Dynamically Load-balanced *p*-adaptive Discontinuous Galerkin Methods using Charm++

Weizhao Li, Adity K. Pandare, Hong Luo, Jozsef Bakosi and Jacob Waltz

Charm++ Workshop 2021
October. 18th, 2021

Outline

- Background
- Governing Equations
- DG Discretization Formulations
  - Weak formulation
- Adaptive Strategies
  - Error indicator based *p*-adaptation
  - Protective layer refinement
- Computation Process
- Numerical Results
- Conclusions

Quinoa

- Computational tools for fluid dynamics
- Written in modern C++
- Production-style, rigorously tested
- Numerical solver for single-material and multi-material hydrodynamics
- Asynchronous, distributed-memory parallel programming
- Fully unstructured tetrahedral mesh support
- Dynamic load balancing and automatic object migration using Charm++
- Open source: https://github.com/quinoacomputing/quinoa

Governing Equations

- The compressible Euler equations can be represented as

$$\frac{\partial \boldsymbol{U}}{\partial t} + \frac{\partial \boldsymbol{F}_k}{\partial x_k} = 0$$

where

$$\boldsymbol{U} = \begin{bmatrix} \rho \\ \rho u_i \\ \rho E \end{bmatrix}, \qquad \mathbf{F} = \begin{bmatrix} \rho u_k \\ \rho u_i u_k + p\delta_{ik} \\ u_{ik}(\rho E + p) \end{bmatrix}$$

- The pressure can be evaluated according to

$$p = (\gamma - 1)\rho\left(E - \frac{1}{2}u_i u_k\right)$$

where γ is the ratio of specific heats

Discontinuous Galerkin Formulation

2.1 Weak formulations

- The weak formulation can be obtained by multiplying test function W and performing integration by parts,

$$\int_{\Omega} \frac{\partial U_h}{\partial t} W_h d\Omega + \int_{\Gamma} F_k(U) n_k W_h d\Gamma - \int_{\Omega} F_k(U) \frac{\partial W_h}{\partial x_k} d\Omega = 0$$

- Assume our high order solution within the cell is represented as,

$$U_h(x,t) = \sum_{j=1}^{N} u_j(t) B_j(x)$$

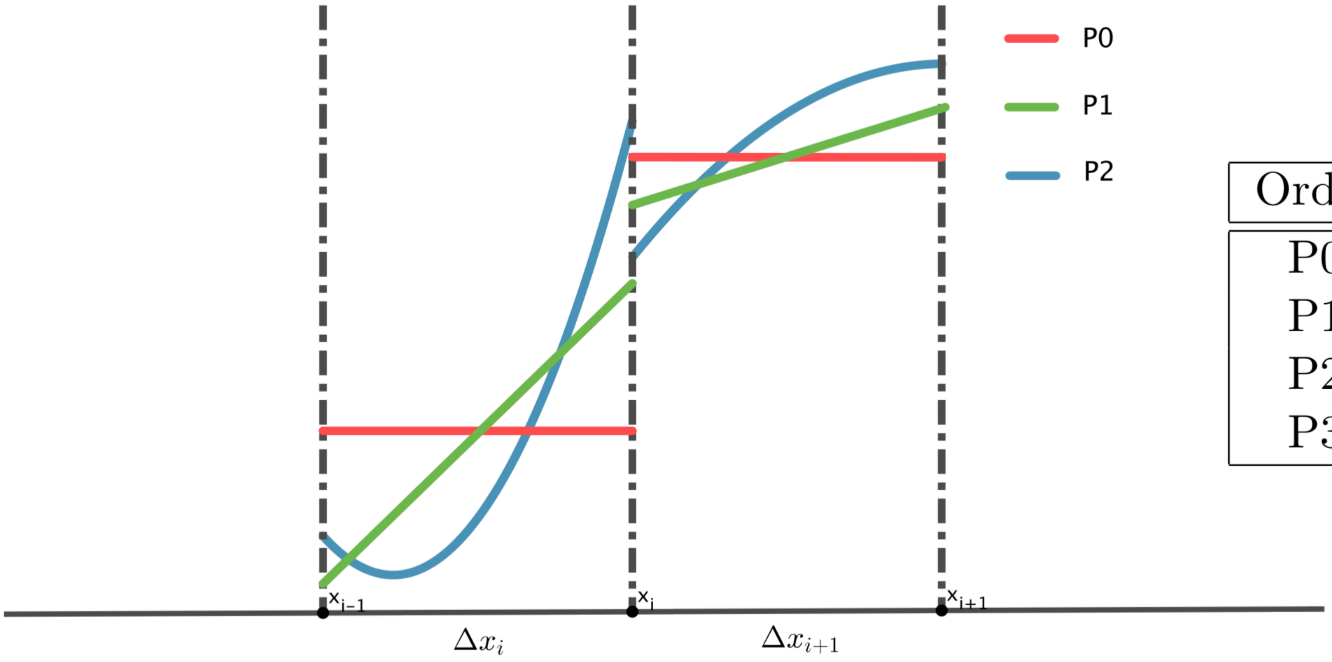
- Apply the solution equation to the above formulation,

$$\left(\int_{\Omega_e} B_i B_j d\Omega \right) \frac{du_j}{dt} + \int_{\Gamma_e} F_k(U_h) n_k B_i d\Gamma - \int_{\Omega_e} F_k(U_h) \frac{\partial B_i}{\partial x_k} d\Omega = 0, 1 \le i \le N$$

Discontinuous Galerkin Formulation

2.1 Weak formulations

- The DG solution: $U_h(x,t) = \sum_{j=1}^{N} u_j(x) B_j(x)$



Order	B_i
P0	$[1]$
P1	$[1, \tilde{x}]$
P2	$[1, \tilde{x}, \tilde{x}^2]$
P3	$[1, \tilde{x}, \tilde{x}^2, \tilde{x}^3]$

Adaptive Strategies

3.1 Error indicator based *p*-adaptation

- Use a posteriori local error indicator to determine where the order of element solution should be refined or coarsened
- The spectral decay indicator is defined as

$$\eta_k = \frac{\int_\Omega \left(\rho_p - \rho_{p-1}\right)^2 d\Omega}{\int_\Omega {\rho_p}^2 d\Omega}$$

Where ρ_p and ρ_{p-1} represent the numerical density with the polynomial order of p and $p - 1$.

- After evaluating adaptive indicators, the following adaptation criterion is used to determine *p*-refinement or coarsening:

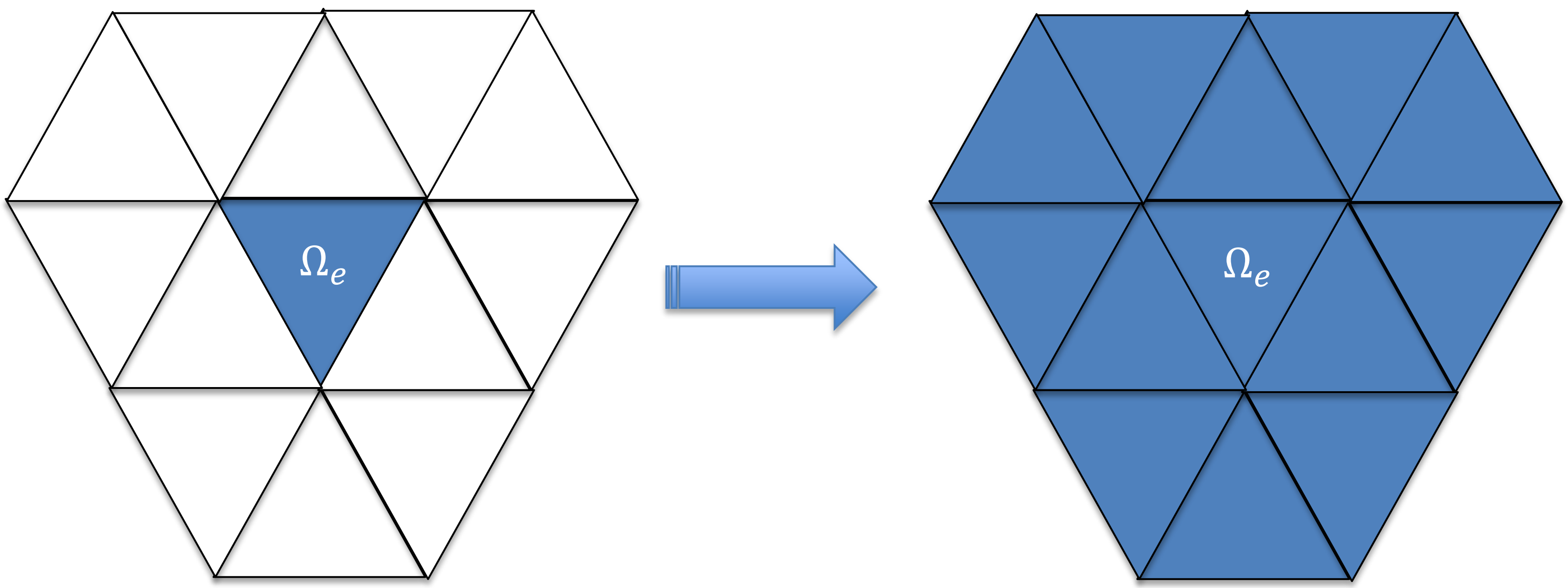
$$\begin{cases} \eta_k \geq \varepsilon_H \Longrightarrow Refine\ if\ p_k < p_{max} \\ \eta_k < \varepsilon_L \Longrightarrow Coarse\ if\ p_k > p_{min} \end{cases}$$

Where ε_H and ε_L are user-input thresholds ($\varepsilon_H > \varepsilon_L$). Both thresholds are case-dependent parameters.

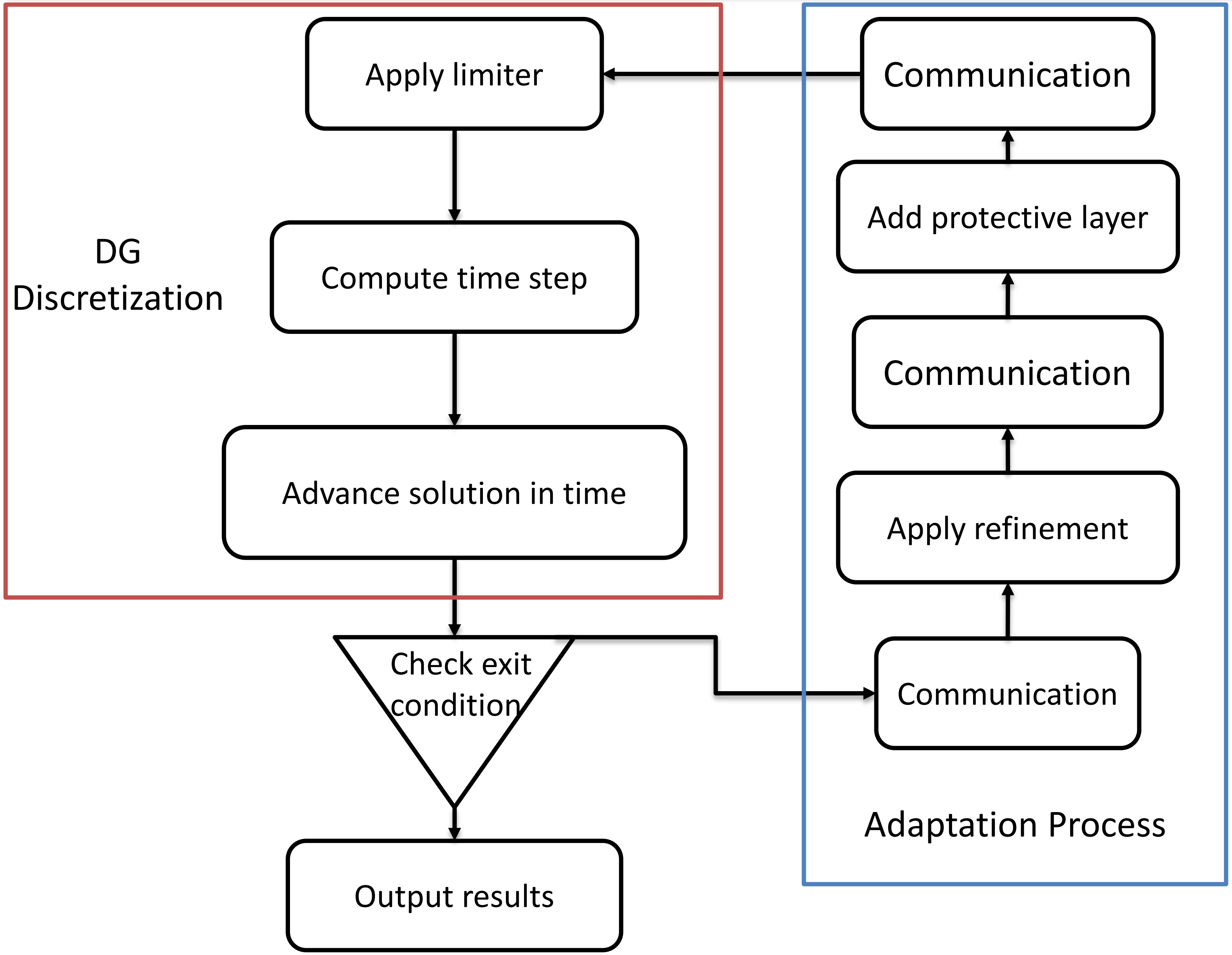
Adaptive Strategies

3.2 Protective layer refinement

- By adding this protective layer, refine all the nodal neighboring elements of the refined element in Ω_e

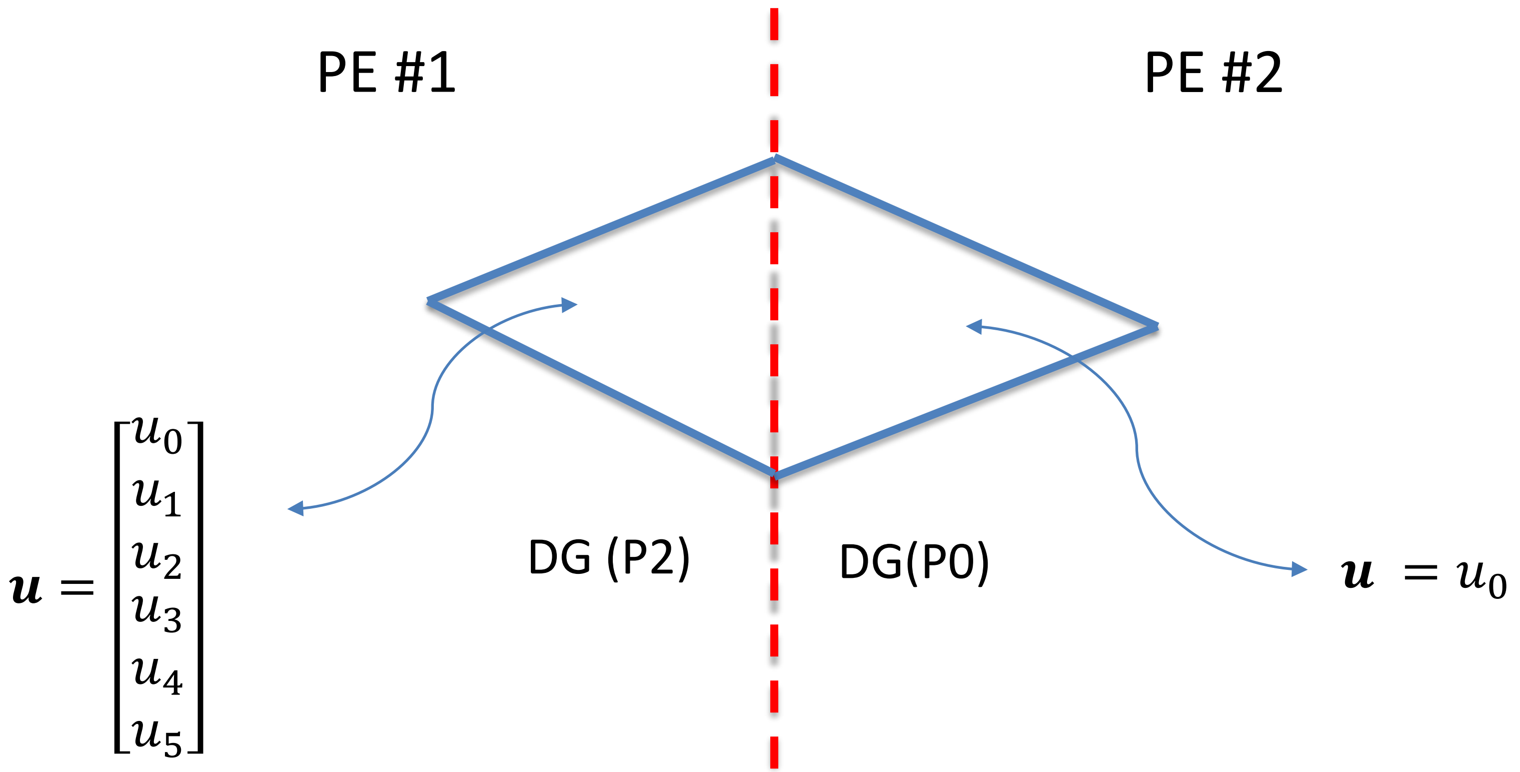


Computation Process



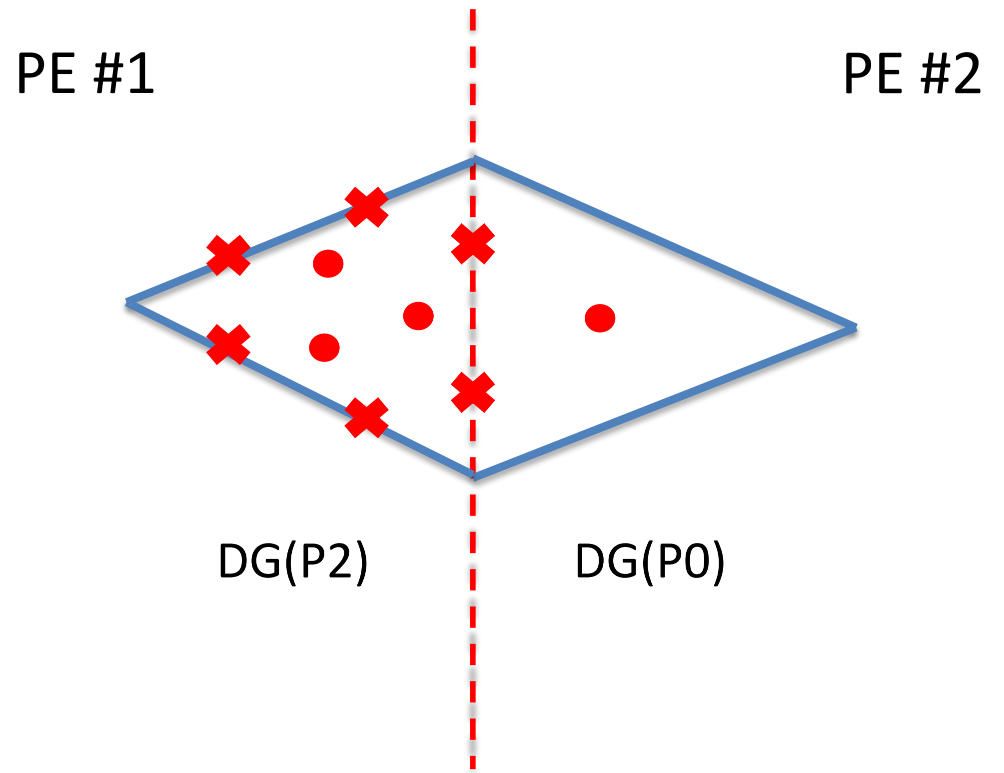
Adaptive Strategies

3.3 Sources of unbalanced load distribution



Adaptive Strategies

3.3 Sources of unbalanced load distribution



$\Longrightarrow$ *Dynamic Load Balancing*

Numerical Results

4.1 Sod shocktube problem

- The initialization condition is given as

$$(\rho, p, u)_L = (1.0,1.0,0.0)$$
$$(\rho, p, u)_R = (1.0,1.0,0.0)$$

- The mesh with 11200 tetrahedra is used here

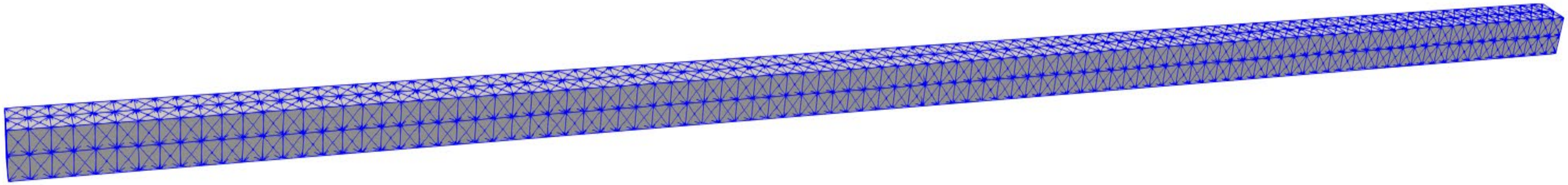


Fig. 1 Mesh for sod shocktube

Numerical Results

4.1 Sod shocktube problem

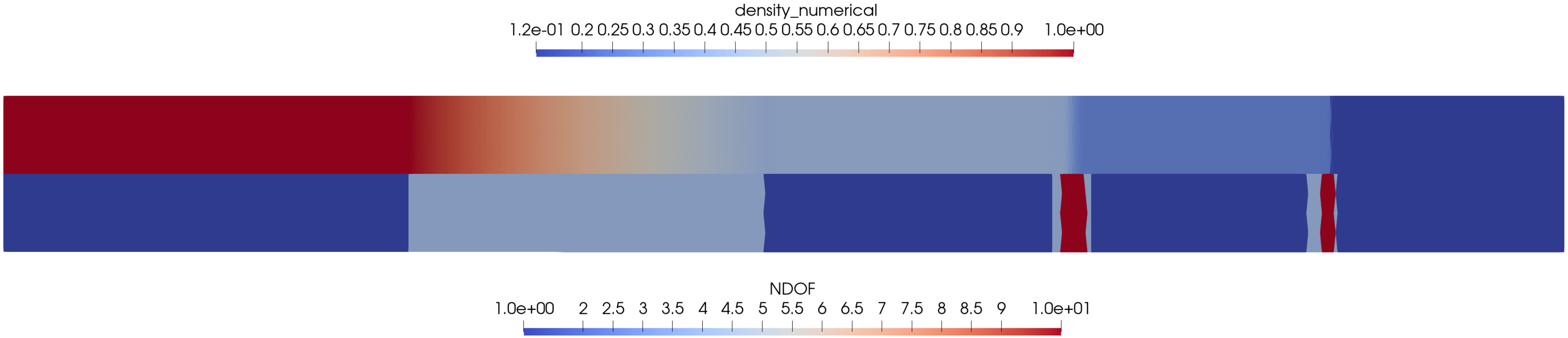


Fig. 2 Numerical distribution for sod shocktube

Numerical Results

4.1 Sod shocktube problem

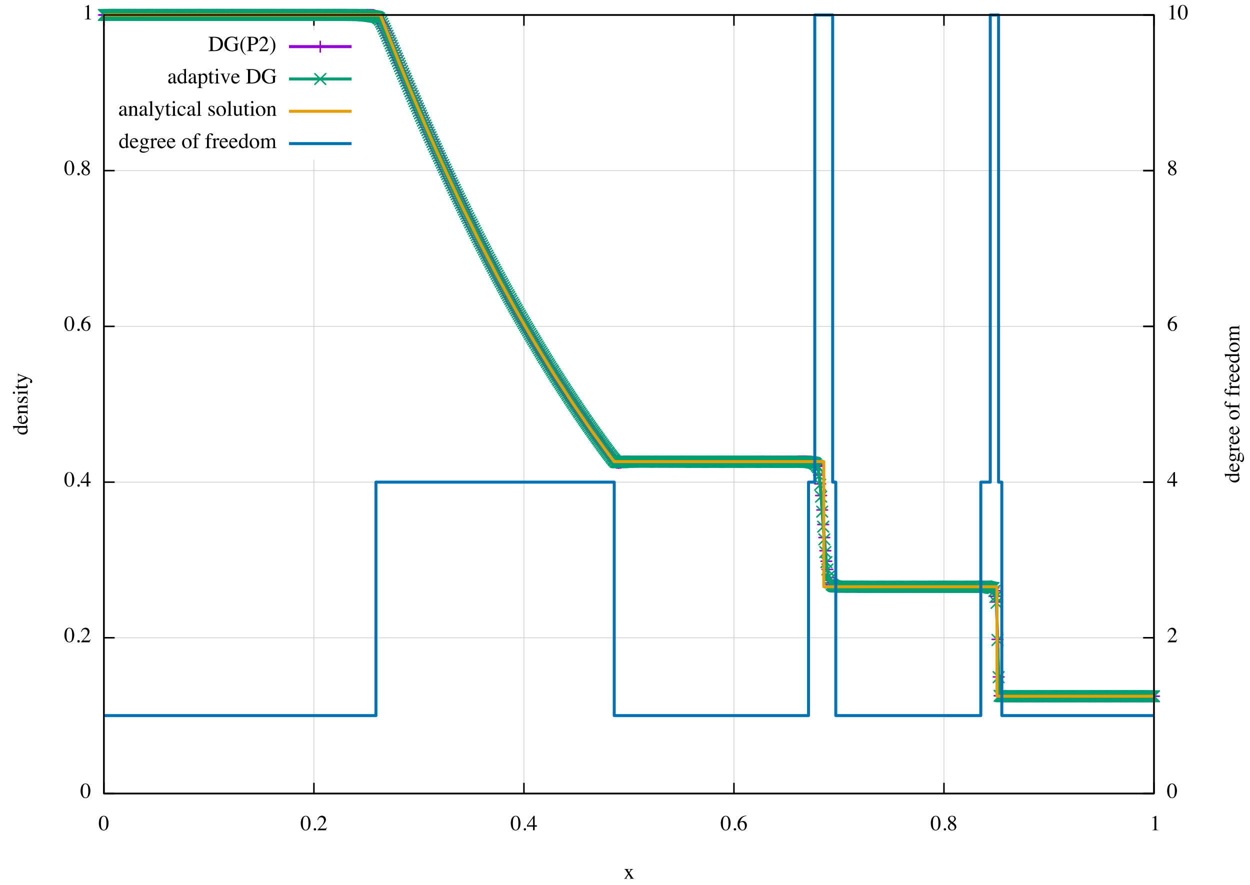


Fig. 3 Numerical distribution for sod shocktube in 1D

Numerical Results

4.1 Sod shocktube problem

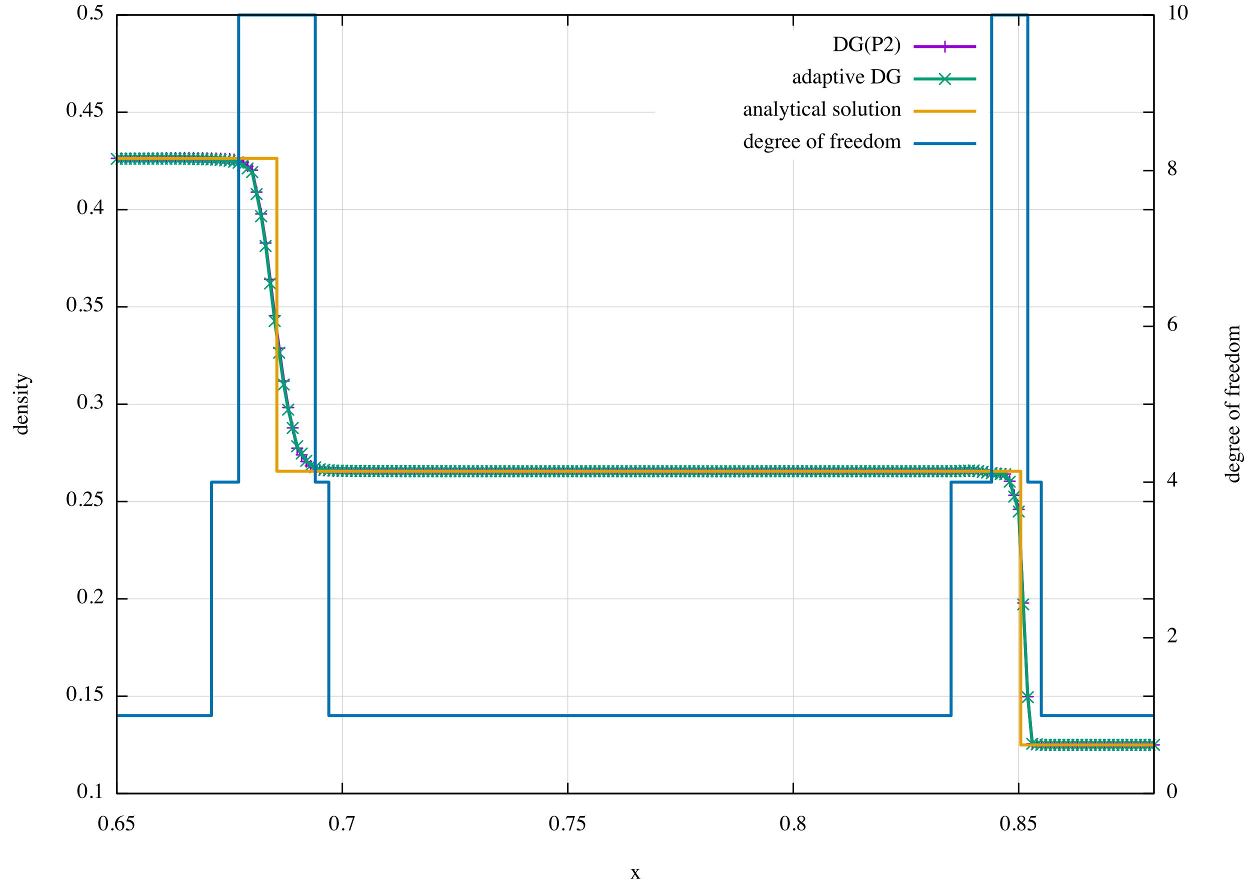


Fig. 4 Numerical distribution for sod shocktube near discontinuities

Numerical Results

4.1 Sod shocktube problem

Case	Configuration	Wall-clock time (m:s)	Speedup relative to case 2
1	128 cores, 128 partitions, DG(P2)	36:44	
2	128 cores, 128 partitions, *p*-adaptive DG	33:52	
3	128 cores, 426 partitions, *p*-adaptive DG	16:56	2.0x
4	128 cores, 426 partitions, *p*-adaptive DG with load balancer	16:17	2.1x

Table 1. Wall-clock time table for sod shocktube

Numerical Results

4.2 Sedov blast problem

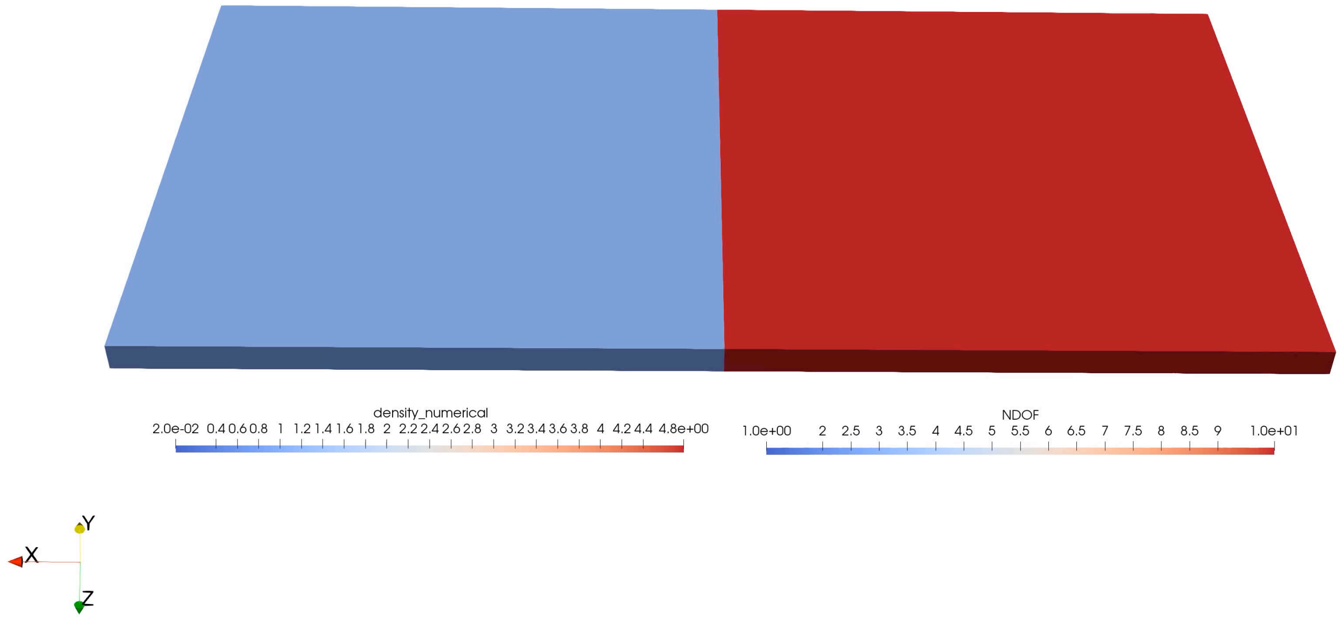
- Sedov blast testcase describes the flow with a strong spherical shock wave.
- The initialization condition is given as

$$u_x = 0$$
$$\rho(x_i) = 1$$
$$p = \begin{cases} 783.4112 & if\ x_i < 0.05 \\ 0 & otherwise \end{cases}$$

- The mesh with 29k tetrahedra is used here
- The goal of this testcase is to assess the capability to capture the strong discontinuities.

Numerical Results

4.2 Sedov blast problem



Numerical Results

4.2 Sedov blast problem

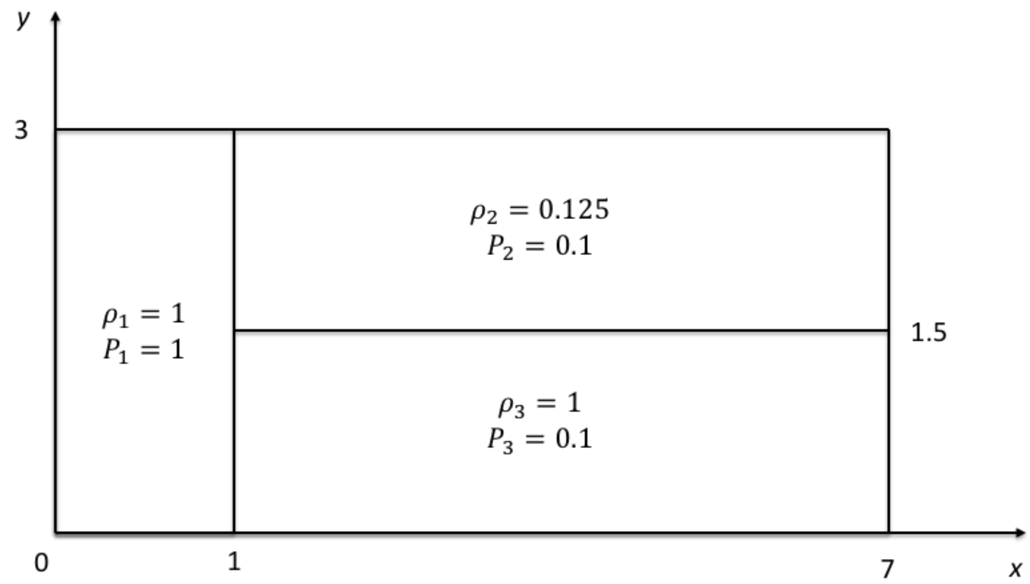
Case	Configuration	Wall-clock time (h:m:s)	Speedup relative to case 3
1	64 cores, 64 partitions, DG(P2)	5:00:29	
2	64 cores, 319 partitions, *DG(P2)*	5:26:5	
3	64 cores, 64 partitions, *p*-adaptive DG	3:13:29	
4	64 cores, 319 partitions, *p*-adaptive DG	1:35:21	2.0x
5	64 cores, 319 partitions, *p*-adaptive DG with load balancer	1:21:42	2.4x

Table 2. Wall-clock time table for Sedov blast at t = 0.01

Numerical Results

4.3 Triple point problem

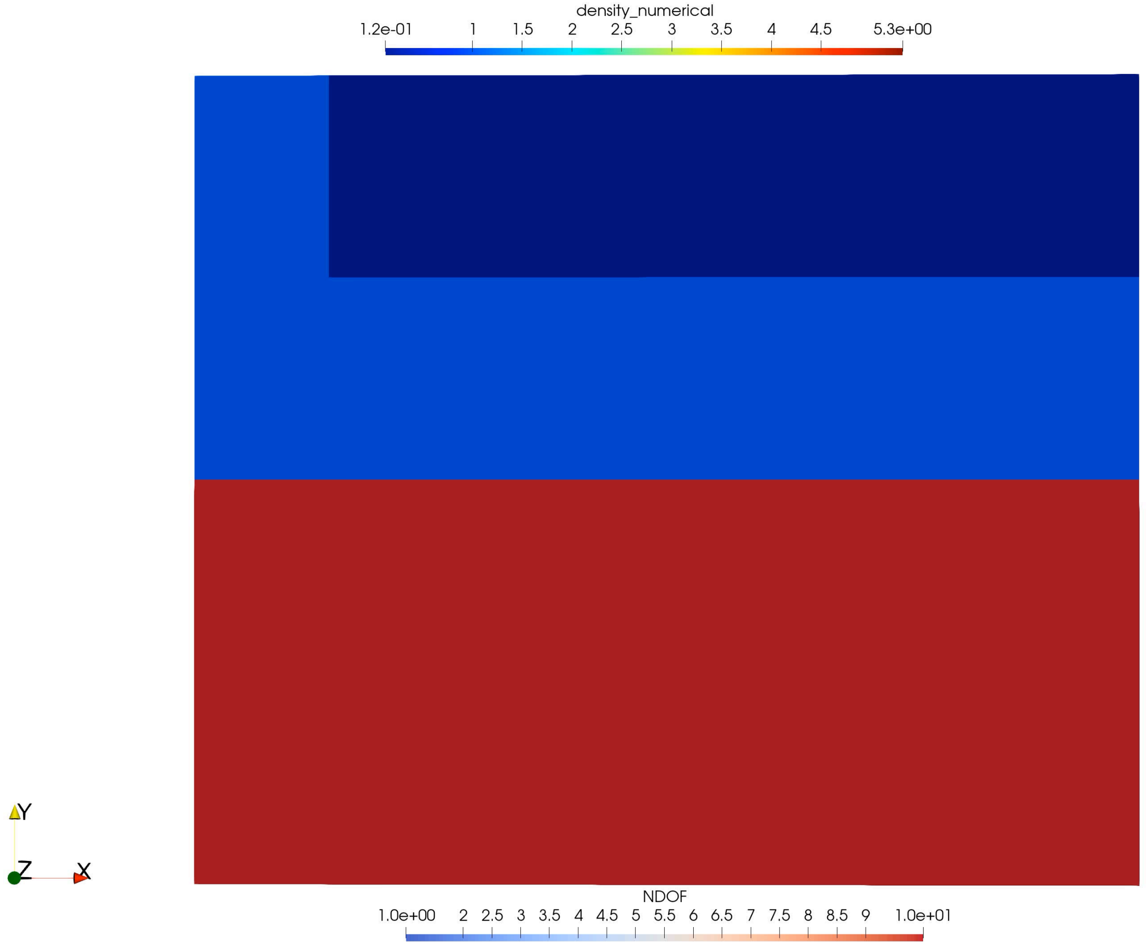
- The triple point problem is three-state two-dimensional Riemann problem
- The initialization condition is given as



- The mesh with 687085 tetrahedra is used here

Numerical Results

4.3 Triple point problem



Numerical Results

4.3 Triple point problem

Case	Configuration	Wall-clock time (h:m:s)	Speedup relative to case 3
1	32 cores, 32 partitions, DG(P2)	5:17:48	
2	32 cores, 32 partitions, *p*-adaptive DG	6:16:12	
3	32 cores, 319 partitions, *p*-adaptive DG	3:37:43	1.7x
4	32 cores, 319 partitions, *p*-adaptive DG with load balancer	2:4:29	3.0x

Table 3. Wall-clock time table for triple point problem at t = 1

Numerical Results

4.3 Triple point problem

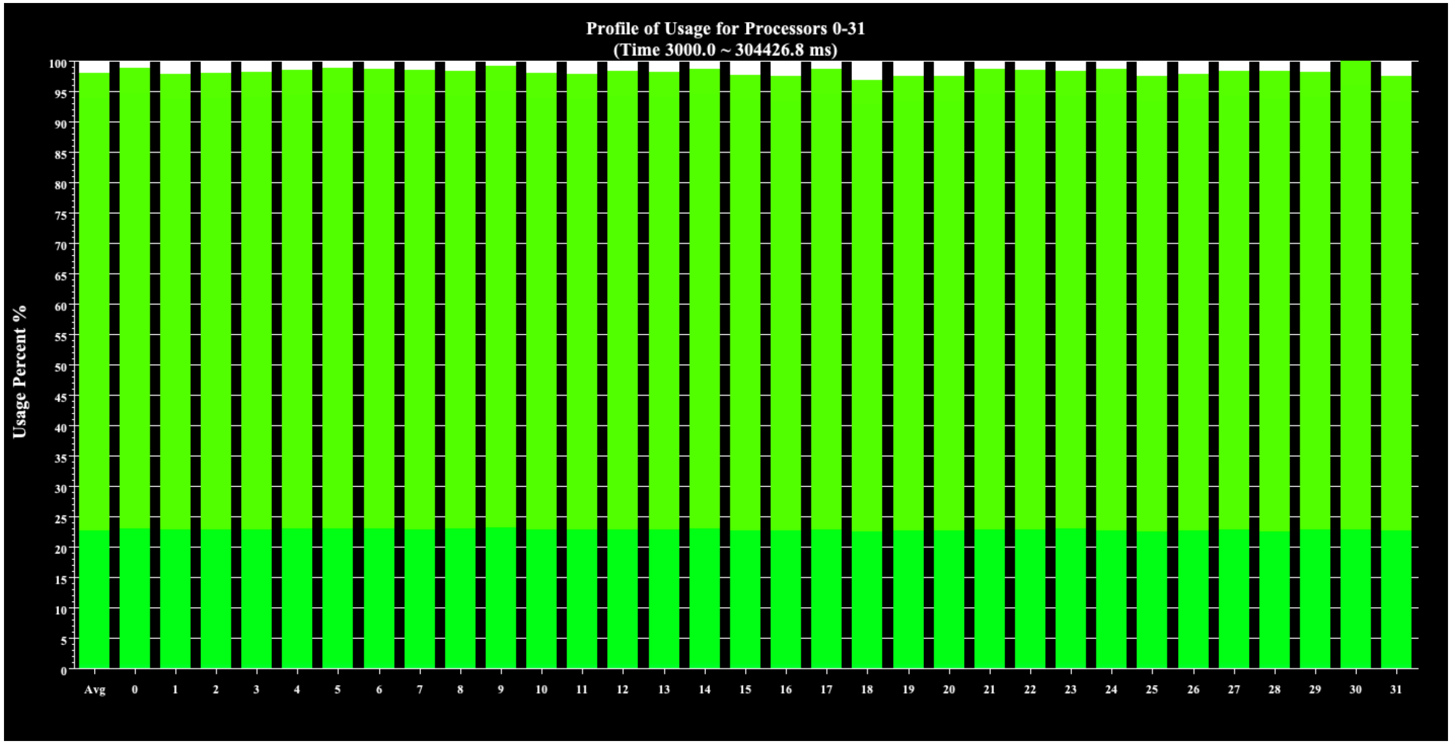


Fig. 5 Usage profile for DG(P2) with 32 cores

Numerical Results

4.3 Triple point problem

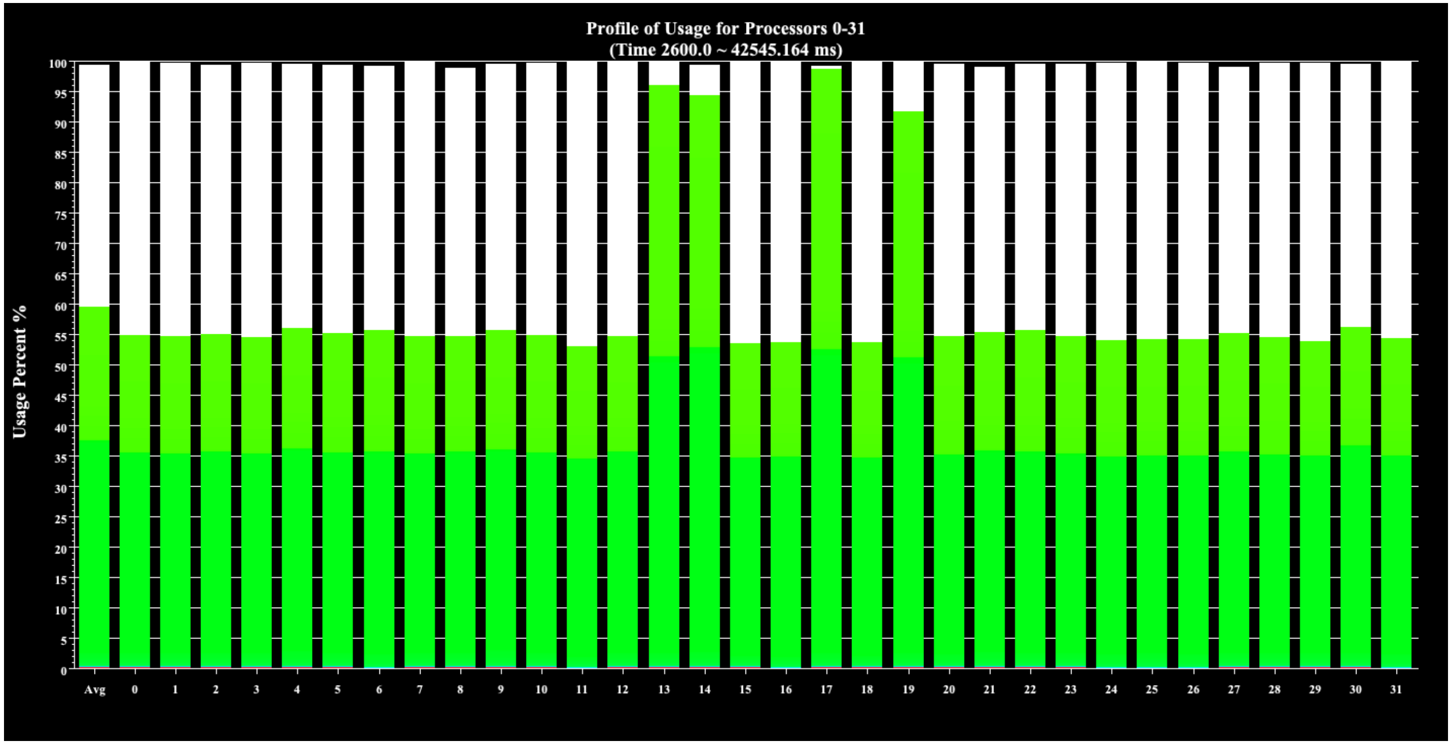


Fig. 6 Usage profile for *p*-adaptive DG with with 32 cores

Numerical Results

4.3 Triple point problem

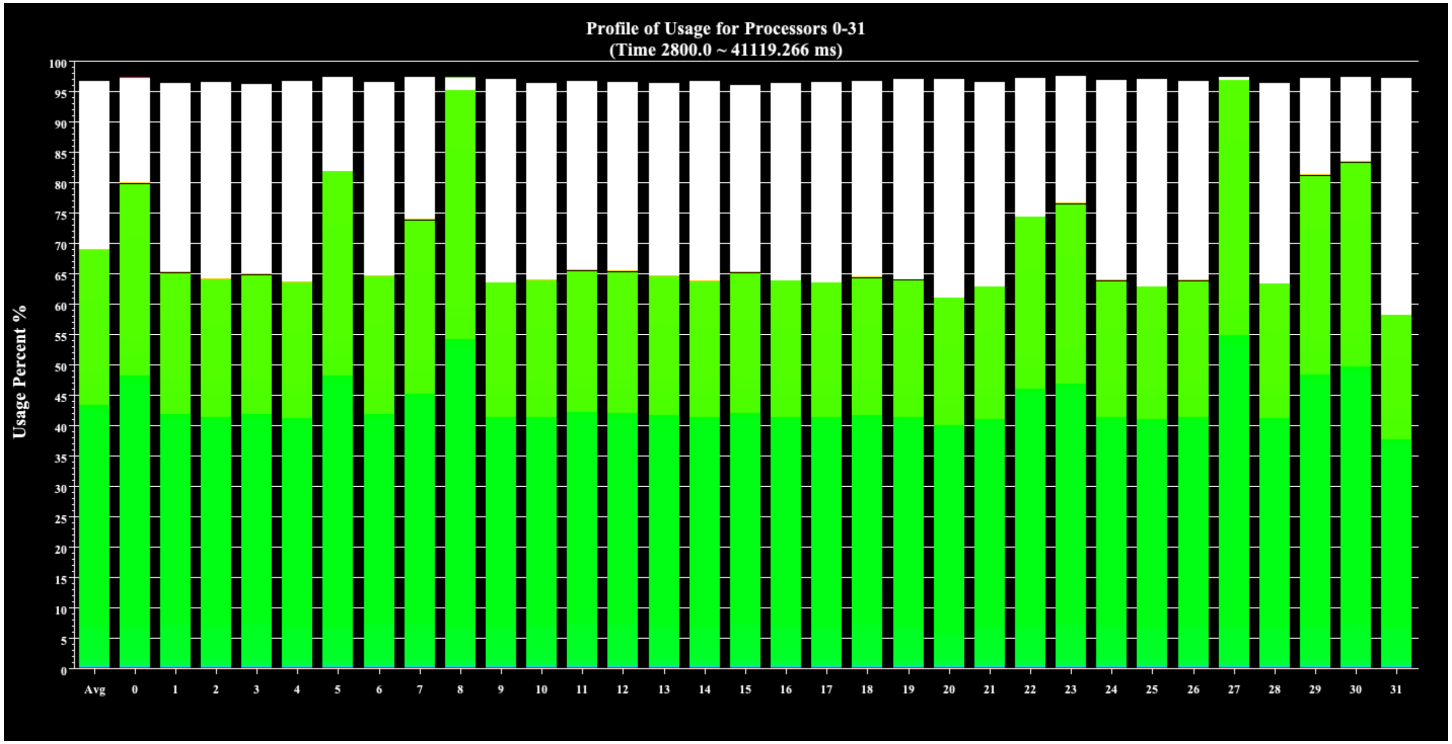


Fig. 7 Usage profile for *p*-adaptive DG with over-decomposition

Numerical Results

4.3 Triple point problem

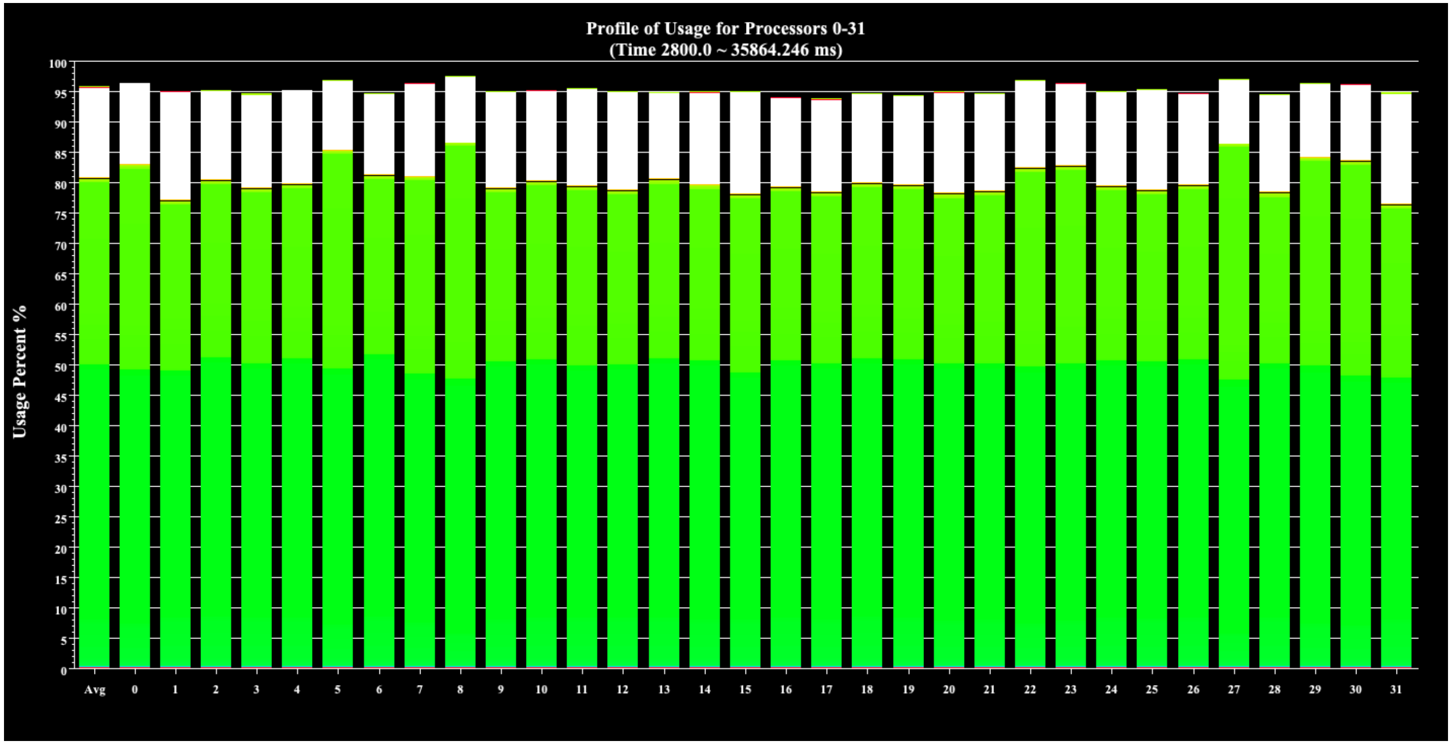


Fig. 8 Usage profile for *p*-adaptive DG with load balancer

Summary

- A *p*-adaptive DG method is developed.
- The developed adaptive method introduces unbalanced load distributions.
- The adaptive scheme combined with load balancing techniques significantly increase the computation efficiency.
- More complex numerical methods will be implemented within this parallel structure to maximize the benefits of dynamic load balancing technique.