate modeling

Parflood



- Finite volume solver
- Solves 2D Shallow water equations to simulate flood propagation
- Runs each computational kernel on GPU
- By means of a multi resolution grid

HyLab
PARMA HYDRAULIC RESEARCH LAB

Dipartimento di Ingegneria e Architettura – DIA
Università di Parma

SCENARIO 1b

Crollo totale ed istantaneo del manufatto regolatore in concomitanza con un'onda di piena in ingresso alla cassa di tempo di ritorno T=200 anni

•Dal modello all'algoritmo

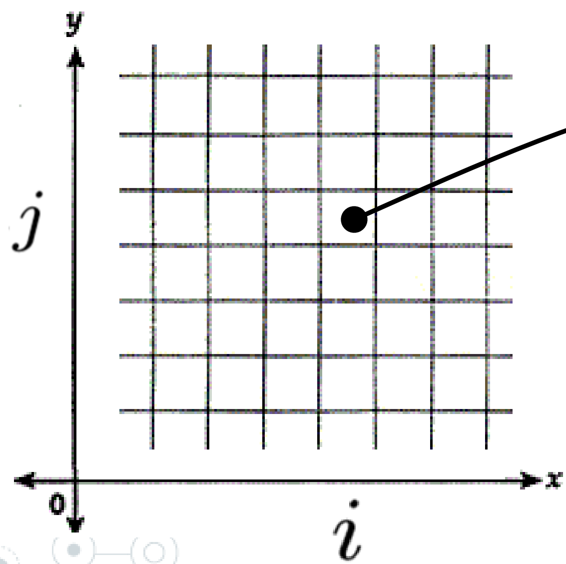
- Shallow Water Equations (SWE) (forma vettoriale)

$$\mathbf{U}_t + \mathbf{F}_x + \mathbf{G}_y = \mathbf{S}(\mathbf{U})$$

$$\mathbf{U} = \begin{bmatrix} h \\ hu \\ hv \end{bmatrix}, \quad \mathbf{F}_x = \begin{bmatrix} hu \\ hu^2 + \frac{1}{2}gh^2 \\ huv \end{bmatrix}, \quad \mathbf{G}_y = \begin{bmatrix} hv \\ hvu \\ hv^2 + \frac{1}{2}gh^2 \end{bmatrix}, \quad \mathbf{S}(\mathbf{U}) = \begin{bmatrix} s_1 \\ s_2 \\ s_3 \end{bmatrix}$$

.SWE

- Il metodo FV discretizza un dominio spaziale attraverso una griglia Cartesiana

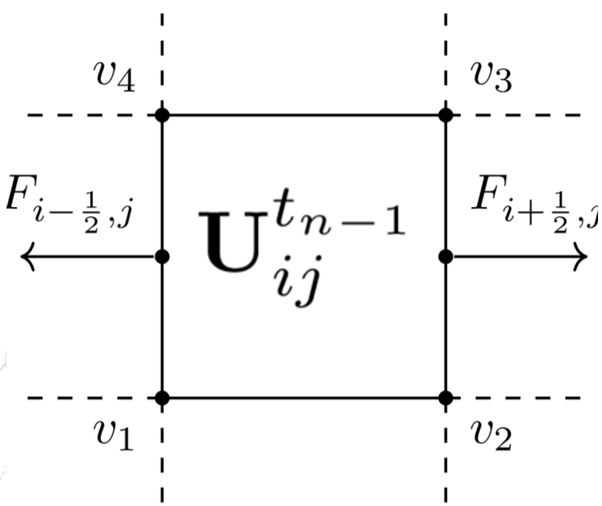


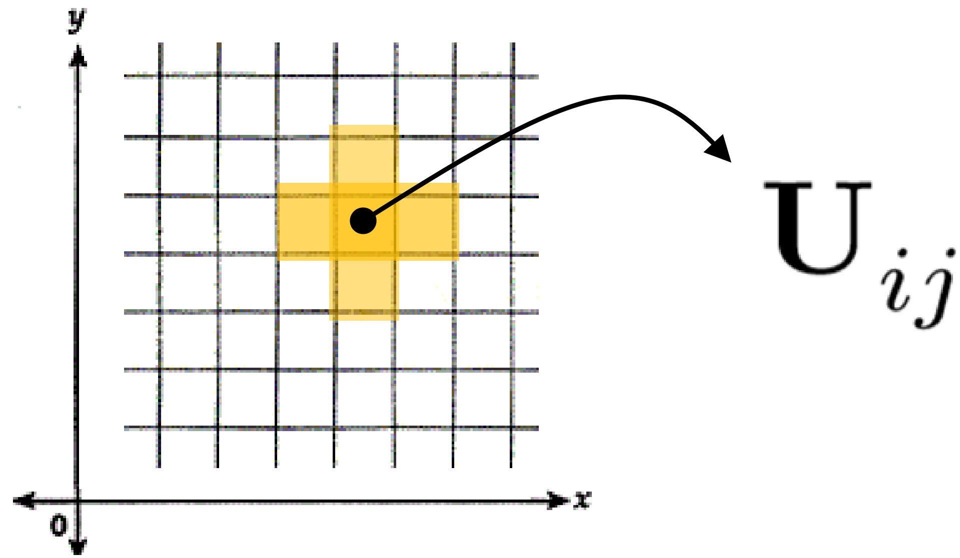
$$\mathbf{U}_{ij} = \begin{bmatrix} h \\ hu \\ hv \end{bmatrix}$$

Calcolo dei Flussi

- Dal punto di vista computazionale i flussi sono funzioni delle variabili conservate

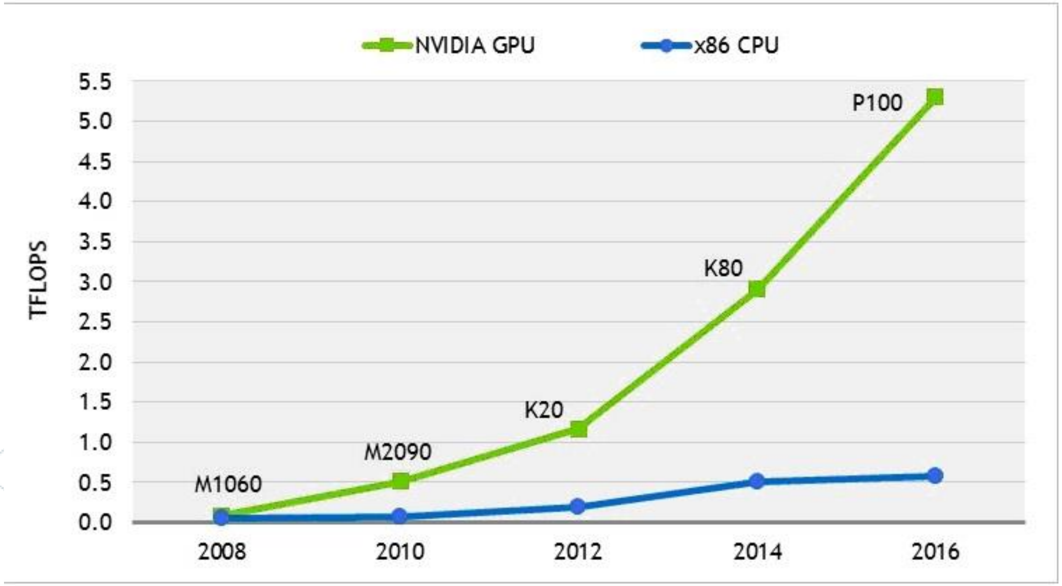
$$\mathbf{F}^{t_{n-1}}_{i+\frac{1}{2},j} = H(\mathbf{U}_{i-l,j}, \ldots, \mathbf{U}_{i,j}, \ldots, \mathbf{U}_{i+r,j})$$





Parallelizzazione

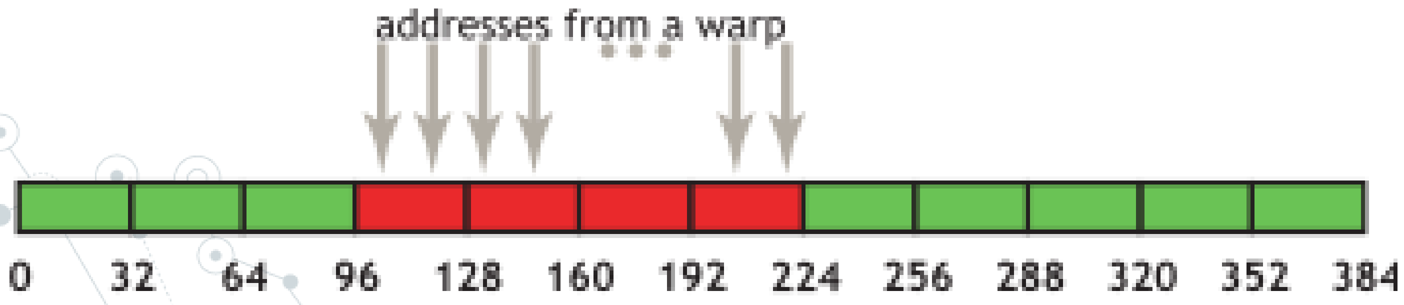
- Il problema è altamente parallelizzabile
- Diverse architetture hardware utilizzabili
- PARFLOOD utilizza le GPU (FLOPS alti)
- La CFD è memory bound quindi è difficile sfruttare appieno i FLOPS delle GPU



•Vantaggi delle griglie Cartesiane

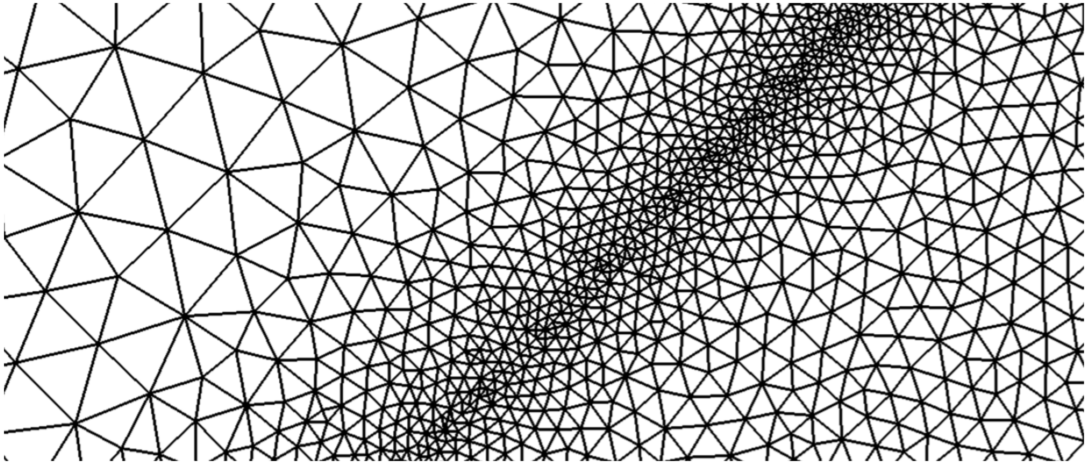
- Coerenza tra modello e algoritmo
- Efficienza grazie alla data locality
- Accessi efficienti alla memoria globale

"..the concurrent accesses of the threads of a warp will coalesce into a number of transactions equal to the number of 32-byte transactions necessary to service all of the threads of the warp.."
[NVIDIA CUDA v10.2.89]



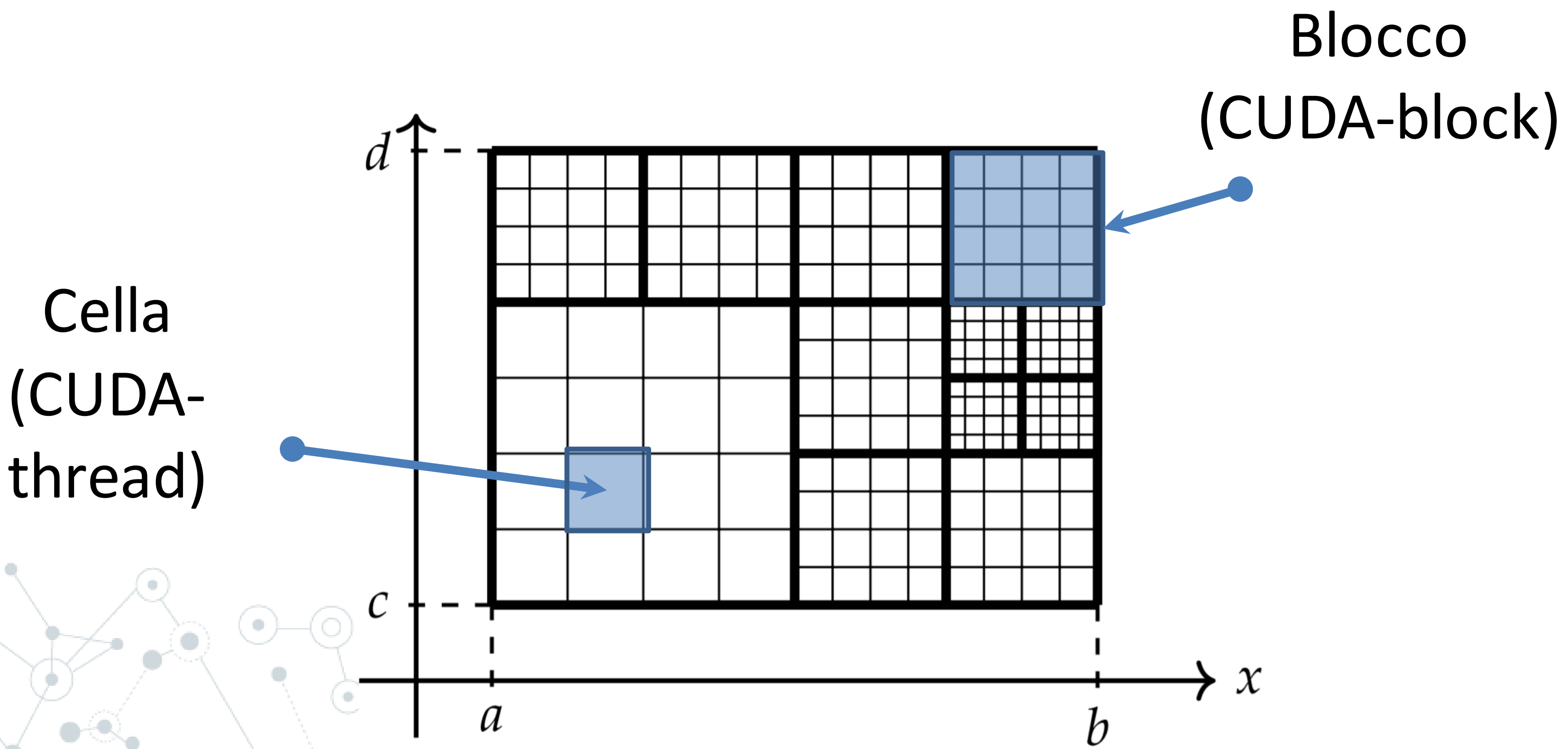
•Svantaggi delle griglie Cartesiane

- Alto numero di celle richiesto per modellare casi reali (i.e., superfici aventi $O(10^7)$ celle)
- L’alternativa è la multi-risoluzione
- In generale multi-risoluzione su GPU non è una buona idea (e.g., accessi in memoria)



Block Uniform Quad-tree Grid (BUQG)

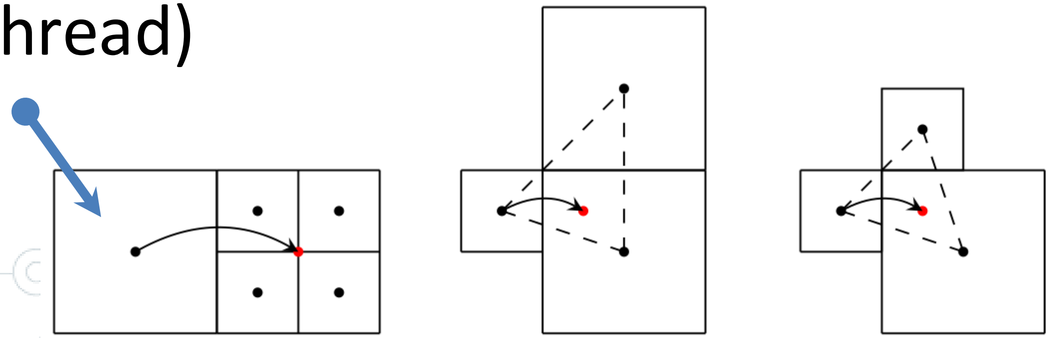
- Parflood trova un compromesso tra griglie multi-risoluzione e Cartesiane



•Block Uniform Quad-tree Grid (BUQG)

- Interpolazioni tra celle adiacenti di risoluzione differente
- Overhead algoritmici superiori rispetto alle griglie Cartesiane

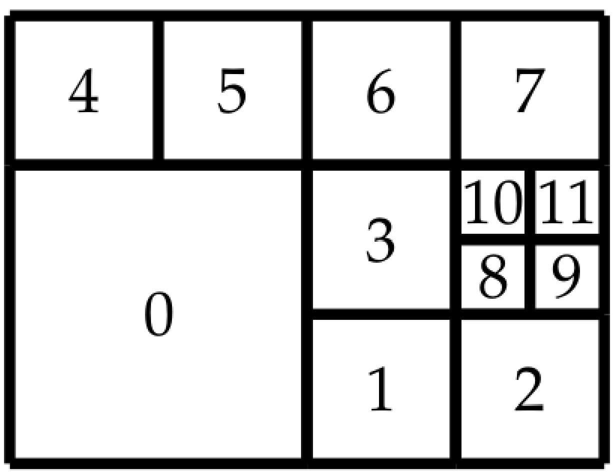
Cella
(CUDA-thread)



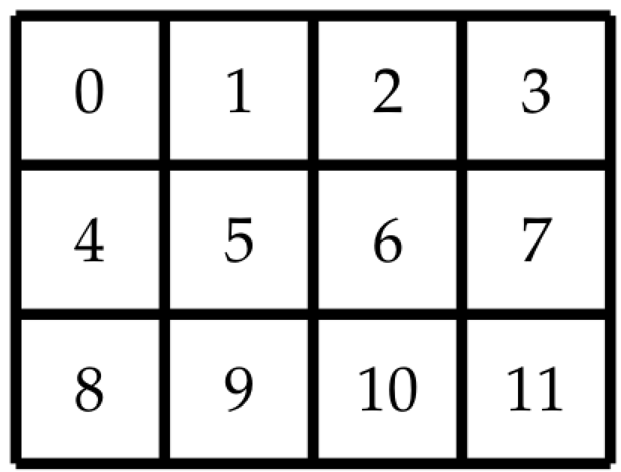
•Memorizzazione delle BUQG

- Parflood trova un compromesso tra griglie multi-risoluzione e Cartesiane
- Data locality persa tra celle di blocchi diversi
- Necessità di un grafo dei neighbors tra blocchi

Rappr. spaziale



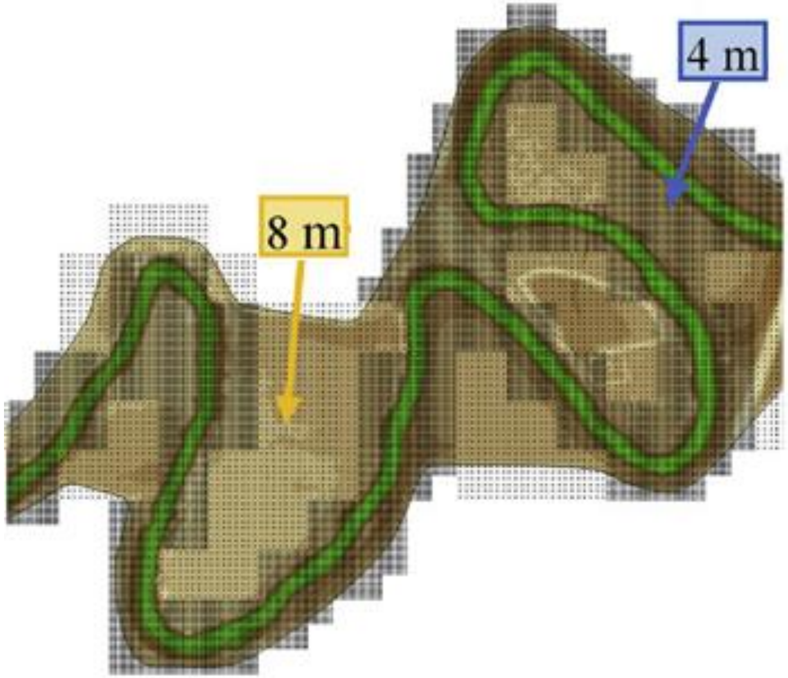
Rappr. di memoria



Block Uniform Quad-tree Grid (BUQG)

- Gli overhead vengono assorbiti dal minor numero di celle da processare
- Mantiene alto il livello di accuratezza

Vacondio R, Dal Palù A, Ferrari A, Mignosa P, Aureli F, Dazzi S. A non-uniform efficient grid type for GPU-parallel Shallow Water Equations models. Environmental Modelling & Software. 2017 Feb 1;88:119-37.



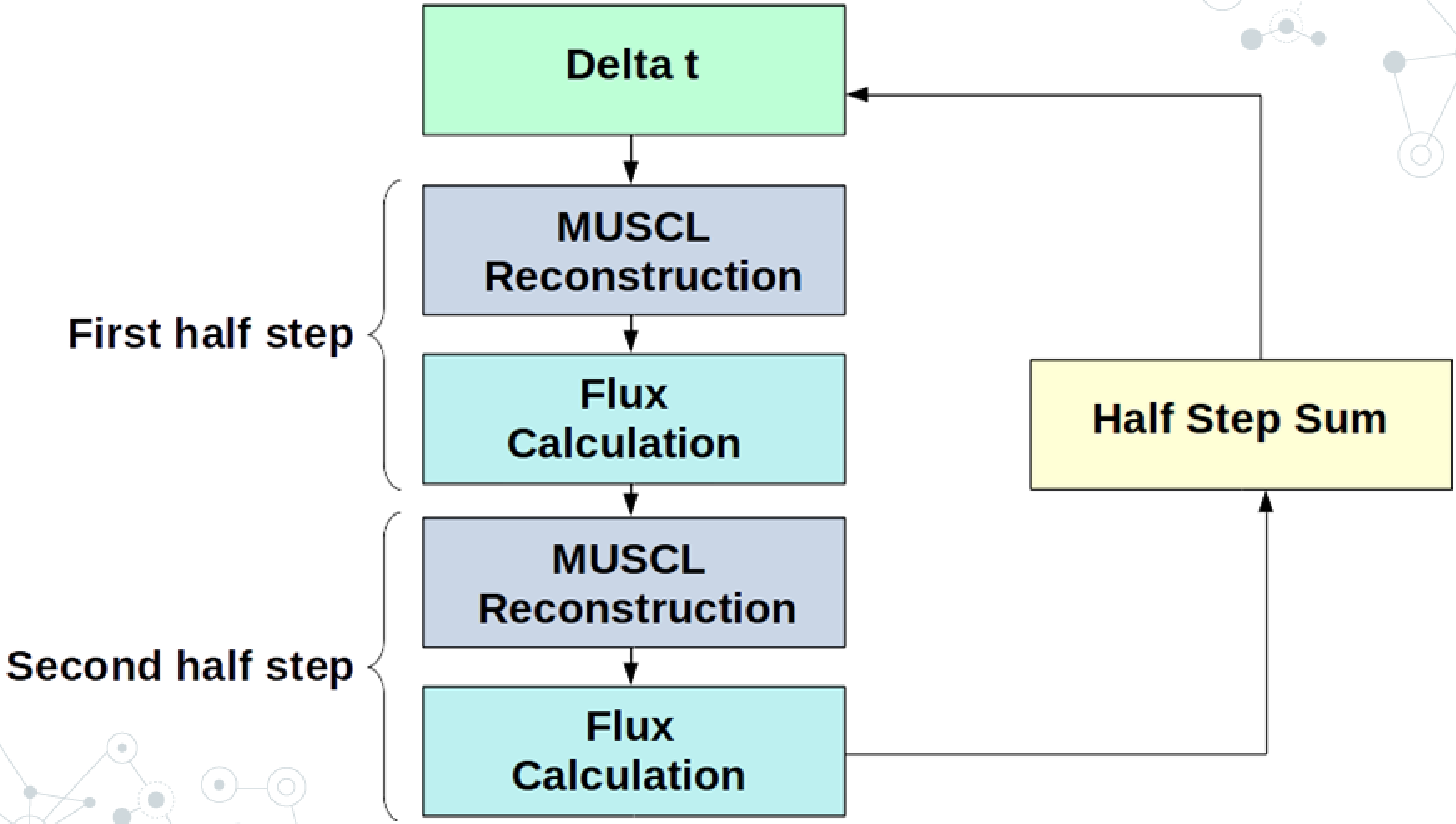
Evaluation of Block Uniform Quad Tree Grids

- Multiple levels of resolution
- Efficient usage of the memory available
- More efficient on GPU than Unstructured Grids

Further details: **Vacondio et al. 2017**

A non-uniform efficient grid type for GPU-parallel Shallow Water Equations models

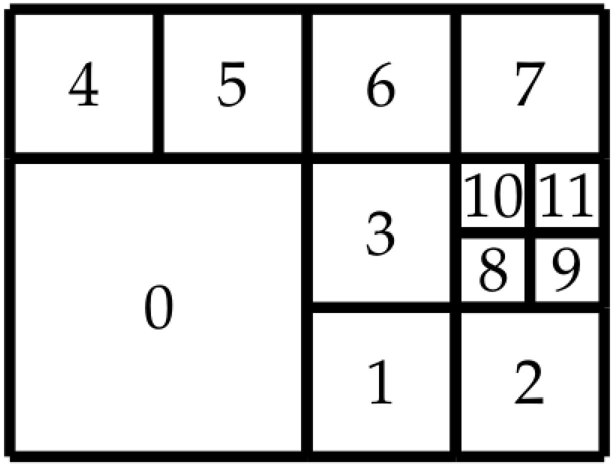
Overall single GPU algorithm



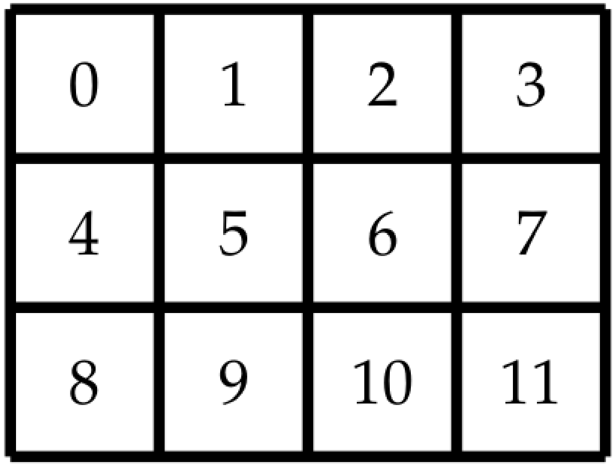
•Memorizzazione delle BUQG

- Parflood trova un compromesso tra griglie multi-risoluzione e Cartesiane
- Data locality persa tra celle di blocchi diversi
- Necessità di un grafo dei neighbors tra blocchi

Rappr. spaziale



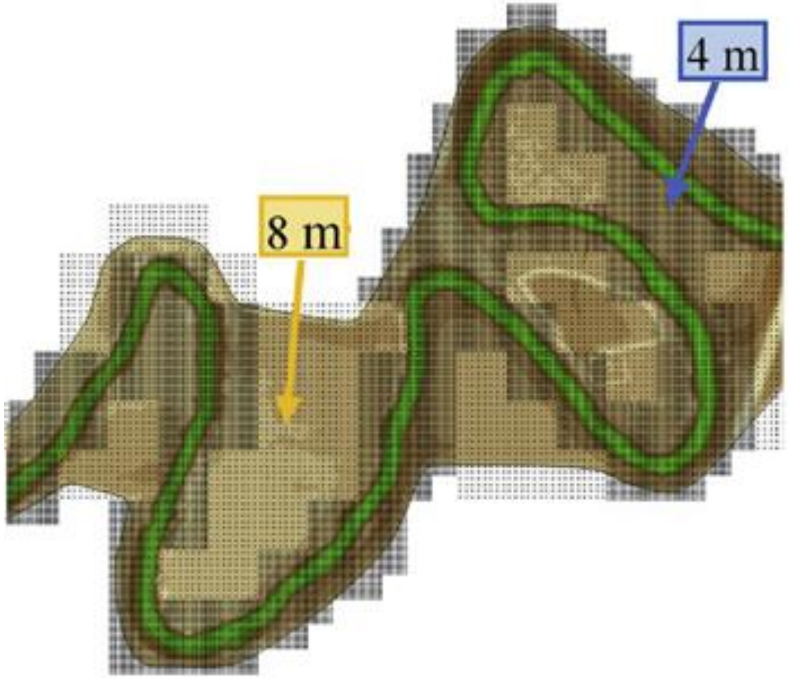
Rappr. di memoria



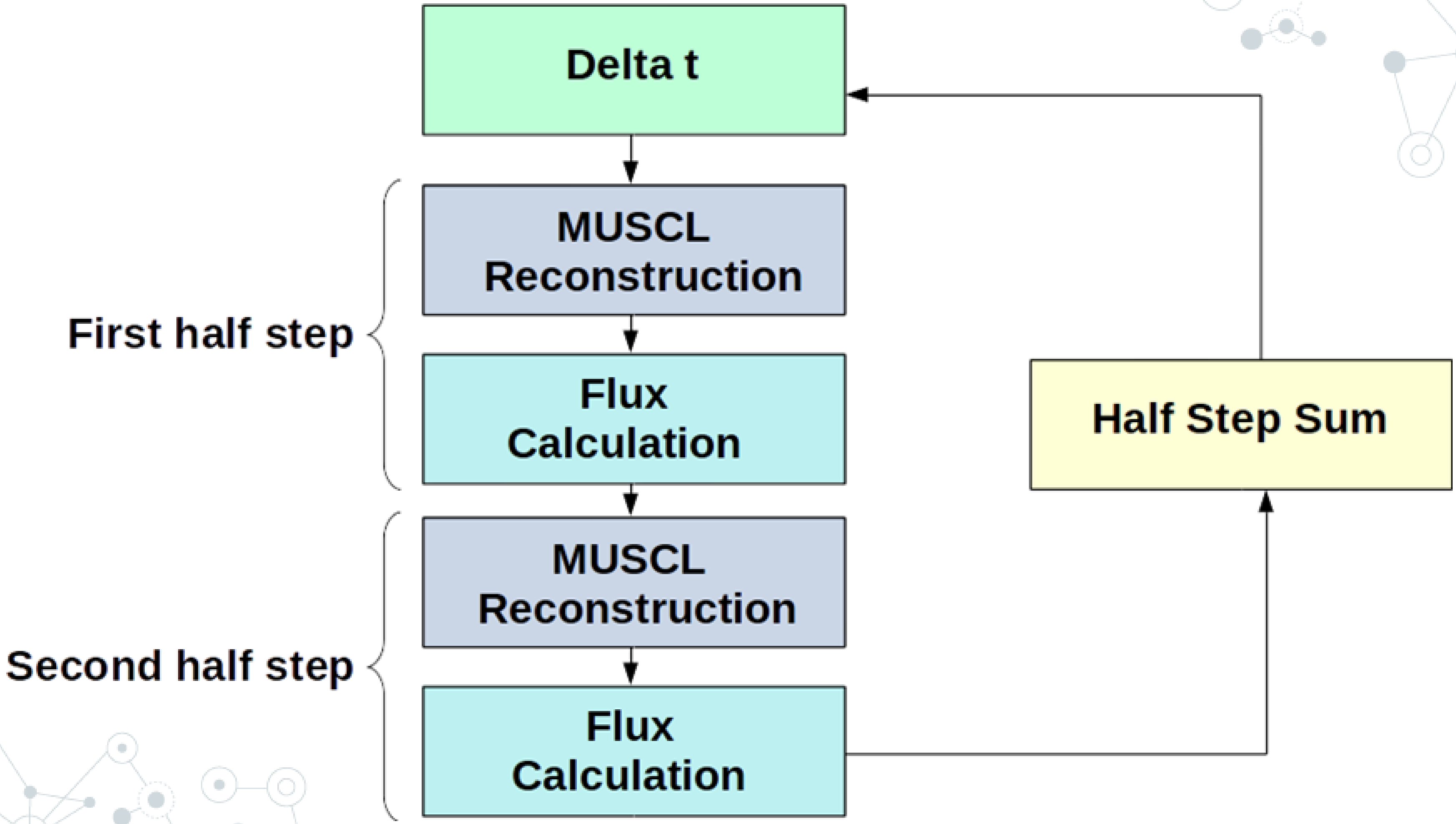
•Block Uniform Quad-tree Grid (BUQG)

- Gli overhead vengono assorbiti dal minor numero di celle da processare
- Mantiene alto il livello di accuratezza

Vacondio R, Dal Palù A, Ferrari A, Mignosa P, Aureli F, Dazzi S. A non-uniform efficient grid type for GPU-parallel Shallow Water Equations models. Environmental Modelling & Software. 2017 Feb 1;88:119-37.



•Algoritmo singola GPU



•Limitazioni singola GPU

Numero GPU	N. Celle (Milioni)	Tempo (Ore)	Superficie (Km^2)
1	10	5-20	10·000

- Obiettivo 1: Diminuire i tempi di calcolo

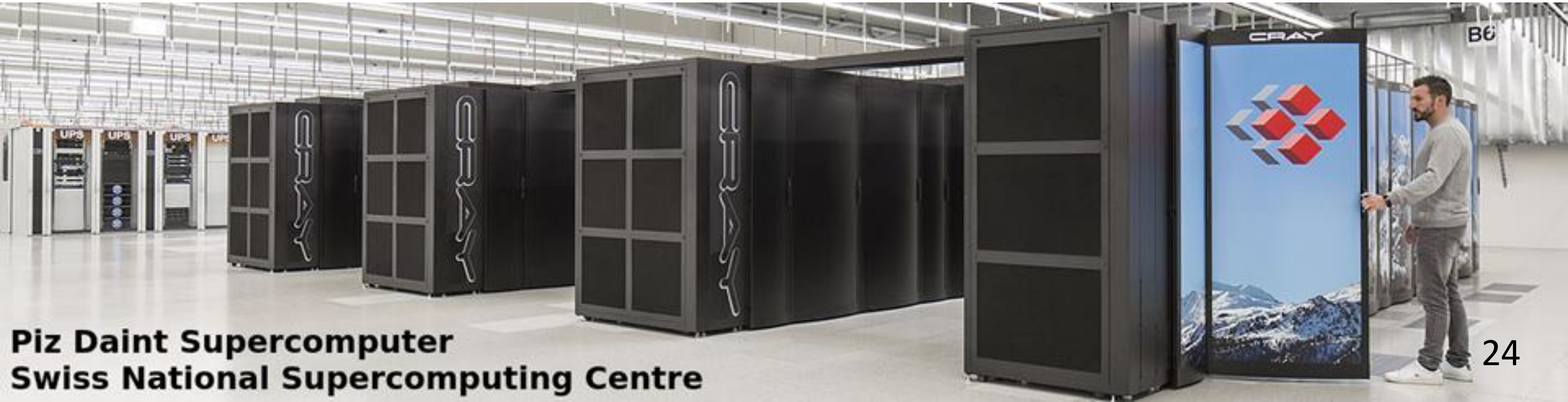
Numero GPU	N. Celle (Milioni)	Tempo (Ore)	Superficie (Km^2)
1	10	5 - 20	10·000
10	10	**<1 - 3**	10·000

•Obiettivo 2: Scalabilità Spaziale

Numero GPU	N. Celle (Milioni)	Superficie (Km^2)
50-100	100	100·000

Extending Parflood on multiple GPUs

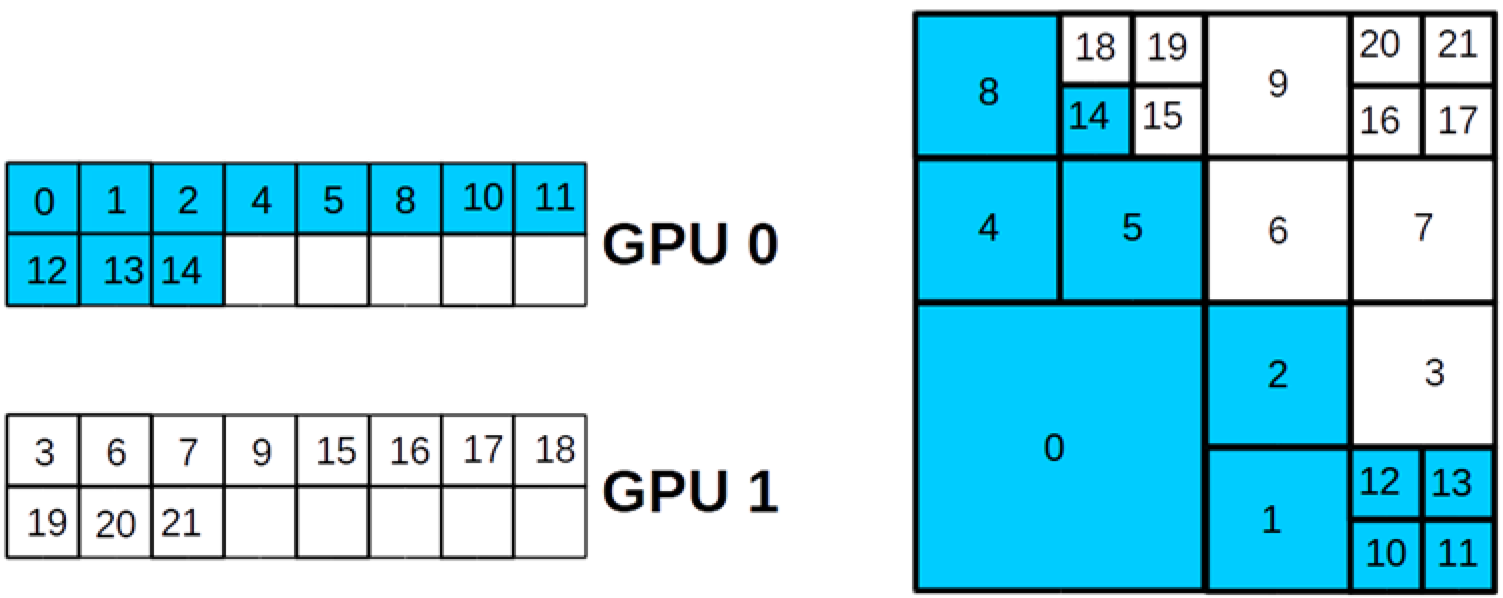
- Further decrease the simulation time
- Overcome the memory limitation of a GPU
- Can be deployed on HPC Systems



Piz Daint Supercomputer
Swiss National Supercomputing Centre

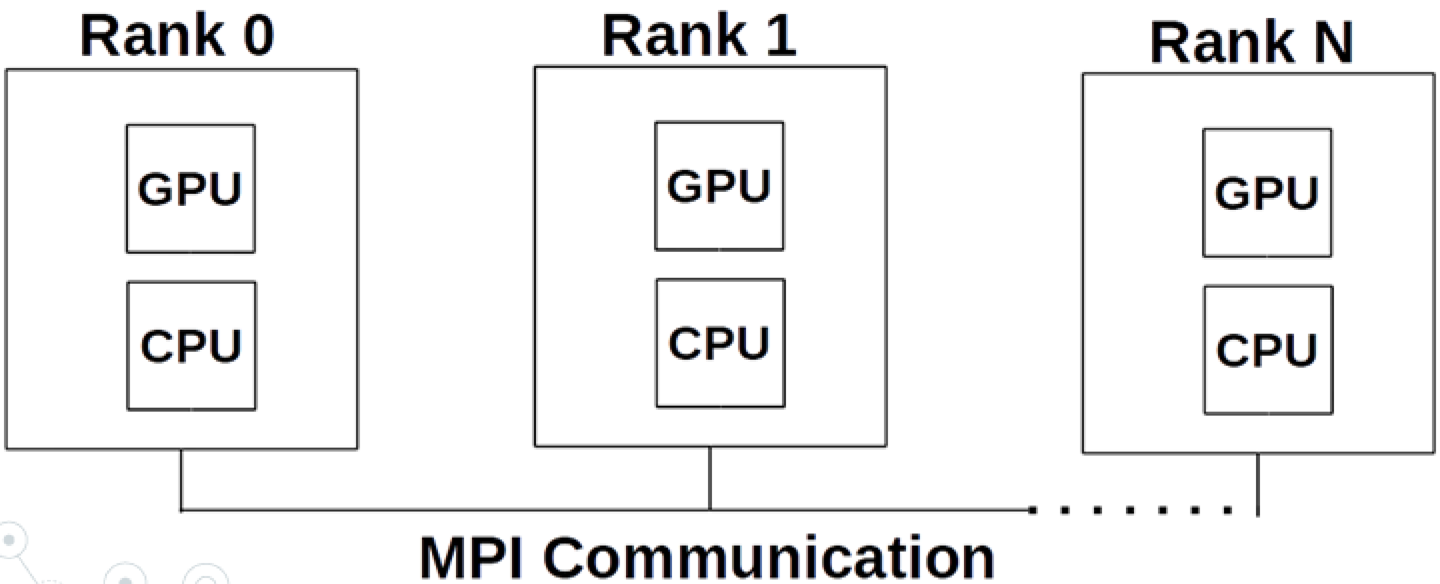
Domain Decomposition on BUQ grids

- Each GPU has its own local grid
- Each block belongs to one single partition



Main Network Requirements

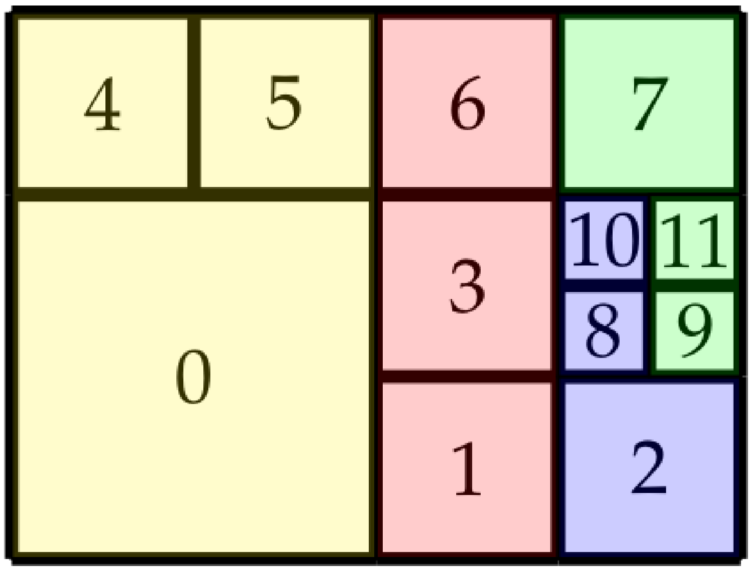
- One MPI process for each partition
- Each partition simulated on a dedicated GPU



1D Partitioning

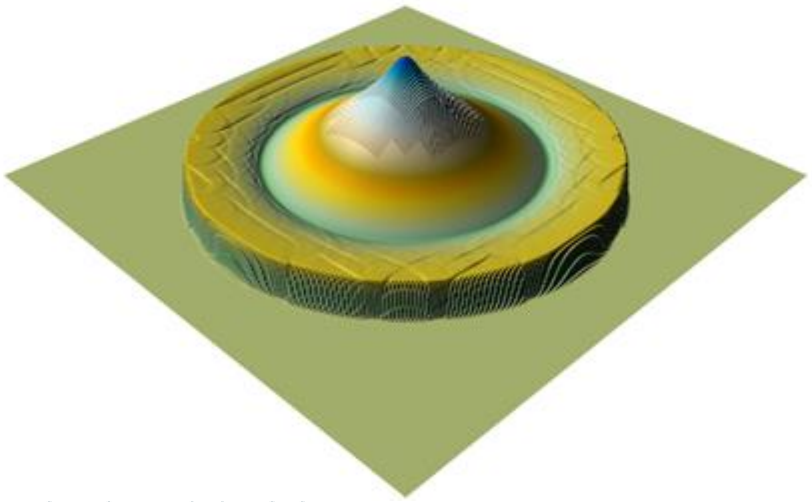
- Sort blocks using a topological ordering

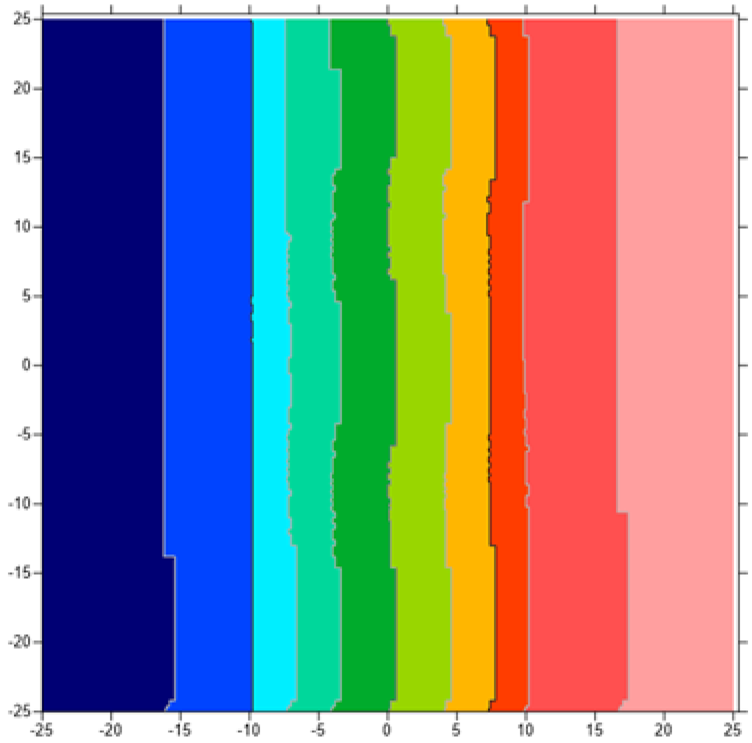
4, 0, 5   1, 3, 6   8, 10, 2   7, 9, 11



1D Partitioning

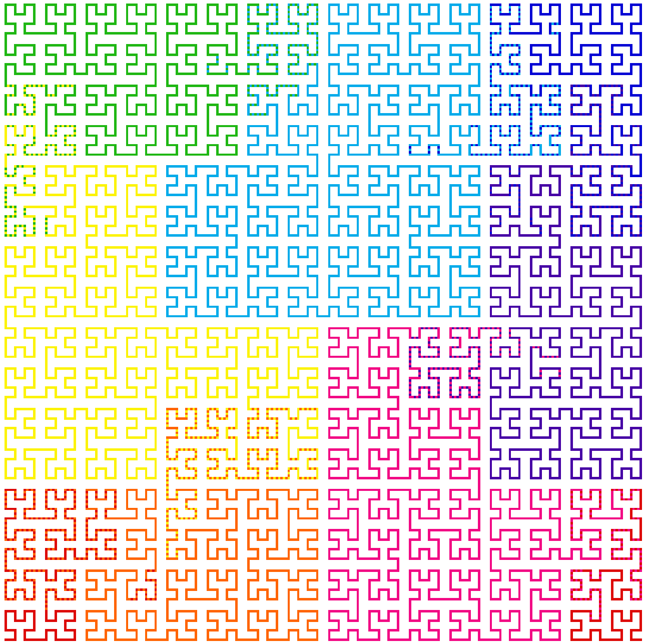
- Simple but inefficient
- By increasing the partition number the border size remains (almost) the same

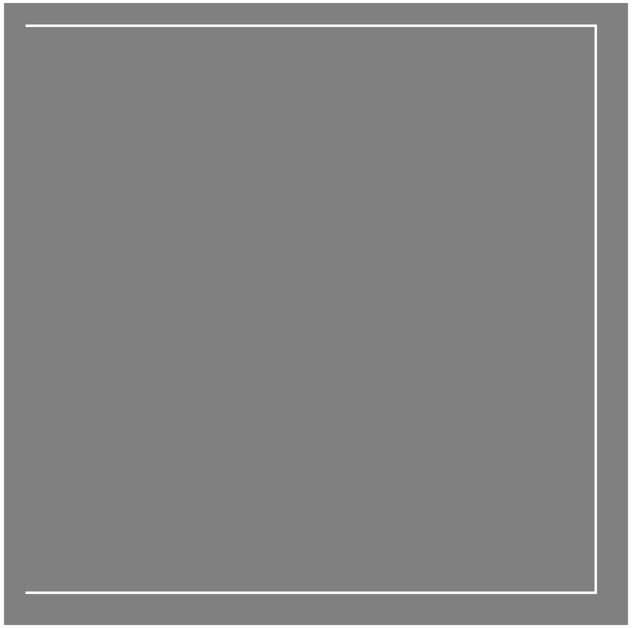




Hilbert Space-Filling Curves (HSFC)

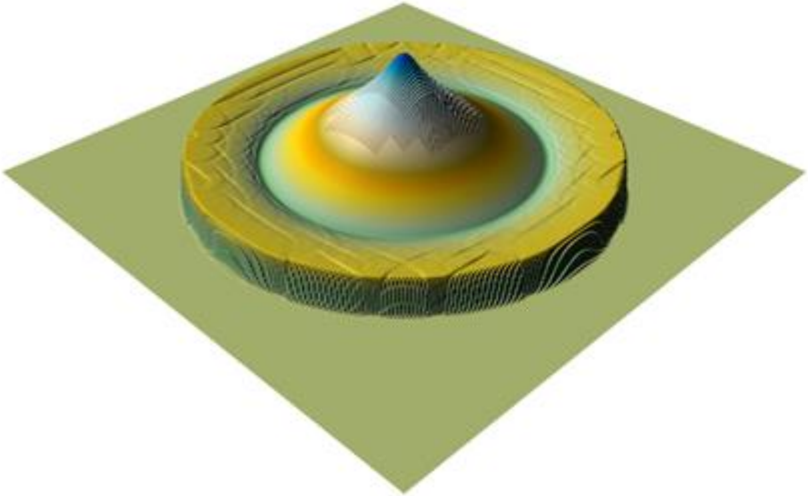
- Defines a mapping between 2D and 1D spaces (Zoltan Library)

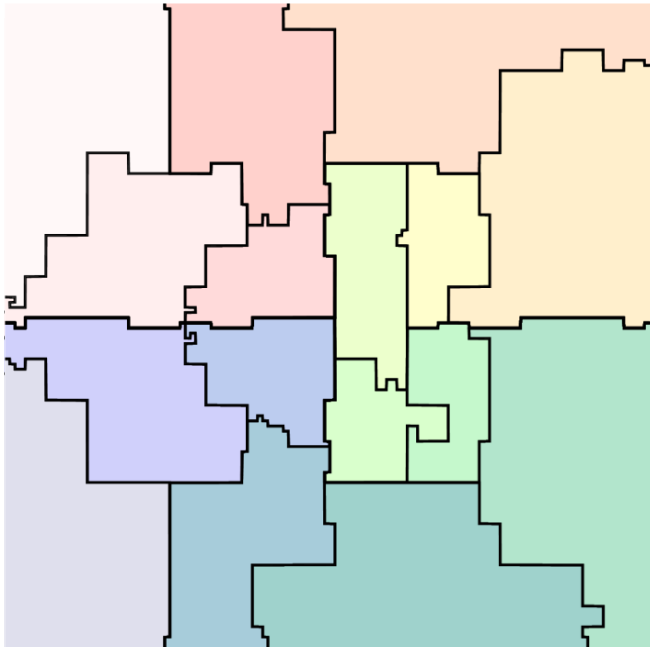




Hilbert Space-Filling Curves (HSFC)

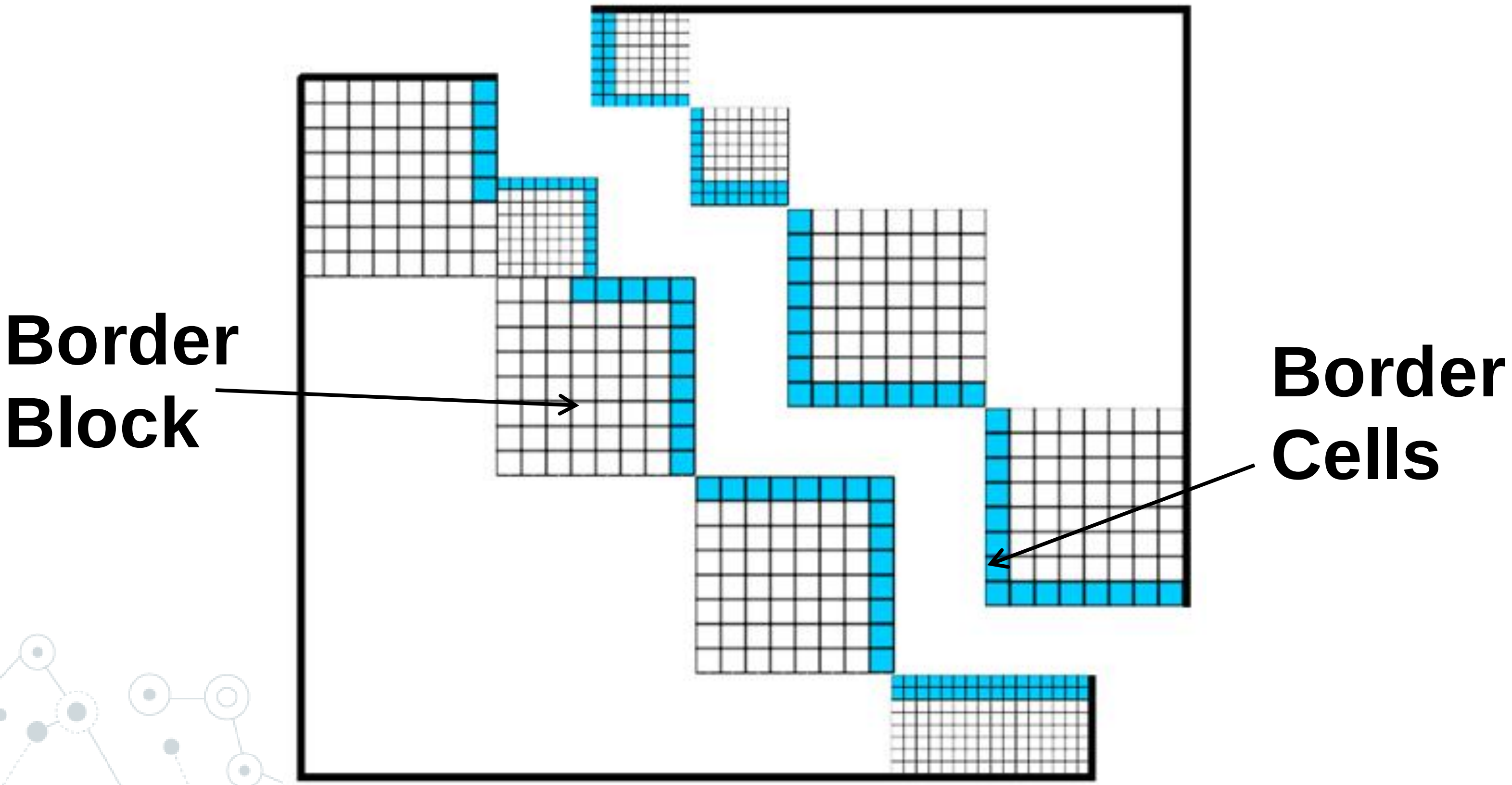
- Border size decreases by increasing the partition number
- More efficient than 1D partitioning



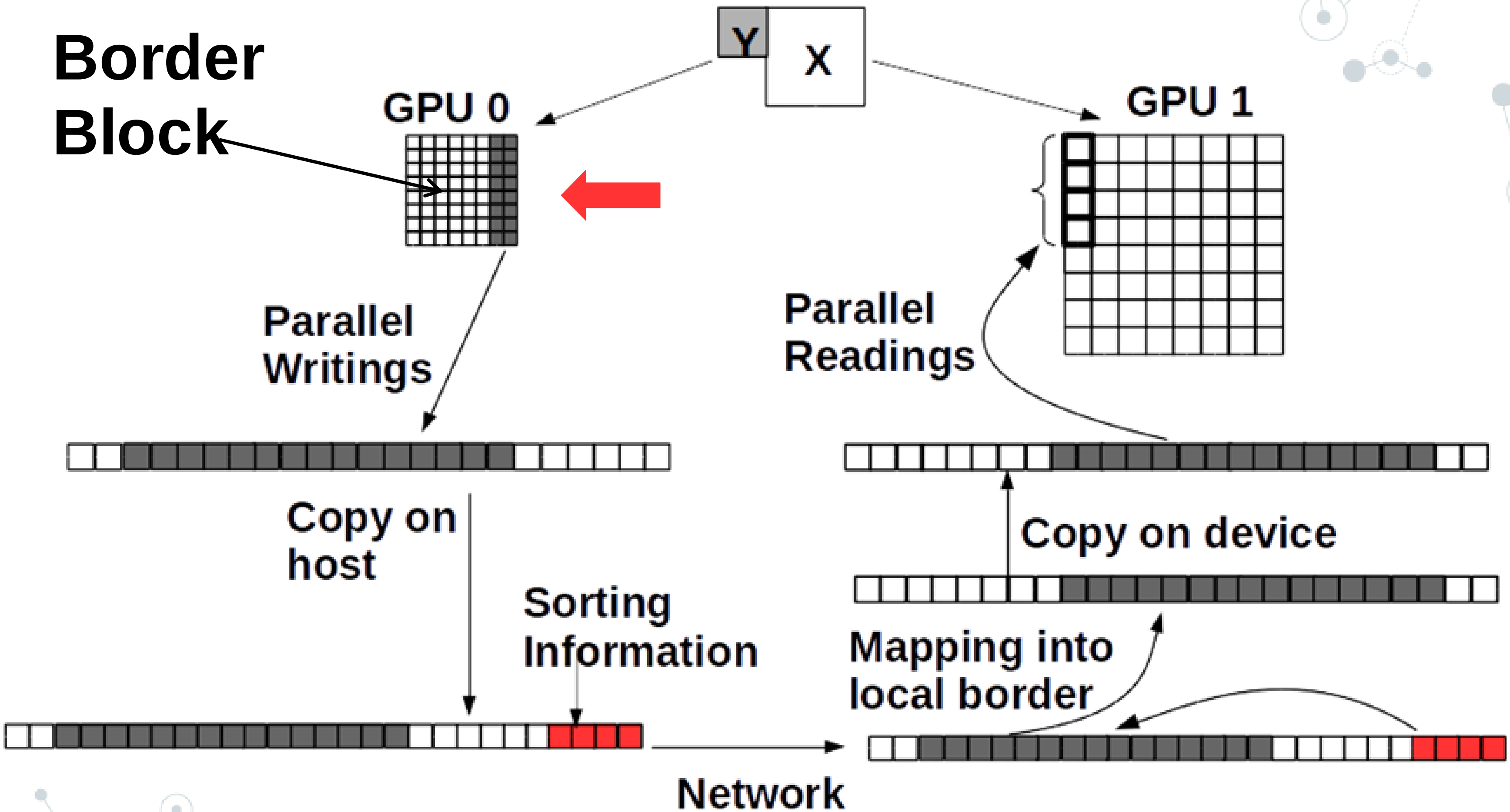


Border Communication

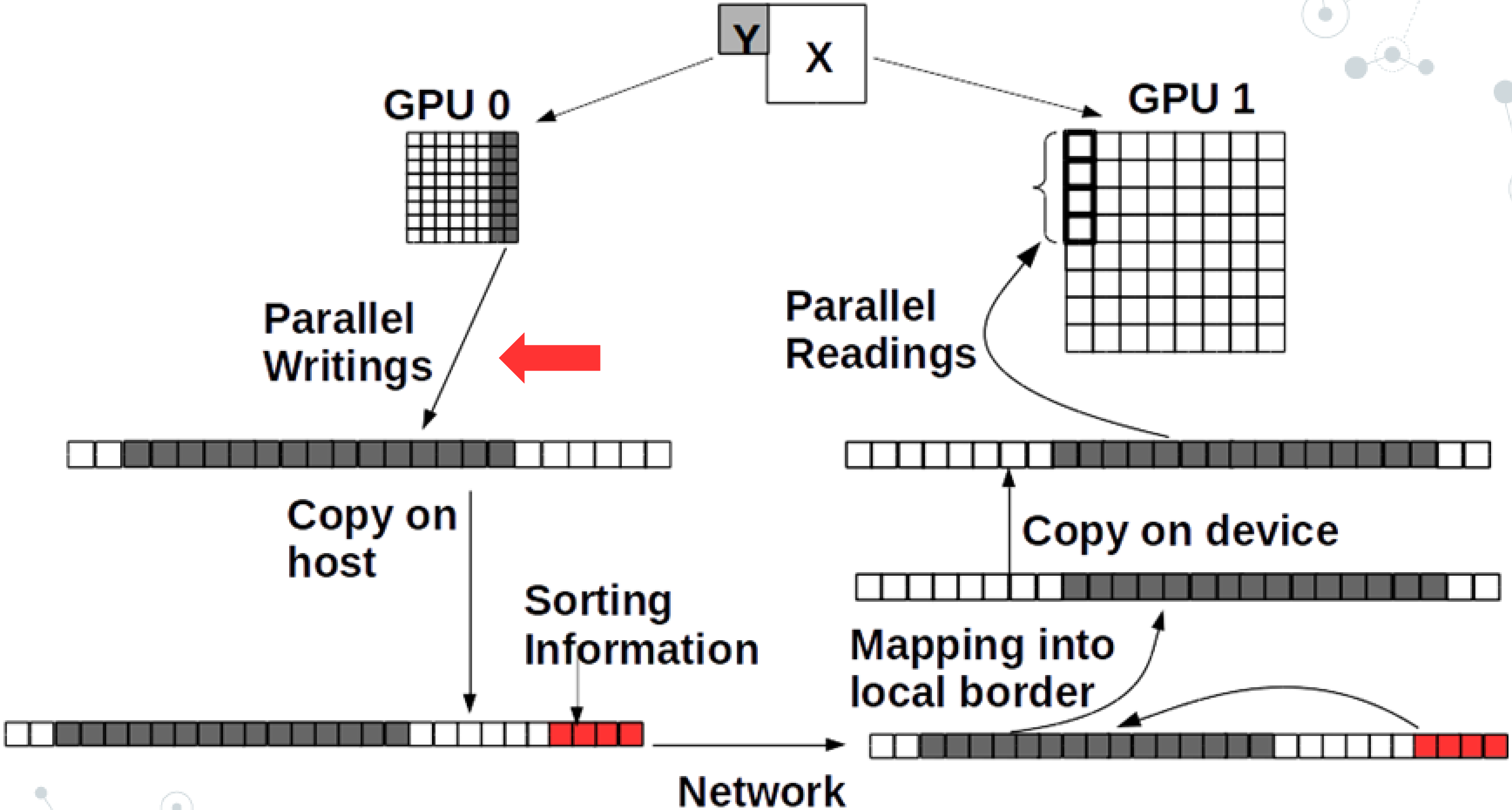
- Neighboring partitions need to exchange border cells



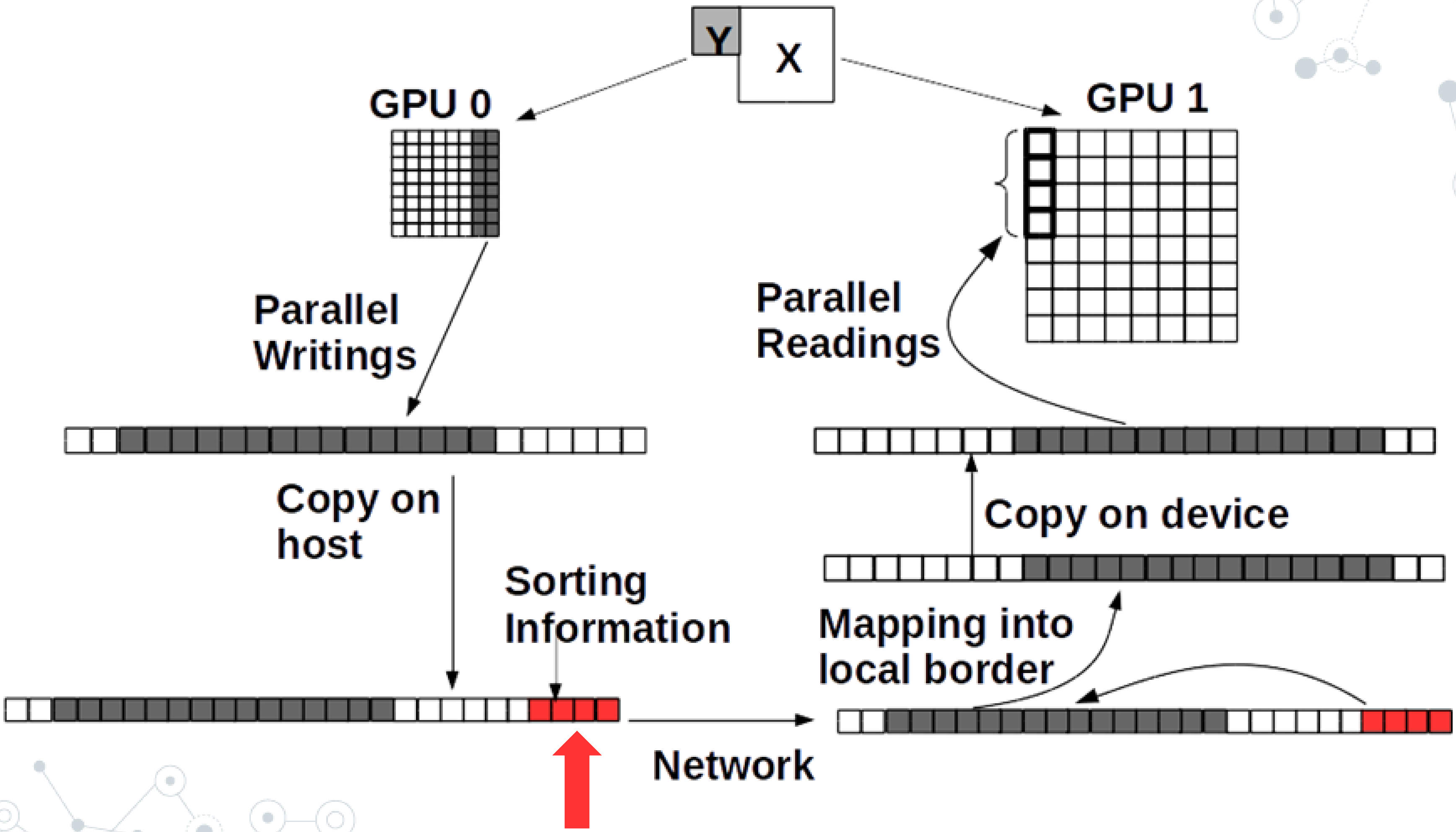
Overall Communication Steps



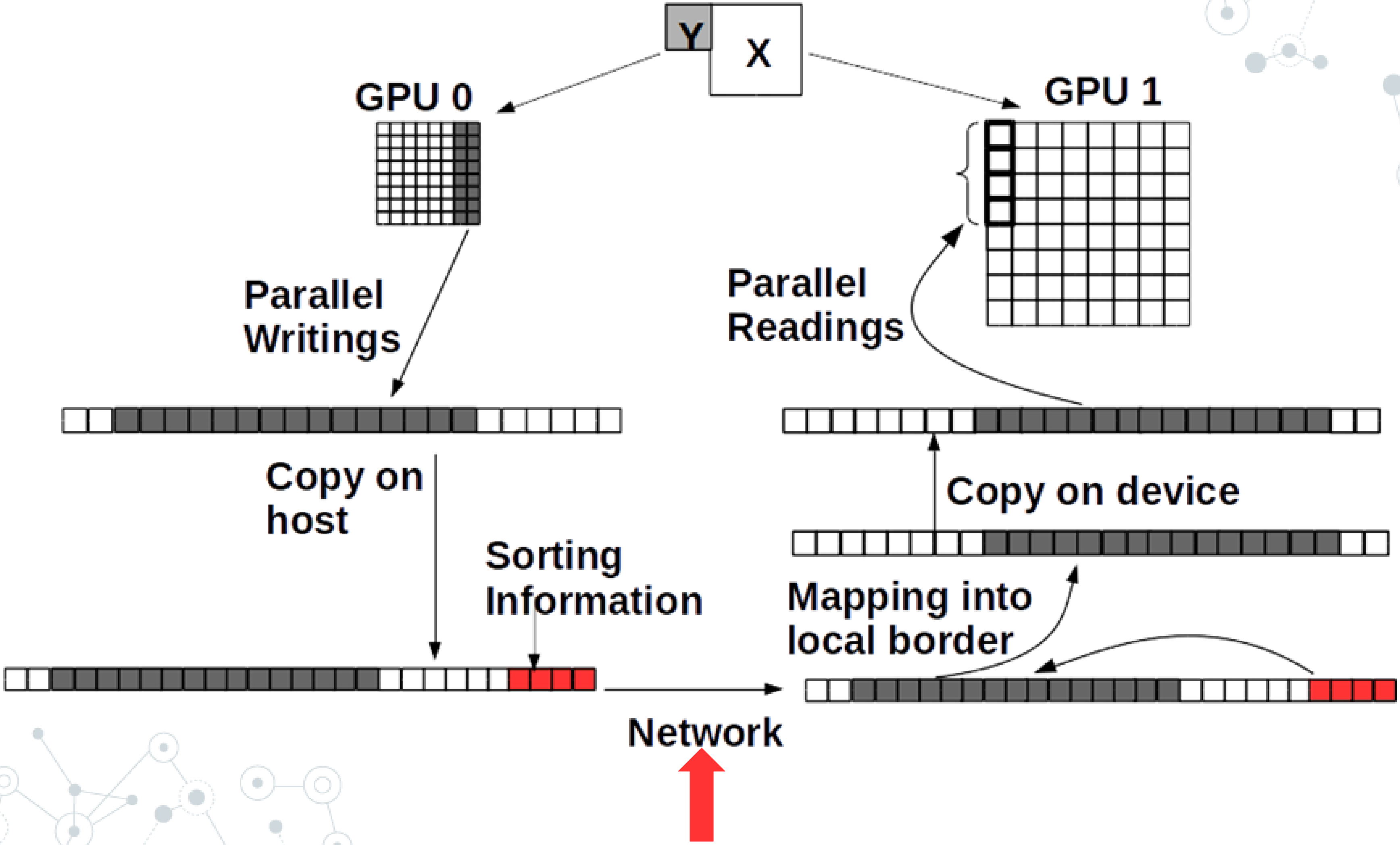
Overall Communication Steps



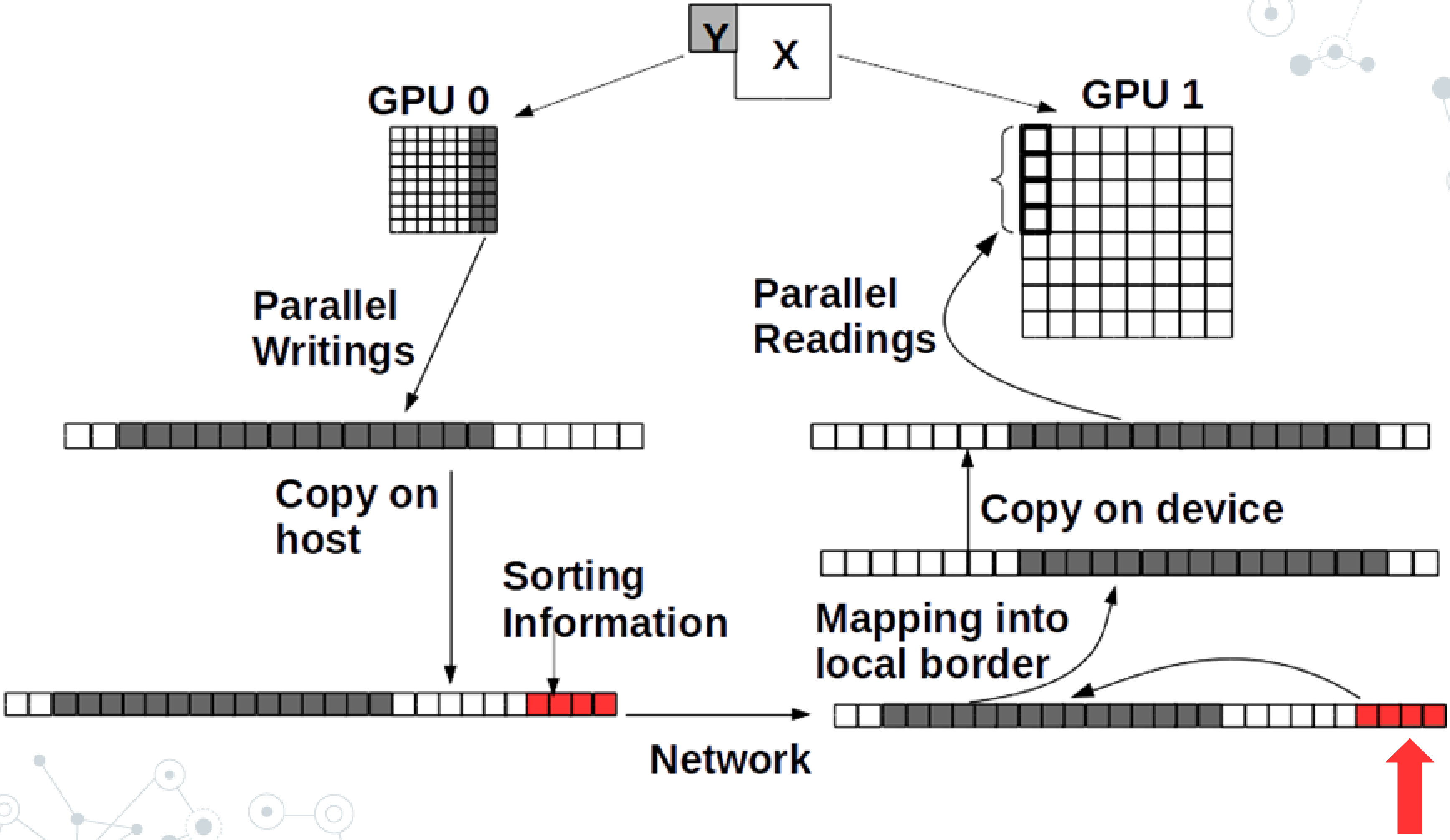
Overall Communication Steps



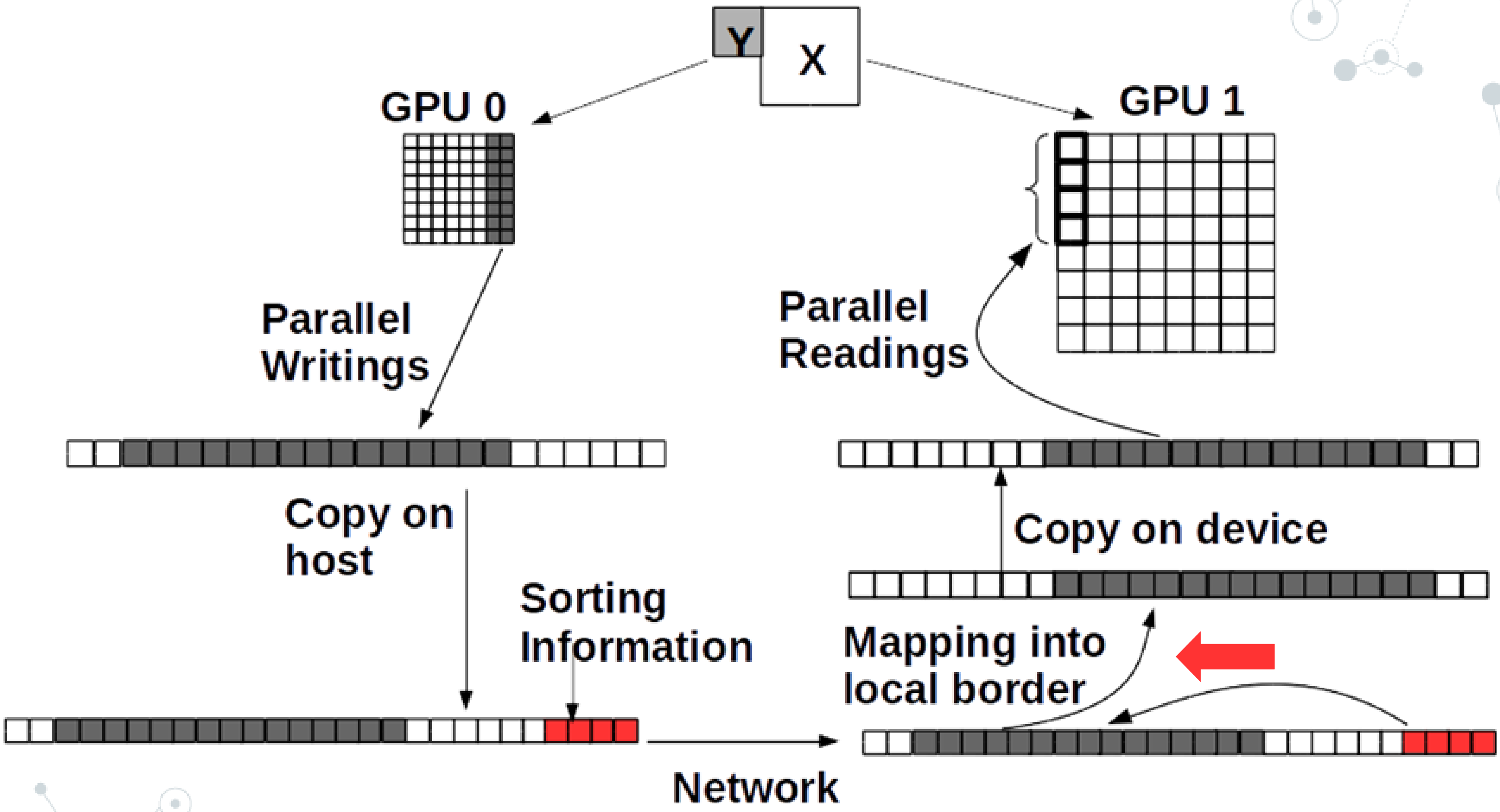
Overall Communication Steps



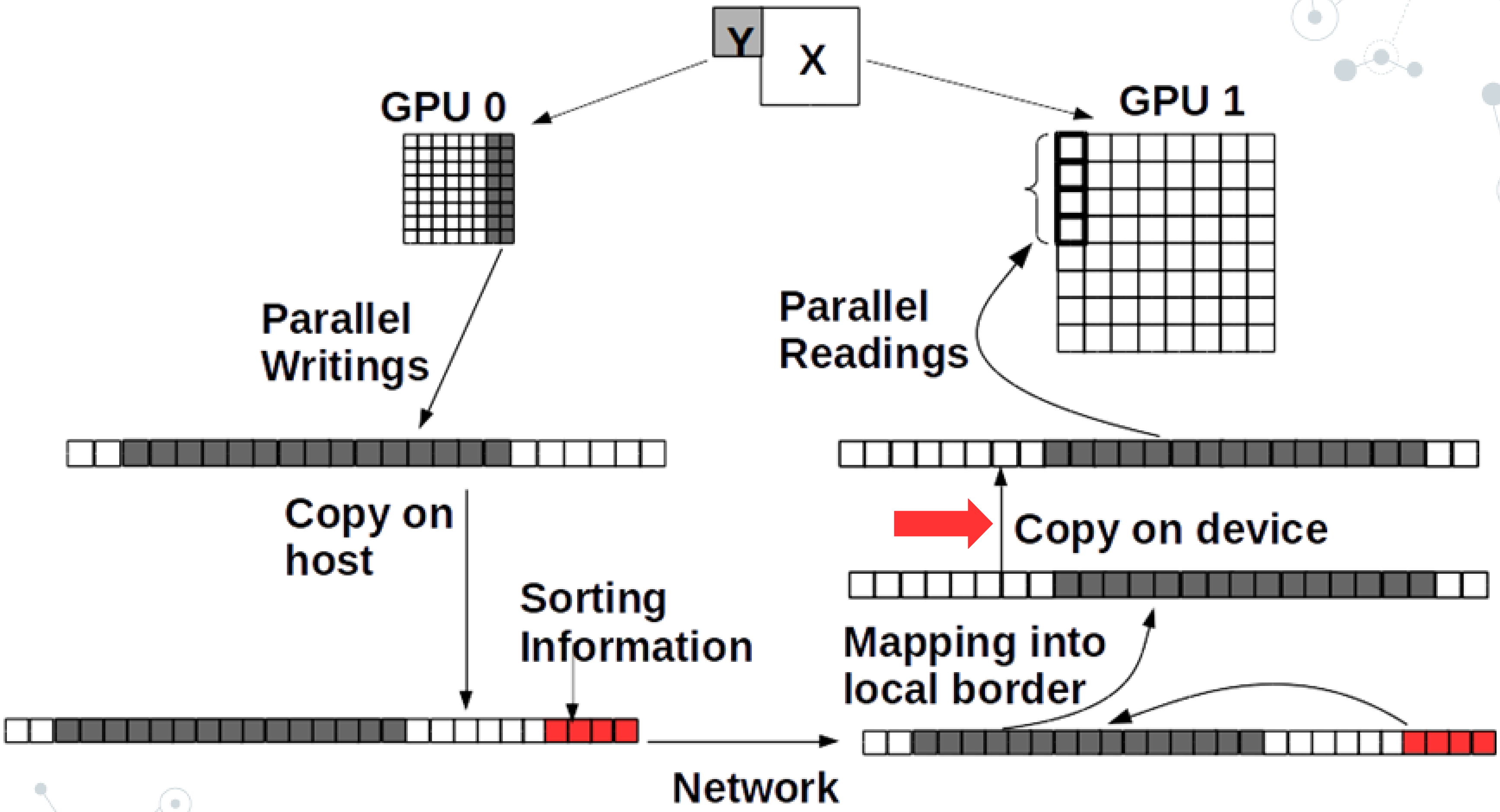
Overall Communication Steps



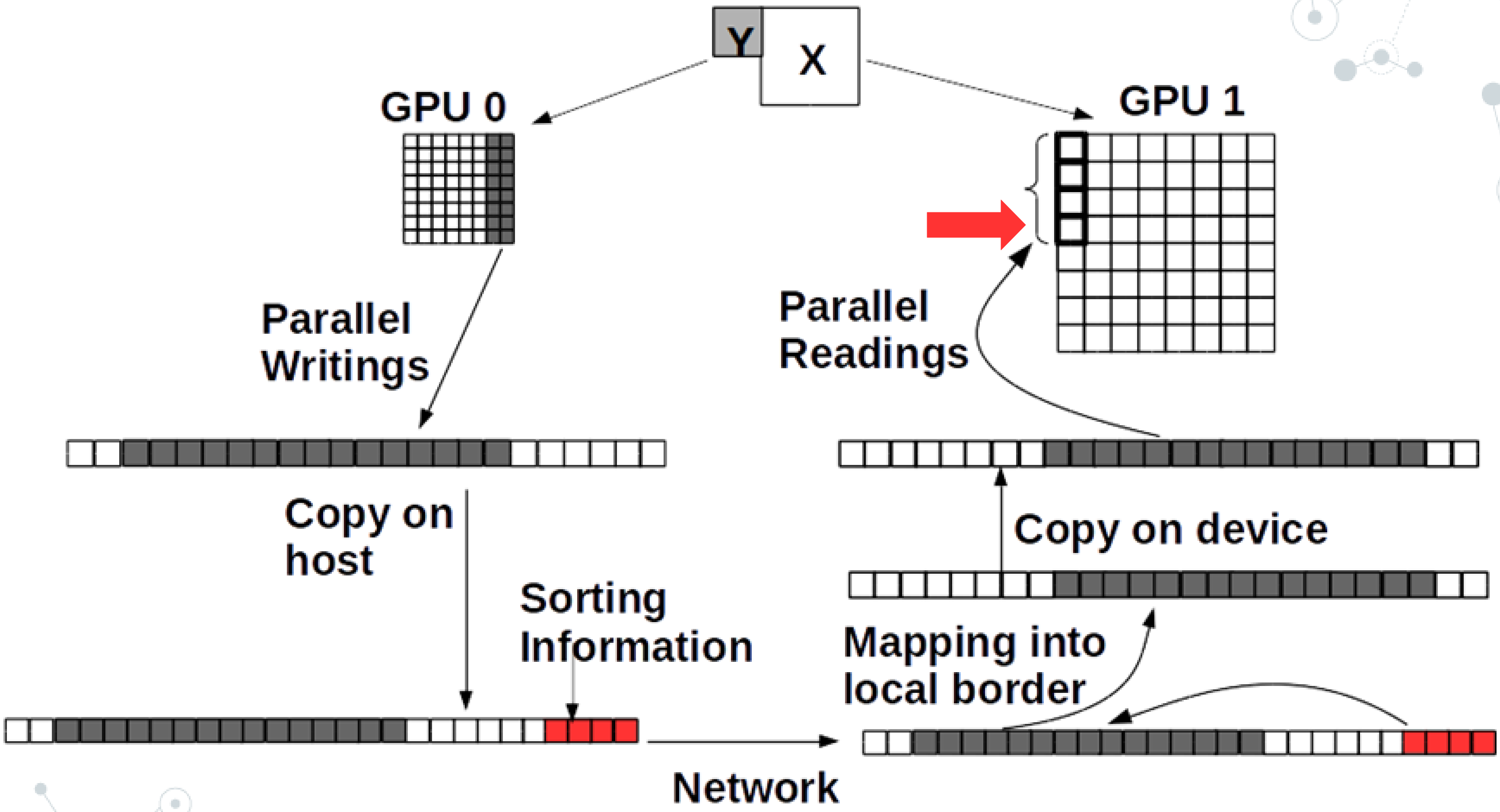
Overall Communication Steps



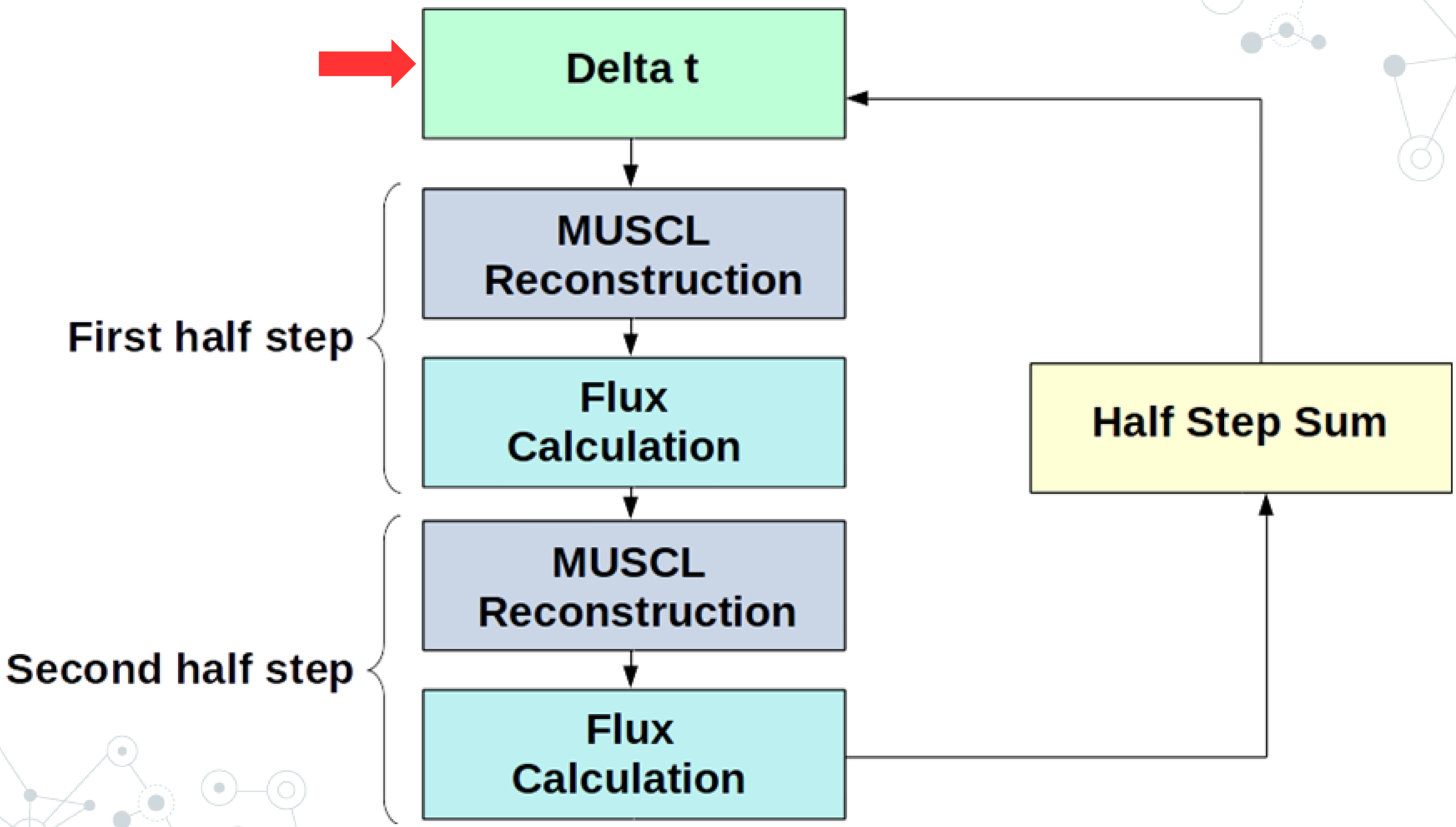
Overall Communication Steps



Overall Communication Steps

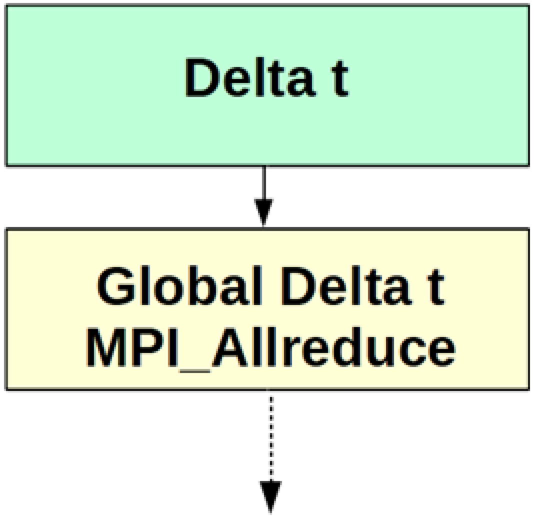


Overall single GPU algorithm

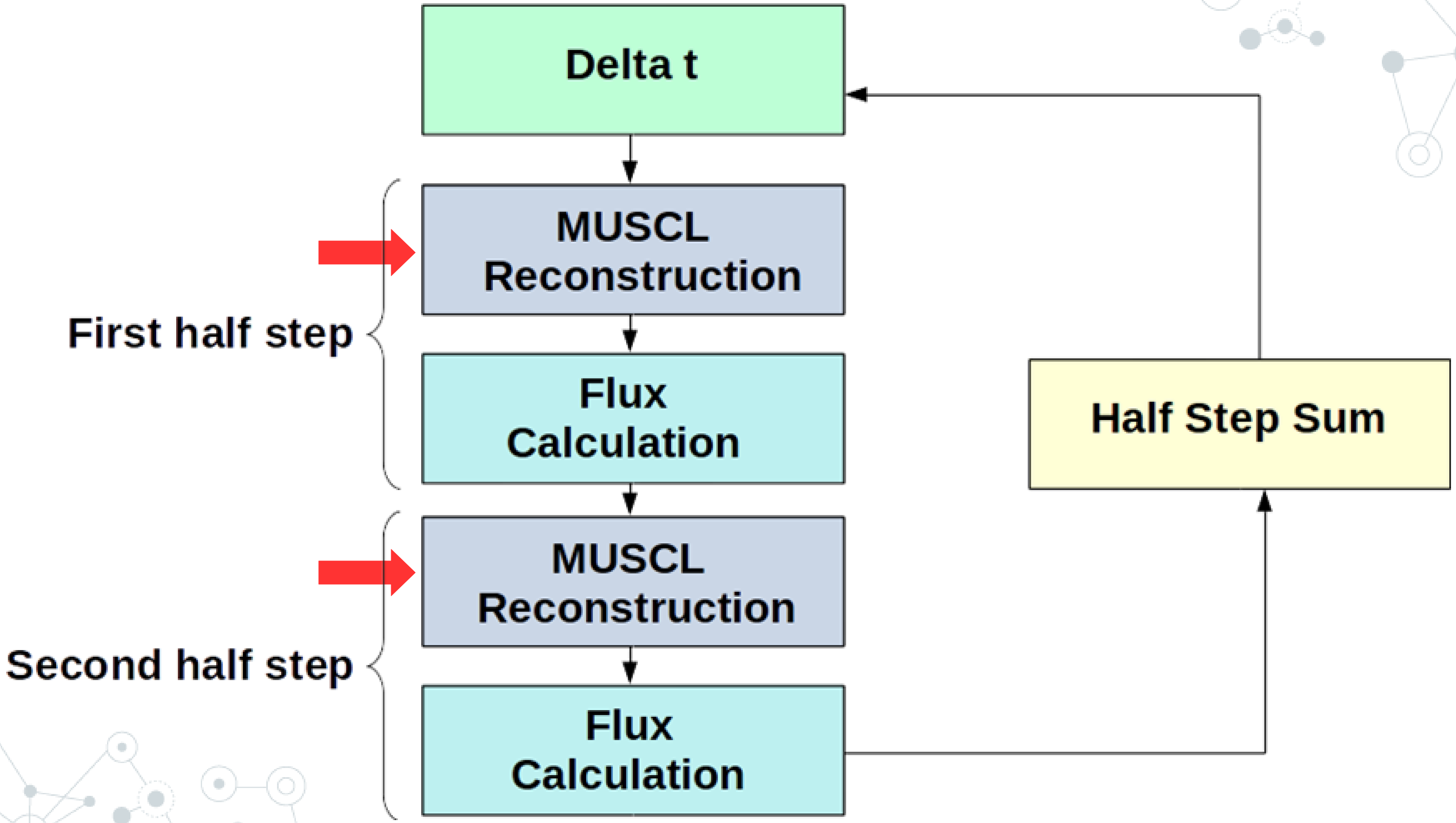


Global delta t Synchronization

- Local Delta t reduction for each partition
- Minimum delta t computed among all partitions
- Collective (blocking) call

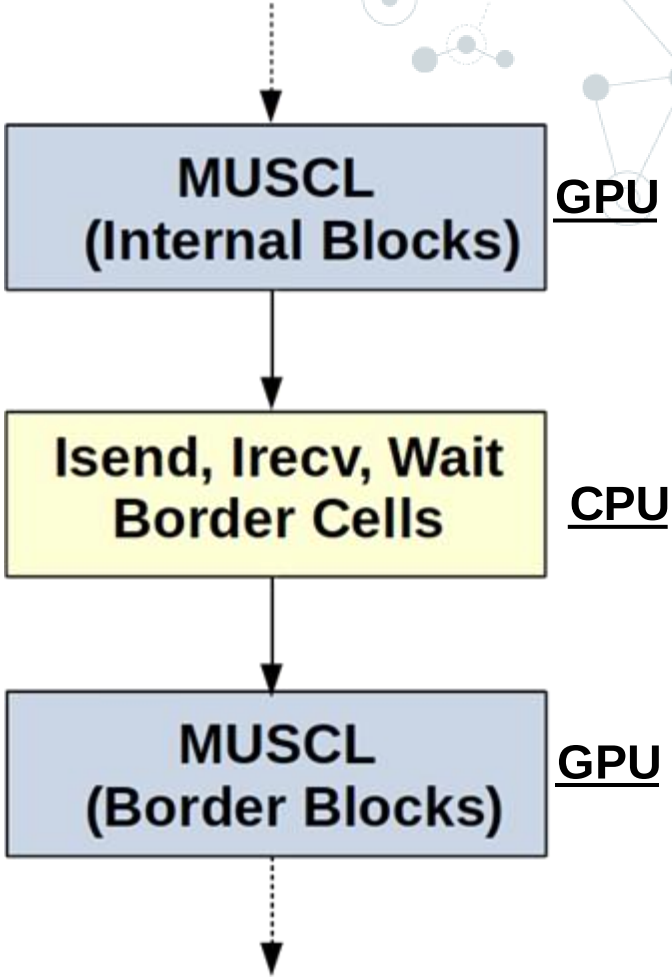


Overall single GPU algorithm



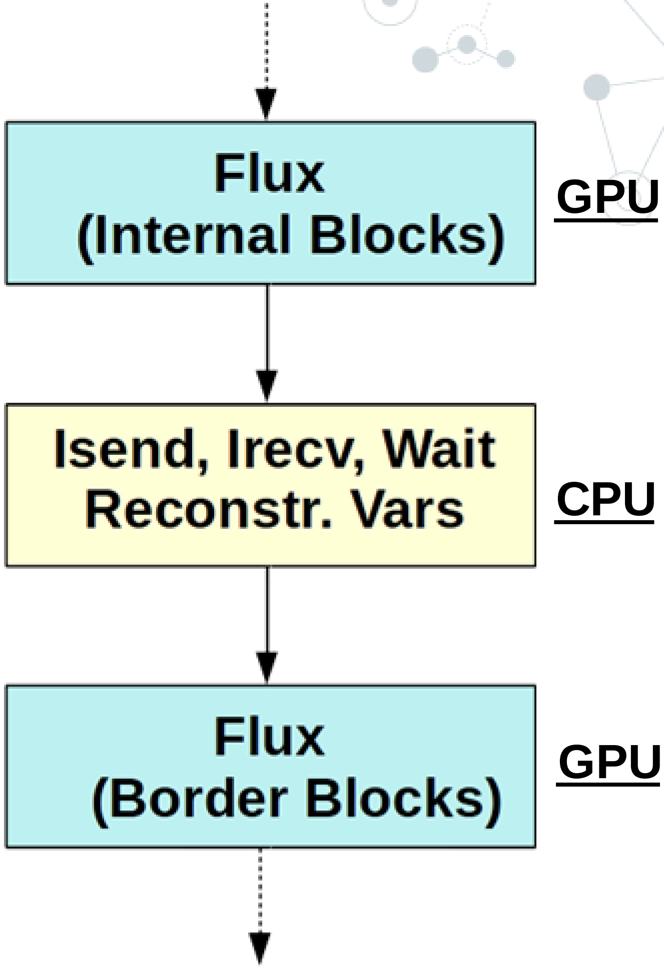
MUSCL Reconstruction on Multi GPU

- MUSCL kernel split into two parts
- Remote data not required by Internal Blocks
- Masking MPI communication with kernel execution



Fluxes on Multi GPU

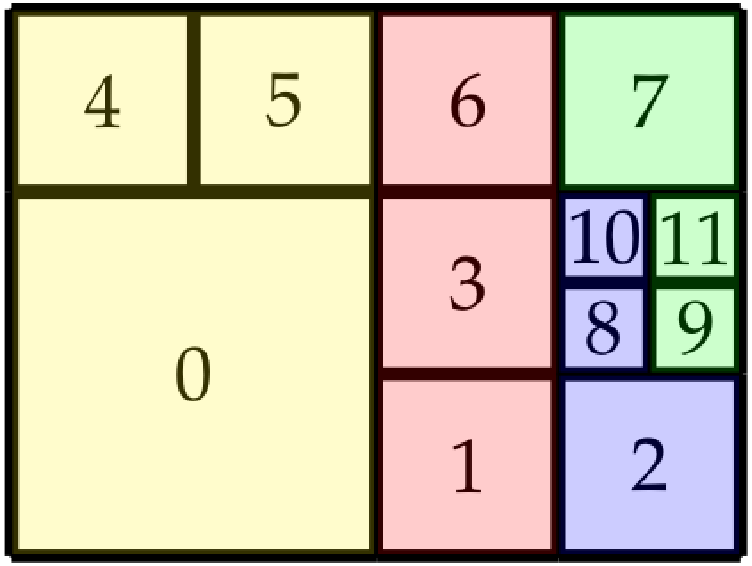
- Flux kernel split into two parts
- Remote data not required by Internal blocks
- Masking MPI communication with kernel execution



•Partizionamento naive1D

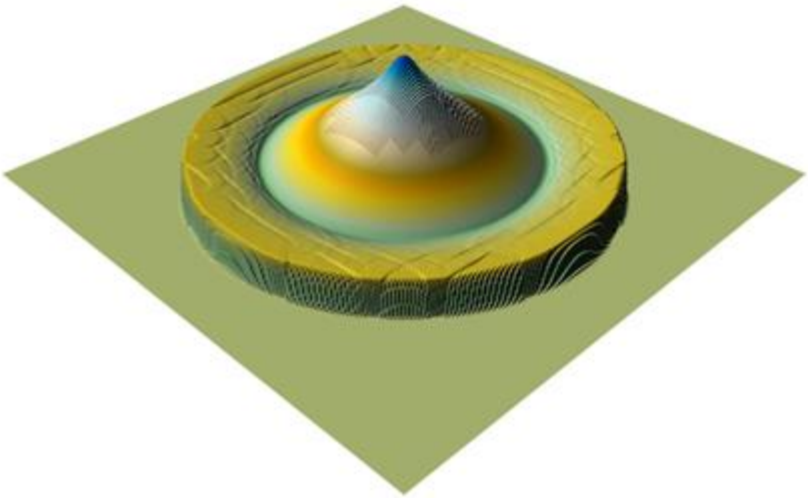
- Utilizzato principalmente per debug e testing. Utile per il confronto con il 2D.

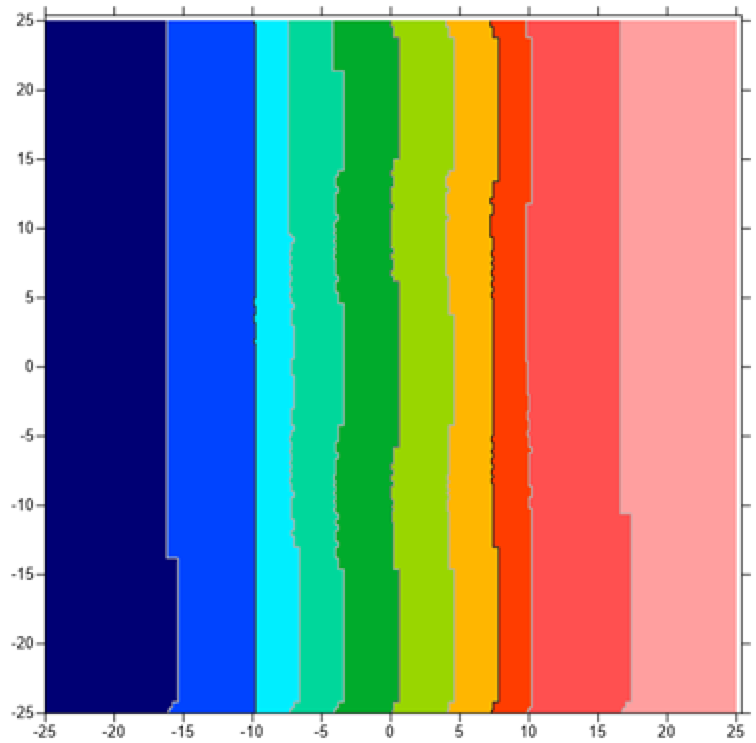
4, 0, 5 1, 3, 6 8, 10, 2 7, 9, 11



•Partizionamento naive1D

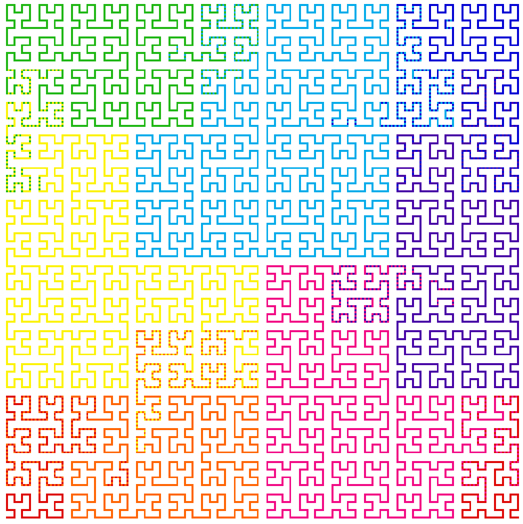
- Svantaggio: aumentando le partizioni la dimensione dei bordi rimane costante

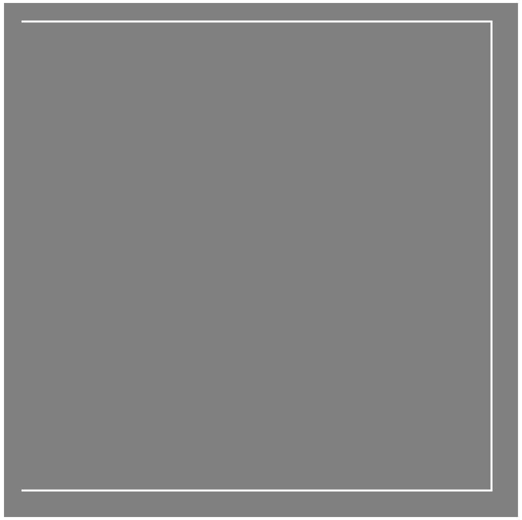




•Hilbert Space Filling Curves (HSFC)

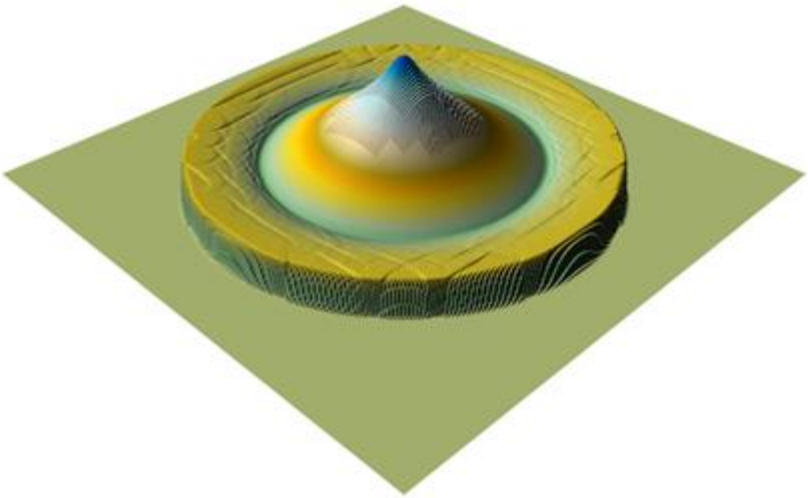
- Punti vicini sulla curva sono vicini anche nello spazio 2D
- Aumentando il numero di partizioni, diminuiscono i bordi
- Integrazione dalle librerie Zoltan

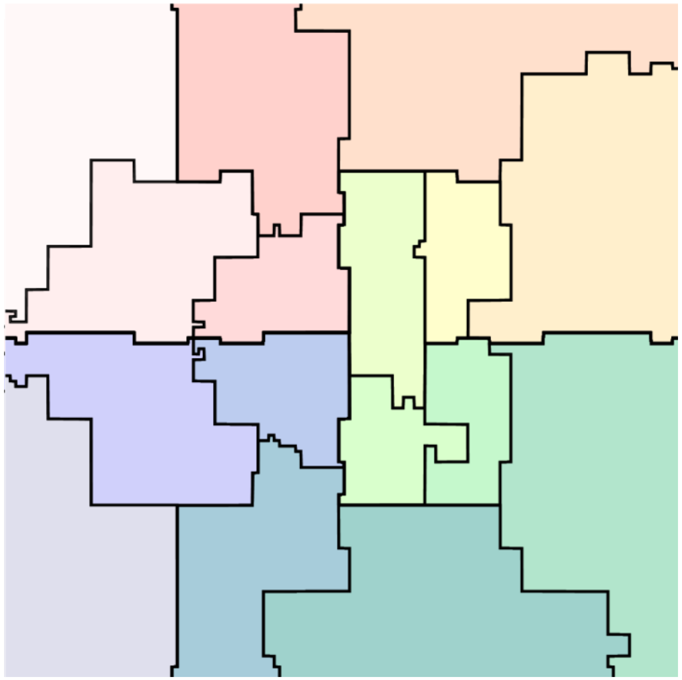




Hilbert Space Filling Curves (HSFC)

- Numero variabile di partizioni vicine





Algoritmo multi-GPU

Algorithm 1 General Simulation Algorithm

Require: $b > 0, T_{max} \geq 0$

$loopCt \leftarrow 1, t \leftarrow 0$
while $t \leq T_{max}$ **do**
    if $(loopCt$ **mod** $b) = 0$ **then** dynLoadBalancing( )
    $\Delta t \leftarrow$ deltaTCalculation( )
    $\Delta t_{min} \leftarrow$ mpiDtReduction(Δ_t)
    $W \leftarrow$cudaMemcpyDeviceToHost( )
    MUSCL(Δt_{min}, InternalBlocks)
    $R \leftarrow$MPIBORDERS(W)
    cudaMemcpyHostToDevice(R)
    MUSCL(Δt_{min}, BorderBlocks, R)
    $W \leftarrow$cudaMemcpyDeviceToHost( )
    Flux(Δt_{min}, InternalBlocks)
    $R \leftarrow$MPIBORDERS(W)
    cudaMemcpyHostToDevice(R)
    Flux(Δt_{min}, borderBlocks, R)
    $W \leftarrow$cudaMemcpyDeviceToHost( )
  ... *Second half step*
    $loopCt \leftarrow loopCt + 1$
    $t \leftarrow t + \Delta t_{min}$

procedure MPIBORDERS(W)
    for all $r \in AdjRanks$ **do**
        mpiIsend(r,W)
        mpiIrecv(r,W_r)
    mpiWaitAll()
    for all $r \in AdjRanks$ **do**
        $processBorders(R, GAM_r, W_r)$
    return R

Algoritmo multi-GPU

Algorithm 1 General Simulation Algorithm

Require: $b > 0, T_{max} \geq 0$

$loopCt \leftarrow 1, t \leftarrow 0$
while $t \leq T_{max}$ **do**
  if $(loopCt$ **mod** $b) = 0$ **then** dynLoadBalancing( )
  $\boxed{\Delta t \leftarrow \text{deltaTCalculation( )}}$
  $\boxed{\Delta t_{min} \leftarrow \text{mpiDtReduction(} \Delta_t \text{)}}$
  $W \leftarrow$cudaMemcpyDeviceToHost( )
  MUSCL(Δt_{min}, InternalBlocks)
  $R \leftarrow$MPIBORDERS(W)
  cudaMemcpyHostToDevice(R)
  MUSCL(Δt_{min}, BorderBlocks, R)
  $W \leftarrow$cudaMemcpyDeviceToHost( )
  Flux(Δt_{min}, InternalBlocks)
  $R \leftarrow$MPIBORDERS(W)
  cudaMemcpyHostToDevice(R)
  Flux(Δt_{min}, borderBlocks, R)
  $W \leftarrow$cudaMemcpyDeviceToHost( )
... *Second half step*
  $loopCt \leftarrow loopCt + 1$
  $t \leftarrow t + \Delta t_{min}$

procedure MPIBORDERS(W)
  for all $r \in AdjRanks$ **do**
    mpiIsend(r,W)
    mpiIrecv(r,W_r)
  mpiWaitAll()
  for all $r \in AdjRanks$ **do**
    $processBorders(R, GAM_r, W_r)$
  return R

Algoritmo multi-GPU

Algorithm 1 General Simulation Algorithm

Require: $b > 0, T_{max} \geq 0$

$loopCt \leftarrow 1, t \leftarrow 0$

while $t \leq T_{max}$ **do**

  if $(loopCt$ **mod** $b) = 0$ **then** dynLoadBalancing( )

  $\Delta t \leftarrow$ deltaTCalculation( )

  $\Delta t_{min} \leftarrow$ mpiDtReduction(Δ_t)

  $W \leftarrow$cudaMemcpyDeviceToHost( )

  MUSCL(Δt_{min}, InternalBlocks)

  $R \leftarrow$MPIBORDERS(W)

  cudaMemcpyHostToDevice(R)

  MUSCL(Δt_{min}, BorderBlocks, R)

  $W \leftarrow$cudaMemcpyDeviceToHost( )

  Flux(Δt_{min}, InternalBlocks)

  $R \leftarrow$MPIBORDERS(W)

  cudaMemcpyHostToDevice(R)

  Flux(Δt_{min}, borderBlocks, R)

  $W \leftarrow$cudaMemcpyDeviceToHost( )

... *Second half step*

  $loopCt \leftarrow loopCt + 1$

  $t \leftarrow t + \Delta t_{min}$

procedure MPIBORDERS(W)

  for all $r \in AdjRanks$ **do**

    mpiIsend(r,W)

    mpiIrecv(r,W_r)

  mpiWaitAll()

  for all $r \in AdjRanks$ **do**

    $processBorders(R, GAM_r, W_r)$

  return R

Algoritmo multi-GPU

Algorithm 1 General Simulation Algorithm

Require: $b > 0, T_{max} \geq 0$

$loopCt \leftarrow 1, t \leftarrow 0$
while $t \leq T_{max}$ **do**
  if ($loopCt$ **mod** b) $= 0$ **then** dynLoadBalancing( )
  $\Delta t \leftarrow$ deltaTCalculation( )
  $\Delta t_{min} \leftarrow$ mpiDtReduction(Δ_t)
  $W \leftarrow$cudaMemcpyDeviceToHost( )
  MUSCL(Δt_{min}, InternalBlocks)
  $R \leftarrow$MPIBORDERS(W)
  cudaMemcpyHostToDevice(R)
  MUSCL(Δt_{min}, BorderBlocks, R)
  $W \leftarrow$cudaMemcpyDeviceToHost( )
  Flux(Δt_{min}, InternalBlocks)
  $R \leftarrow$MPIBORDERS(W)
  cudaMemcpyHostToDevice(R)
  Flux(Δt_{min}, borderBlocks, R)
  $W \leftarrow$cudaMemcpyDeviceToHost( )
... *Second half step*
  $loopCt \leftarrow loopCt + 1$
  $t \leftarrow t + \Delta t_{min}$

procedure MPIBORDERS(W)
  for all $r \in AdjRanks$ **do**
    mpiIsend(r,W)
    mpiIrecv(r,W_r)
  mpiWaitAll()
  for all $r \in AdjRanks$ **do**
    $processBorders(R, GAM_r, W_r)$
  return R

Algoritmo multi-GPU

Algorithm 1 General Simulation Algorithm

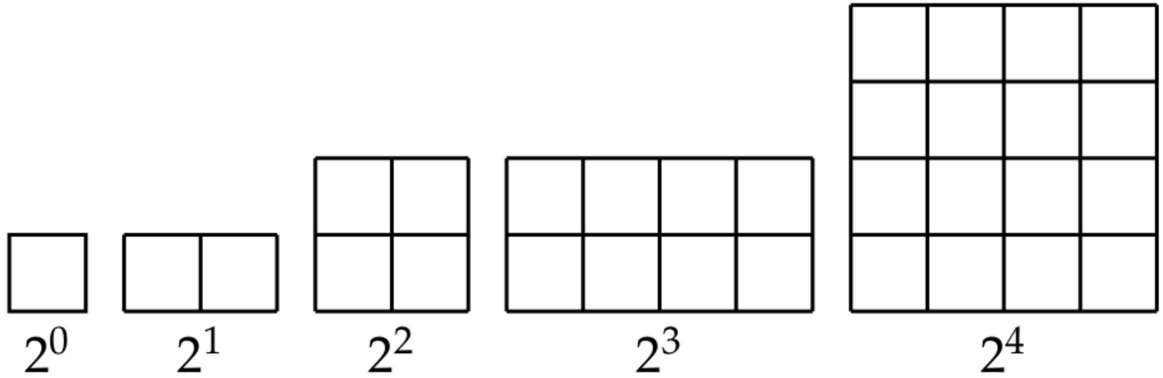
Require: $b > 0, T_{max} \geq 0$

$loopCt \leftarrow 1, t \leftarrow 0$
while $t \leq T_{max}$ **do**
    if $(loopCt$ **mod** $b) = 0$ **then** dynLoadBalancing( )
    $\Delta t \leftarrow$ deltaTCalculation( )
    $\Delta t_{min} \leftarrow$ mpiDtReduction(Δ_t)
    $W \leftarrow$cudaMemcpyDeviceToHost( )
    MUSCL(Δt_{min}, InternalBlocks)
    $R \leftarrow$MPIBORDERS(W)
    cudaMemcpyHostToDevice(R)
    MUSCL(Δt_{min}, BorderBlocks, R)
    $W \leftarrow$cudaMemcpyDeviceToHost( )
    Flux(Δt_{min}, InternalBlocks)
    $R \leftarrow$MPIBORDERS(W)
    cudaMemcpyHostToDevice(R)
    Flux(Δt_{min}, borderBlocks, R)
    $W \leftarrow$cudaMemcpyDeviceToHost( )
... *Second half step*
    $loopCt \leftarrow loopCt + 1$
    $t \leftarrow t + \Delta t_{min}$

procedure MPIBORDERS(W)
    for all $r \in AdjRanks$ **do**
        mpiIsend(r,W)
        mpiIrecv(r,W_r)
    mpiWaitAll()
    for all $r \in AdjRanks$ **do**
        $processBorders(R, GAM_r, W_r)$
    return R

Weak Scalability Test

- Si fissa la dimensione della partizione
- Si generano modelli con un numero crescente di partizioni

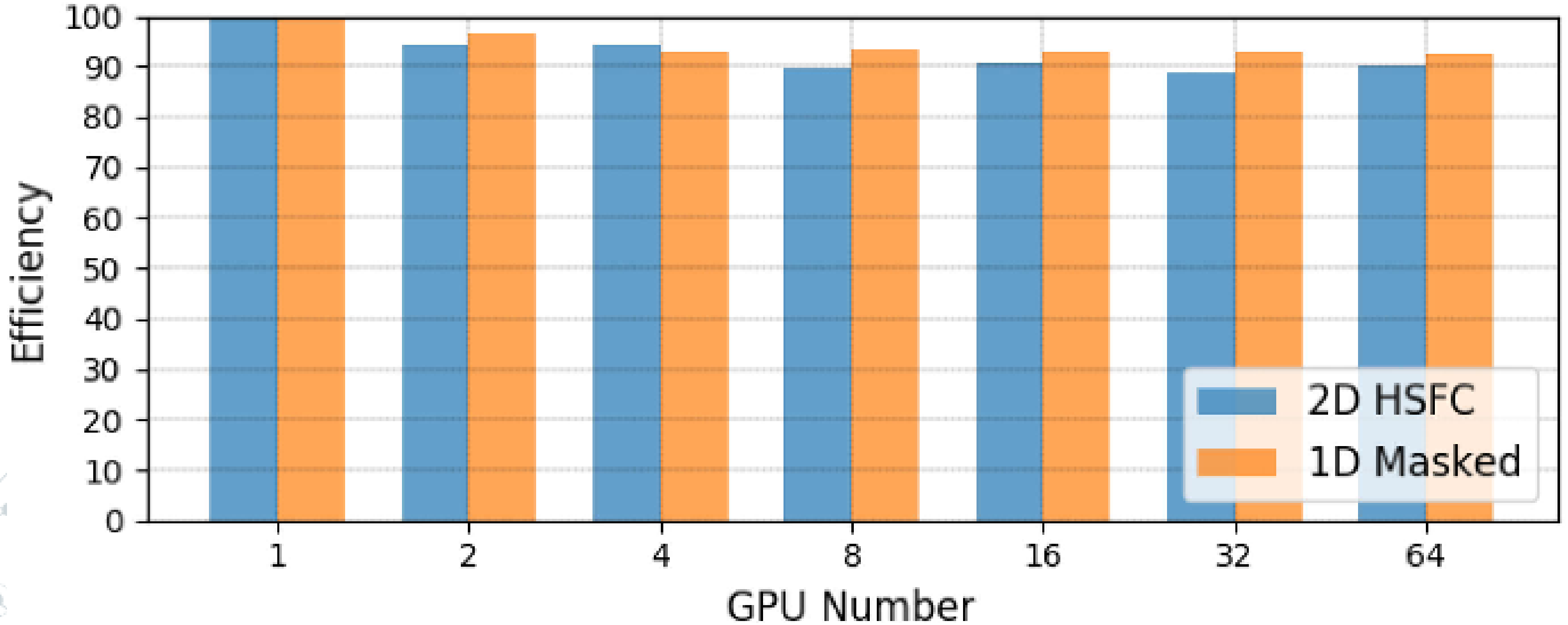


- Il modello con N partizioni viene simulato su N GPU

Weak Scalability Test

- 8 milioni di celle per partizione
- Valutazione dei costi di comunicazione e sincronizzazione dell’algoritmo 2D rispetto all’1D

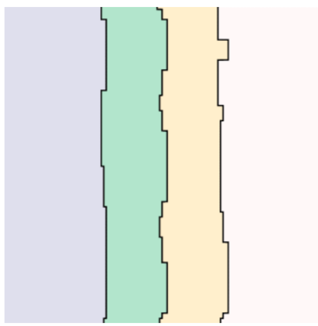
$$Efficiency = T_1/Tn$$



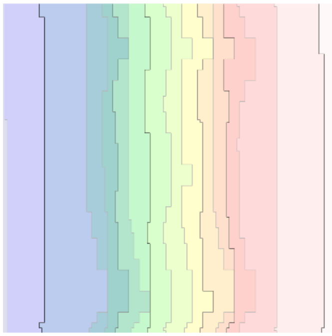
.Strong Scaling Test

- 1D

4 GPU



16 GPU

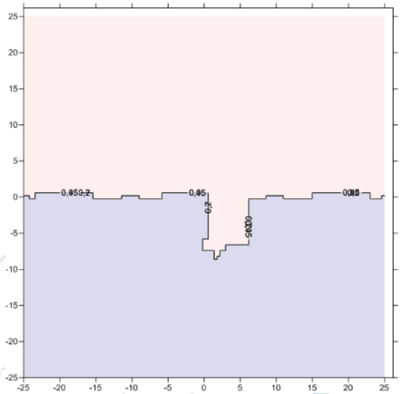


.....

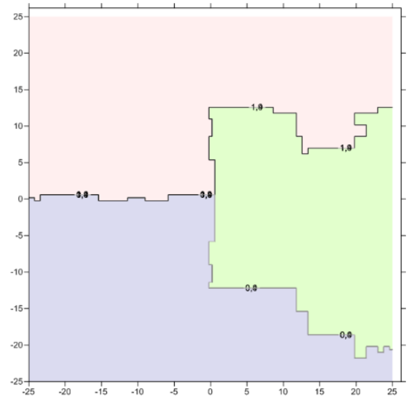
...

- 2D (HSFC)

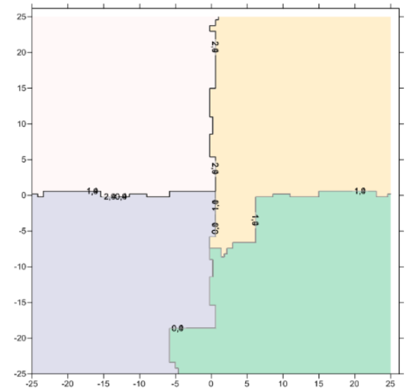
2 GPU



3 GPU



4 GPU

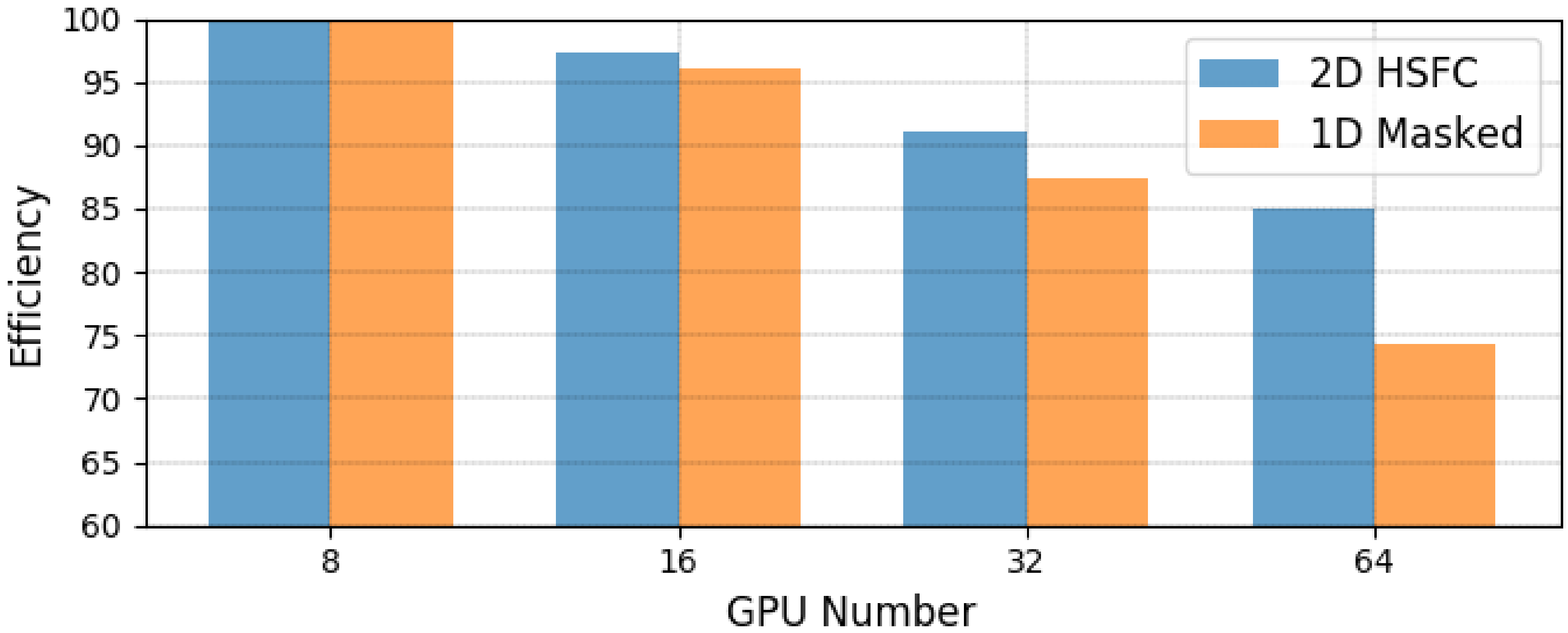


...

Strong Scaling Test

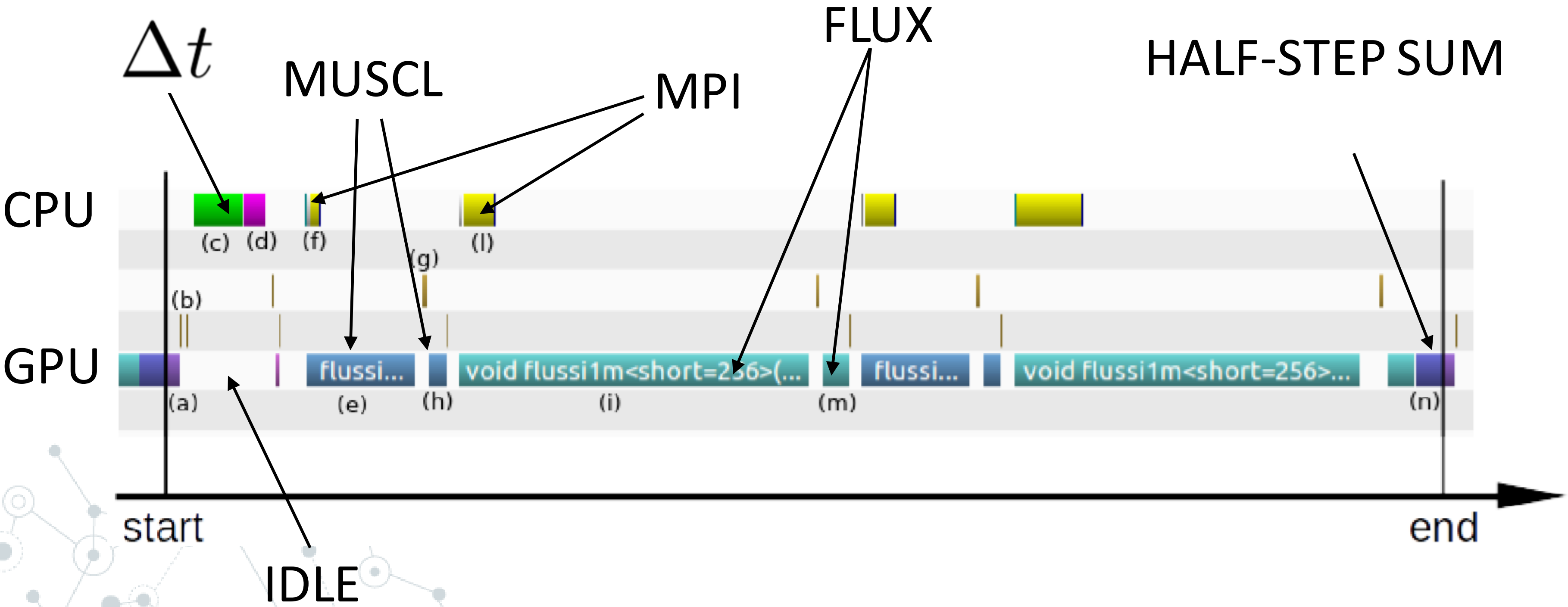
- 85 milioni di celle
- Valutazione benefici sui tempi di calcolo dell’algoritmo 2D rispetto all’1D

$$Efficiency = T_1/(nTn)$$



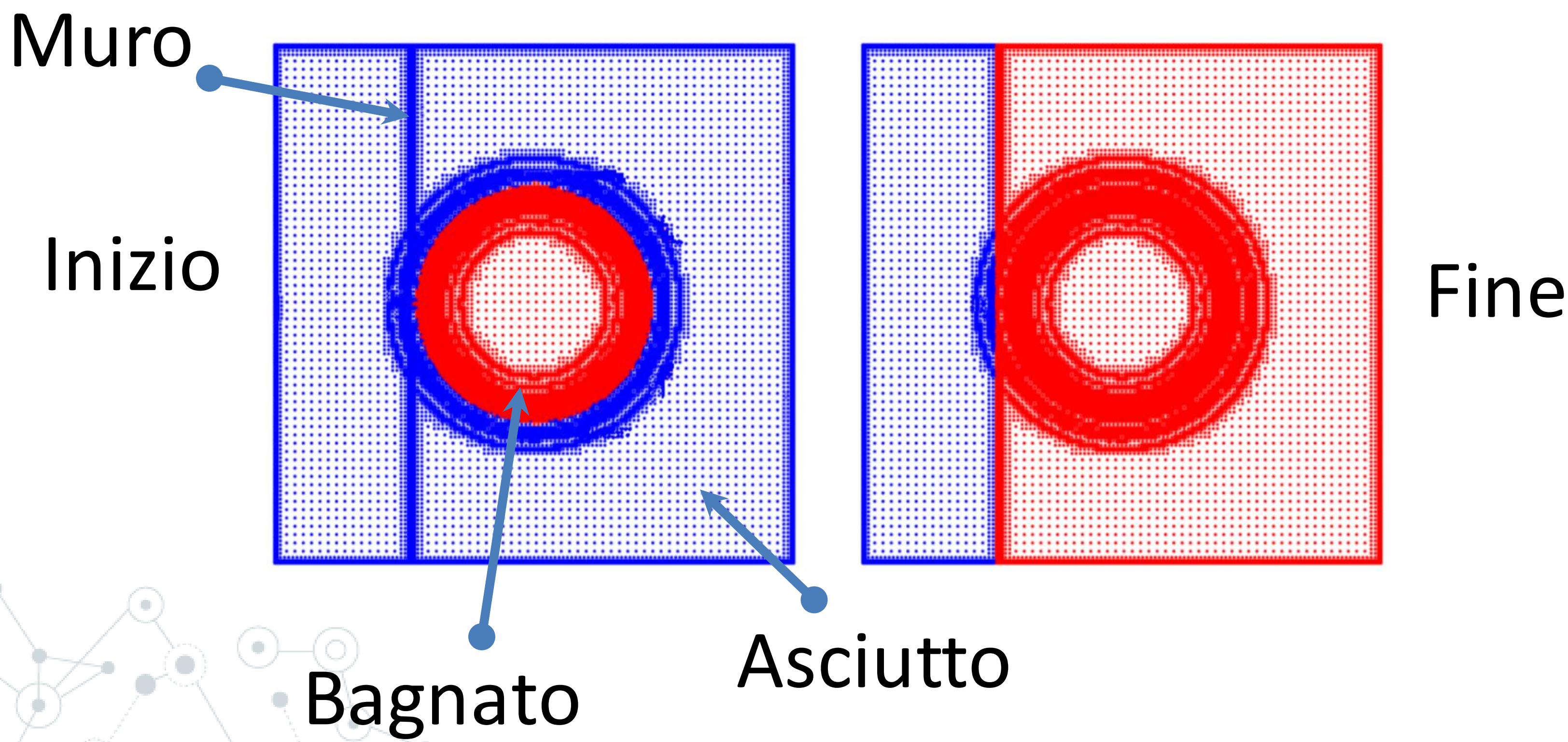
Osserviamo il time-step più in dettaglio…

- Verifica di mascheramento MPI
- Dati ottenuti con nvprof, nvtx e nvvp

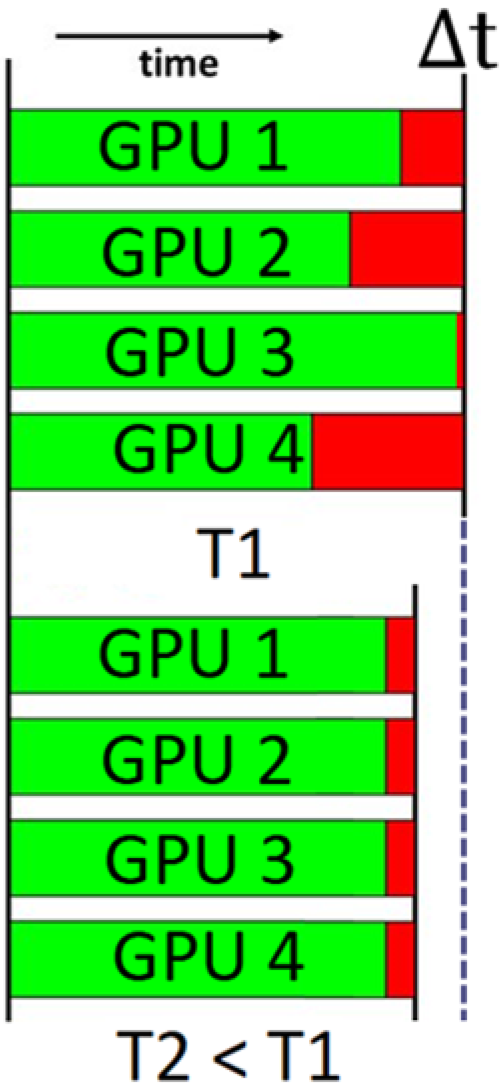


•Fronti dry-wet e bilanciamento del carico

- Numero variabile di celle bagnate
- Il tempo di calcolo di una partizione dipende dal numero di celle bagnate

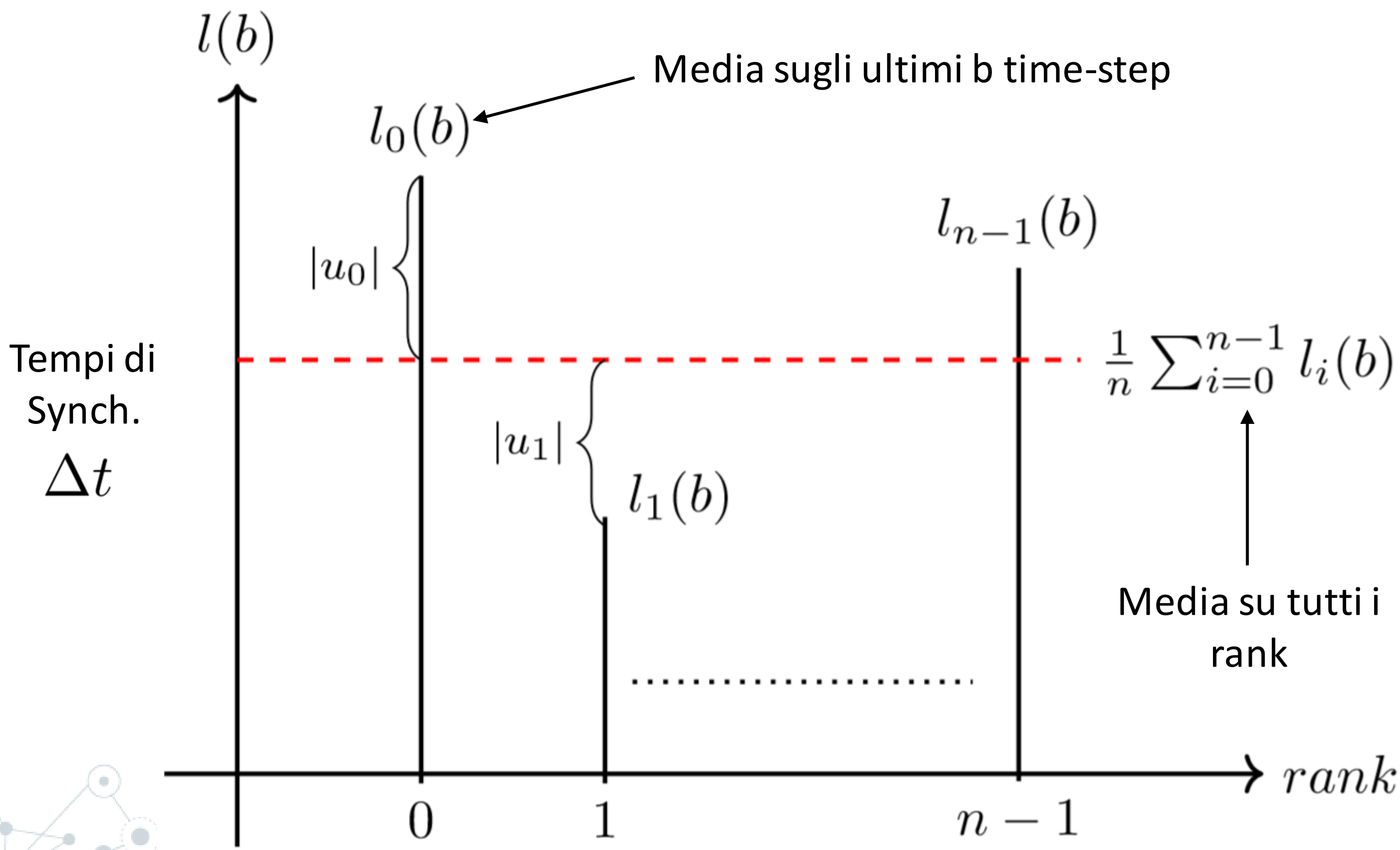


•Fronti dry-wet e bilanciamento del carico



- La sincronizzazione del delta t causa tempi di inattività per alcune GPU
- L’idea è di redistribuire il carico minimizzando il tempi di sincronizzazione

•Fronti dry-wet e bilanciamento del carico



- Euristica di bilanciamento del carico

$$\boxed{\sum_{i=0}^{n-1} w_i = 1, \quad w \in [0,1]}$$

$$u_p := l_p(b) - \frac{1}{n} \sum_{i=0}^{n-1} l_i(b)$$

$$z_p := \frac{u_p}{max_{0 \leq i \leq n-1}\{L(b)\}}, \quad z_p \in [-1,1]$$

$$w'_p := w_p + z_p \varepsilon, \quad 0 \leq \varepsilon < 1, \; w'_p \in [0,1]$$

$$\sum_{i=0}^{n-1} w'_i = 1$$

Euristica di bilanciamento del carico

$$\sum_{i=0}^{n-1} w_i = 1, \quad w \in [0,1]$$

tempo di sincronizz. Del proc. *p*

$$u_p := \boxed{l_p(b)} - \frac{1}{n}\sum_{i=0}^{n-1} l_i(b)$$

$$z_p := \frac{u_p}{max_{0 \leq i \leq n-1}\{L(b)\}}, \quad z_p \in [-1,1]$$

$$w'_p := w_p + z_p\varepsilon, \quad 0 \leq \varepsilon < 1, \; w'_p \in [0,1]$$

$$\sum_{i=0}^{n-1} w'_i = 1$$

Euristica di bilanciamento del carico

$$\sum_{i=0}^{n-1} w_i = 1, \quad w \in [0,1]$$

$$u_p := l_p(b) - \frac{1}{n}\sum_{i=0}^{n-1} l_i(b)$$

$$\boxed{z_p := \frac{u_p}{max_{0 \le i \le n-1}\{L(b)\}}, \quad z_p \in [-1,1]}$$

Tempo tot. del ciclo del proc. *p*

$$w'_p := w_p + z_p \varepsilon, \quad 0 \le \varepsilon < 1, \; w'_p \in [0,1]$$

$$\sum_{i=0}^{n-1} w'_i = 1$$

Euristica di bilanciamento del carico

$$\sum_{i=0}^{n-1} w_i \; = \; 1, \quad w \; \in \; [0,1]$$

$$u_p := l_p(b) - \frac{1}{n}\sum_{i=0}^{n-1} l_i(b)$$

$$z_p := \frac{u_p}{max_{0\leq i\leq n-1}\{L(b)\}}, \quad z_p \in [-1,1]$$

$$\boxed{w'_p := w_p + z_p\varepsilon, \quad 0 \leq \varepsilon < 1, \; w'_p \in \; [0,1]}$$

$$\sum_{i=0}^{n-1} w'_i \; = \; 1$$

Euristica di bilanciamento del carico

$$\sum_{i=0}^{n-1} w_i = 1, \quad w \in [0,1]$$

$$u_p := l_p(b) - \frac{1}{n}\sum_{i=0}^{n-1} l_i(b)$$

$$z_p := \frac{u_p}{max_{0 \le i \le n-1}\{L(b)\}}, \quad z_p \in [-1,1]$$

$$w'_p := w_p + z_p\varepsilon, \quad 0 \le \varepsilon < 1, \; w'_p \in [0,1]$$

$$\boxed{\sum_{i=0}^{n-1} w'_i = 1}$$

•Output da Zoltan

- I pesi $w_1,..,w_n$ vengono dati in input a Zoltan il quale ritorna le informazioni su come spostare il carico

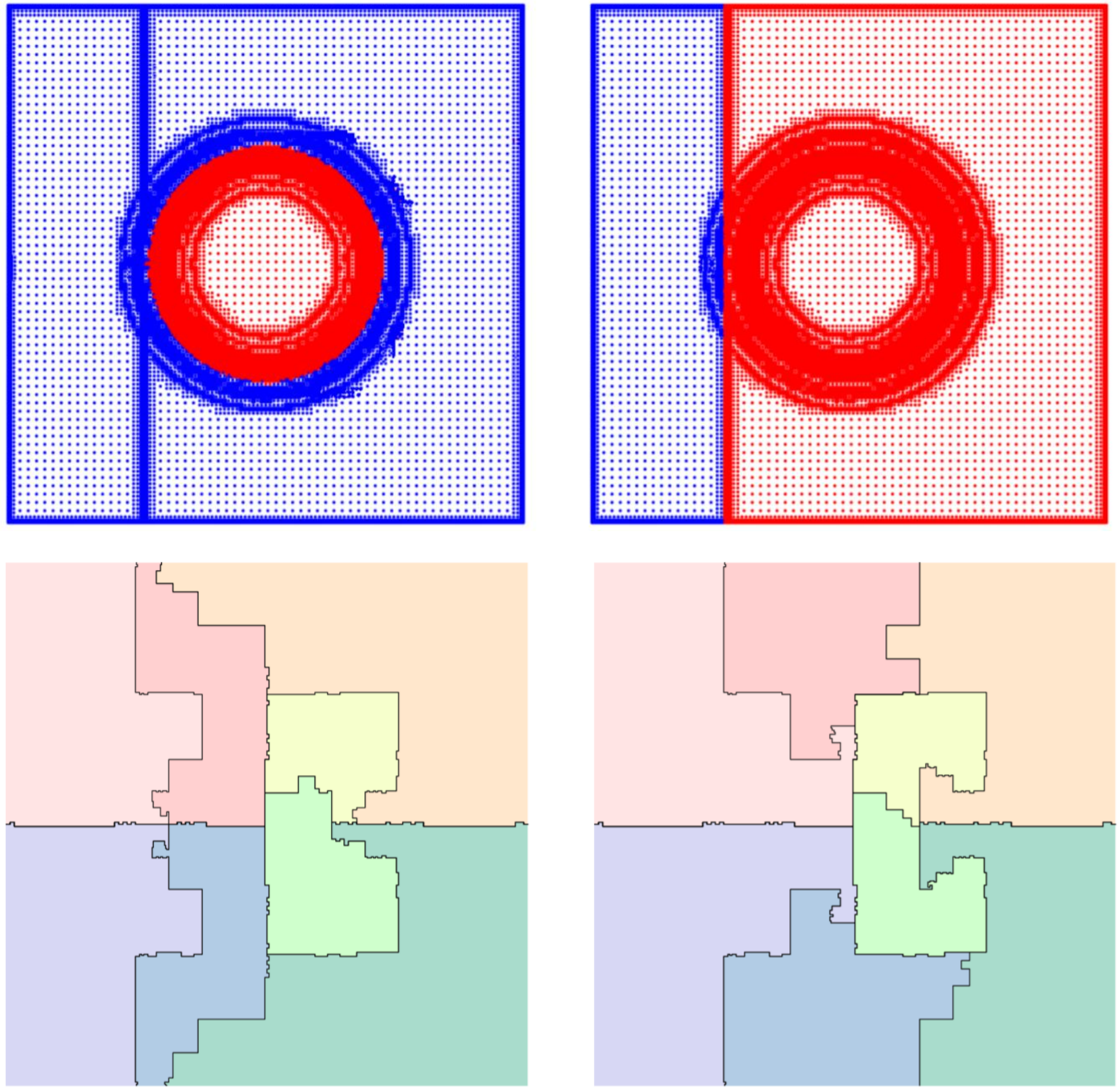
```
struct PartitionChanges{

  int changes;
  int num_import;
  int num_export;

  ZOLTAN_ID_PTR import_global_ids;
  ZOLTAN_ID_PTR import_local_ids;
  ZOLTAN_ID_PTR export_global_ids;
  ZOLTAN_ID_PTR export_local_ids;

};
```

Blocchi importati

Blocchi esportati

Euristica di bilanciamento del Carico

Inizio

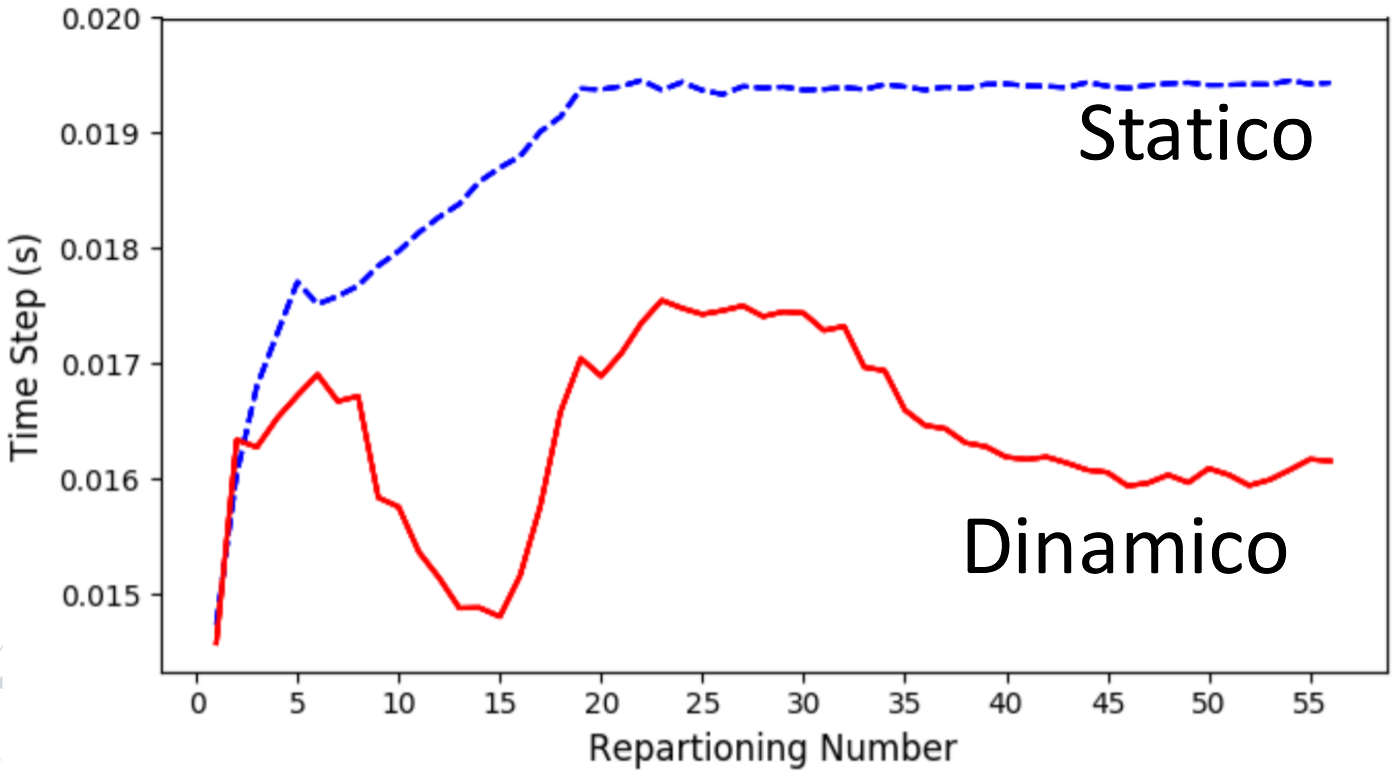


Fine

Risultati (DLB)

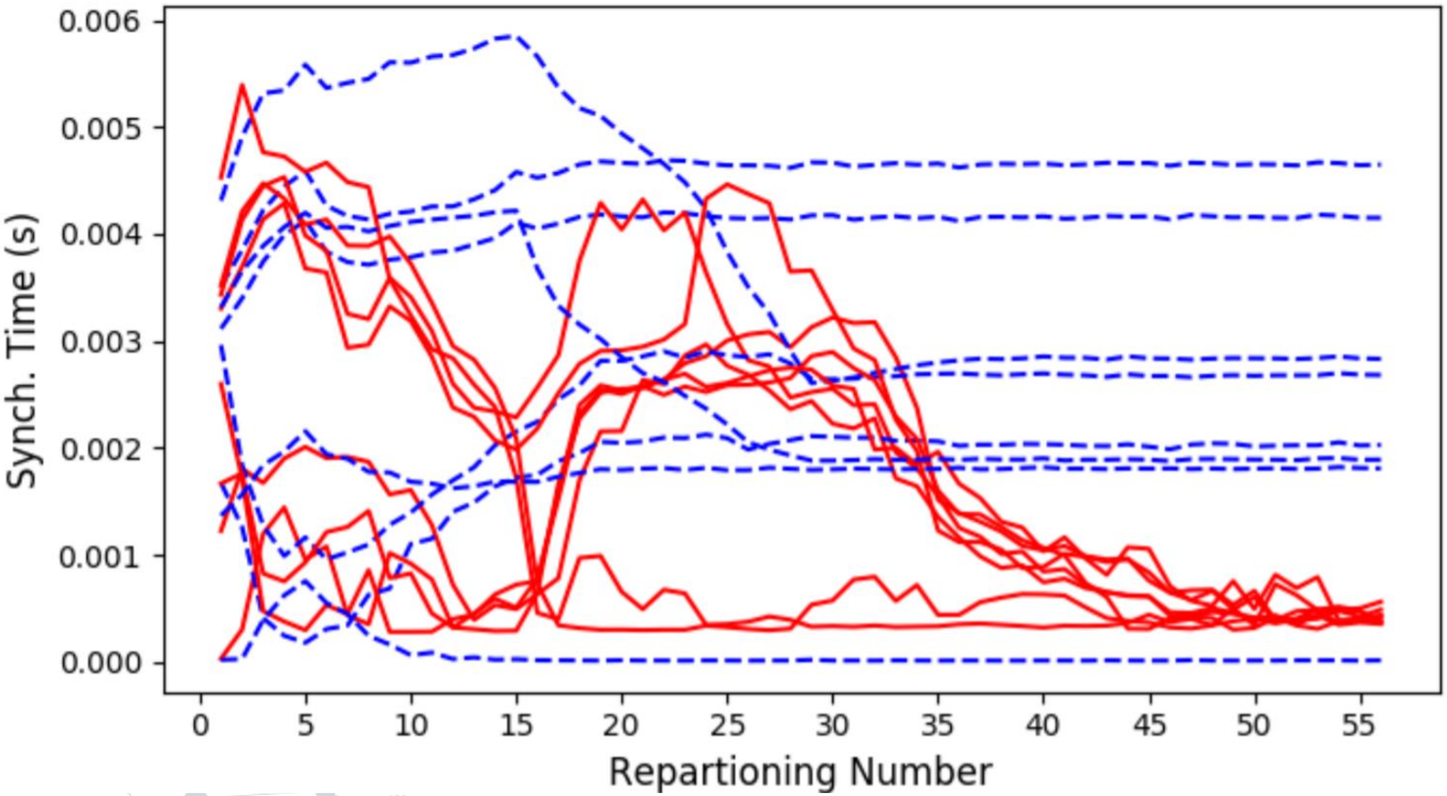
- Convergenza nel caso dinamico
- Test eseguito su Dam-Break circolare,

  24 M celle, 8 GPU



Risultati (DLB)

- Convergenza a un carico bilanciato nel caso dinamico (Tempi di synch.)
- Test eseguito su Dam-Break circolare, 24 M celle, 8 GPU



Conclusioni

- Punto di partenza: solver SWE 1-GPU
- Punto di arrivo: solver multi-processo (MPI)

  multi-GPU

Principali Risultati ottenuti:

- Weak Scaling 90% (HSFC) 64 GPU
- Strong Scaling 85% (HSFC) 64 GPU
- DLB convergente

I modelli presi in considerazione sono sintetici.

Conclusioni

Numero GPU	N. Celle (Milioni)	Tempo (Ore)	Superficie (Km^2)
1	10	5 - 20	10·000
10	10	**<1 - 3**	10·000

Numero GPU	N. Celle (Milioni)	Superficie (Km^2)
64	**500**	**$O(100 \cdot 10^3)$**

Conclusions

- BUQ grids are a good compromise between Uniform Cartesian Grids and Unstructured Grids
- Border communication can be masked by kernel execution
- 2D partitioning outperforms 1D partitioning
- Dynamic load balancing outperforms static partitioning