A proposal for the implementation of discontinuous Galerkin methods for elliptic problems on GPUs

Paolo Gorlani - Mathematical Engineering MSc - Politecnico di Milano
paolo.gorlani@mail.polimi.it

Introduction

The implementation of Finite Element methods on GPUs encounters two critical points:

- the assembly of the global stiffness matrix, which implies non-homogeneous memory accesses to data locations assigned to the degrees of freedom shared by different elements;
- the storage and the algebraic operations (such as the matrix-vector product) of a *sparse* stiffness matrix.

Both points make it difficult to fully exploit the memory bandwidth of GPUs because memory operations do not coalesce.

Our proposal, which can be applied to discontinuous Galerkin methods, maximizes the coalescence of the global memory accesses and completely avoids the thread divergence.

⋆ ⋆ ⋆

Mathematical problem

We consider the following boundary value problem

Problem 1. *Given a limited domain $\Omega \subset \mathbb{R}^2$; the functions $\lambda, f : \Omega \to \mathbb{R}$ and $g : \partial\Omega \to \mathbb{R}$; find the solution $u : \Omega \to \mathbb{R}$ that satisfies:*

$$\begin{aligned} -\Delta u + \lambda u &= f \quad \textit{in } \Omega \\ u &= g \quad \textit{in } \partial\Omega \end{aligned}$$

Jump and average operators

The set $\mathcal{F}_h$ contains the face of elements belonging to the mesh $\mathcal{K}_h$. Specifically,

$$\mathcal{F}_h = \mathcal{F}_h^i \cup \mathcal{F}_h^b \quad ,$$

where $\mathcal{F}_h^i$ collects the interfaces between elements and $\mathcal{F}_h^b$ collects the faces included in $\partial\Omega$.

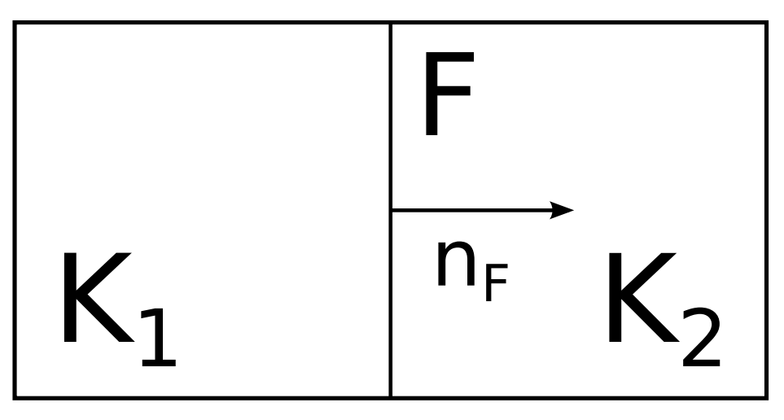

Figure 1

Let $K_1, K_2 \in \mathcal{K}_h$ such that $F = K_1 \cap K_2 \in \mathcal{F}_h^i$. The outward normal vector $\mathbf{n}_F$ is oriented form K_1 to K_2.

Let $v : \Omega \to \mathbb{R}$ be a regular enough function which admits a two-valued trace on $\mathcal{F}_h^i$. The averages $\{\{\cdot\}\}$ and the jumps $[\![\cdot]\!]$ of v on $F \in \mathcal{F}_h^i$ are defined as

$$\begin{aligned} \{\{v\}\} &= \frac{v|_{K_1} + v|_{K_2}}{2} , \\ [\![v]\!] &= v|_{K_1} - v|_{K_2} . \end{aligned}$$

Symmetric Interior Penalty Galerkin method

Problem 1 has been discretized with the Symmetric Interior Penalty Galerkin method, which reads as follows

Problem 2. *Find $u \in V_h$ such that*

$$a_h(u, v) = F_h(v) \qquad \forall v \in V_h .$$

The space V_h is the *finite dimensional piecewise discontinuous polynomial space* defined as:

$$V_h = \left\{ v \in L^2(\Omega) : v|_K \in \mathbb{P}^p(K) \quad \forall K \in \mathcal{K}_h \right\}$$

The bilinear form $a : V_h \times V_h \to \mathbb{R}$ can be expressed as

$$a_h(u,v) = \sum_{K \in \mathcal{K}_h} \gamma^K(u,v) + \sum_{F \in \mathcal{F}_h^i} \phi^F(u,v) + \sum_{F \in \mathcal{F}_h^b} \xi^F(u,v) ,$$

where

$$\begin{aligned} \gamma^K(u,v) &= \int_K \nabla u \nabla v + \lambda uv , \\ \phi^F(u,v) &= \int_F \frac{\eta}{h_F} [\![u]\!][\![v]\!] - \{\{\nabla u \cdot \mathbf{n}_F\}\}[\![v]\!] - \{\{\nabla v \cdot \mathbf{n}_F\}\}[\![u]\!] , \\ \xi^F(u,v) &= \int_F \frac{\eta}{h_F} uv - \nabla u \cdot \mathbf{n}_F v - \nabla v \cdot \mathbf{n}_F u . \end{aligned}$$

The linear functional $F : V_h \to \mathbb{R}$ reads as follows

$$F_h(v) = \int_\Omega fv - \sum_{F \in \partial\Omega} \int_F \Big(\nabla v \cdot n_F + \frac{\eta}{h_F} v \Big) g .$$

Stiffness matrix structure

Let B^K be the set of the basis functions on $K \in \mathcal{K}_h$. Let $K_1, K_2 \in \mathcal{K}_h$ such that $F = K_1 \cap K_2 \in \mathcal{F}_h^i$ as shown in Figure 1.

The local matrices Γ^K and Φ^F are defined as

$$\begin{aligned} \Gamma^K_{\mathbf{ij}} &= \gamma^K(\varphi_\mathbf{i}, \varphi_\mathbf{j}) \qquad && \forall \varphi_\mathbf{i},\ \varphi_\mathbf{j} \in B^K , \\ \Phi^F_{\mathbf{ij}} &= \phi^F(\varphi_\mathbf{i}, \varphi_\mathbf{j}) \qquad && \forall \varphi_\mathbf{i},\ \varphi_\mathbf{j} \in B^{K_1} \cup B^{K_2} . \end{aligned}$$

The local matrix Φ^F can be partitioned in

$$\Phi^F_{\mathbf{ij}} = \begin{cases} \Phi^F_{\mathbf{ij}}|^+_+ = \phi^F(\varphi_\mathbf{i}, \varphi_\mathbf{j}) & \text{if } \varphi_\mathbf{i},\ \varphi_\mathbf{j} \in B^{K_1} \\ \Phi^F_{\mathbf{ij}}|^+_- = \phi^F(\varphi_\mathbf{i}, \varphi_\mathbf{j}) & \text{if } \varphi_\mathbf{i} \in B^{K_1},\ \varphi_\mathbf{j} \in B^{K_2} \\ \Phi^F_{\mathbf{ij}}|^-_- = \phi^F(\varphi_\mathbf{i}, \varphi_\mathbf{j}) & \text{if } \varphi_\mathbf{i},\ \varphi_\mathbf{j} \in B^{K_2} \\ \Phi^F_{\mathbf{ij}}|^-_+ = \phi^F(\varphi_\mathbf{i}, \varphi_\mathbf{j}) & \text{if } \varphi_\mathbf{i} \in B^{K_2},\ \varphi_\mathbf{j} \in B^{K_1} \end{cases} .$$

The stiffness matrix of Problem 2 is a block matrix. The main diagonal blocks contain the contributions of Γ^K and of $\Phi^F_{\mathbf{ij}}|^+_+$ or $\Phi^F_{\mathbf{ij}}|^-_-$ (depending on $\mathbf{n}_F$). The matrix $\Phi^F_{\mathbf{ij}}|^+_-$ or $\Phi^F_{\mathbf{ij}}|^-_+$ (depending on $\mathbf{n}_F$) is present in correspondence of the *dof* of the mesh elements which share $F \in \mathcal{F}_h^i$.

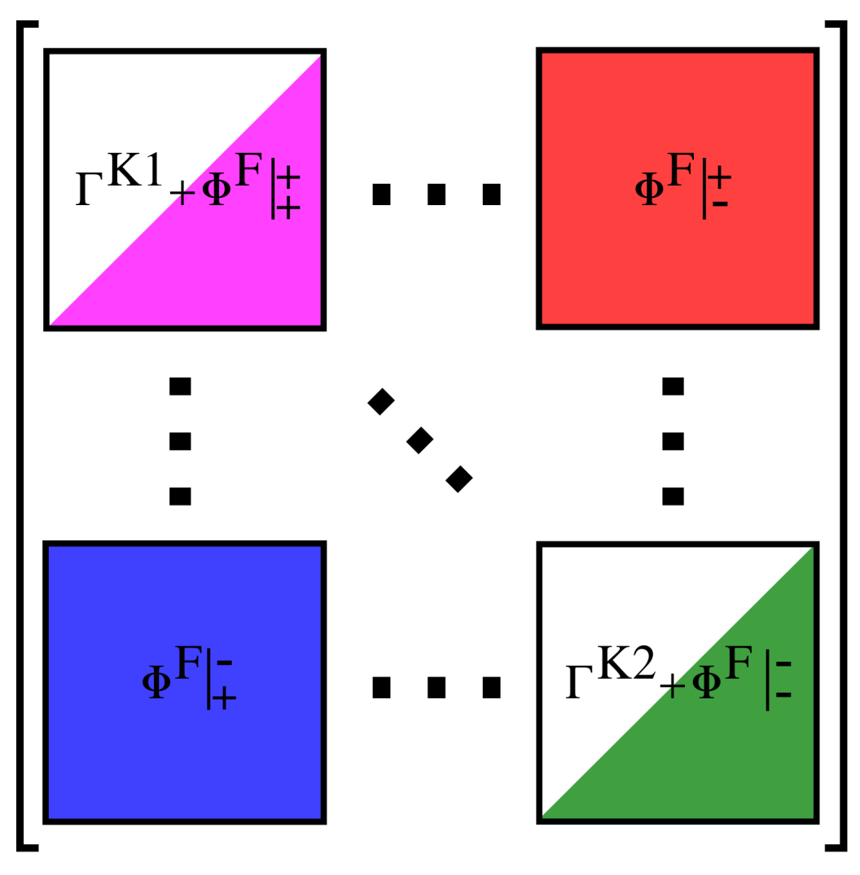

Figure 2: Stiffness matrix structure.

⋆ ⋆ ⋆

Matrix vector product implementation

The matrix vector product is the key step in any iterative solver. **Our proposal consists of merging the generation of the local matrix components and the assembly of the global stiffness matrix into the matrix vector product**, which is performed by directly applying the local matrices to the vector of the degrees of freedom (*dof*). In such a way, we avoid the storage of the global stiffness matrix, saving device memory and preventing non-coalescing memory accesses.

Data structure of *dof* vectors

Consider a structured quadrilateral mesh. For a two-dimensional problem, we save the vector of the *dof* using a three-dimensional data structure on the device memory. This data structure has three strides: the first is 1, the second is the number of elements per row, the third is the total number of mesh elements. So, the first two indexes select a mesh element, the third index selects a specific *dof* within the element.

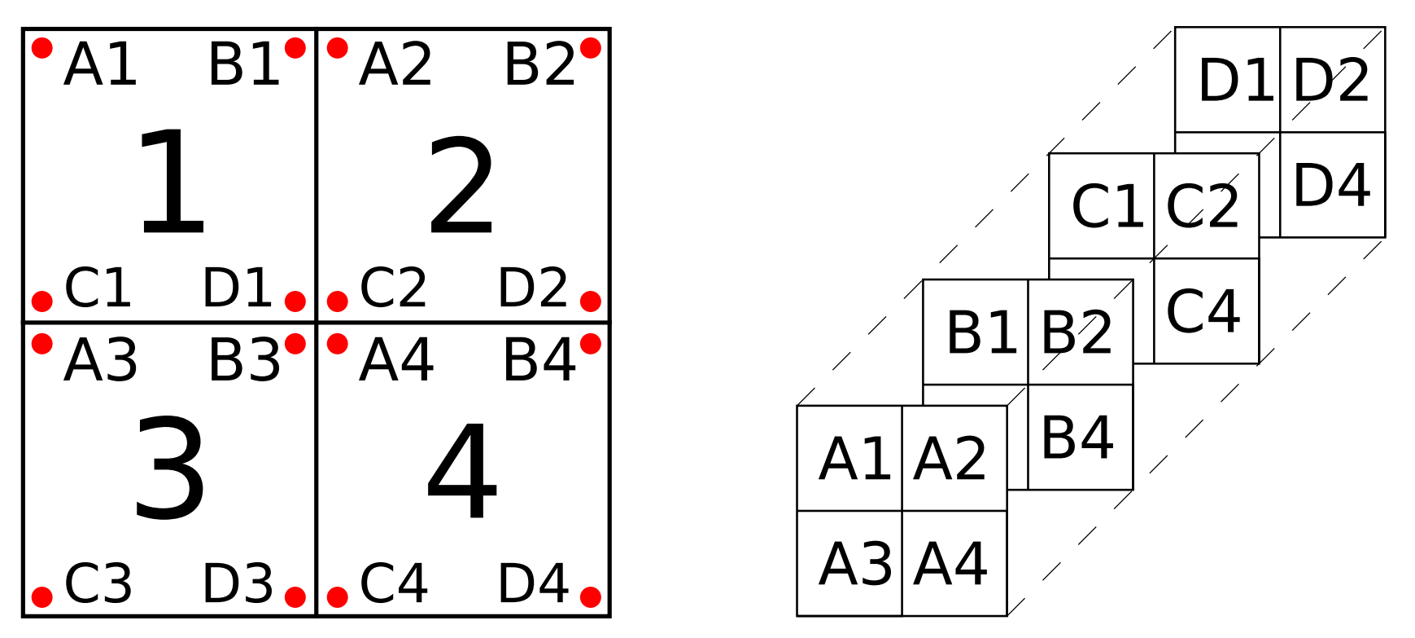

Figure 3: Example of data structure [right] for a 4-element mesh [left] with 4 *dof* for each element.

The structure of the SIPG method stiffness matrix and the structure of the vector of *dof* let possible to **implement the matrix vector product as a stencil code**.

Multi-GPU

In order to exchange the halos of the local portions of the *dof* vector between the GPUs, we have employed the GCL library, which is part of GridTools. The halos have been declared easily with the following lines of code.

```
GCL::array<GCL::halo_descriptor,3> halo_dsc;
halo_dsc[0] = GCL::halo_descriptor(1, 1, 1, x-2, x);
halo_dsc[1] = GCL::halo_descriptor(1, 1, 1, y-2, y);
halo_dsc[2] = GCL::halo_descriptor(0, 0, 0, de, de);
```

Where `x`, `y` are the local mesh sizes and `de` is the number of *dof* per element.

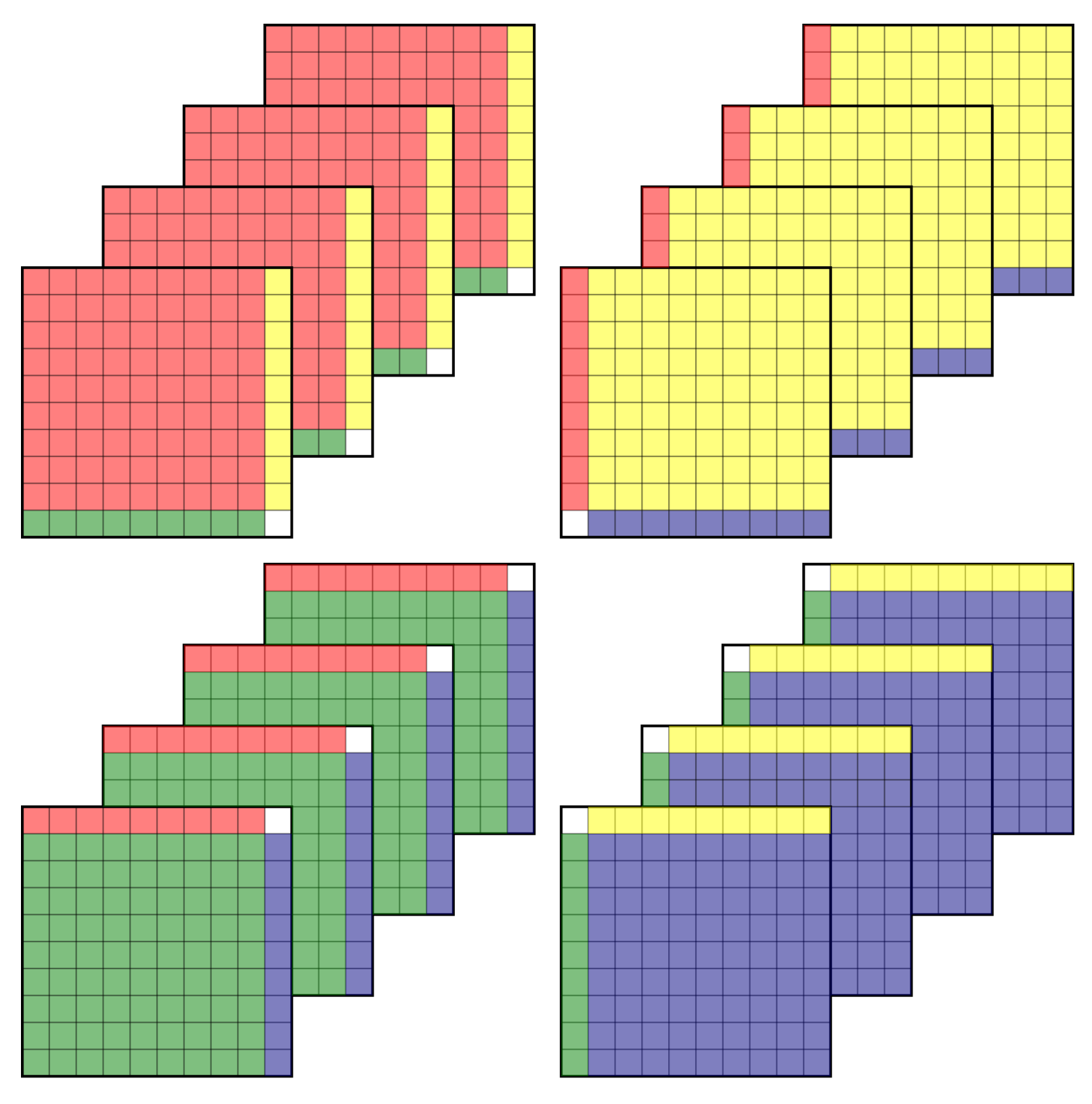

Figure 4: Halo exchange between 4 GPUs.

We have evaluated the strong and the weak scaling of the multi-GPU code. Strong scaling means measuring the execution times for a fixed size problem while increasing the number of GPUs. We have measured the speedup, which is defined as

$$speedup = \frac{\tau(1)}{\tau(n)}$$

where $\tau(n)$ is the execution time on n GPUs.

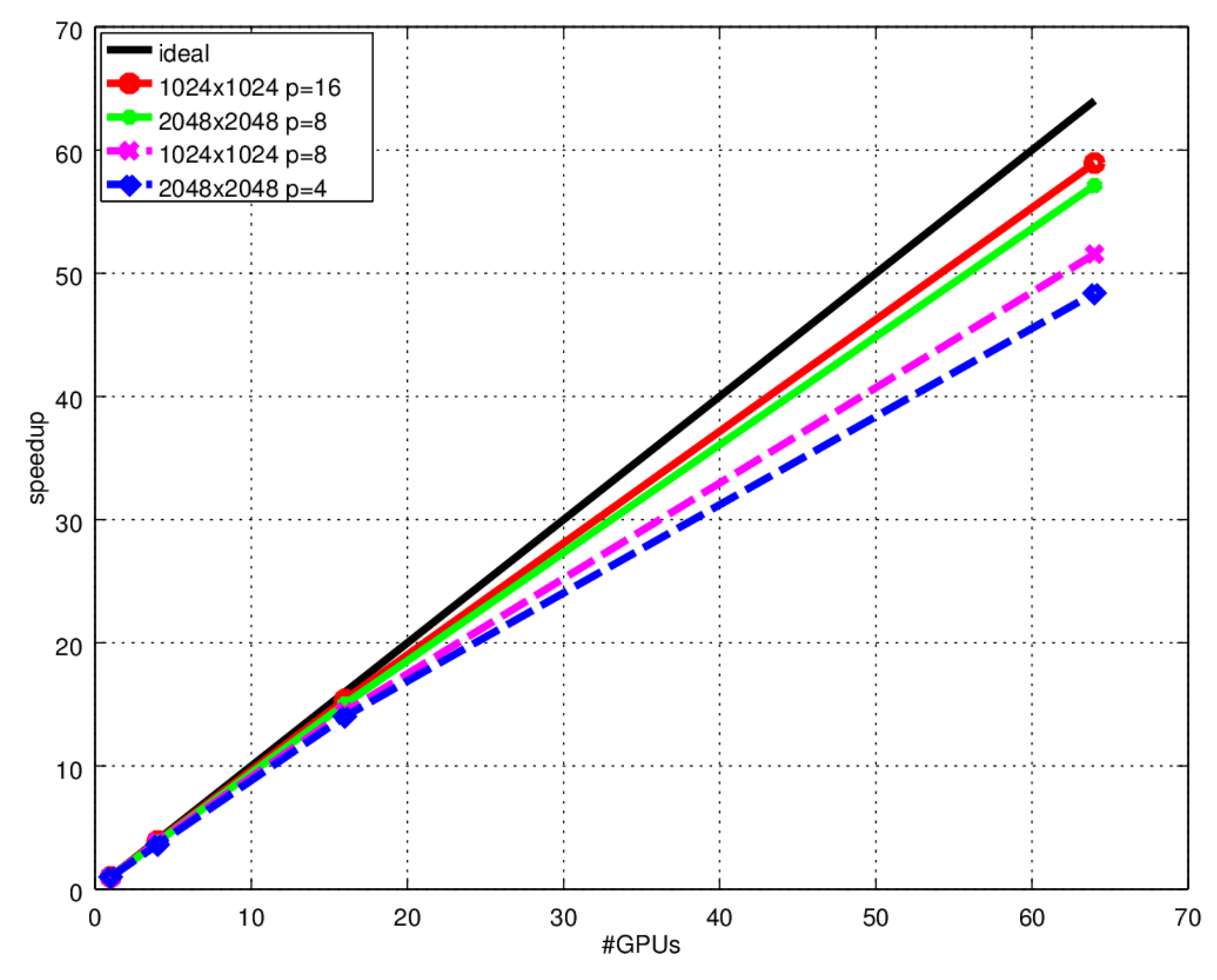

Figure 5: Strong scaling

In weak scaling, the problem size assigned to each GPU stays constant, so that using additional GPUs makes the size of the problem grow. The weak scaling efficiency ε is defined as

$$\varepsilon = \frac{\bar{\tau}(1)}{\bar{\tau}(n)} ,$$

where $\bar{\tau}(n)$ is the execution time for problems of size increasing with the number of GPUs n.

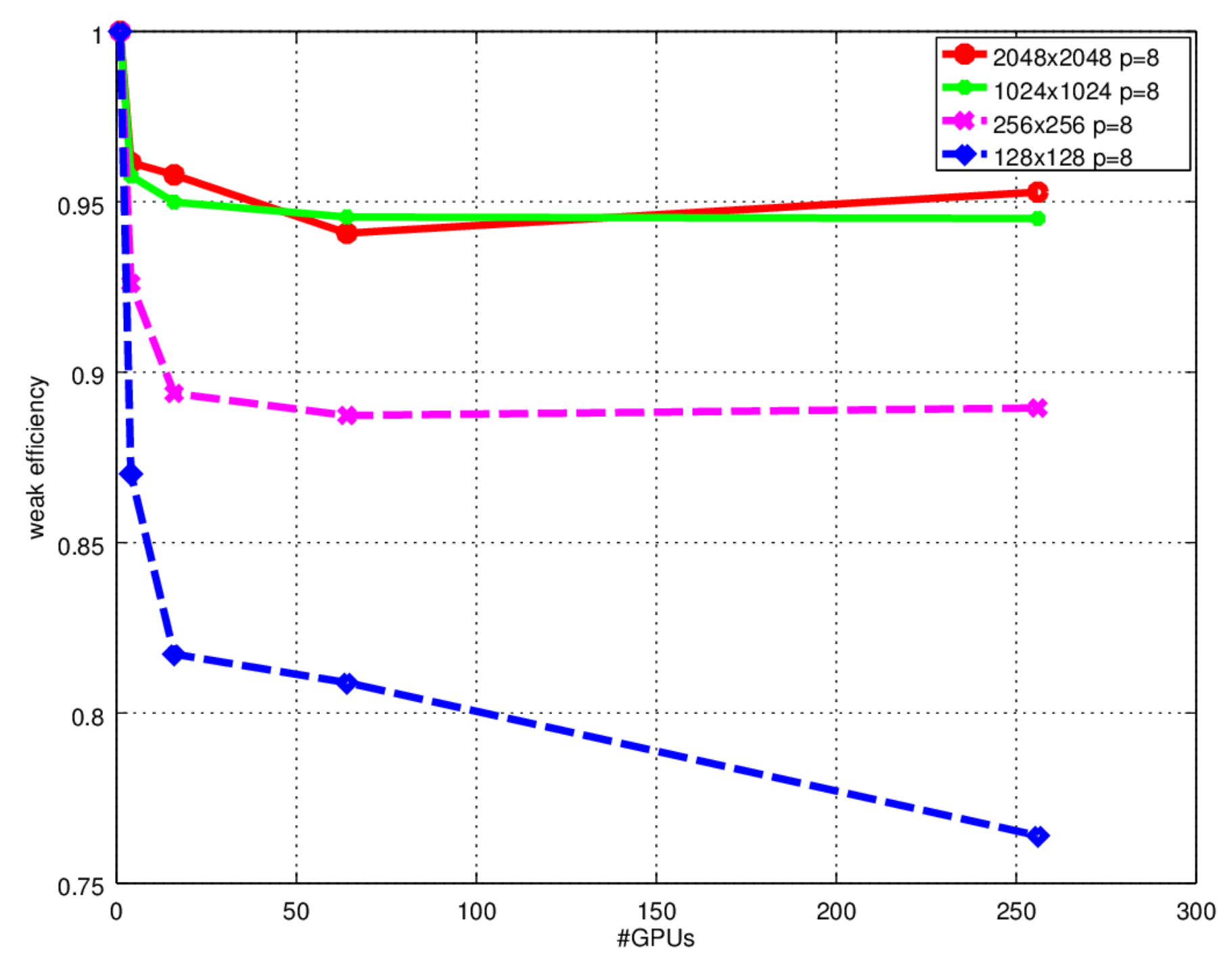

Figure 6: Weak scaling

⋆ ⋆ ⋆

Conclusions

We have applied our proposal to the solution of a Poisson problem with Dirichlet boundary conditions. The linear system obtained has been solved on GPUs. The code has been developed in the CUDA programming language and first validated with respect to the correctness of the implementation. Subsequently, performance analyses were carried out. The developed kernels reach a high occupancy and exploit most of the available memory bandwidth.

⋆ ⋆ ⋆

Acknowledgements

This project has been developed as master thesis by Paolo Gorlani. Thesis supervisors have been Prof. Luca Bonaventura and Prof.ssa Paola Francesca Antonietti, both of them are professors of the Laboratory for Modeling and Scientific Computing (MOX) in the Department of Mathematics at Politecnico di Milano.

We acknowledge the help and support in our development of the Centro Svizzero di Calcolo Scientifico, that has provided Piz Daint system; William Sawyer, computational scientist at CSCS, who has shown his interest for this project and has provided useful advices; Mauro Bianco, computational scientist at CSCS, who has developed and has supplied the GCL library used in the multi-GPU version of the code.