# Design and Optimization of OpenFOAM-based CFD Applications for Hybrid and Heterogeneous HPC Platforms

**Amani AlOnazi**[*], David E. Keyes[*], Alexey Lastovetsky[†], Vladimir Rychkov[†]

[*]Extreme Computing Research Center, KAUST, Thuwal, Saudi Arabia,
[†]Heterogeneous Computing Laboratory, UCD, Dublin, Ireland,

*amani.alonazi@kaust.edu.sa*

May 21, 2014

# Overview

# Motivation

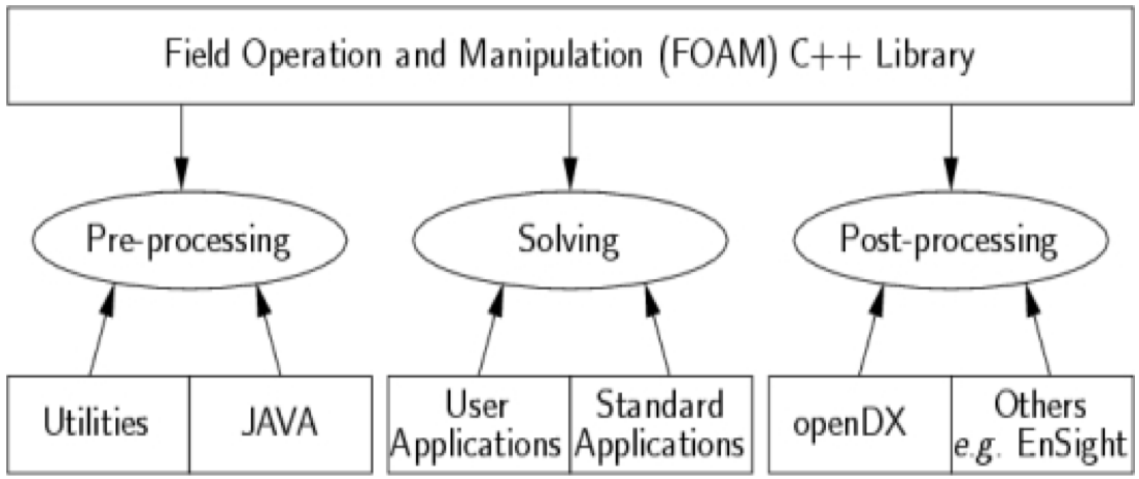
- Hardware changes have to be taken into accounts
  - Parallelism and heterogeneity in modern HW
- Per-processor performance on heterogeneous systems
- Algorithms and codes have to be redesigned

▷ The heterogeneity of these platforms leads to several challenges and much contemporary attention is devoted to new software solutions. This trend in the HPC platforms invites redesign of the CFD packages or the algorithms themselves to use these platforms efficiently.

" I would rather have today's algorithms on yesterday's computers than vice versa."

# OpenFOAM CFD Package

- Open source Field Operation And Manipulation (OpenFOAM) is a library written in C++ used to solve PDEs
- It covers a wide range of solvers employed in CFD, such as Laplace and Poisson equations, incompressible flow, multiphase flow, and user defined models
- Free, open source, parallel, modular, and flexible
- Large, user-driven support community



# OpenFOAM Parallel Approach

## MPI Bulk Synchronous Parallel Model

- OpenFOAM uses MPI to provide parallel multi-processors functionality
- The mesh and its associated fields are divided into subdomains and allocated to separate processors (domain decomposition)

## Domain Decomposition Methods

It provides different utilities for partitioning the domain, such as:

- Simple
- Hierarchal
- METIS
- SCOTCH

The communication between the subdomains is implicitly handled within the package

# OpenFOAM Selected Applications

**icoFoam** The incompressible laminar Navier-Stokes equations can be solved by icoFoam, which applies the PISO algorithm in time stepping loop.

$$\nabla \cdot \mathbf{u} = 0$$

$$\frac{\partial \mathbf{u}}{\partial t} + \nabla \cdot (\mathbf{uu}) - \nabla \cdot (\nu \nabla \mathbf{u}) = -\nabla p$$

- $p$: **CG**
- $u$: **Bi-CGSTAB**

**laplacianFoam** The solver is used to find the solution of the Laplacian equation. The equation contains one variable, a passive scalar, for instance, a temperature, $T$.

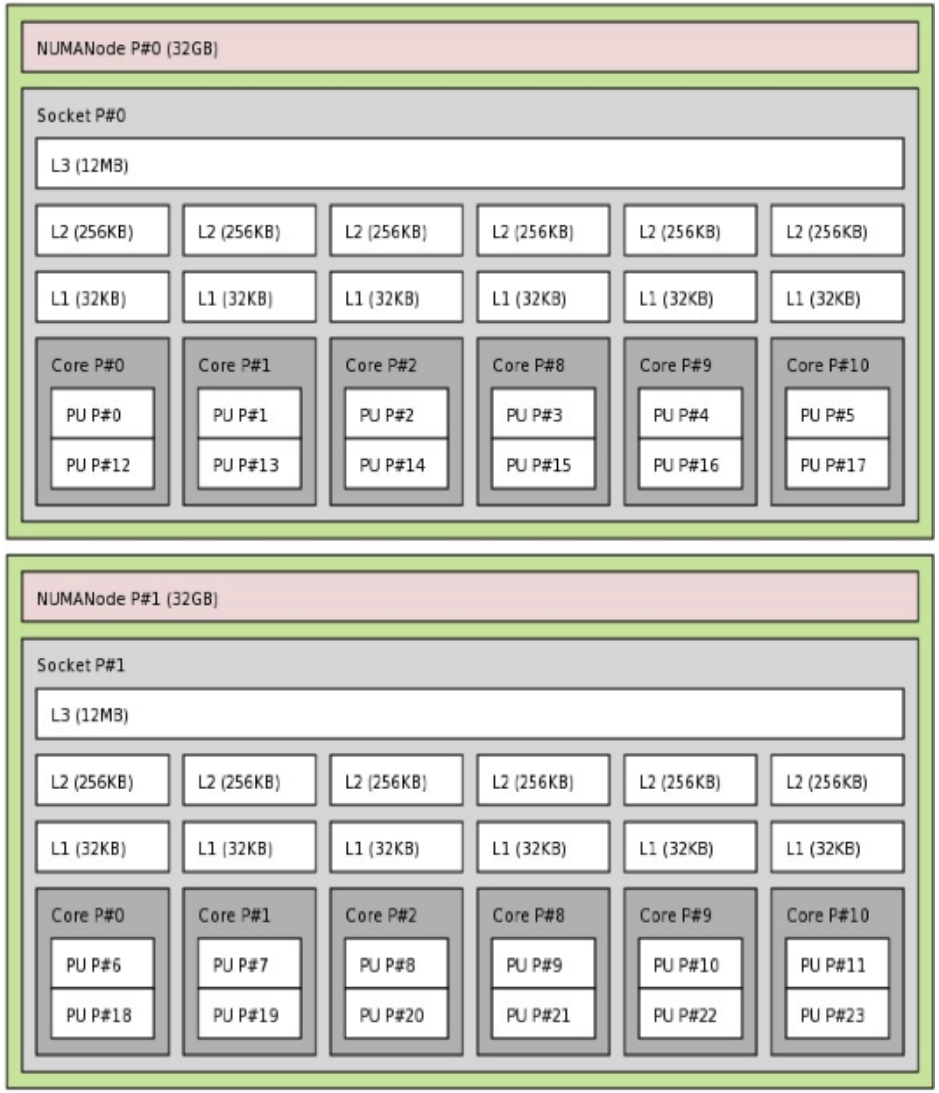
$$\frac{\partial T}{\partial t} - \nabla^2 (D_T \cdot \mathbf{T}) = 0$$

- $T$: **CG**

# Performance Analysis: Platform

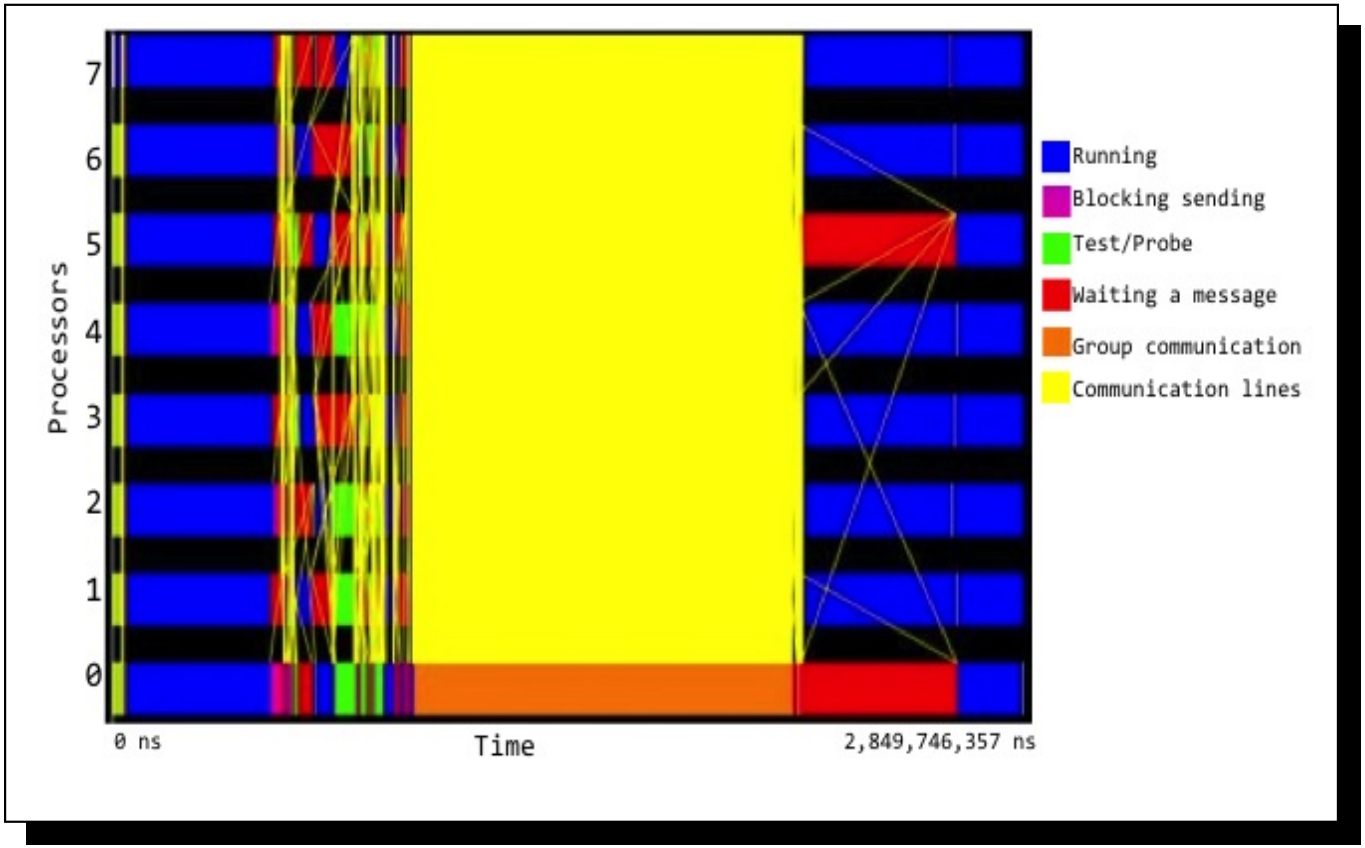
A single node consisting of two sockets, each socket is connected to a GPU device (Tesla C2050)

The socket has 6 dual cores Intel Xeon CPU X5670 @ 2.93GHz



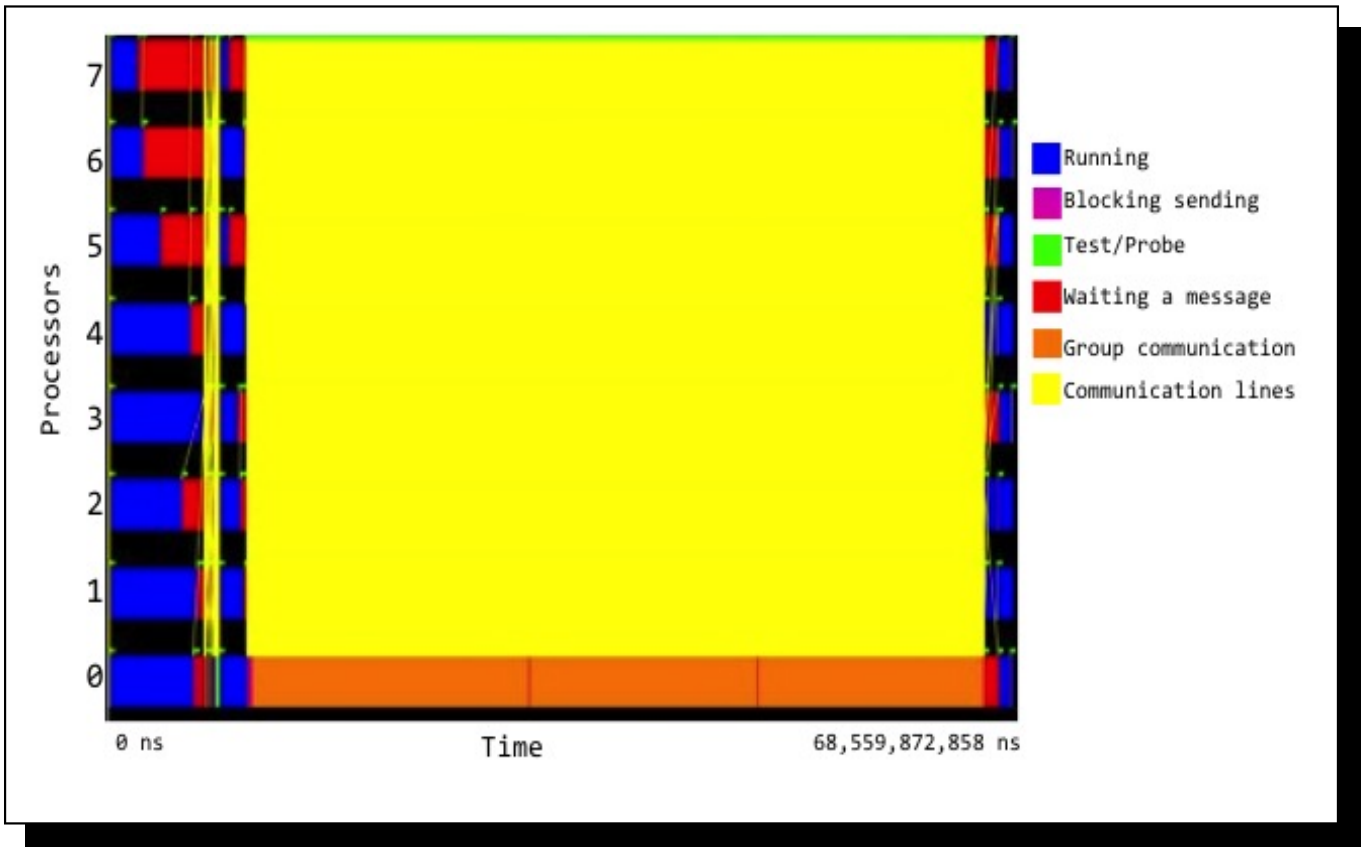
# laplacianFoam: 3D Heat Equation

Heat equation test case solved over a three-dimensional slab geometry using the laplacianFoam.



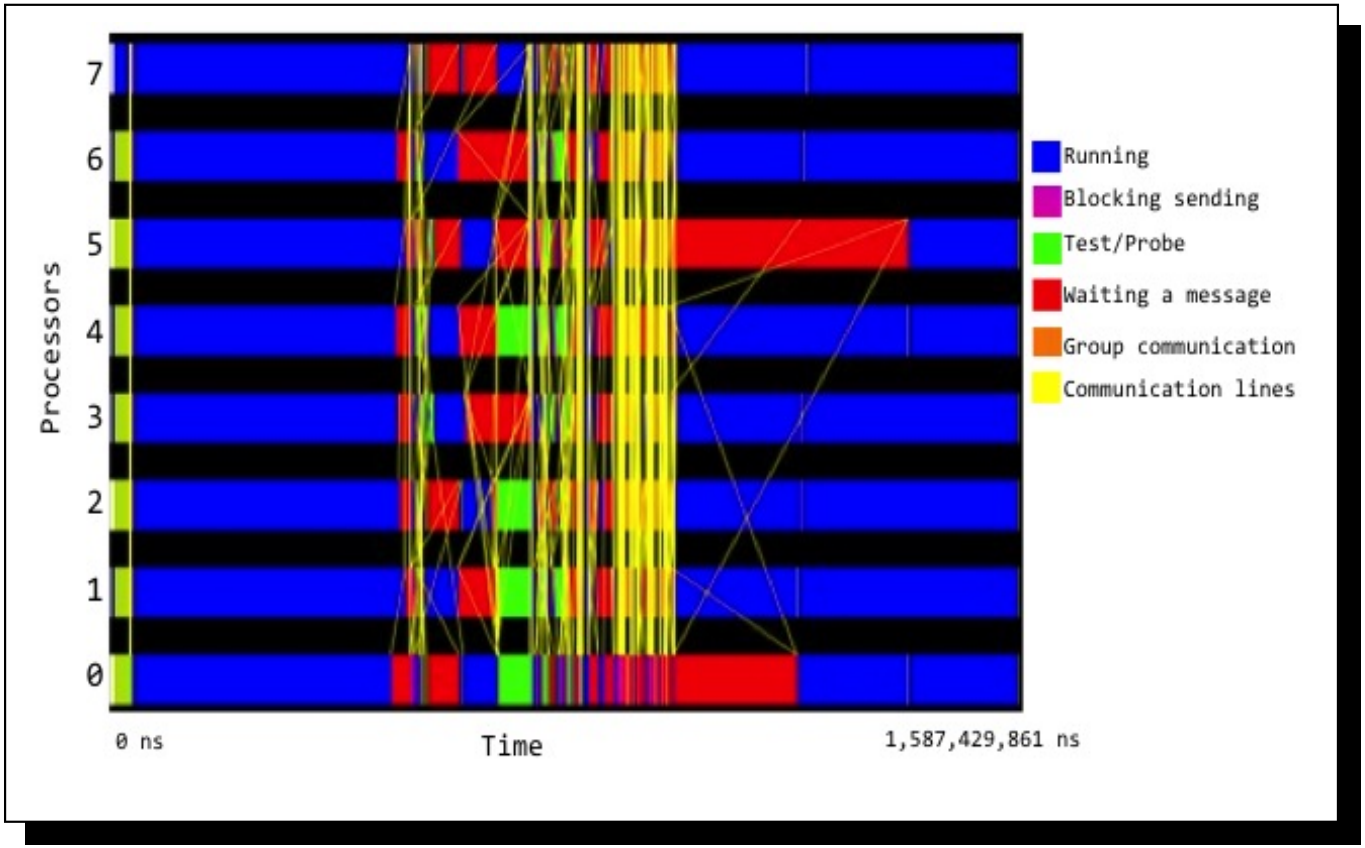
# icoFoam: Lid-driven Cavity flow

The lid-driven cavity flow test case contains the solution of a laminar, isothermal and incompressible flow over a three-dimensional cubic geometry. The top boundary of the cube is a wall that moves in the $x$ direction, whereas the rest are static walls.



# Conjugate Gradient  1 iteration

A single iteration of the conjugate gradient solver using a matrix derived from the 3D heat equation.



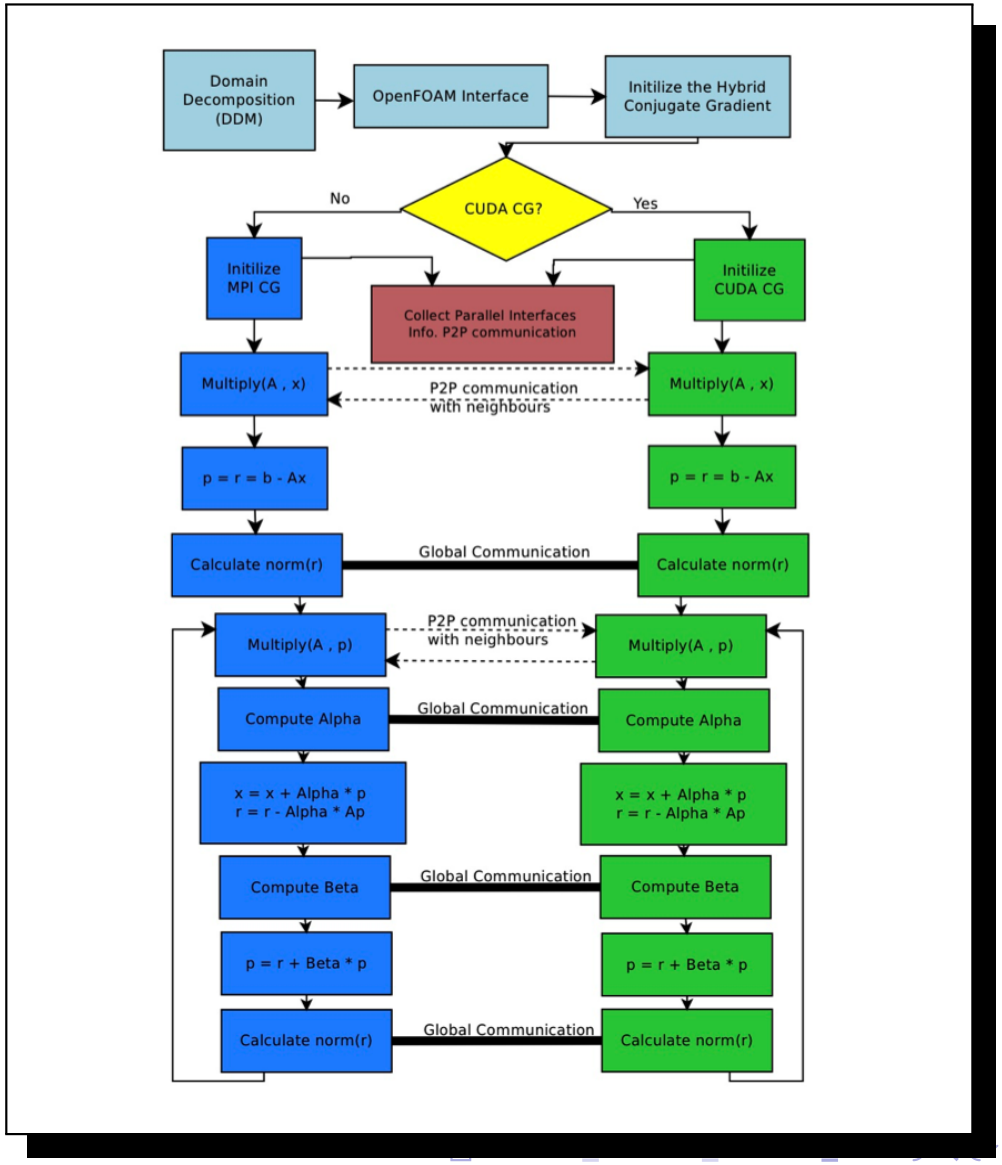
# Proposed Optimizations

- Hybrid CG Solver
- Hybrid Pipelined CG Solver
- Heterogenous Decomposition

“ Future computing architectures will be hybrid systems with parallel-core GPUs working in tandem with multi-core CPUs. ”

# Hybrid CG

- Multicore/MultiGPU
- CUDA Kernel is based on CUSP and Thrust calls.
- CSR Format in the GPU and Skyline Format in the CPU



# Hybrid Pipelined CG

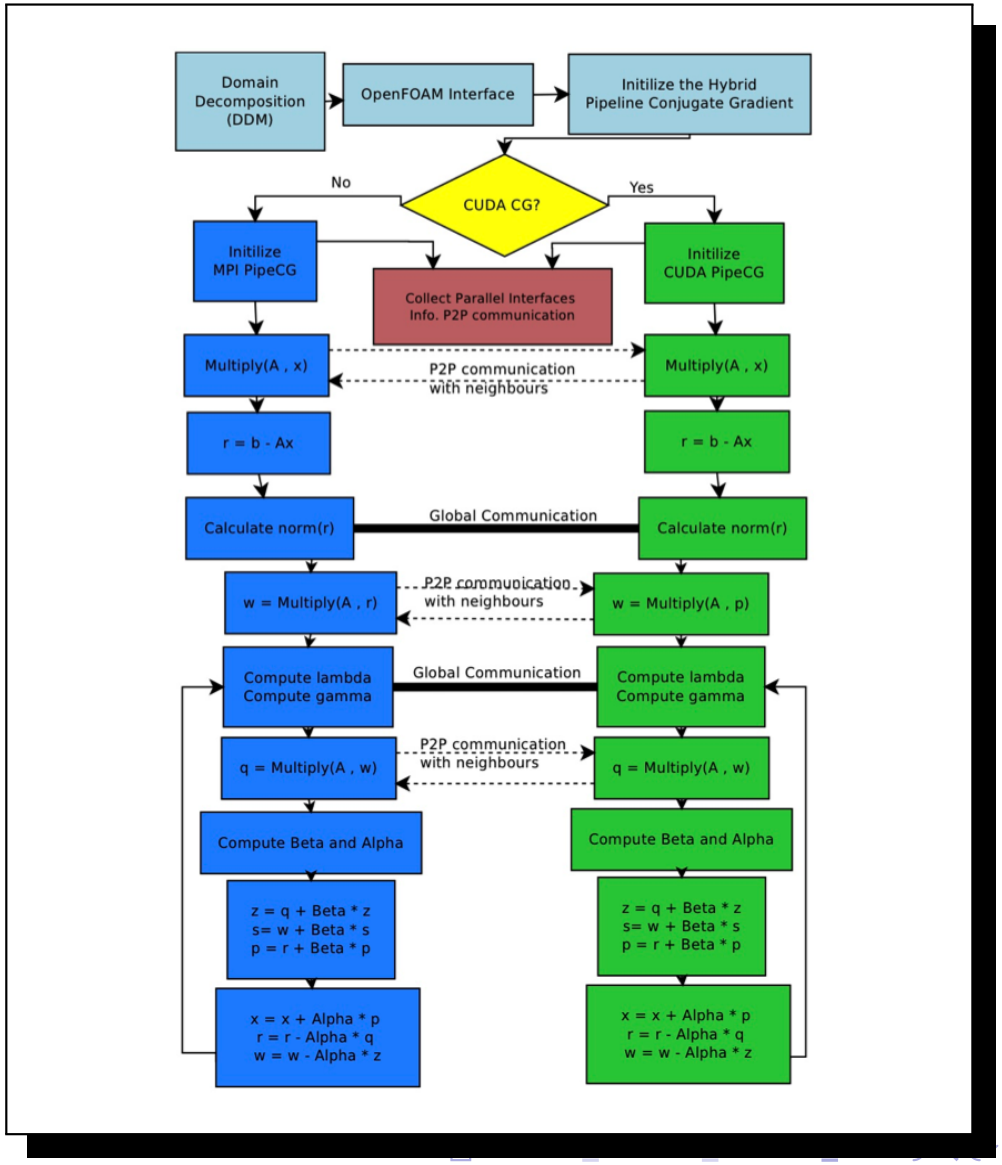
- Recently introduced by Ghysels and Vanroose*
- Reduces the global communication to 1 time instead of three
- Offers better scalability at the price of extra computations

$r_0 = b - Ax$
$w_0 = Ar_0$
$k = 0$
**while** $\|r\|_2 \geq \tau$ **do**
  $\lambda_k = dotc(r_k, r_k)$
  $\delta = dotc(w_k, r_k)$
  $q = SpMV(A, w_k)$
  **if** (k >0) **then**
    $\beta = \lambda_k$ / $\lambda_{k-1}$
    $\alpha = \lambda_k$ / ($\delta$ - ($\beta$*$\lambda_k$ )/$\alpha$)
  **else**
    $\beta = 0$
    $\alpha = \lambda_k$ / $\delta$
  **end if**
  $z = q + \beta * z$
  $s = w + \beta * s$
  $p = r + \beta * p$
  $x = x + \alpha * p$
  $r = r - \alpha * s$
  $w = w - \alpha * z$
**end while**

* P. Ghysels and W. Vanroose, Hiding global synchronization latency in the preconditioned conjugate gradient algorithm, Parallel Computing, 2013.

# Hybrid Pipelined CG

- Multicore/MultiGPU
- CUDA Kernel is based on CUSP and Thrust calls.
- CSR Format in the GPU and Skyline Format in the CPU



# Heterogenous Decomposition

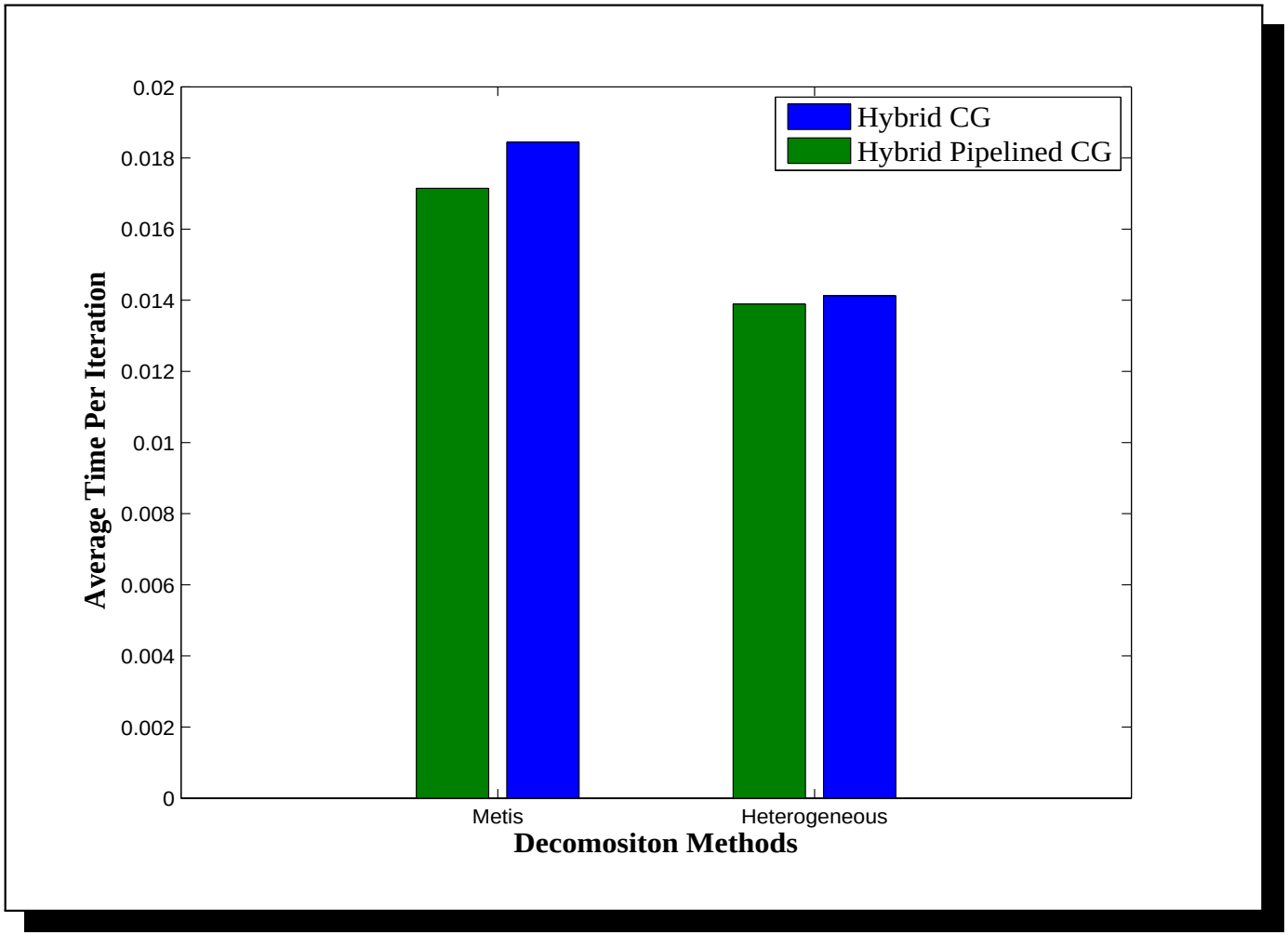
- The main idea is to adequately partition and assign the subdomain to the processor in proportion to its performance
  - more powerful processors get larger subdomains
- Combines the performance model and the METIS/SCOTCH library
- The load of the processor will be balanced if the number of computations performed by each processor is accordance to its speed on execution the kernel

$$s_i(n_i) = \frac{n_i + 2F_i}{T(n_i)}$$

$$r_i(n_i) = \frac{s_i(n_i)}{\sum_p s_i(n_i)}$$

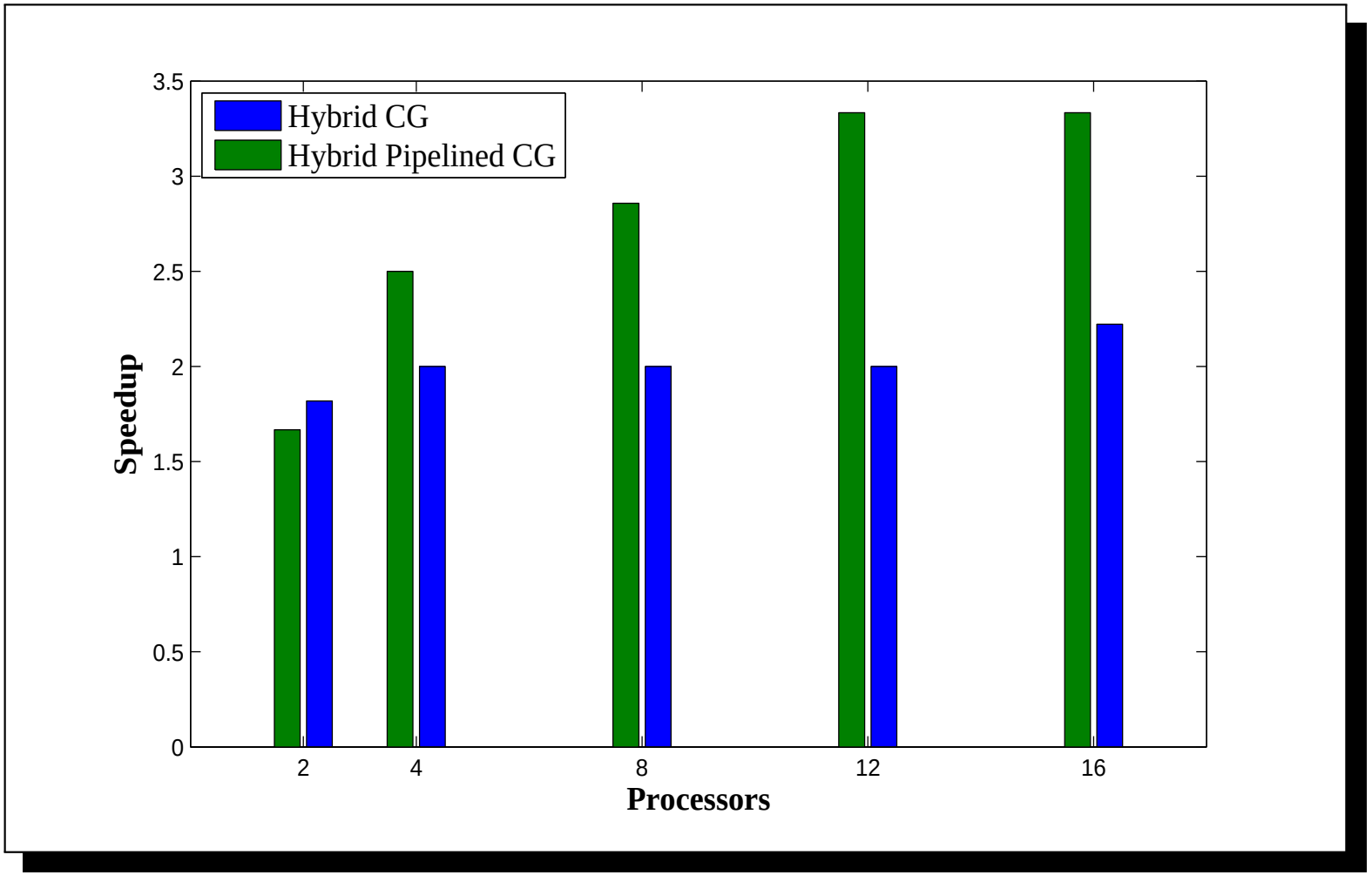
$$n_{i,new} = N * r_i(n_i)$$

# Experimental Results: Heterogenous Decomposition



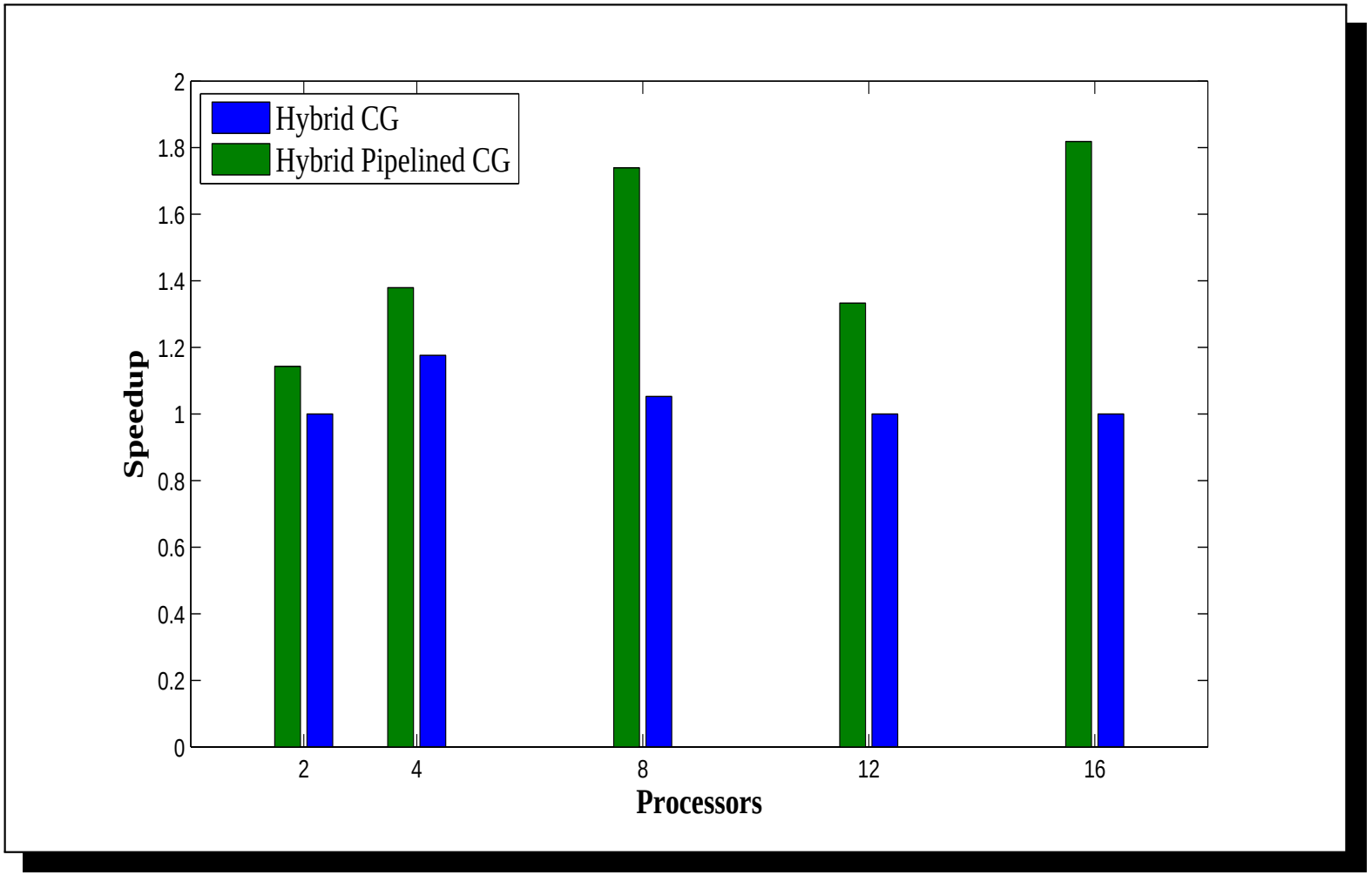
# Experimental Results: Speedup icoFoam I

N=100k



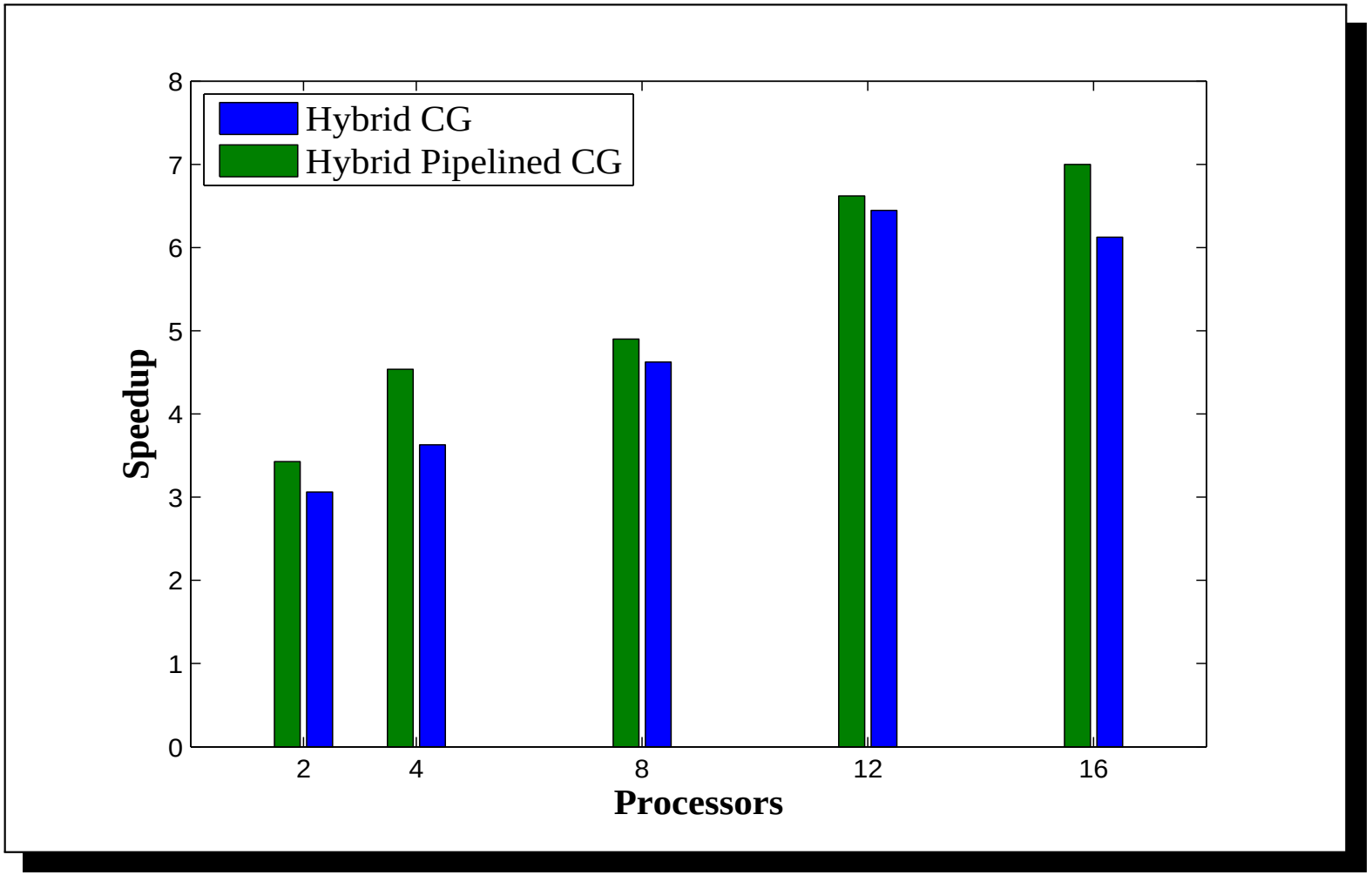
# Experimental Results: Speedup icoFoam II

N=1M



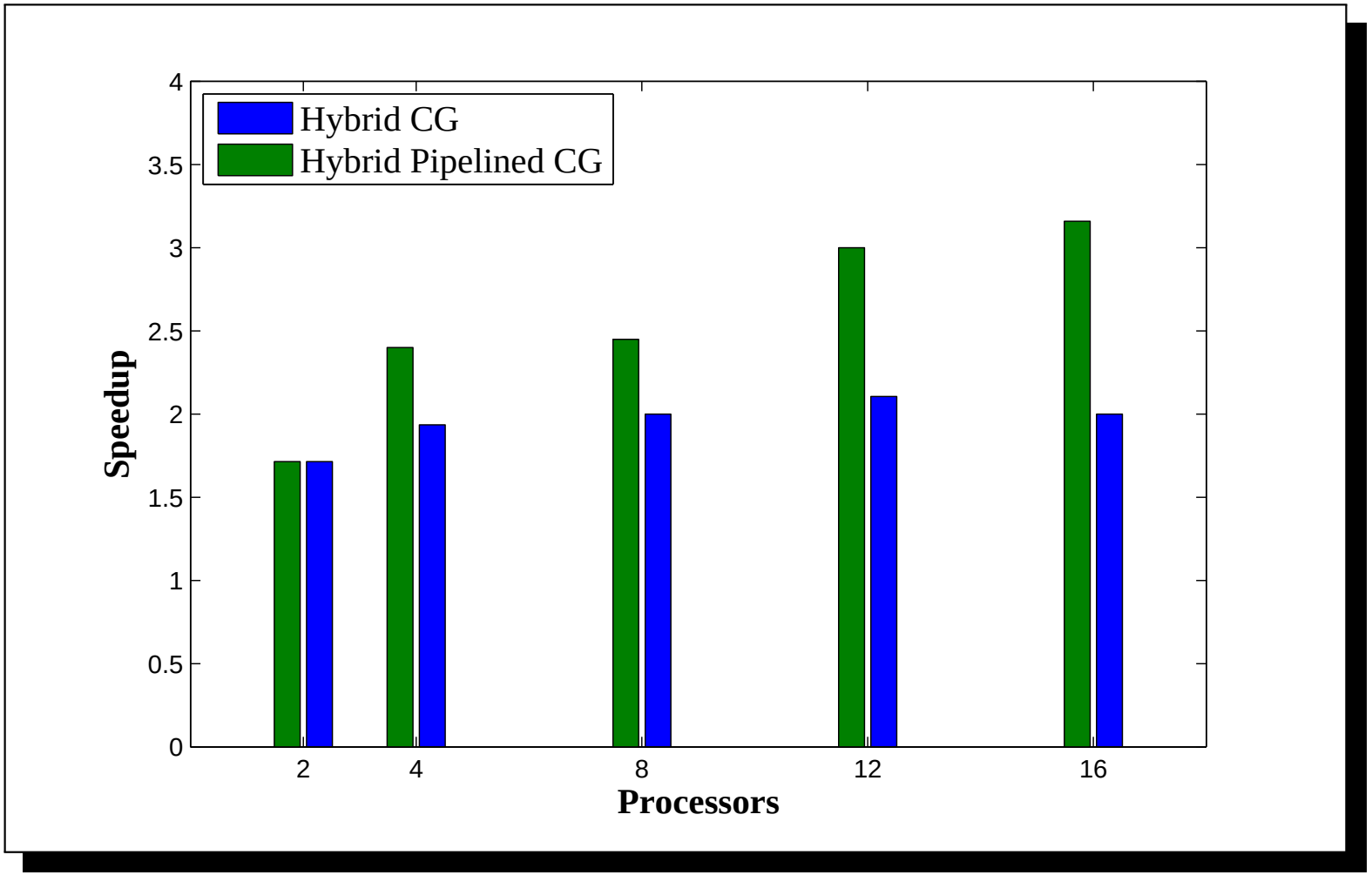
# Experimental Results: Speedup laplacianFoam I

N=100k

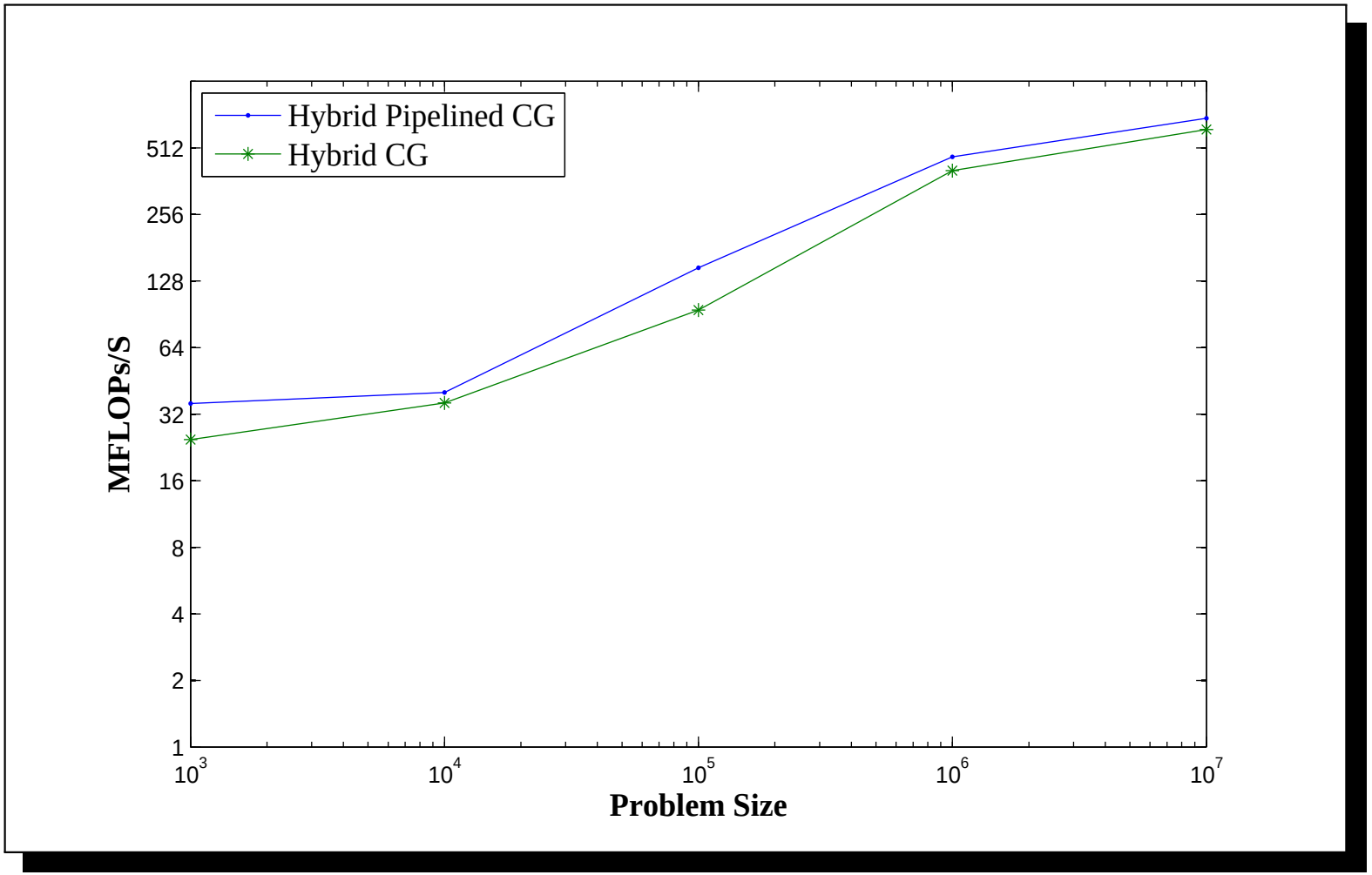


# Experimental Results: Speedup laplacianFoam II

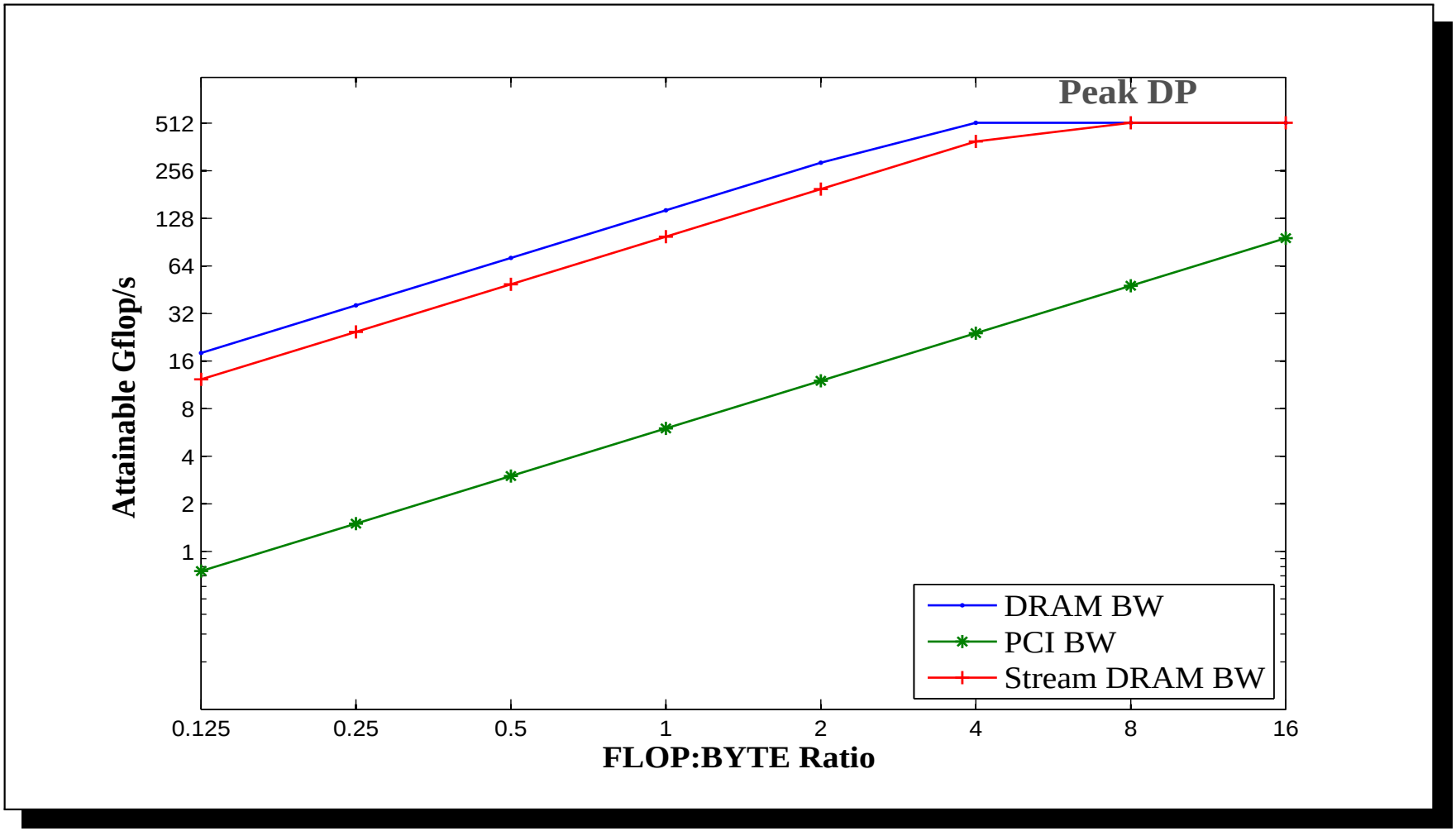
N=1M



# Experimental Results: Performance



# Experimental Results: Roofline



# Conclusion

- Memory bound applications, such as the OpenFOAM selected solvers, can take better advantage of the full hardware potential, which is now complex, hybrid and heterogeneous, if all resources are taken into accounts in a holistic approach.
- Vector reduction kernel performs $n \times 1$ memory transactions, with $n$ the vector size and cannot be combined with other operations $\rightarrow$ low arithmetic intensity, low memory throughput and poor scalability when increasing number of GPU/CPU.
- The need for dynamic load balancing scheduling, which adaptively balances the workload during the run-time, by memory-aware work stealing.
- The experimental results show that the hybrid implementation of both solvers significantly outperforms state-of-the-art implementations of a widely used open source package.

# Thank You!

# Backup Slides.

# PISO Algorithm

- Pressure Implicit with Splitting of Operators (PISO)*
  1. Set the boundary conditions.
  2. Solve the discretized momentum equation to compute an intermediate velocity field.
  3. Compute the mass fluxes at the cells faces.
  4. Solve the pressure equation.
  5. Correct the mass fluxes at the cell faces.
  6. Correct the velocities on the basis of the new pressure field.
  7. Update the boundary conditions.
  8. Repeat from 3 for the prescribed number of times.
  9. Increase the time step and repeat from 1.

*J. H. Ferziger, M. Peric, Computational Methods for Fluid Dynamics, Springer, 3rd Ed., 2001. H. Jasak, Error Analysis and Estimation for the Finite Volume Method with Applications to Fluid Flows, Ph.D. Thesis, Imperial College, London, 1996. http://openfoamwiki.net