Parallel Sparse Linear Solver GMRES for GPU Clusters with Compression of Exchanged Data

J. M. Bahi, R. Couturier, L. Ziane Khodja

Laboratoire d'Informatique de l'Université de Franche-Comté (LIFC)
IUT Belfort-Montbéliard, France

9th International Workshop on Algorithms, Models and Tools for Parallel Computing on Heterogeneous Platforms
Bordeaux, France

August 29th 2011

Our objectives

- Before:
  - Solving large sparse linear systems
  - Parallel GMRES solver on GPU clusters
  - Sparse banded matrices
- After:
  - All types of sparse matrices (large bandwidths!)
  - Parallel sparse matrix-vector products on a GPU cluster:
    - Pb: overheads in CPU/CPU and GPU/CPU communications
    - Solution: data compression of shared vectors

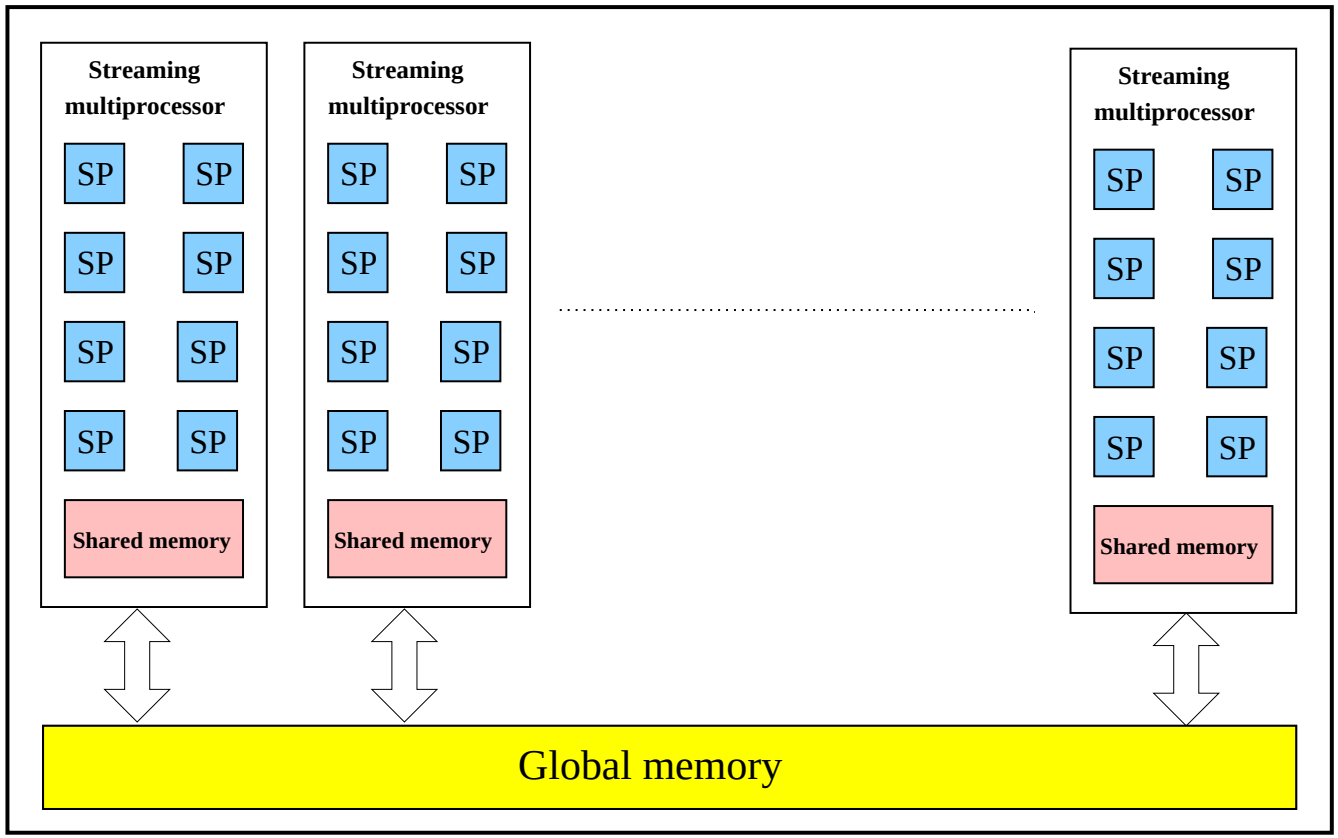
Outline

- **GPU clusters**
- **GMRES solver and its improvements**
- **Experimental results**
- **Conclusion and future work**

GPU clusters

Graphic Processing Unit: GPU

- Initially designed for the 3D visualization in real-time
- Today, high performance accelerator for data-parallel and intensive tasks



GPGPU programming
CUDA (Compute Unified Device Architecture programming)

- Nvidia CUDA programming environment: extension of C language
- GPU is viewed as a co-processor to the CPU
- Kernels: data-parallel and data-intensive functions of an application
- In CUDA program:
  - Host (CPU):
    - Controls the application execution
    - Executes the sequential code written in C
    - Launches all kernels on the GPU
  - GPU:
    - Executes the kernels
    - Grid of thread blocks: SIMT platform
    - Returns the final results of a kernel execution to the CPU

GPU cluster architecture

- To exploit simultaneously: the memory capability & the high performance computing of several GPUs

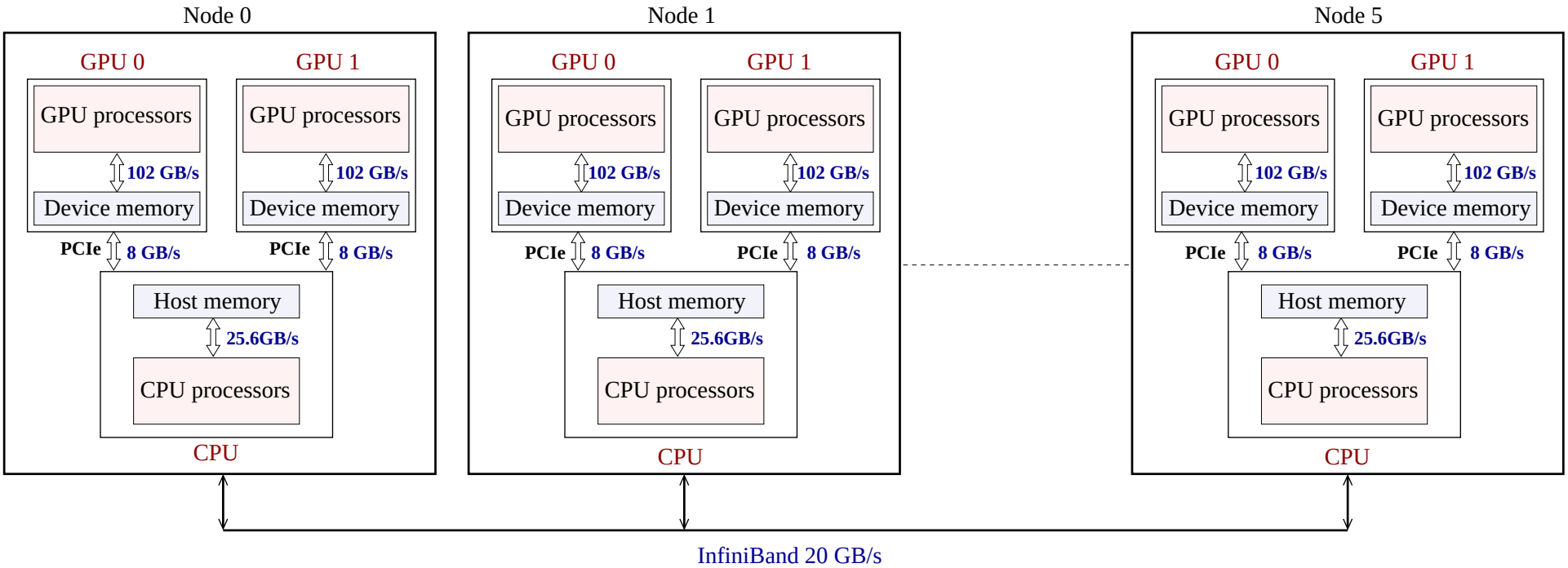


Figure: GPU cluster of IUT Belfort-Montbéliard

GMRES solver and its improvements

GMRES method

Generalized Minimal RESidual solver

- Iterative method developped by Saad & Schultz in 1986
- Generalization of the MINRES method
- Generic and efficient method to solve:
  - nonsymmetric & non-Hermitian problems
  - definite & indefinite symmetric problems

Sequential GMRES algorithm with restarts

1: ***Start***: Choose x_0 and compute: $r_0 = M^{-1}(b - Ax_0)$ and $v_1 = r_0/\|r_0\|$

2: ***Iterate***: Arnoldi process

For j=1,2,. . . ,m do:

- $h_{i,j} = (M^{-1}Av_j, v_i)$, i=1,2,. . . ,j
- $w_j = M^{-1}Av_j - \sum_{i=1}^{j} h_{i,j}v_i$
- $h_{j+1,j} = \|w_j\|$
- $v_{j+1} = w_j/\|w_j\|$

3: *From the approximate solution*: $x_m = x_0 + V_m y_m$

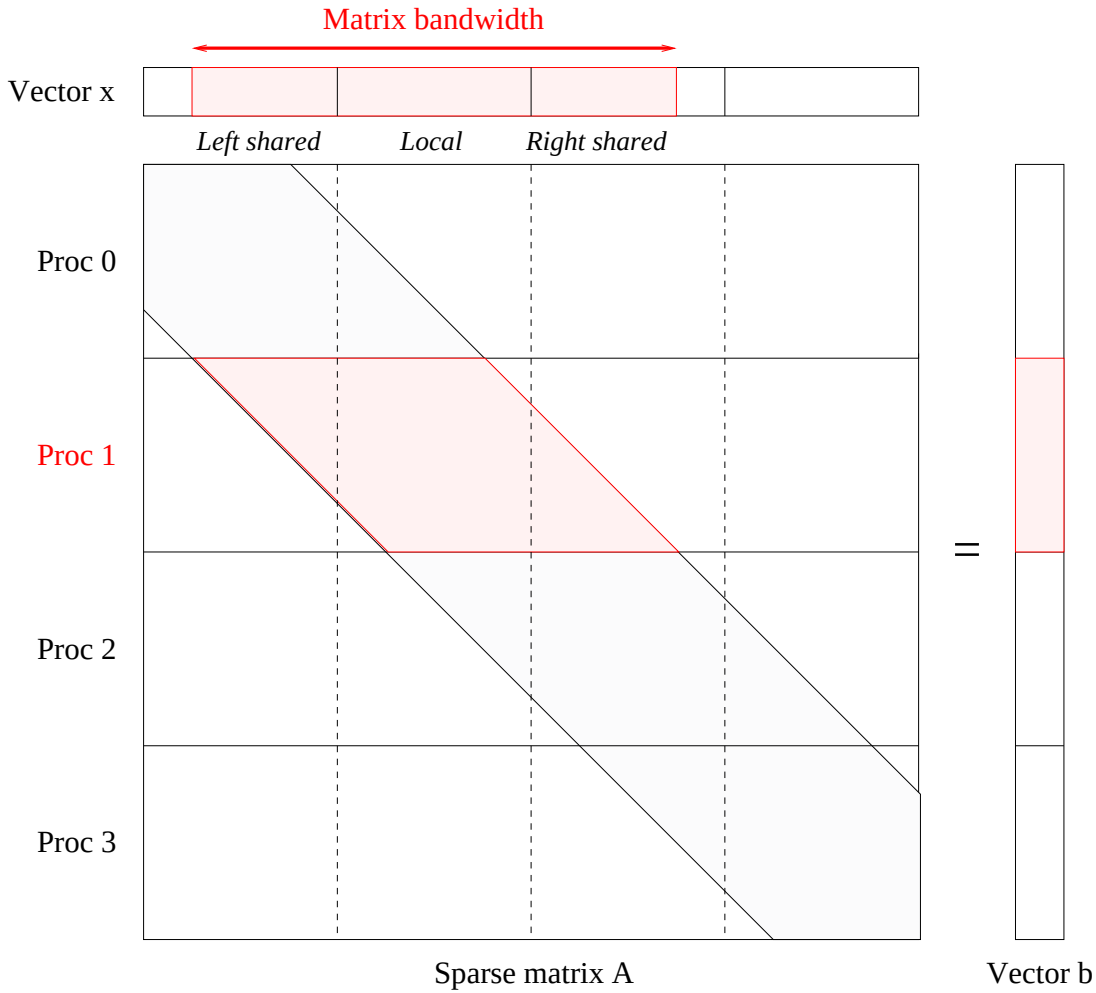
where y_m minimizes $\|\beta e_1 - \bar{H}_m y\|$, such that: $\beta = \|r_0\|$ and $y \in R^m$

4: ***Restart***:

- Compute $r_m = M^{-1}(b - Ax_m)$
- if satisfied then stop
- else compute $x_0 := x_m$, $v_1 := r_m/\|r_m\|$ and go to 2

GMRES parallelization on a GPU cluster

- Partitioning of $Ax = b$ into p sub sparse linear systems: $A_k x_k = b_k$, $k = 1, 2, \ldots, p$ (number of GPUs on the cluster)



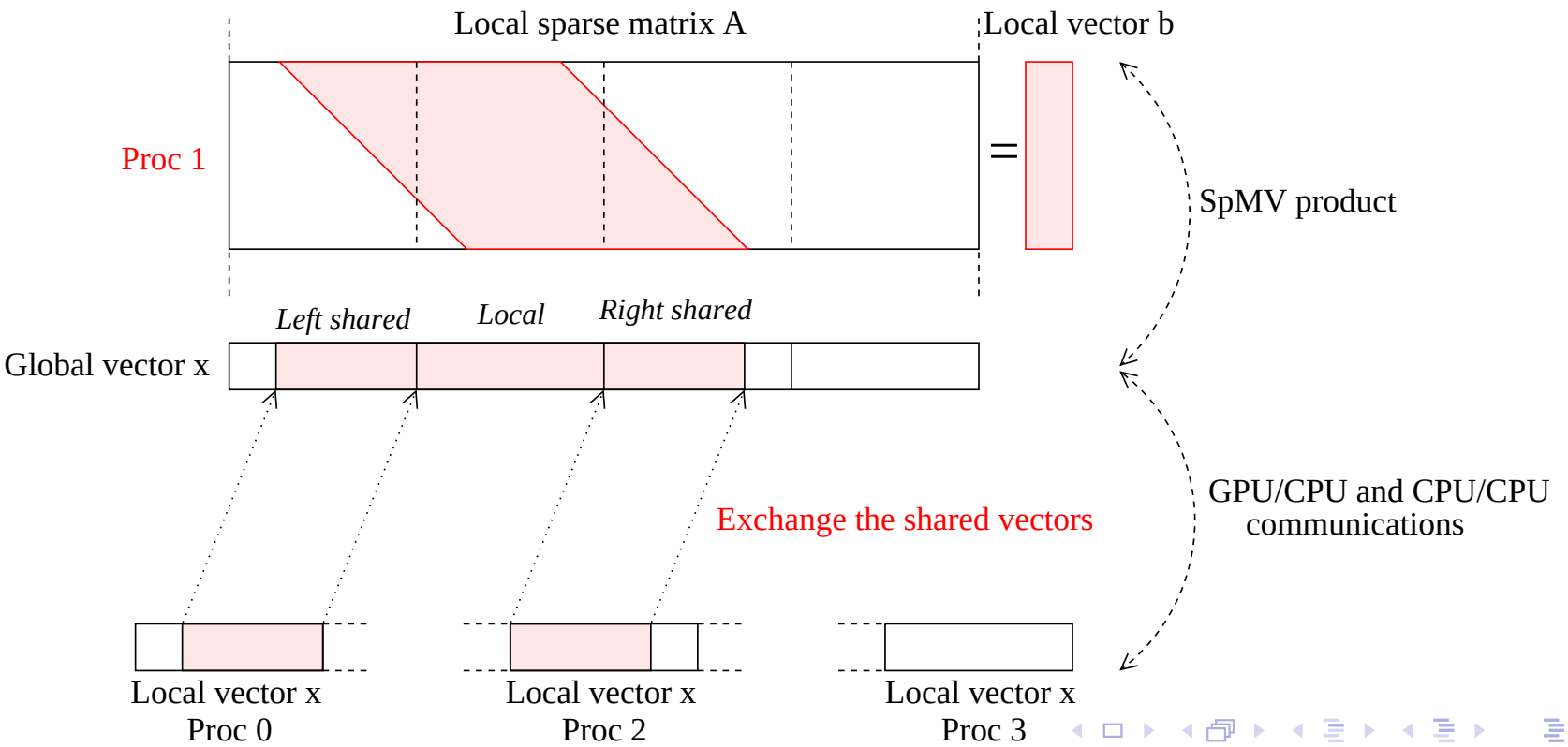
GMRES parallelization on a GPU cluster

- Accelerating the mathematical operations on the GPUs:
  - Sparse matrix-vector product (SpMV)
  - Dot product, Euclidean norm, AXPY operation
  - Other kernels: scalar-vector product, solving least-squares problem, solution vector updates
- Synchronizations over the GPU cluster by the CPUs:
  - Reduction operations: MPI_Allreduce()
  - Parallel SpMV: exchanging shared vectors of unknowns

The most expensive step of the parallel GMRES algorithm

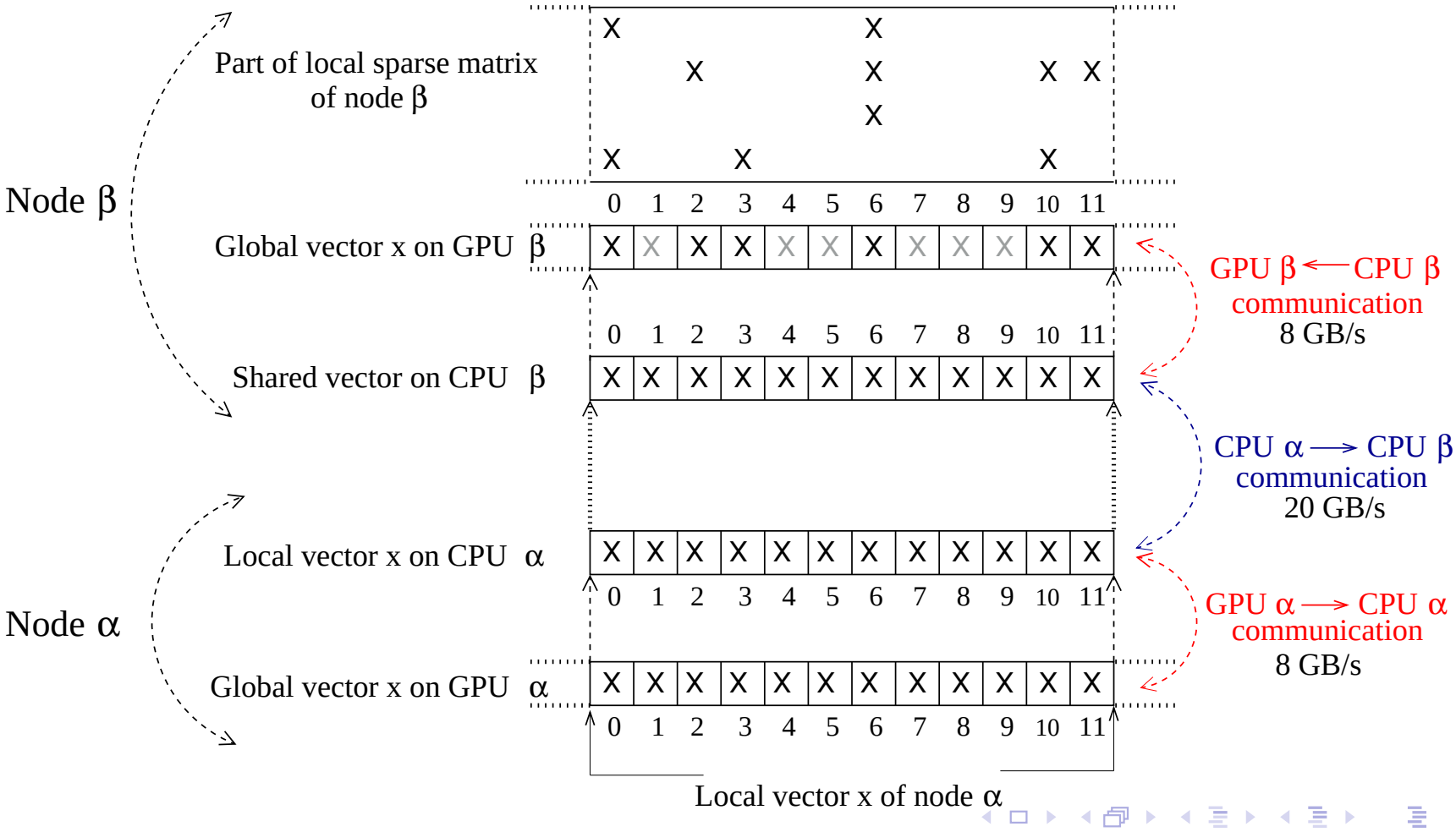
- Parallel sparse matrix-vector product (SpMV) on a GPU cluster:

```
1: gpu_to_cpu();      /* GPU→CPU comm. (8 GB/s) */
2: MPI_Alltoallv();   /* CPU↔CPU comm. (20 GB/s) */
3: cpu_to_gpu();      /* GPU←CPU comm. (8 GB/s) */
4: SpMV();
```



Slow data exchanges of the shared vectors

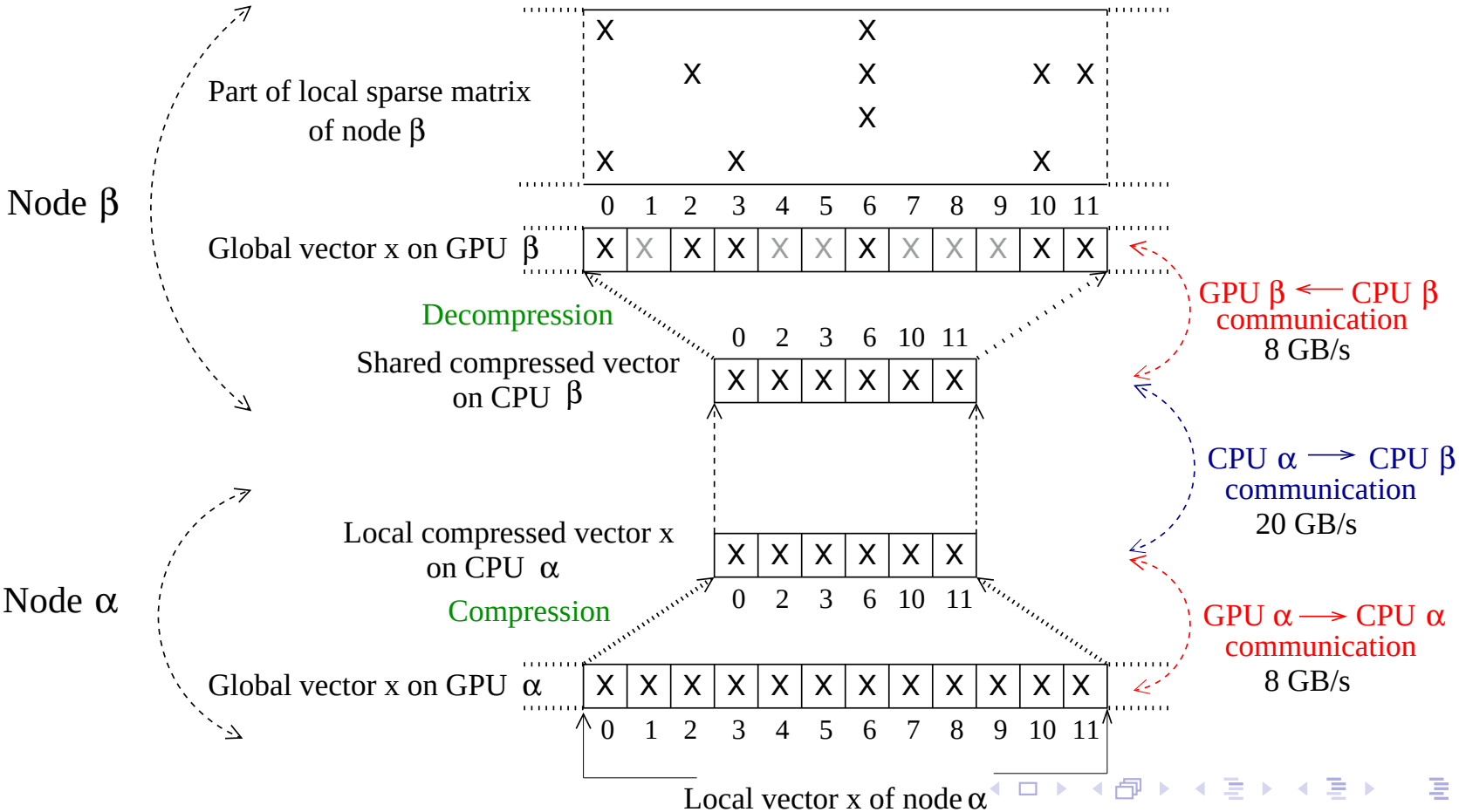
- Parallel SpMV does not need all unknown values $\Rightarrow$ loss of performance!
  - loss of performance = 6 elements $\times$ (2 $\cdot$ throughput of GPU/CPU comm. + throughput of CPU/CPU comm.)



Enhanced data exchanges of the shared vectors

- Solution:
  - Compression of the shared vectors before the sending
  - Decompression of the shared vectors after the reception

⇒ Minimize the communication overheads of solving large sparse systems



Enhanced parallel SpMV on a GPU cluster

- Accelerating the *compression* and *decompression* functions on GPUs

- Parallel SpMV algorithm:

```
1: compression();     /*kernel on GPUs*/
2: gpu_to_cpu();
3: MPI_Alltoallv(); /*done on CPUs*/
4: cpu_to_gpu();
5: decompression(); /*kernel on GPUs*/
6: SpMV();            /*kernel on GPUs*/
```

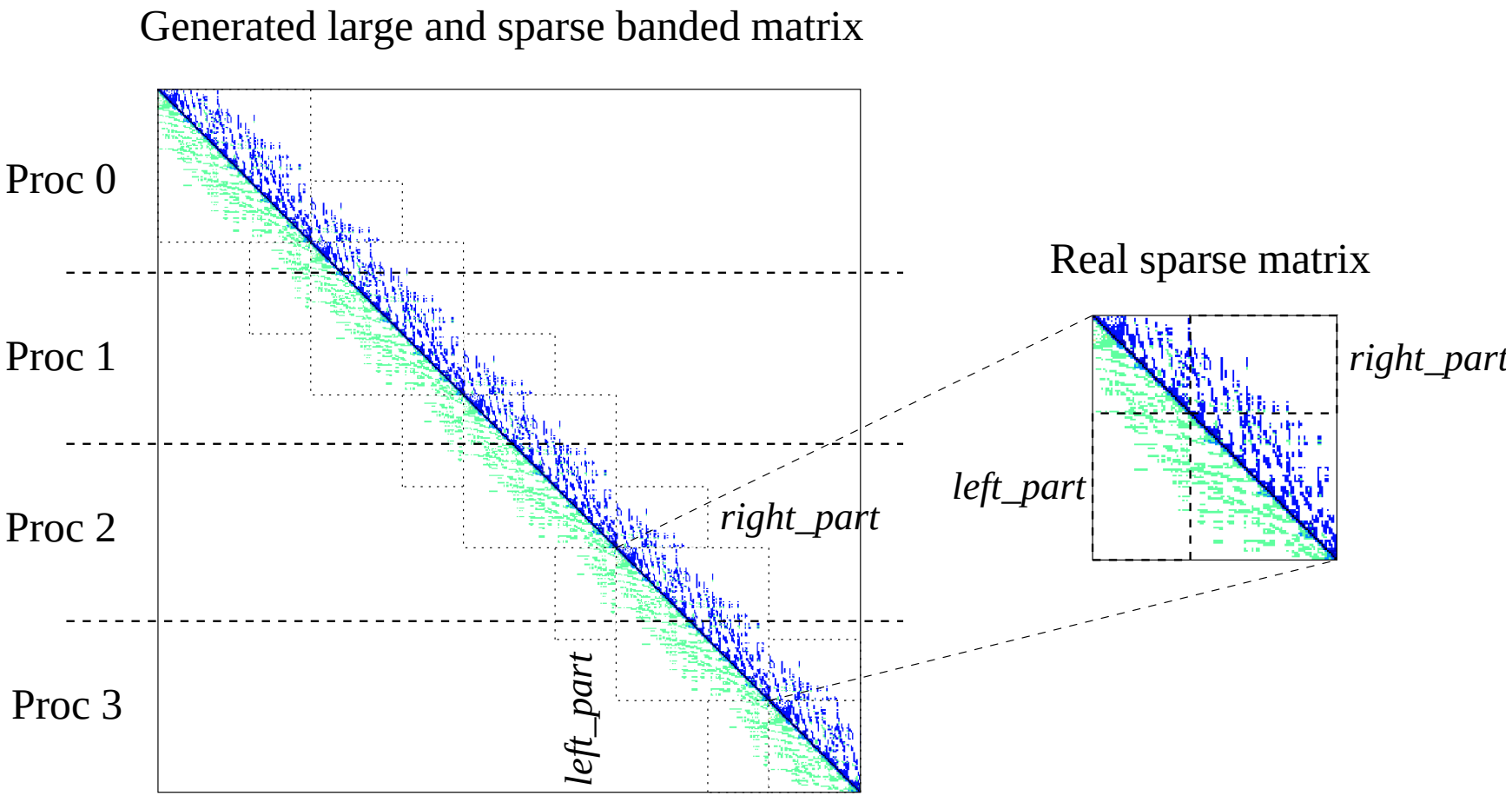
Experimental results

Our GPU cluster

- GPU cluster:
  - InfiniBand
  - Six Quad-Core Xeon E5530
  - Two Tesla C1060 GPUs per CPU
  - $\rightarrow$ Cluster of 12 GPUs
- Performance measure: speed-ups $\frac{T_{CPU}}{T_{GPU}}$ of the cluster of 12 GPUs compared to:
  - Cluster of 12 CPU cores
  - Cluster of 24 CPU cores
- Comparison between the two versions of GMRES solver (*without* and *with* compression/decompression operations)

Sparse matrices of tests

- Real-world sparse matrices of the Davis' collection
- Automatic generation of large sparse matrices



Sparse matrices of the experimental tests

- Sparse matrices generated from those of the Davis's collection
- Size of the test sparse matrices: 90 million

Matrix	Nb. Nonzeros	Bandwidth
ecology2	449,729,174	1,002
stomach	1,277,498,438	22,868
shallow_water2	360,751,026	23,212
language	276,894,366	398,626
G3_circuit	443,429,071	525,429
cage14	1,674,718,790	1,266,626
thermal2	643,458,527	1,928,223

Speed-ups of GMRES solver *without* and *with* data compression

- Precision on GPU cluster: 5.75e-8 to 4.27e-15
- Difference GPU/CPU solutions: 3.78e-10 to 1.35e-18

Matrix	vs. 12 CPUs		vs. 24 CPUs	
	- comp.	+ comp.	- comp.	+ comp.
ecology2	8.46	13.33	5.86	9.18
stomach	8.53	10.66	5.88	7.35
shallow_water2	9.83	12.89	6.49	8.92
language	8.81	11.93	5.89	8.05
G3_circuit	8.02	12.37	5.50	8.53
cage14	6.93	9.11	5.30	6.21
thermal2	6.56	10.70	4.34	7.05

Conclusion & future work

- Conclusion:
  - Both versions of GMRES are faster on GPU clusters
  - GMRES with data compression is more efficient
  - Data compression minimizes: GPU/CPU & CPU/CPU communication overheads

- Future work:
  - Other structures: matrices with large bandwidths
  - Data partitioning: minimizing the communication volume

Thank you for your attention!