Task-parallel *in situ* data compression for large-scale computational fluid dynamics simulations

PAW-ATM Virtual Workshop 2020

Heather Pacella[1], Alec Dunton[2], Alireza Doostan[2], and Gianluca Iaccarino[1]

[1]Stanford University

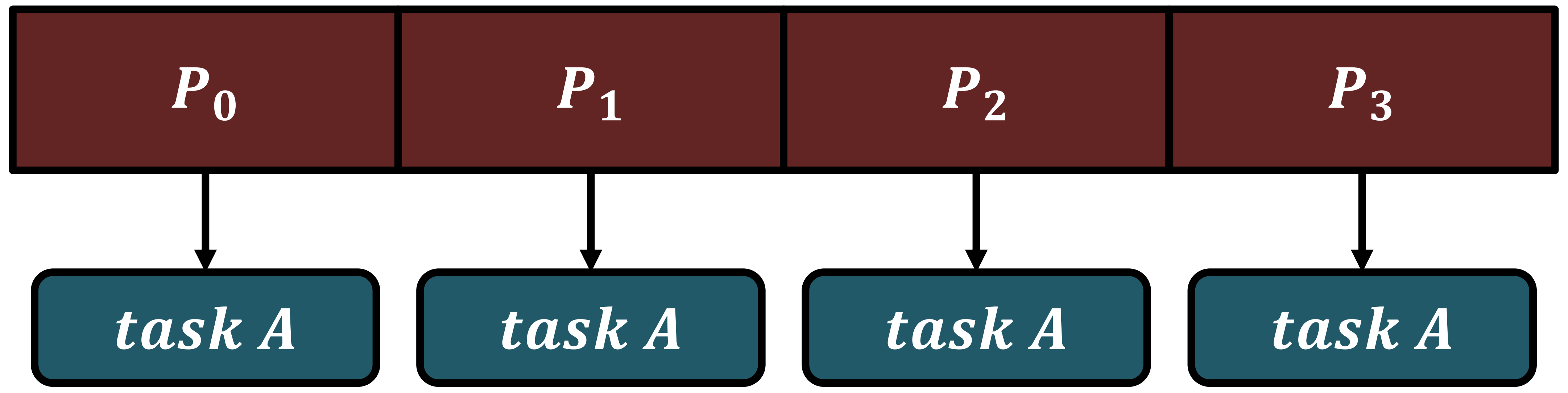
[2]University of Colorado, Boulder

12 November 2020

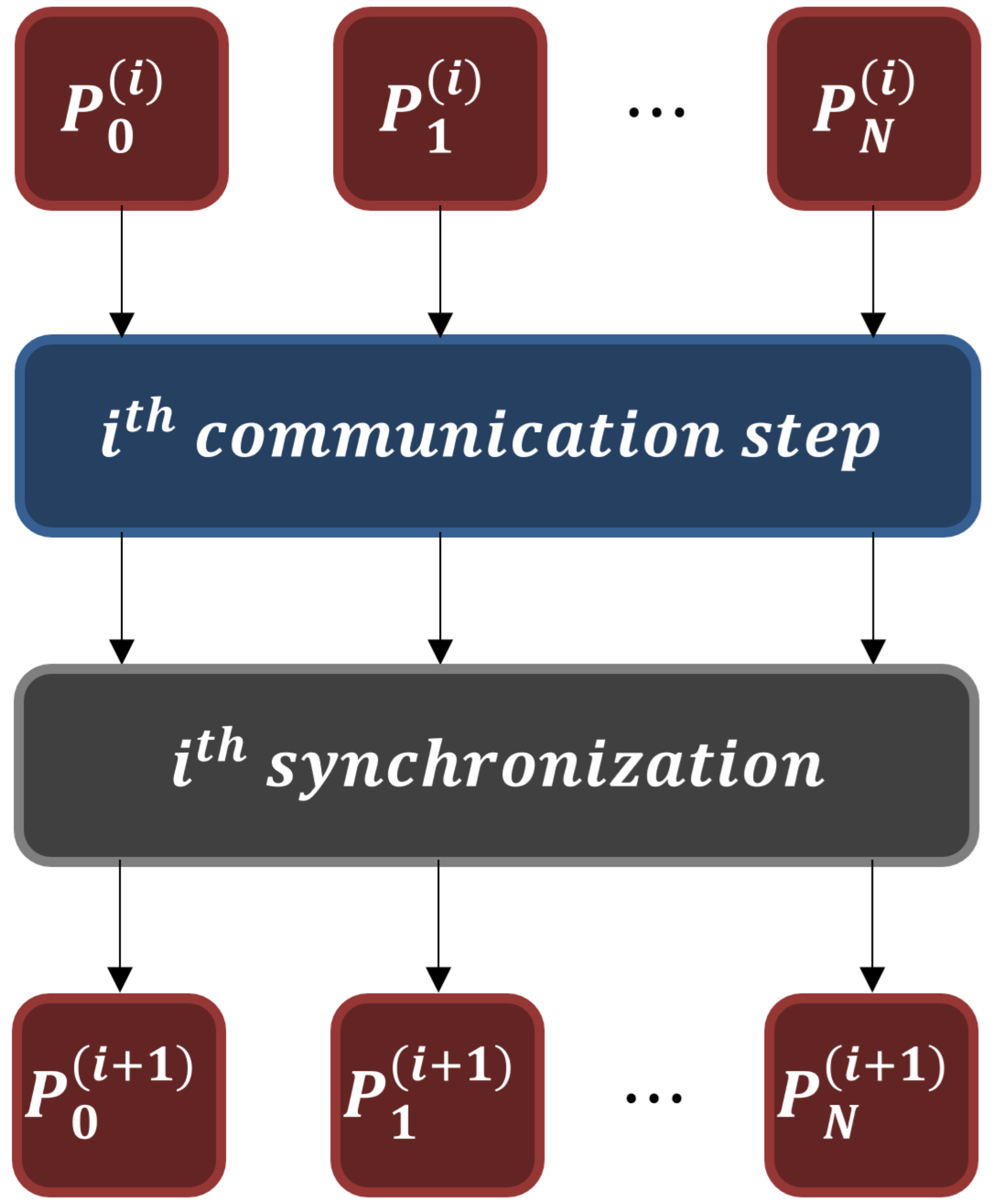
Computational Fluid Dynamics (CFD)

- **CFD simulations** involve numerical solution of a set of PDEs on a discretized grid that represents the spatial domain of a fluid systems.

- For large simulations, this discretized domain is partitioned into subdomains of data, which are then distributed across the available computing resources.
  - Simulations tend to be **data-parallel** and are typically implemented using ***synchronous*** or ***bulk-synchronous*** programming models.
  - For each time step of the simulation, identical sets of instructions are executed on the blocks of the domain, then there is a communication stage as data between processors is exchanged.



data-parallel execution



Stanford University

Motivation for Data Compression

- CFD simulations can generate large quantities of data. Additionally, if large ensembles of these model evaluations are required for inference, UQ, or optimization methods, existing memory capacity is quickly exceeded.

- Over the next decade, exascale supercomputers will provide a 1,000-10,000 times increase in floating-point performance, with memory increase of only a factor of 100 [1].

- To address this, we implement *in situ* compressive algorithms within the solvers themselves, **which allows us to store these simulations at a fraction of the memory cost**.

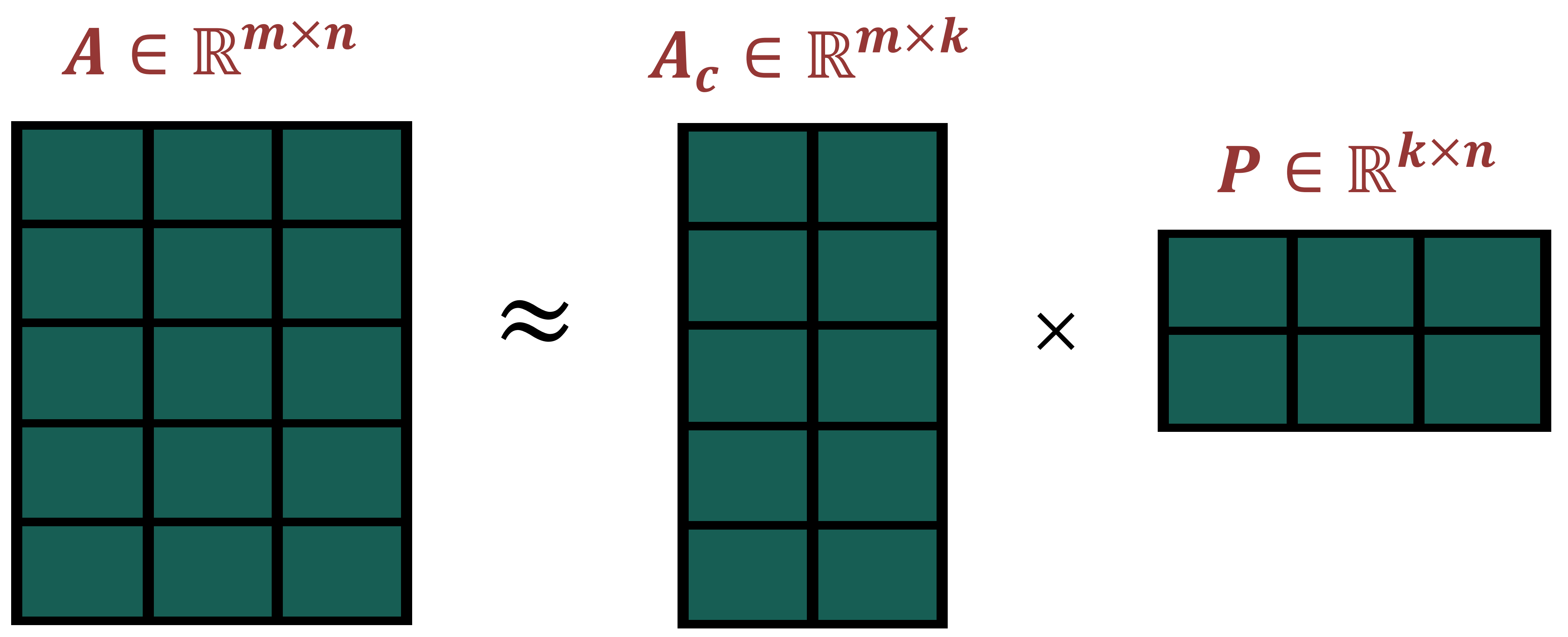
- We are interested in lossy compression, which provides an approximation of the original data but large compression ratios via the construction of *low-rank matrix approximations.*
    - ➢ ***Interpolative decomposition*** (ID) is a type of lossy compression.

[1] Ang et. al,. (2012) Workshop on extreme-scale solvers: transition to future architectures U.S. D.O.E.

Interpolative Decomposition (ID) methods

- Interpolative decomposition (ID) forms a low rank approximation $\boldsymbol{A} \approx \boldsymbol{A_c P}$ where:
  - ➢ The matrix $\boldsymbol{A_c} \equiv \boldsymbol{A}(:, \boldsymbol{\mathcal{J}})$ is a subset of the columns of the original data matrix $\boldsymbol{A}$, whose column indices are denoted by index vector $\boldsymbol{\mathcal{J}}$.
  - ➢ $\boldsymbol{P}$ is a (non-sparse) coefficient expansion matrix.
  - ➢ The rank of this approximation, k, is controlled by either a user-provided target rank or a user-provided tolerance.

- The matrices $\boldsymbol{A_c}$ and $\boldsymbol{P}$ are formed from a pivoted QR performed on $\boldsymbol{A}$ using a modified Gram-Schmidt process.



$\boldsymbol{A} \in \mathbb{R}^{\boldsymbol{m \times n}}$
$\approx$
$\boldsymbol{A_c} \in \mathbb{R}^{\boldsymbol{m \times k}}$
$\times$
$\boldsymbol{P} \in \mathbb{R}^{\boldsymbol{k \times n}}$

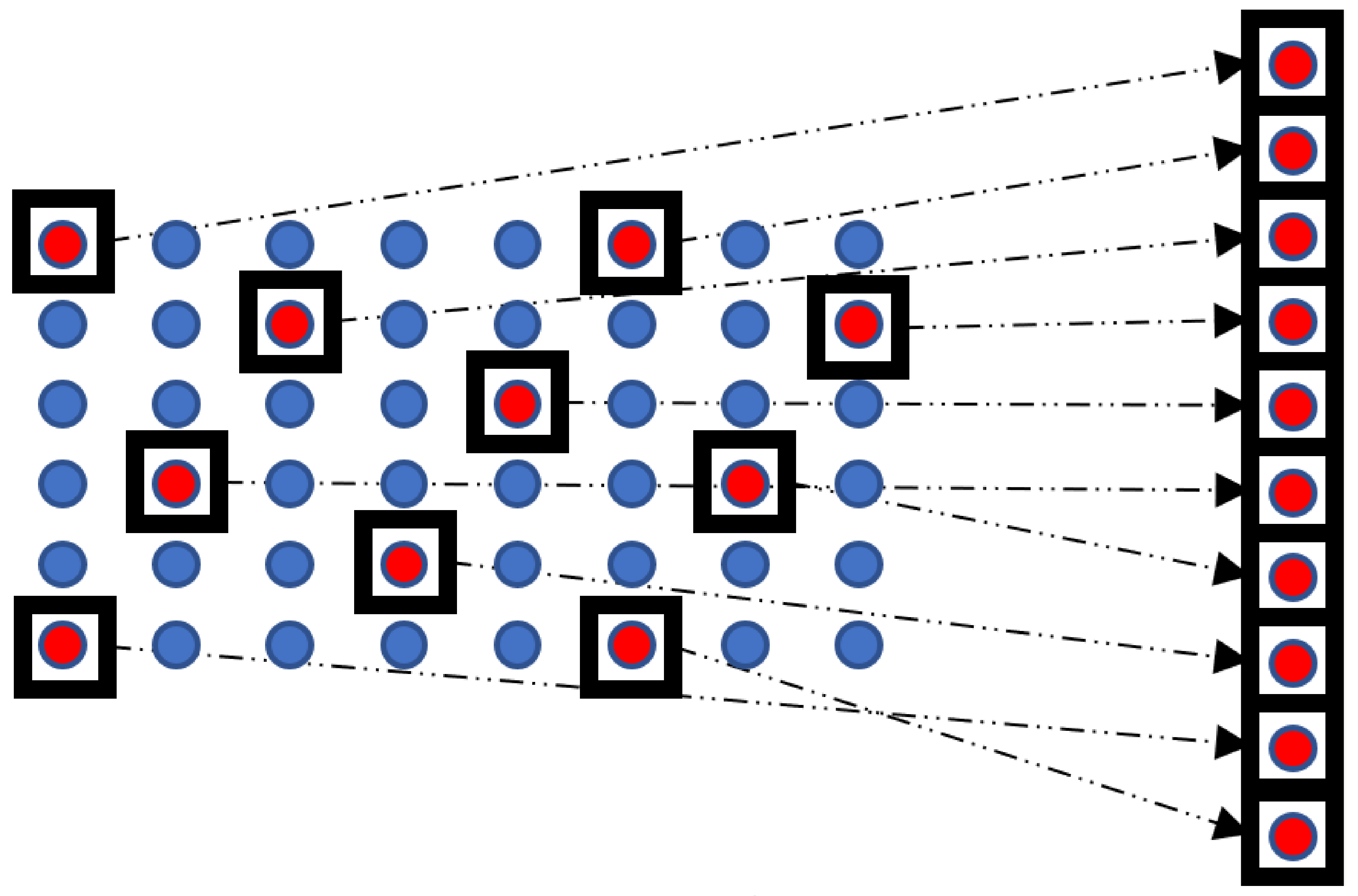
$\boldsymbol{mk, nk \ll mn}$

- There are multiple ways to form the interpolative decomposition; in this talk, we will be concerned with the Single Pass Interpolative Decomposition (SPID).

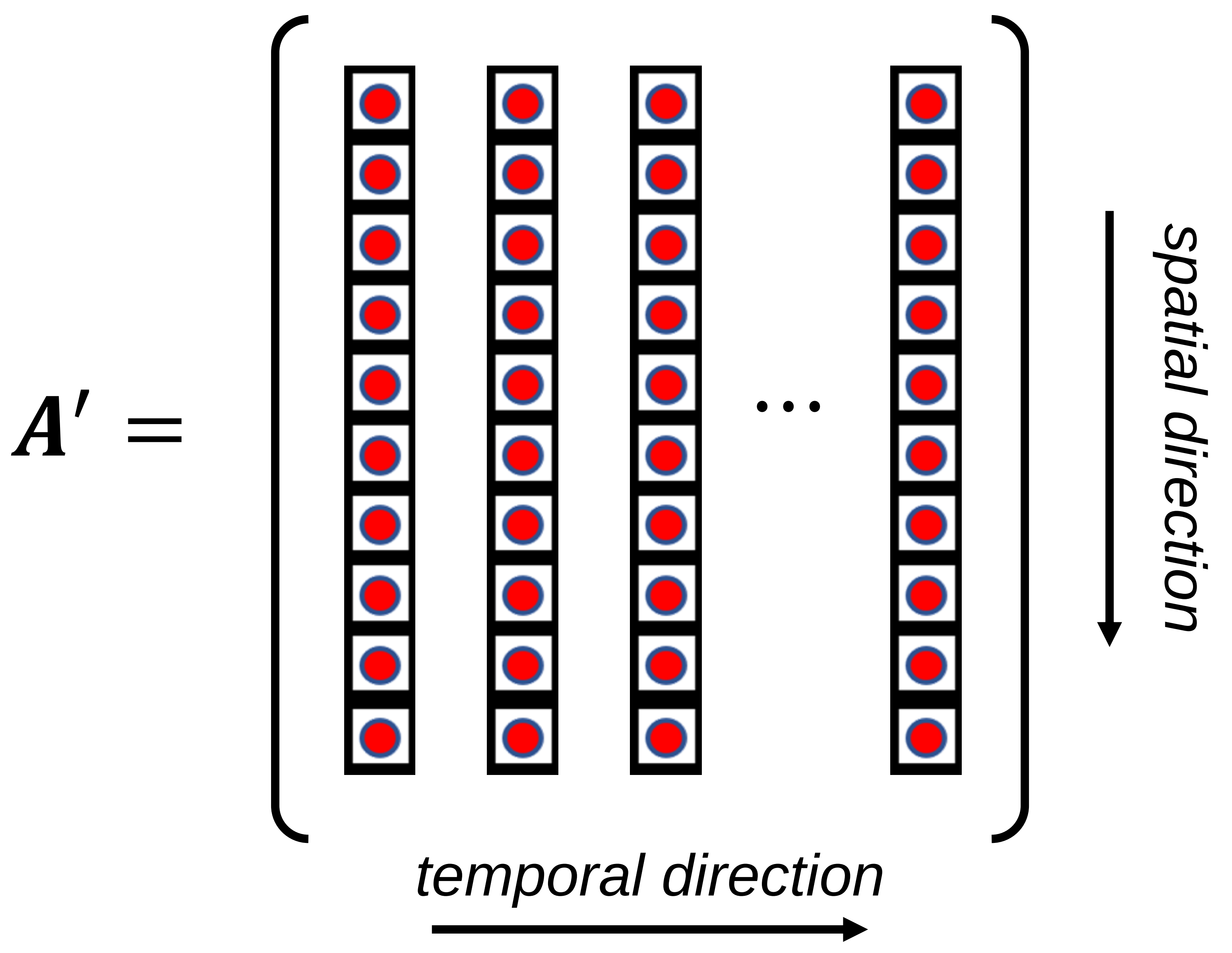
[2] Cheng, H., et al. (2005) On the compression of low rank matrices. *SIAM J. Sci. Comput.* 26

Interpolative Decomposition (ID) methods

- For time-dependent physical systems, each column of $\boldsymbol{A}$ is a flattened spatial domain at a specific time step.
  - ➢ The row index corresponds to physical space (x, y, z) and the column index corresponds to temporal space (Δt_n).
  - ➢ In this case, each column of $\boldsymbol{A_c}$ corresponds to an "important" time step of the simulation.
- To reduce storage requirements and overall dimensionality when storing the entries of $\boldsymbol{A}$, the spatial data can be subsampled and stored in the matrix $\boldsymbol{A'}$.
  - ➢ For SPID, this results in a coarse grid compression, $\boldsymbol{A} \approx \boldsymbol{M A'_c P}$, where $\boldsymbol{A'_c}$ is a subset of the columns of the subsampled data matrix $\boldsymbol{A'}$ and $\boldsymbol{M}$ is an interpolation operator back to the original grid.



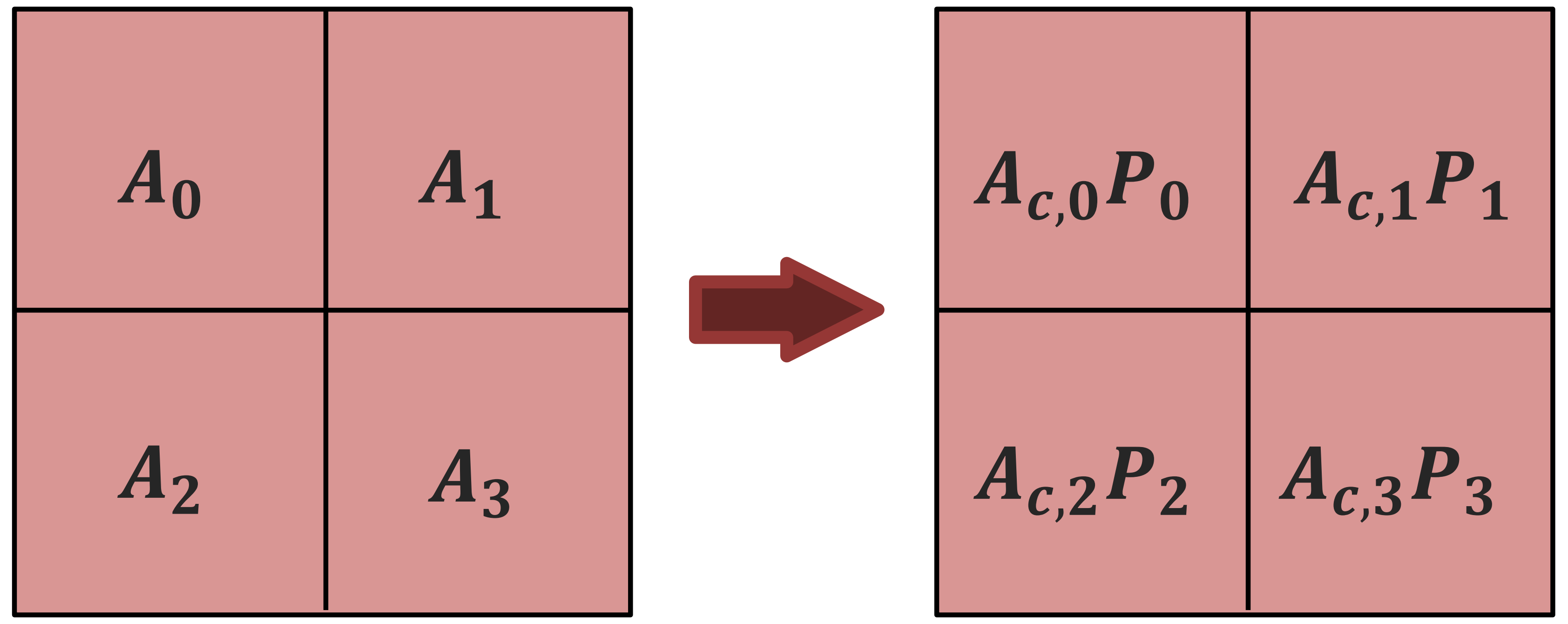
spatial domain at the i^{th} time step



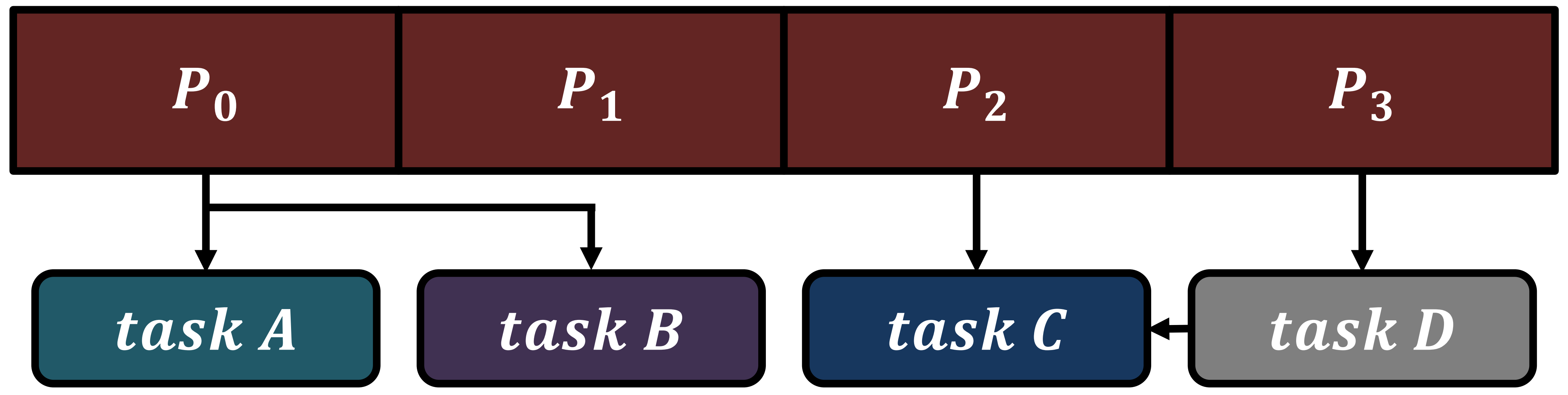
Task-based parallelism and ID

- When simulating large physical systems, the domain is partitioned into subregions and each of these subregions is operated on (for the most part) in parallel.

- Interpolative decomposition is also applied to these domain subregions; depending on the temporal evolution of the physical system, different subregions may have different important time realizations. This means that the computational cost (and dimensions) of $\boldsymbol{A_c P}$ can differ between the regions.



- Task-based parallelism parallelizes over tasks instead of data. The task launches are **asynchronous**, and eliminate computational bottlenecks.



Stanford University

Parallelizing ID within CFD Applications

- As previously presented, the data matrix $\boldsymbol{A}$ is formed by collecting all data from the CFD simulation and then performing the SPID.
    - However, even with physical subsampling, the size of this matrix can exceed available memory.
    - Also forces the ID simulation to be a “serial addition” to the CFD simulation.

- It is desirable to find a way to interleave the ID algorithm with the fluid solver, so that the ID computations can start while the flow solver is still running. We do this via a “two-stage” process.

Stage 1:

Partition the matrix $\boldsymbol{A}$ (which is still being formed) along the temporal direction into $\boldsymbol{N}$ subdomains of size $\boldsymbol{A} \in \mathbb{R}^{\boldsymbol{m}\times\frac{\boldsymbol{n}}{\boldsymbol{N}}}$; apply the SPID algorithm to each of these blocks as they are formed.

At the end of the CFD simulation, the following matrix is formed:

$$\boldsymbol{A} \approx \boldsymbol{A}' = [\boldsymbol{A}_0(:,\mathcal{I}_0)\boldsymbol{P}_0 \quad \cdots \quad \boldsymbol{A}_{N-1}(:,\mathcal{I}_{N-1})\boldsymbol{P}_{N-1}] = \boldsymbol{A}^{1\prime}\boldsymbol{P}^{1\prime}$$

where

$$\boldsymbol{A}^{1\prime} = [\boldsymbol{A}_0(:,\mathcal{I}_0) \quad \cdots \quad \boldsymbol{A}_{N-1}(:,\mathcal{I}_{N-1})], \qquad \boldsymbol{P}^{1\prime} = \mathbf{diag}(\boldsymbol{P}_0,\cdots,\boldsymbol{P}_{N-1})$$

Parallelizing ID within CFD Applications

Stage 2:

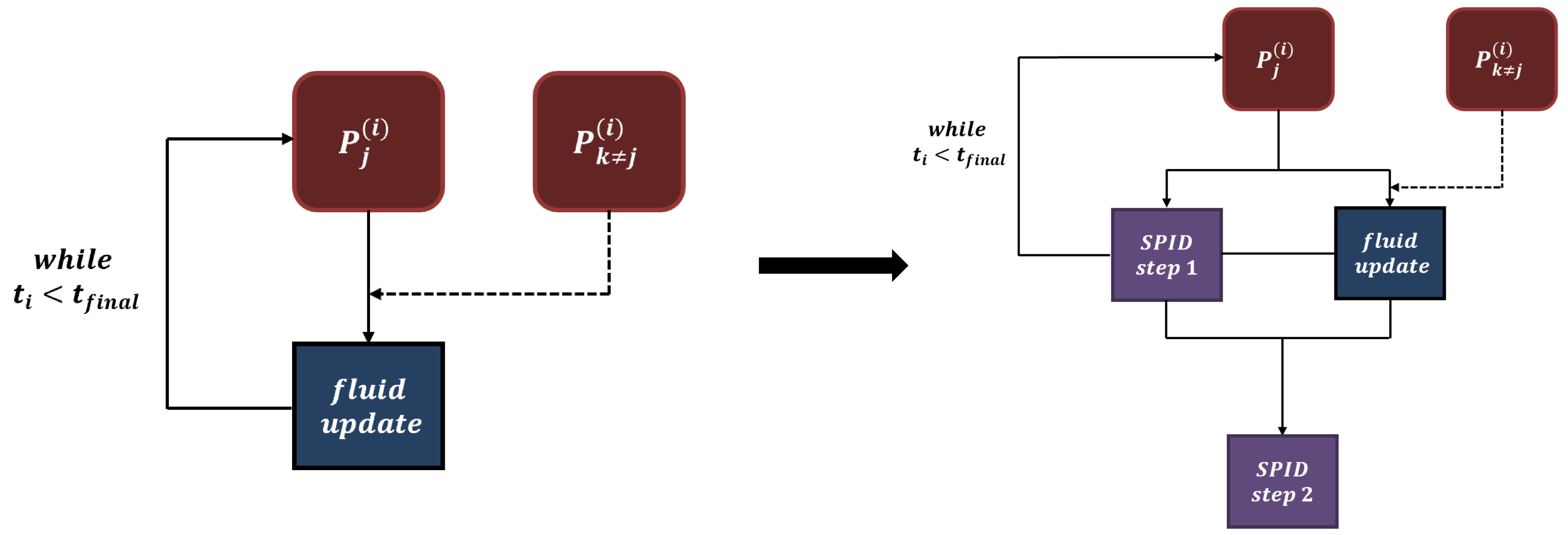
To achieve additional compression after the completion of the flow solver, apply the SPID algorithm again to the matrix $\boldsymbol{A}^{1\prime}$. This results in:

$$\boldsymbol{A}^{1\prime} \approx \boldsymbol{A}^{2\prime}\boldsymbol{P}^{2}, \quad \text{where } \boldsymbol{A}^{2\prime} = \boldsymbol{A}^{1\prime}(:, \mathcal{J}_U)$$

Then the final compressed result is:

$$\boldsymbol{A} \approx \boldsymbol{A}^{1\prime}\boldsymbol{P}^{1\prime} \approx \left(\boldsymbol{A}^{2\prime}\boldsymbol{P}^{2}\right)\boldsymbol{P}^{1\prime} \equiv \boldsymbol{A}^{2\prime}\boldsymbol{P}^{2\prime}$$

- Now, most of the ID work is completed while the flow solver is still running. If there is enough latency in the communication step of the flow solver, the cost of the data compression can be hidden.



Legion: An Alternative to MPI+X

- **Legion** is a task-based parallel programming system. It is an open-source, collaborative effort between Stanford University, LANL, NVIDIA Research, and SLAC.

- We chose this (over other programming models) because:
  - A single Legion code can be ported to multiple architectures (laptops, large computing clusters, CPUs, GPUs, a mixture of CPUs and GPUs, etc.).
  - The Legion runtime dynamically allocates tasks, which allows for dynamic load balancing with little to no user input.
  - Regent, a high-level programming language, allows for the extraction of implicit parallelism from a serial code.
  - The ability to write a custom mapper allows for fine-grain control over how an application runs on various machines.
  - There are several tools available to the user to visualize the exact execution of an application: **Legion Spy** and **Legion Prof**.

Task-based parallelism with Legion

- **Legion Spy** is a tool that can be used to show task dependencies in a visual manner. Independent tasks are displayed as stacked boxes, while the arrows between them denote dependence.

- Spy results can be used to determine why tasks are not executing in parallel, as well as why tasks are delayed.

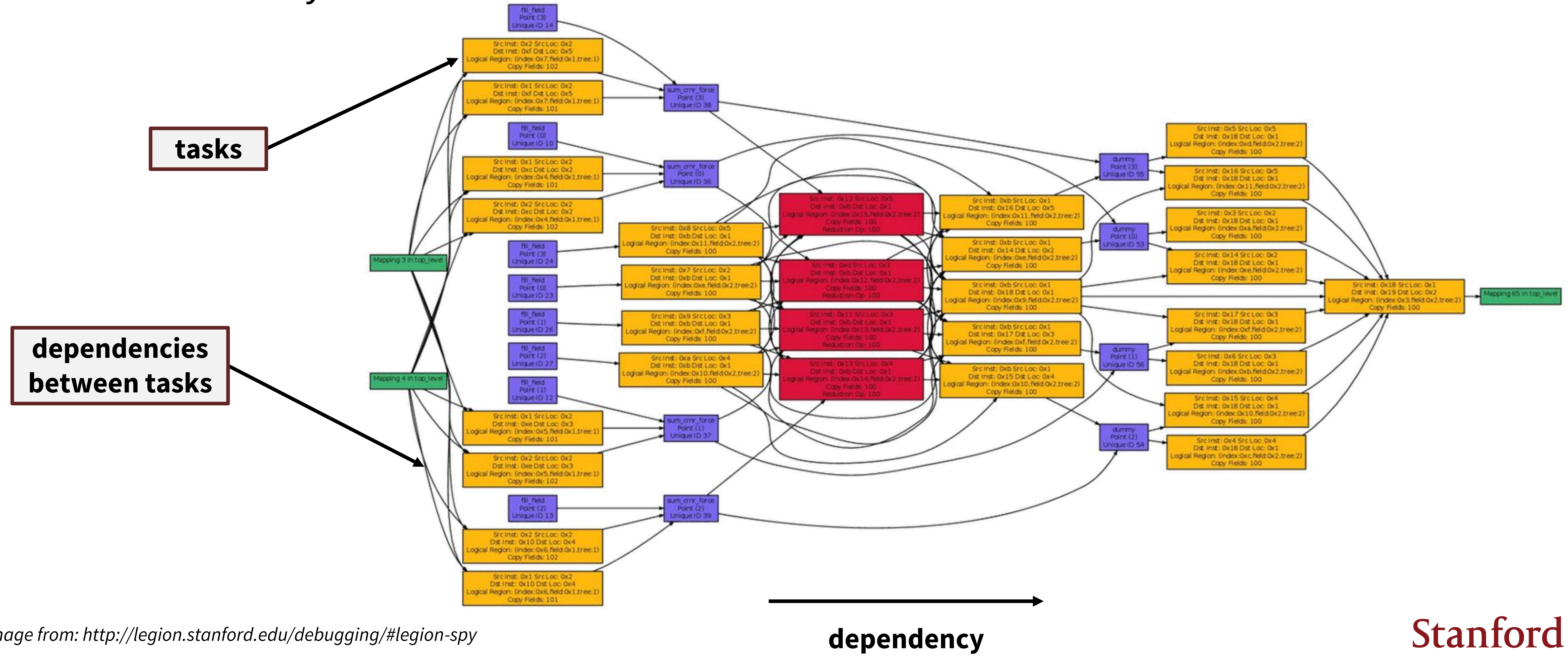


image from: http://legion.stanford.edu/debugging/#legion-spy

Stanford University

Legion: An Alternative to MPI+X

- **Legion Prof** shows overall resource utilization, as well as more detailed information such as start and end times of individual tasks or dependence analysis.

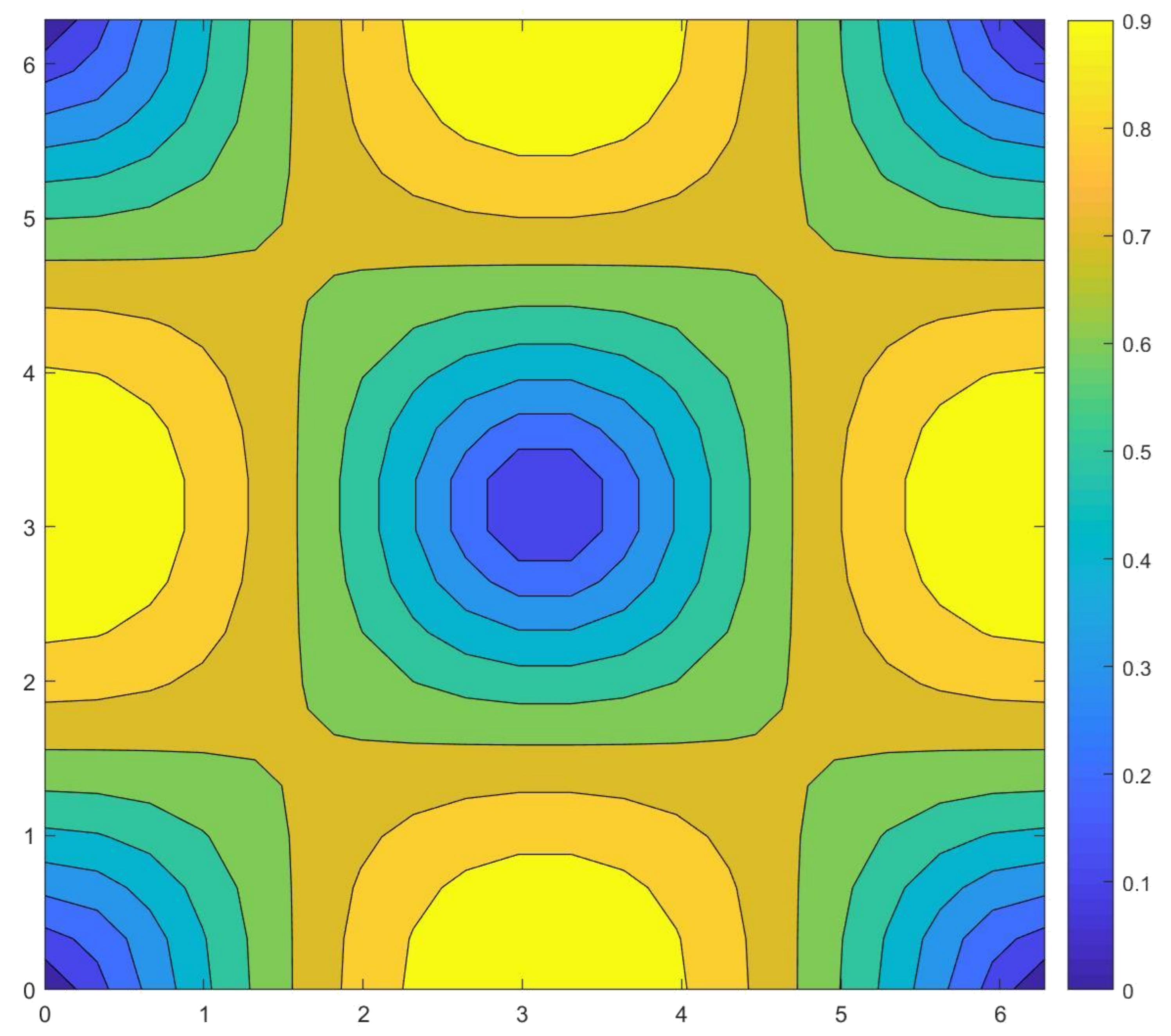
resource utilization
idle time
tasks
runtime utilization
time
node 0 (CPU)
CPU Proc 1
CPU Proc 2
CPU Proc 3
CPU Proc 4
CPU Proc 5
CPU Proc 6
CPU Proc 7
CPU Proc 8
node 0 (Utility)

Stanford University

Numerical Studies: Incompressible Taylor-Green Vortex

- An incompressible, 3D Taylor-Green vortex was simulated on both structured and unstructured grids to study how well our SPID algorithm approximates a rank-one system. The exact solution of the vortex is:

$$\begin{cases} u(x,y,t) = \cos(x)\sin(y)\,e^{-2\nu t} \\ v(x,y,t) = -\sin(x)\cos(y)\,e^{-2\nu t} \\ p(x,y,t) = \dfrac{\rho}{4}(\cos(2x)\cos(2y))e^{-4\nu t} \end{cases}$$



- The domain is doubly-periodic, with $0 \leq x, y \leq 2\pi$.

- The structured grid was linearly spaced, while the unstructured grid was generated via random numbers on the interval $2\pi[0,1]$.

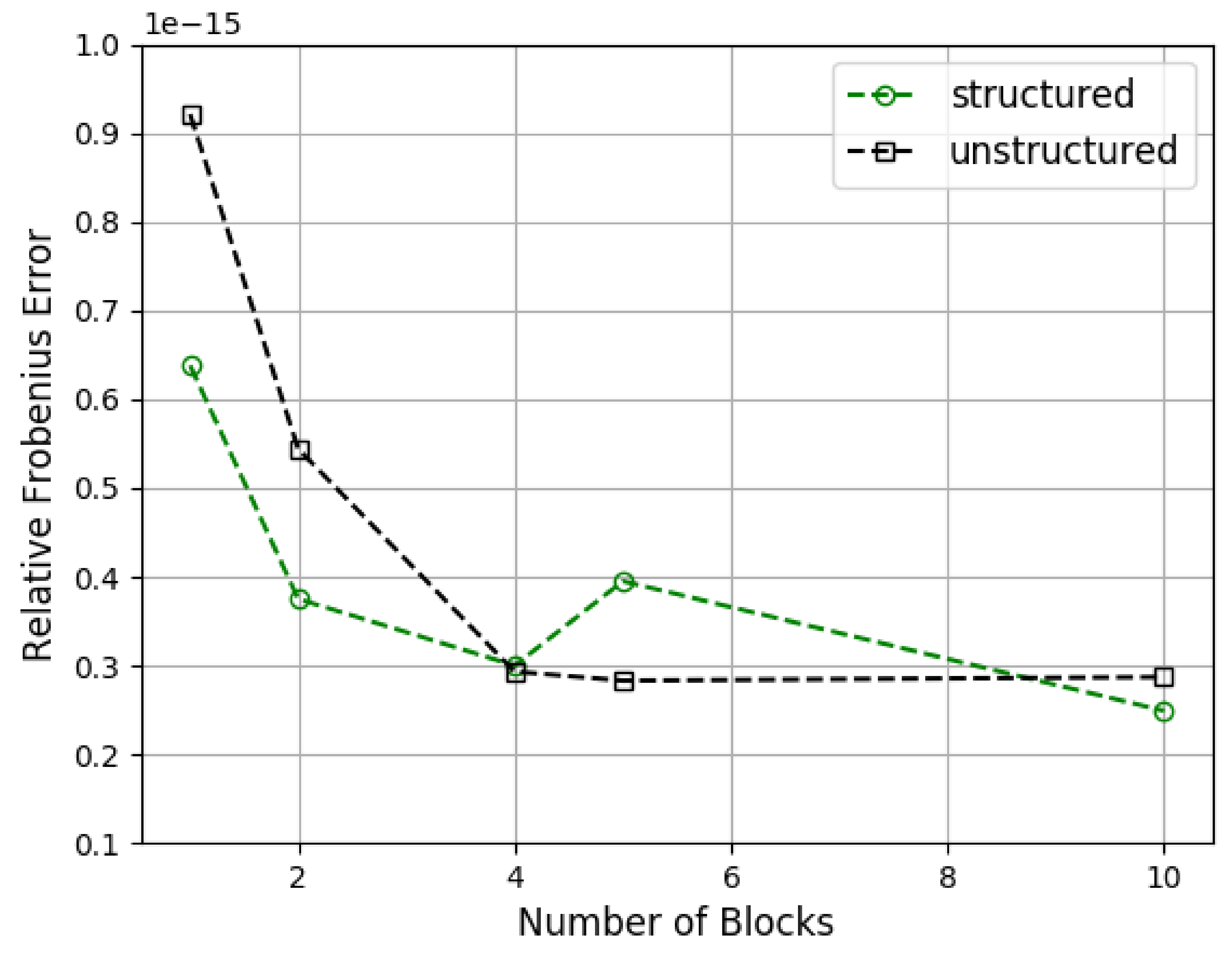
Stanford University

Numerical Studies: Incompressible Taylor-Green Vortex

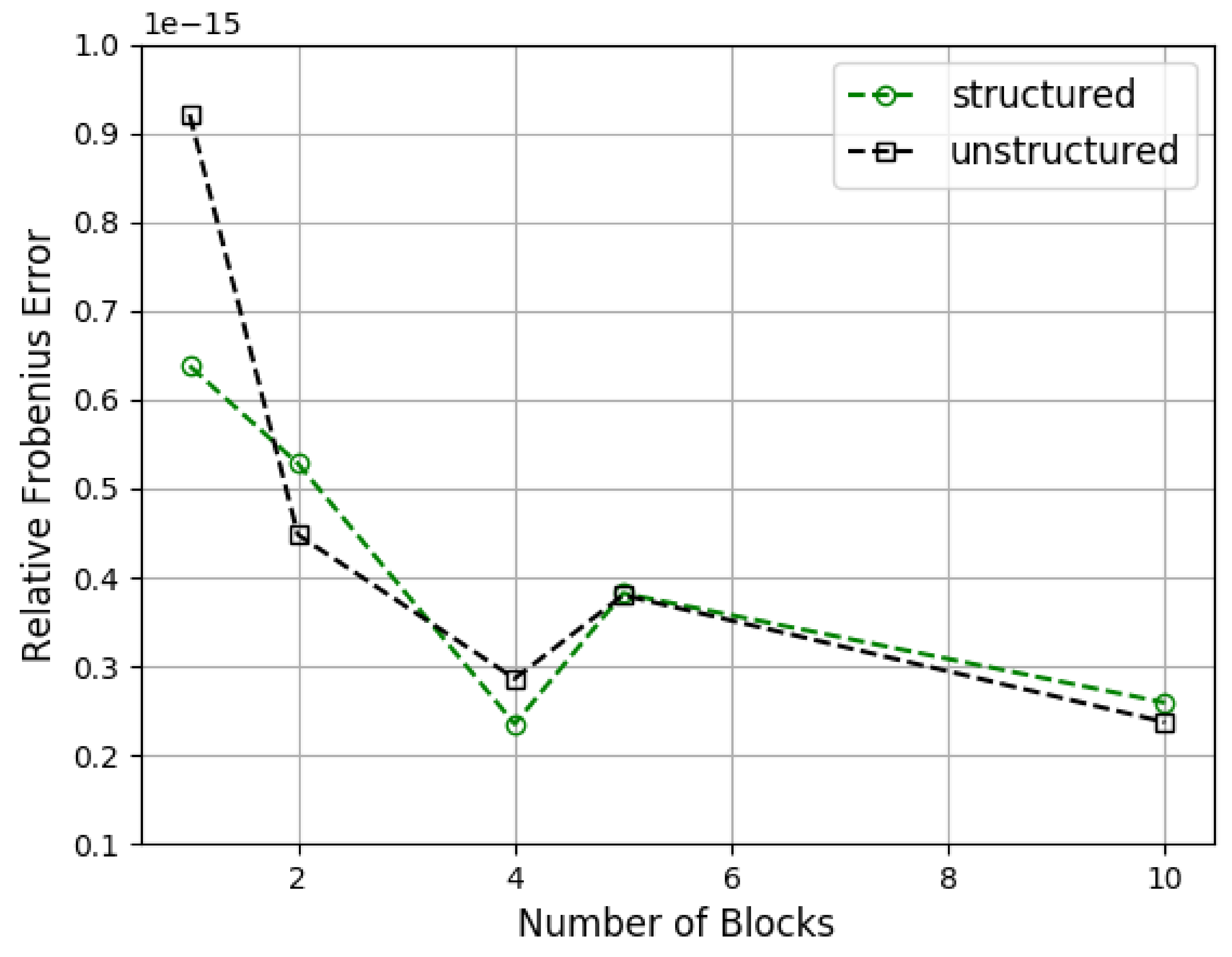
- The SPID algorithm was applied to both a structured and an unstructured grid on the domain $[0, 2\pi)$ with 20^2 points for 100 "time steps". The reconstructed solution was compared to the exact solution via the normalized Frobenius norm of the error:

$$E = \frac{\left\| \boldsymbol{A}_{\boldsymbol{exact}} - \boldsymbol{A}_{\boldsymbol{approx}} \right\|_{FR}}{\left\| \boldsymbol{A}_{\boldsymbol{exact}} \right\|_{FR}}$$

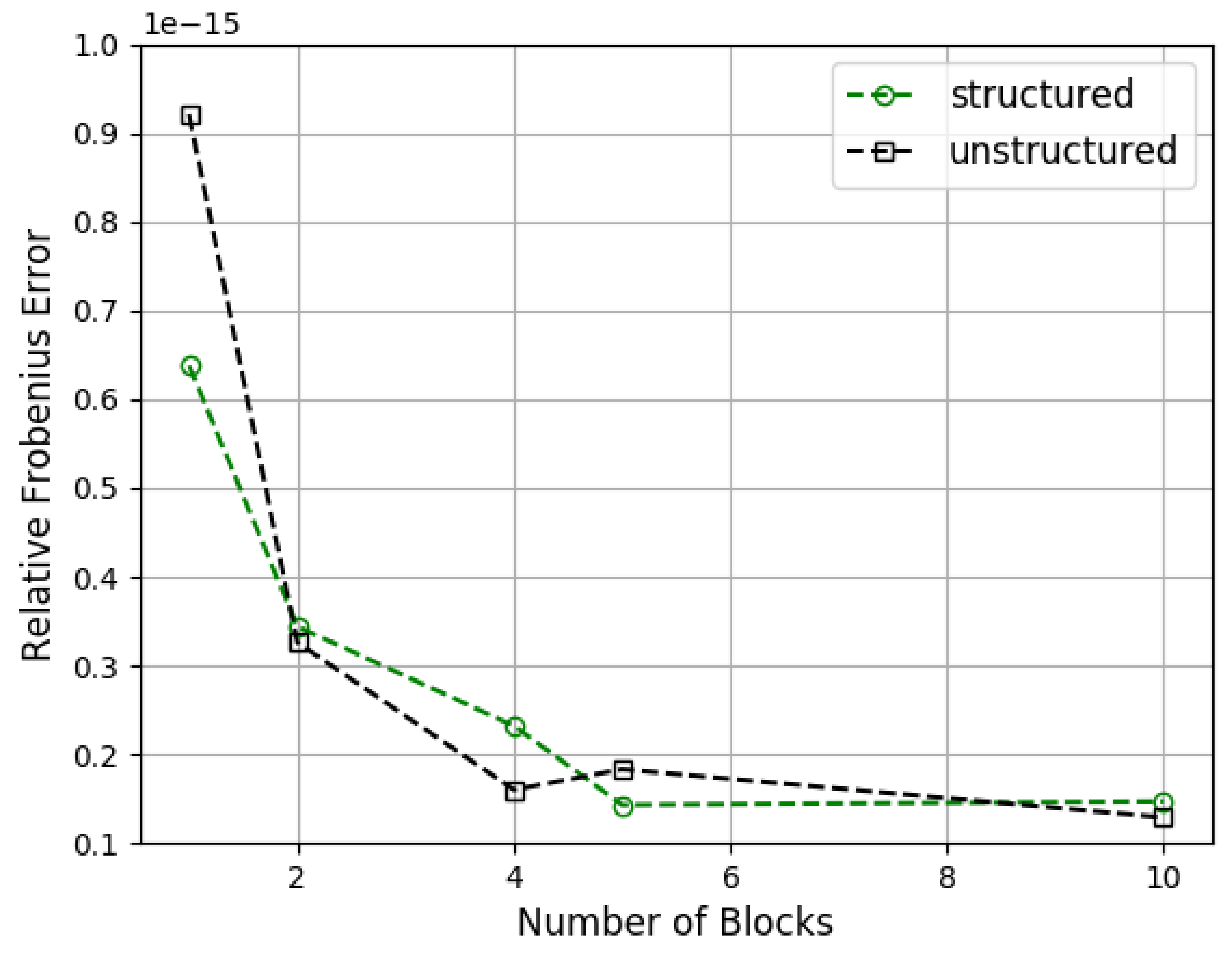
- The error in u as a function of partitions in the spatial direction is shown below:



u error as a function of blocks in the x_1-direction



u error as a function of blocks in the x_2-direction



u error as a function of blocks in the x_1, x_2-directions

Stanford University

Numerical Studies: Incompressible Taylor-Green Vortex

- For both grid types, the resulting ID approximation is rank 1, corresponding to a compression factor of 100.
  - ➢ **The SPID algorithm can achieve a large compression ratio even when there is not an underlying order to the the points of the spatial domain.**

- The error as a function of the number of domain partitions decreased as the number of physical partitions increased.
  - ➢ This behavior is typically seen due to the structure of the data matrix $\boldsymbol{A}$; adjacent row entries correspond to adjacent points in the spatial domain, and adjacent column entries correspond to adjacent points in the temporal domain. Thus, partitioning the data matrix allows for the extraction of additional low-rank behavior.
  - ➢ **Partitioning the spatial domain can increase both computational efficiency and the compression ratio without sacrificing accuracy.**

Numerical Studies: High-Order Navier-Stokes Solver

- Regent implementation is based on an existing solver used by NASA Ames Research Center [6].

- Sixth-order finite volume Navier-Stokes solver for a Cartesian grid, uses a seven-point stencil for flux reconstructions.

$$\frac{\partial \boldsymbol{q}}{\partial t} + \frac{\partial \boldsymbol{f_1}}{\partial x_1} + \frac{\partial \boldsymbol{f_2}}{\partial x_2} + \frac{\partial \boldsymbol{f_3}}{\partial x_3} = \frac{\partial \boldsymbol{g_1}}{\partial x_1} + \frac{\partial \boldsymbol{g_2}}{\partial x_2} + \frac{\partial \boldsymbol{g_3}}{\partial x_3}$$

- The solution is advanced in time using a low-storage third-order Runge-Kutta method.

$$\frac{d\boldsymbol{q}_j}{dt}\Delta V = -R(\boldsymbol{q}_j)$$

$$\boldsymbol{q}_j^{*} = \boldsymbol{q}_j^{(n)} - a_1 \frac{\Delta t}{\Delta V} R(\boldsymbol{q}_j^{(n)})$$

$$\boldsymbol{q}_j^{**} = \boldsymbol{q}_j^{(n)} - a_2 \frac{\Delta t}{\Delta V} R(\boldsymbol{q}_j^{(n)})$$

$$\boldsymbol{q}_j^{***} = \boldsymbol{q}_j^{*} - a_3 \frac{\Delta t}{\Delta V} R(\boldsymbol{q}_j^{**})$$

$$\boldsymbol{q}_j^{(n+1)} = \boldsymbol{q}_j^{*} - a_4 \frac{\Delta t}{\Delta V} R(\boldsymbol{q}_j^{***})$$

[6] Leffell, J. et al. (2016) Towards Efficient Parallel-in-Time Simulation of Periodic Flow. The 54th AIAA Aerospace Sciences Meeting.

Numerical Studies: High-Order Navier-Stokes Solver

- Because this code is high-order (a 7-point stencil) and uses an RK3 method, a **large amount of communication** is required for each time step.

- Below is the Legion Spy analysis for a single time step; each arrow corresponds to a task dependency that requires data communication from the stencil points.

RK stage 1
RK stage 2
RK stage 3

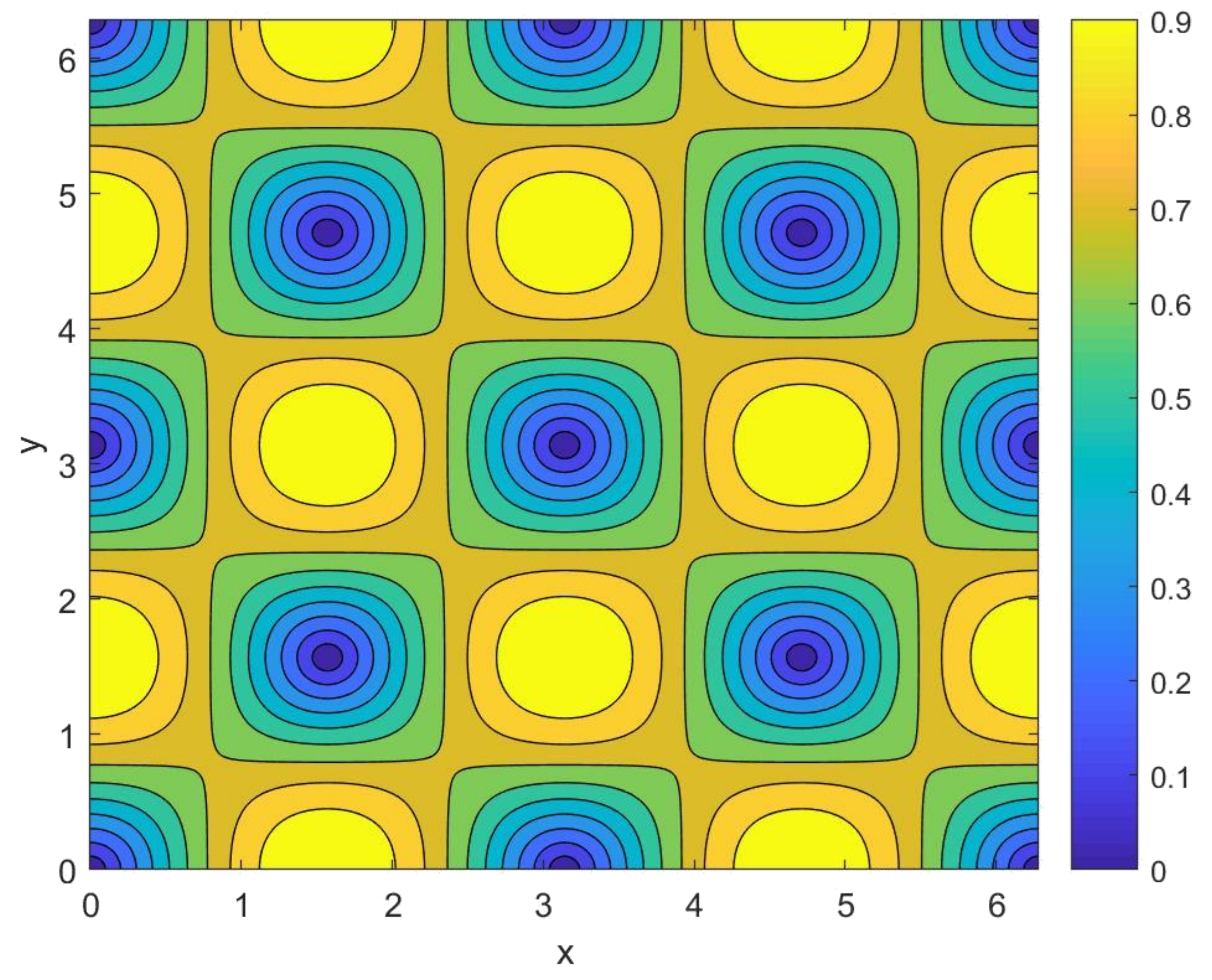
Stanford University

Numerical Studies: High-Order Navier-Stokes Solver

- A compressible, 3D Taylor-Green vortex was simulated to study the efficiency of the SPID algorithm embedded within a fluid solver.

- The domain was triply-periodic, with $0 \leq x, y, z \leq 2\pi$. The initial conditions used (from [7]) are:

$$\begin{cases} u(t_0) = u_0 \sin(x)\cos(y)\cos(z) \\ v(t_0) = -u_0 \cos(x)\sin(y)\cos(z) \\ w(t_0) = 0 \\ p(t_0) = p_0 + \frac{\rho_0}{16}(\cos(2x) + \cos(2y))(\cos(2z) + 2) \end{cases}$$



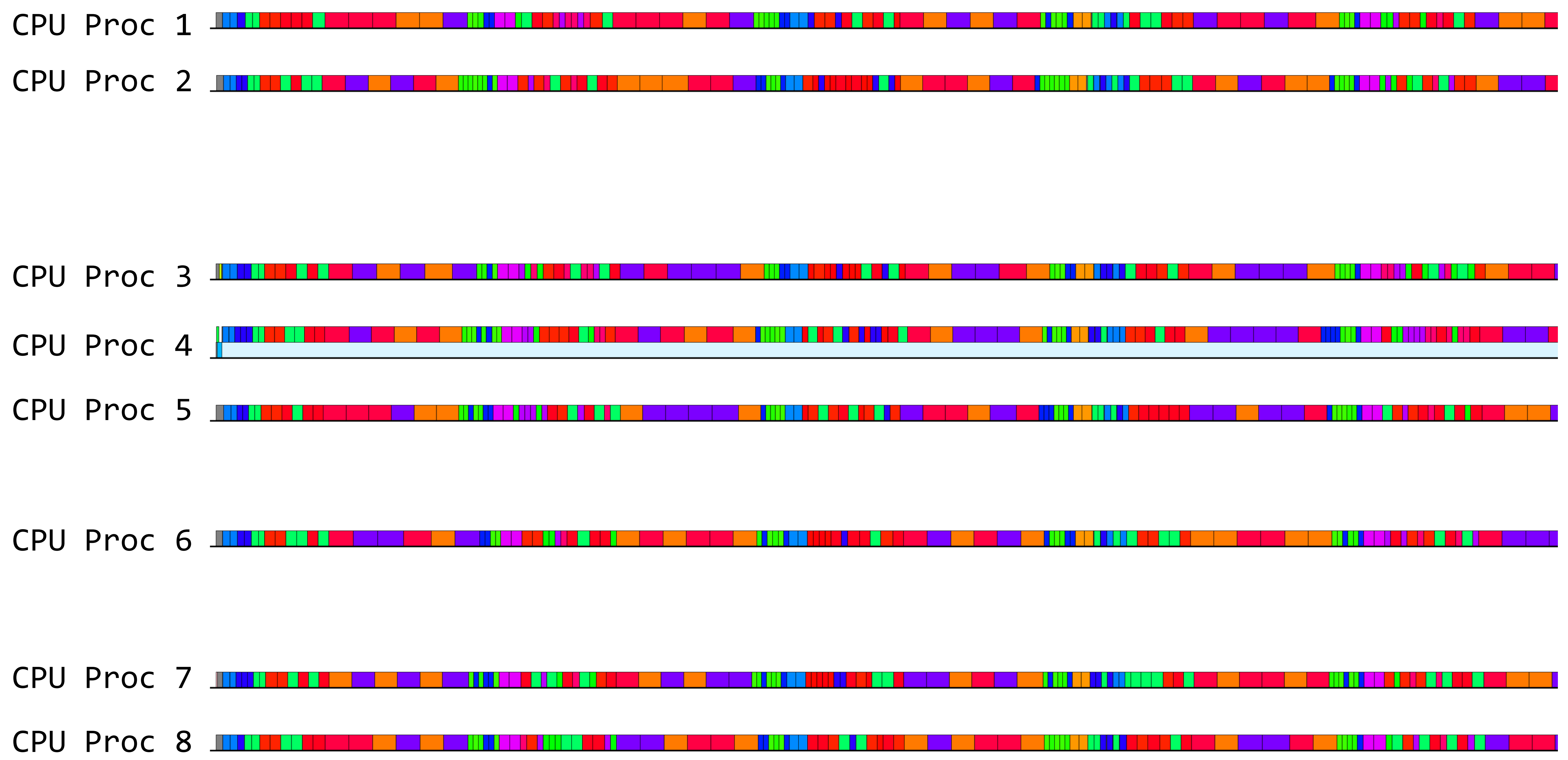
with $u_0 = 1$, $p_0 = 100$, $\rho_0 = 1$, and $T_0 = 300$ K.

- The viscosity was selected such that the Reynolds number of the flow was $\mathcal{O}(100)$.

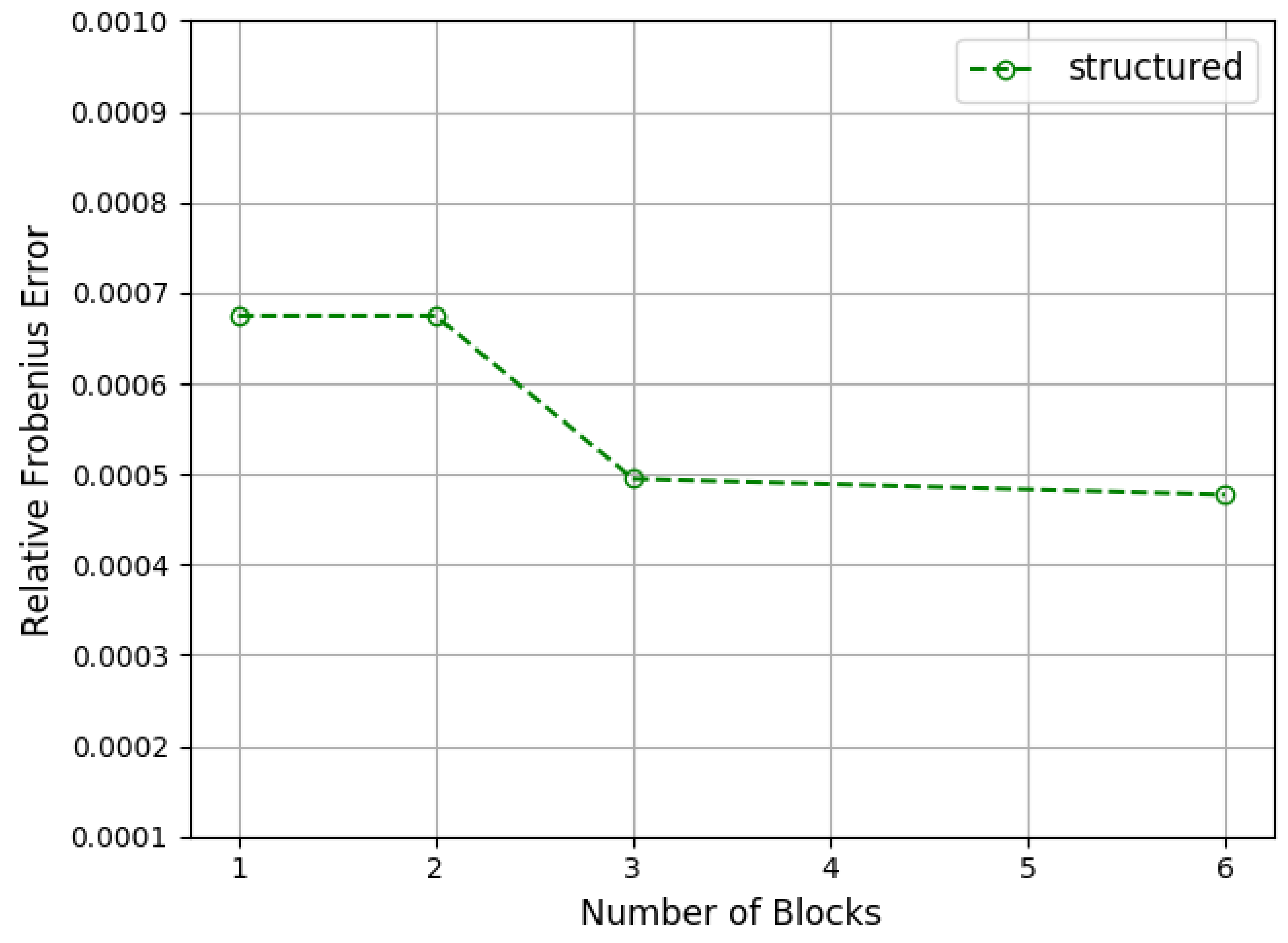
[7] Bull, J.R. & Jameson, A. (2014) Simulation of the Compressible Taylor Green Vortex using High-Order Flux Reconstruction Schemes. *AIAA Aviation.*

Numerical Studies: High-Order Navier-Stokes Solver

- The SPID algorithm was incorporated into the Navier-Stokes solver (NSID). Simulations were run for a 3D compressible Taylor-Green vortex with 48^3 domain points and for 100 time steps.
  - The u error as a function of partitions in the spatial direction was computed. As expected, the number of partitions did not increase the error of the ID approximation.
  - The resulting rank of the ID approximation in each case was 1.



Profiler Results for Node 0 of a 4-node run

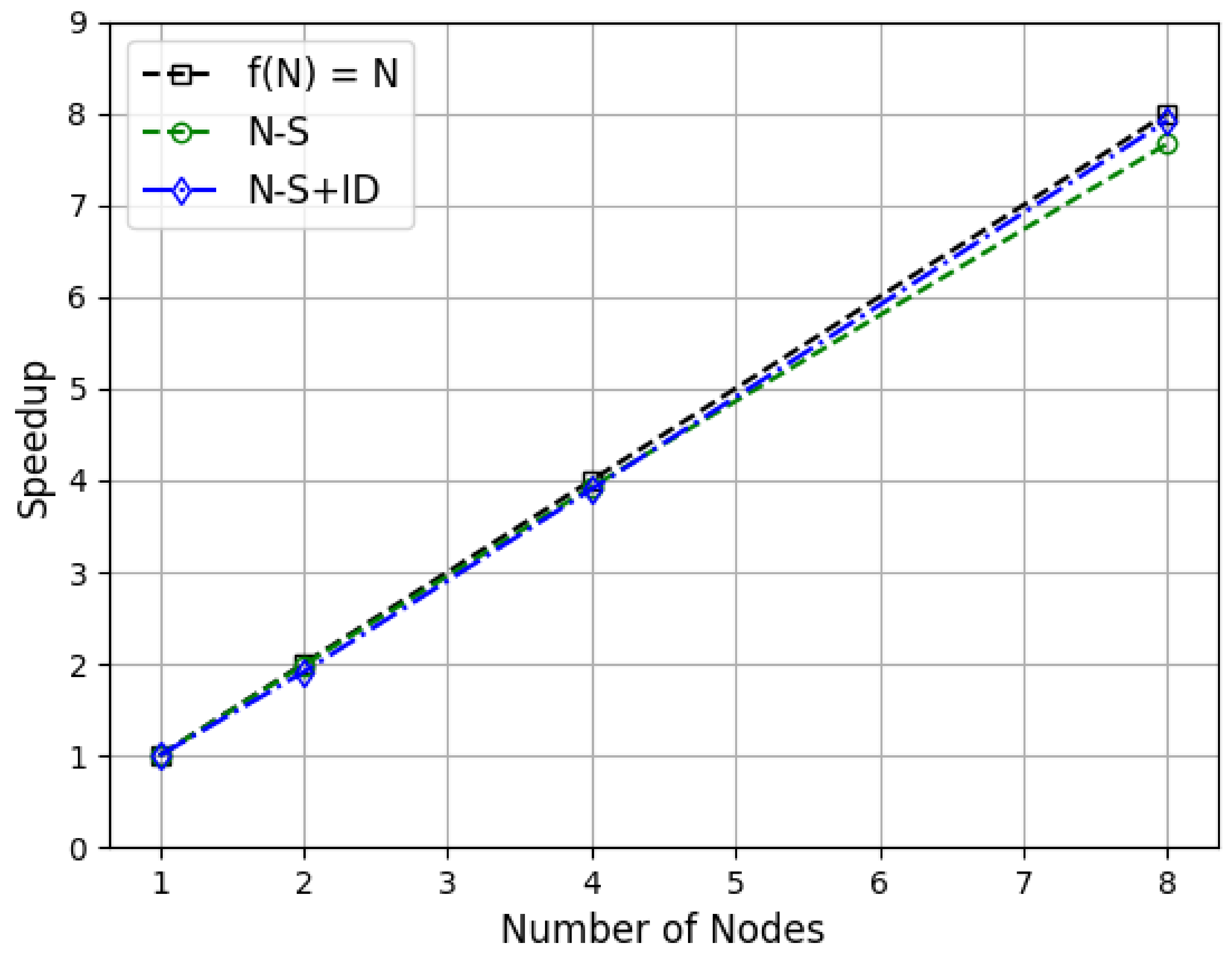


u error as a function of blocks in the x_1, x_2, x_3 - directions

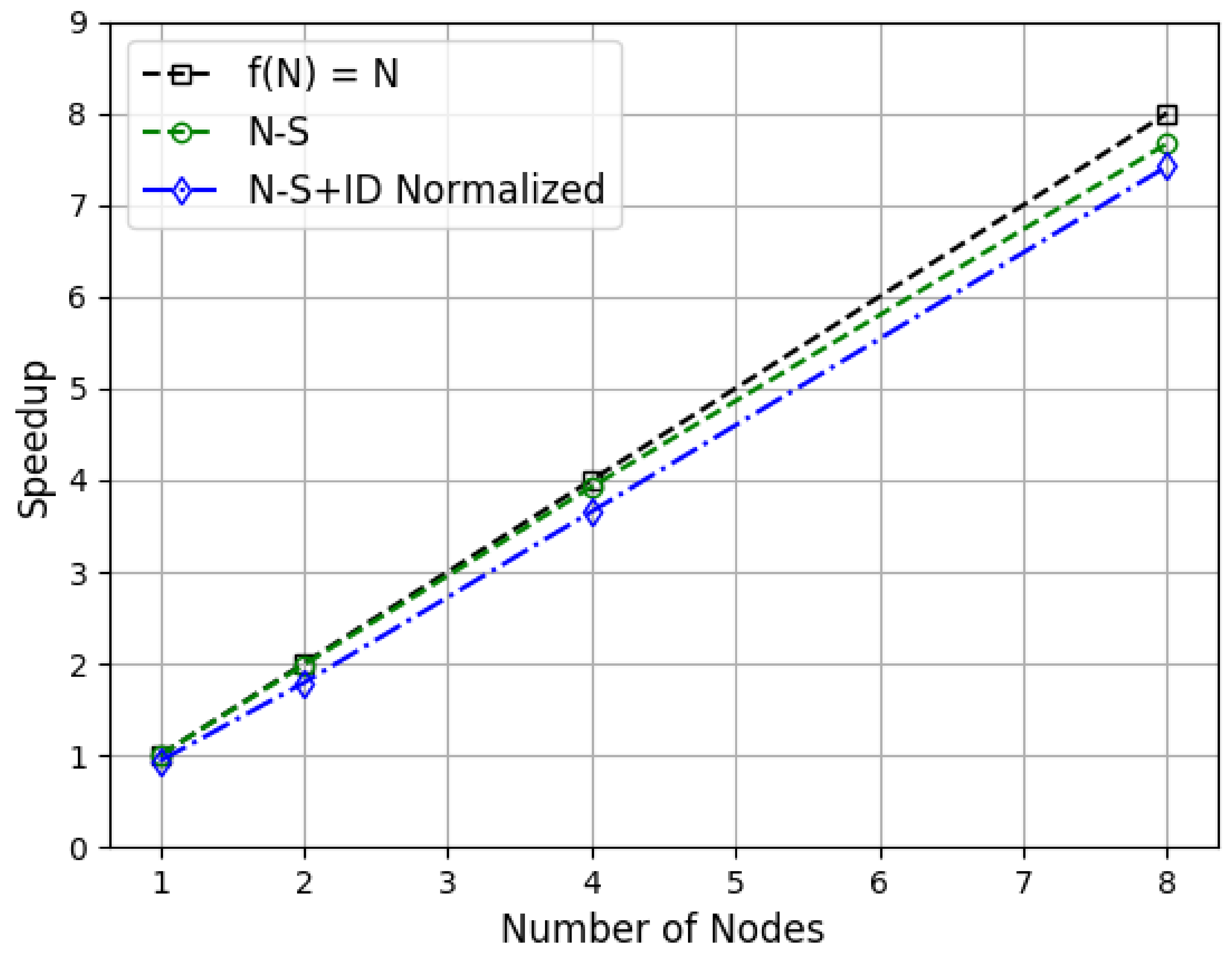
Stanford University

Numerical Studies: High-Order Navier-Stokes Solver

- Strong scaling results for a 128^3 domain with 4^3 physical partitions and 10 time steps are shown below.
  - ➢ Though both codes scaled well, the NSID code scaled slightly better than the NS code.
  - ➢ The increase in runtime of the NSID code was, at most, 10% that of the NS code.
  - ➢ **Introducing the ID algorithm did not have negative impacts on scaling or overall runtime.**



Strong Scaling



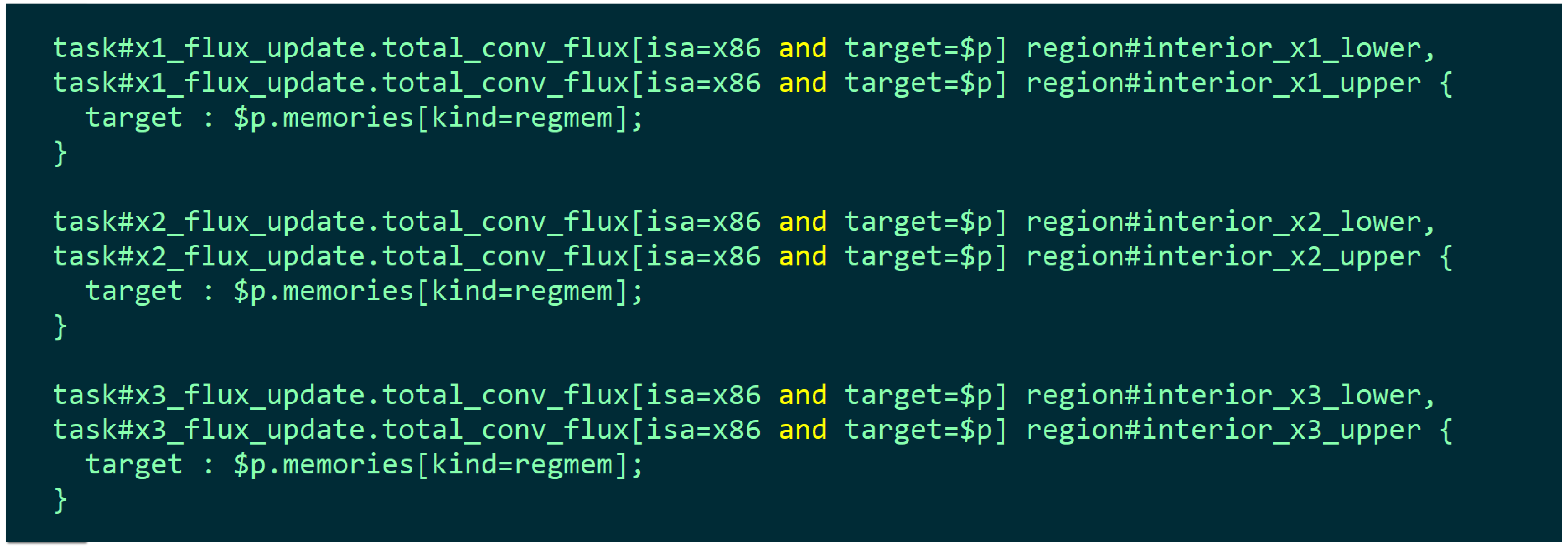
Strong Scaling normalized to the 1-node NS runtime

Conclusions and Future Work

- Interpolative decomposition methods allow for easily implemented lossy data compression, and they are well-suited to a task-parallel environment.

- The SPID algorithm was directly incorporated into a high-order Navier-Stokes solver, and were able to provide a high amount of compression for a Taylor-Green vortex test case.

- Current work is on developing a custom mapper for the Navier-Stokes ID solver to more precisely control the distribution of the CFD and ID tasks across various architectures.



```
task#x1_flux_update.total_conv_flux[isa=x86 and target=$p] region#interior_x1_lower,
task#x1_flux_update.total_conv_flux[isa=x86 and target=$p] region#interior_x1_upper {
  target : $p.memories[kind=regmem];
}

task#x2_flux_update.total_conv_flux[isa=x86 and target=$p] region#interior_x2_lower,
task#x2_flux_update.total_conv_flux[isa=x86 and target=$p] region#interior_x2_upper {
  target : $p.memories[kind=regmem];
}

task#x3_flux_update.total_conv_flux[isa=x86 and target=$p] region#interior_x3_lower,
task#x3_flux_update.total_conv_flux[isa=x86 and target=$p] region#interior_x3_upper {
  target : $p.memories[kind=regmem];
}
```

- This work will be extended to ensemble simulations.

Stanford University

Questions?

email: hpacella@stanford.edu

Legion: http://legion.stanford.edu/
Regent: http://regent-lang.org/
exascale computing at Stanford: http://exascale.stanford.edu

Stanford University