Implementation of a Finite Volume Unstructured CFD Solver Using Cluster Based Parallel Computing

A thesis defense
Presented in partial fulfillment
For the requirements of the degree
Masters of Engineering
In Mechanical Engineering
American University of Beirut
July 2005

By Tony Y. Saad
Advised by
Prof. Marwan Darwish

http://tsaad.utsi.edu - tsaad@utsi.edu

Plan

- Introduction
- Objectives
- The Finite Volume Method
- Iterative Solvers
- Parallel Computing
- Our Implementation
- Results
- Conclusions

Introduction

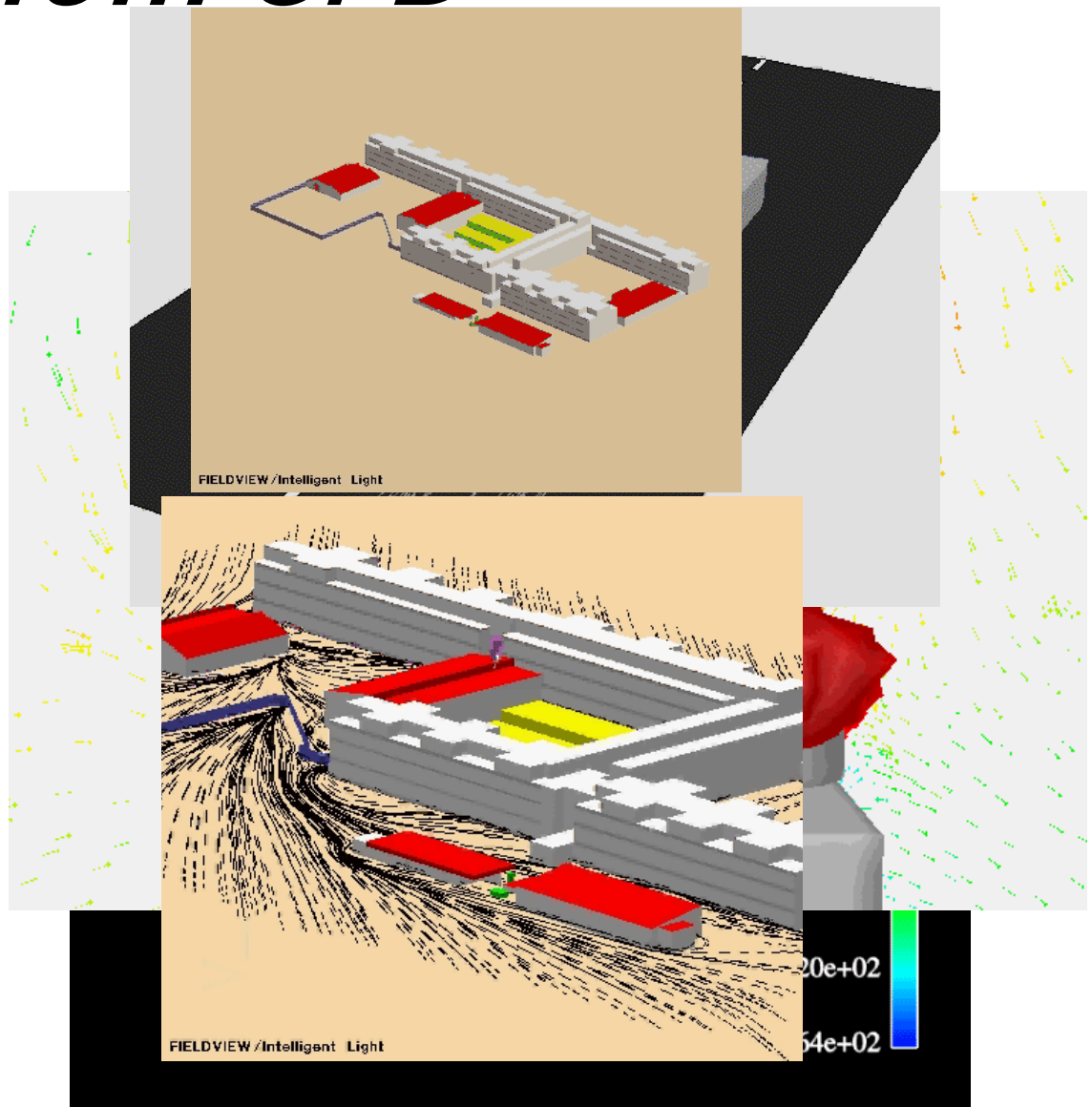
- CFD: a tool not a toy!
- Parallel computing: the future of scientific computing
- Parallel CFD: the ultimate combination

Introduction: CFD

- Computational Fluid Dynamics (CFD) is an advanced computing technology developed from traditional fluid mechanics.
- It uses a set of numerical methods for the solution of PDEs arising from general transport phenomena
- Methods include: Finite Difference, Finite Volume, Finite Element, Boundary Element, Spectral…
- CFD is being used to model several phenomena and to design advanced equipment

Introduction: CFD

- Aerospace
- Appliances
- Sports
- Automotive
- Environment…



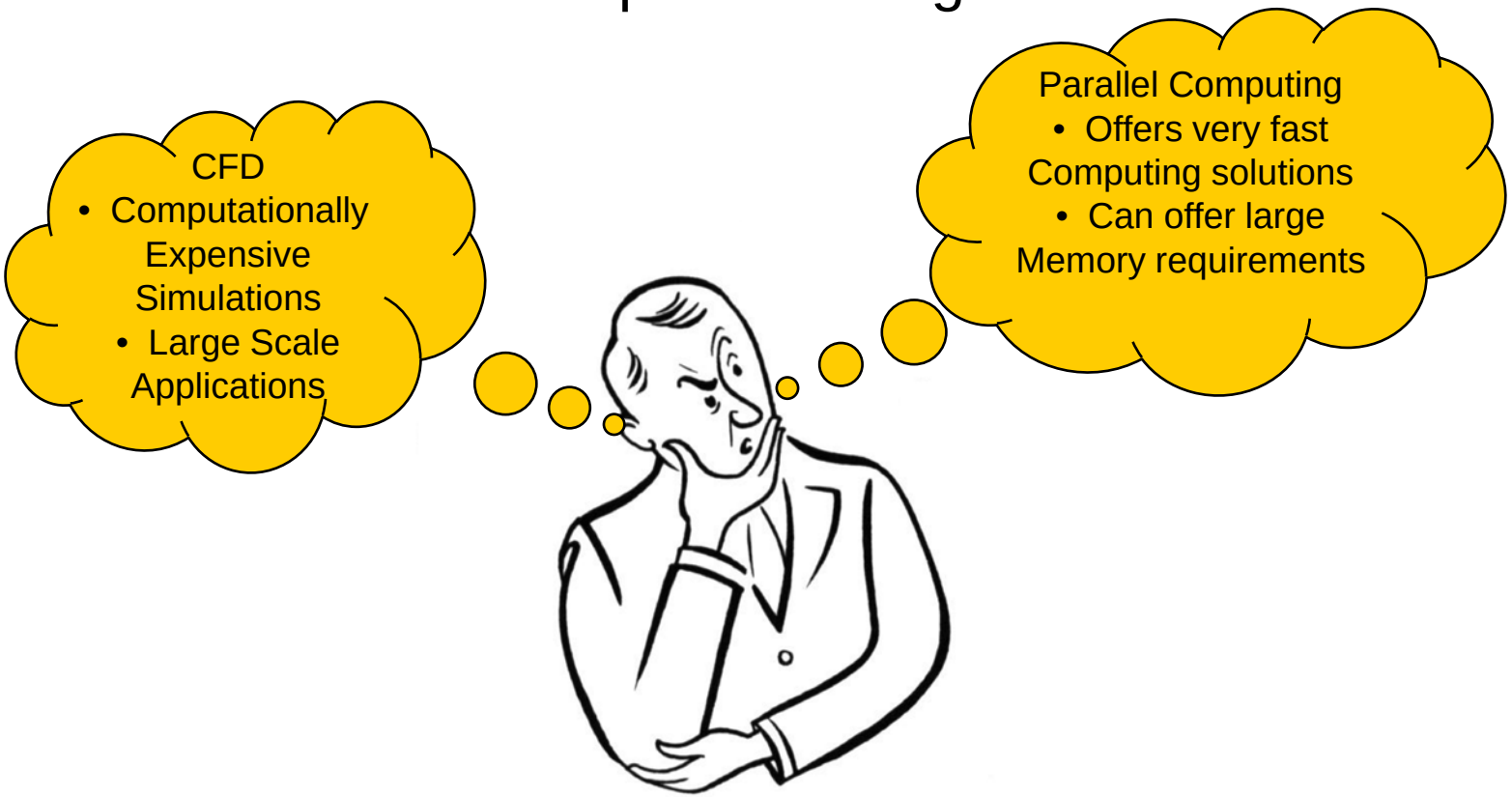
Introduction: Parallel

- The increase in processing power will eventually reach a limit
- Alternative: Parallel Computing
- Parallel computing stands for the collaboration of several processors in the solution of a certain problem

QuickTime™ and a
TIFF (Uncompressed) decompressor
are needed to see this picture.

Introduction: Parallel CFD

- So… How do these pieces fit together?



Objectives

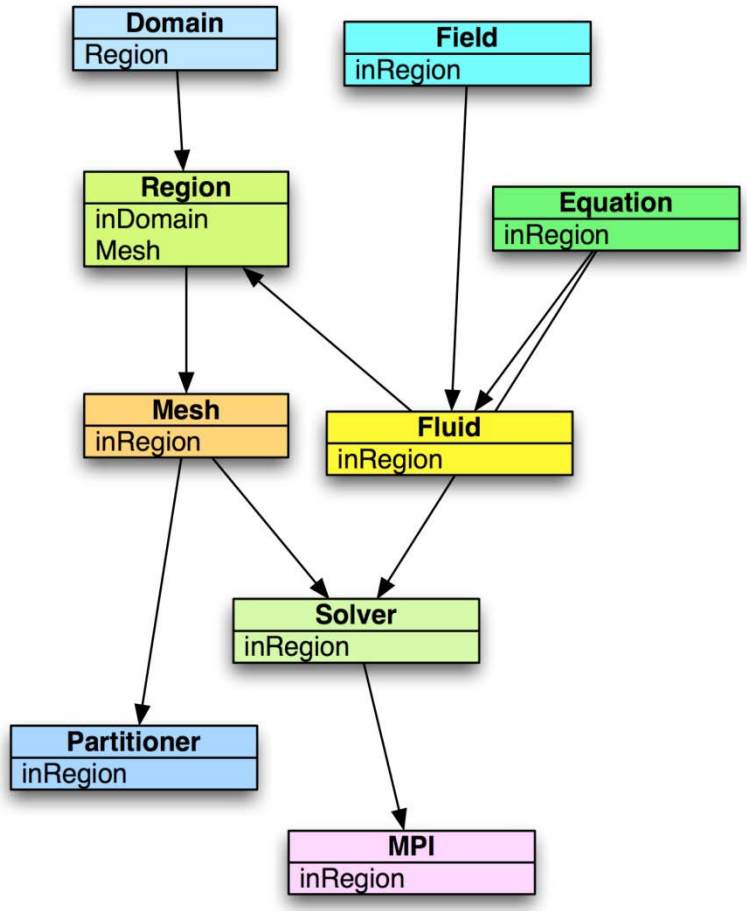
- Current Research Trends
- Our Objectives: *… I do mind your publishing faster than you think!*

Objectives: Some History

- Problems with iterative solvers: they tend to decrease the efficiency of the parallel code
- Problems with scalability and robustness: most codes work well for a narrow range of processors
- Problems with code migration and transparency: most parallel codes are tailored to simulate a small set of phenomena

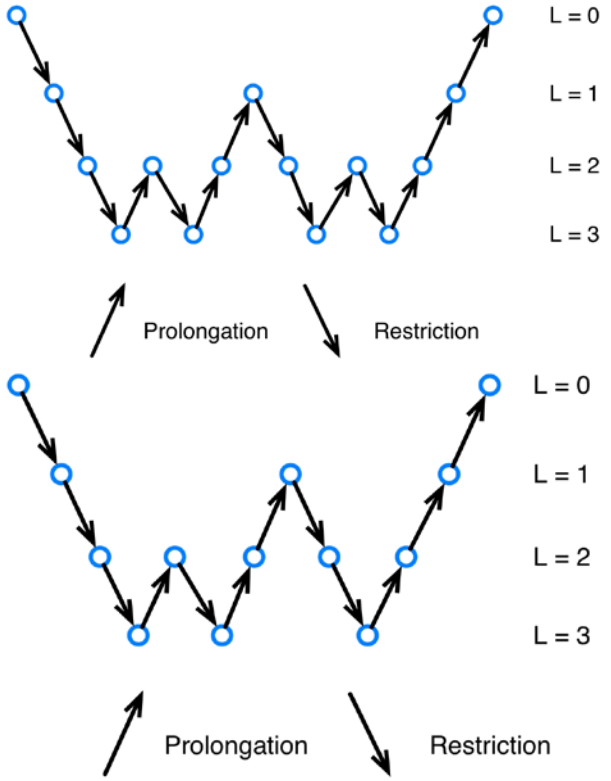
Objectives

- Implement an object oriented parallel CFD code



Objectives

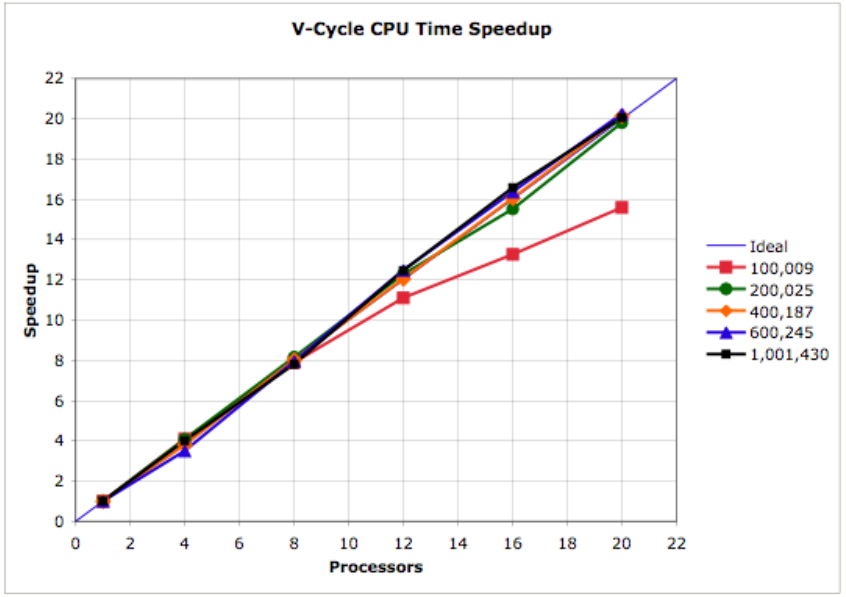
- Implement an object oriented parallel CFD code
- Implement a parallel AMG solver using V, F and W cycles



Objectives

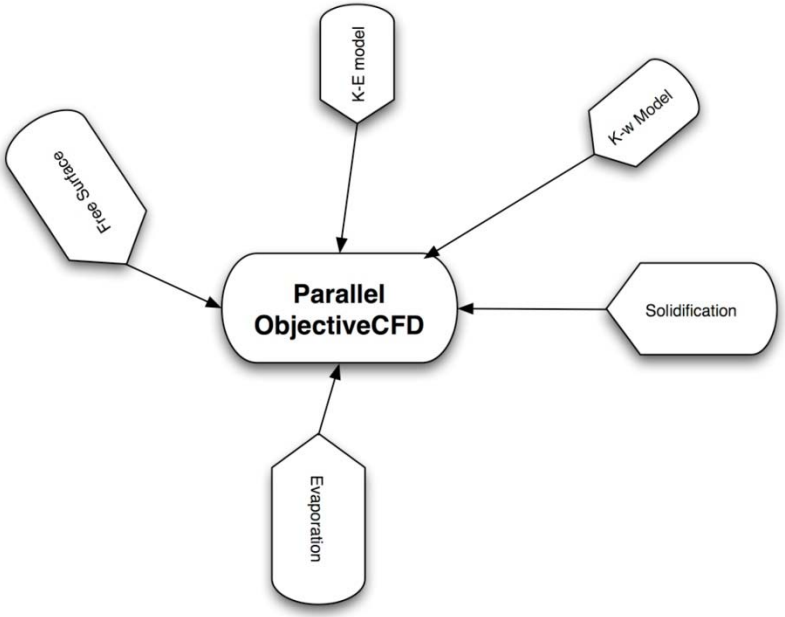
- Implement an object oriented parallel CFD code
- Implement a parallel AMG solver using V, F and W cycles
- Achieve scalability while maintaining robustness

$$S(n) = \frac{\text{Execution time using one processor}}{\text{Execution time using n processors}} = \frac{t_s}{t_p}$$



Objectives

- Implement an object oriented parallel CFD code
- Implement a parallel AMG solver using V, F and W cycles
- Achieve scalability while maintaining robustness
- Code Transparency



The FV Method

- The FV Method is a numerical technique aimed at the solution of PDE's, and specifically tuned to those arising form fluid flow and related phenomena
- The target PDE of the FVM is the scalar transport equation

$$\underbrace{\frac{\partial \rho \phi}{\partial t}}_{\text{Transient Term}} + \underbrace{\nabla \cdot \left(\rho \boldsymbol{u} \phi \right)}_{\text{Convection Term}} = \underbrace{\nabla \cdot \left(\Gamma \nabla \phi \right)}_{\text{Diffusion Term}} + \underbrace{S_{\phi}}_{\text{Source Term}}$$

The FV Method

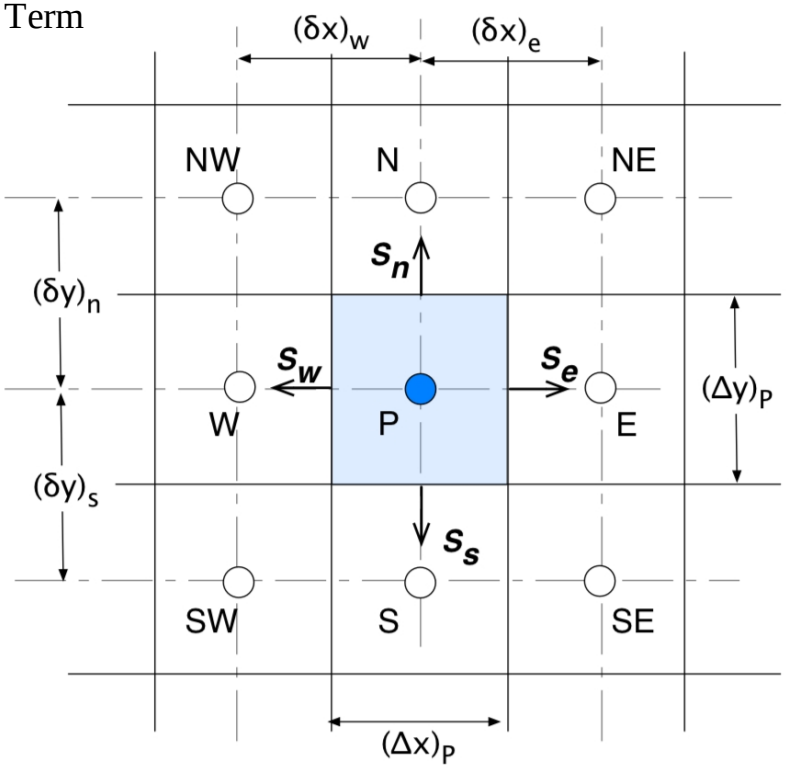
$$\underbrace{\frac{\partial \rho \phi}{\partial t}}_{\text{Transient Term}} + \underbrace{\nabla \cdot \left(\rho \boldsymbol{u} \phi \right)}_{\text{Convection Term}} = \underbrace{\nabla \cdot \left(\Gamma \nabla \phi \right)}_{\text{Diffusion Term}} + \underbrace{S_\phi}_{\text{Source Term}}$$

$$\left(\frac{a_P^t + a_P^{CD}}{urf} \right) \phi_P + \sum_{i=NB(P)} a_{Ni}^{CD} \phi_{Ni} = Q_P V_P + a_P^t \phi_P^0 + \frac{1-urf}{urf} \left(a_P^t + a_P^{CD} \right) \phi_P^*$$

$$a_p \phi_P + \sum_{i=NB(P)} a_{Ni} \phi_{Ni} = b_P$$

$$a_{Ni} = -\Gamma_f \frac{E_f}{d_{PNi}} - \left\| -\dot{m}_f, 0 \right\| \qquad a_P = a_P^t - \sum_{i=NB(P)} a_{Ni} + \sum_{i=nb(P)} \dot{m}_{fi}$$

$$b_P = Q_P V_P + a_P^t \phi_P^0 + \frac{1-urf}{urf} \left(a_P^t + a_P^{CD} \right) \phi_P^* + \sum_{i=nb(P)} \left(\Gamma \nabla \phi \right)_{fi} \cdot \boldsymbol{T}_{fi}$$

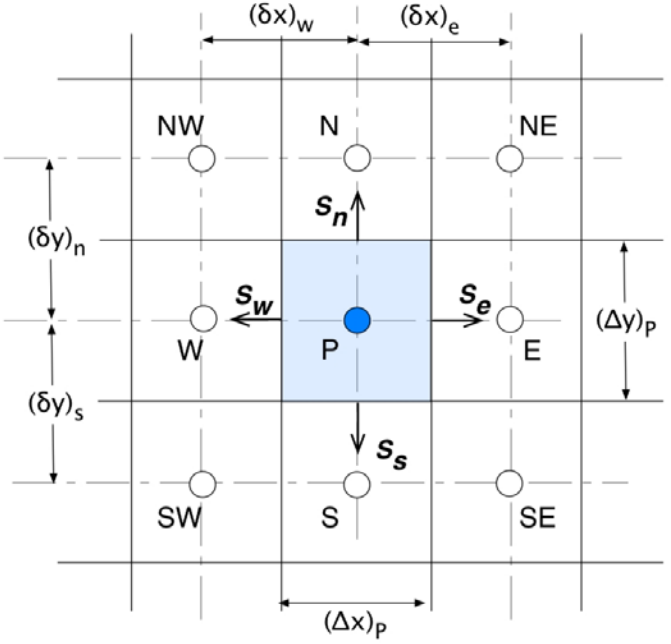


The FV Method

- For each control volume we end up with an Algebraic Equation
- The system of equations is then solved using an Iterative Solver

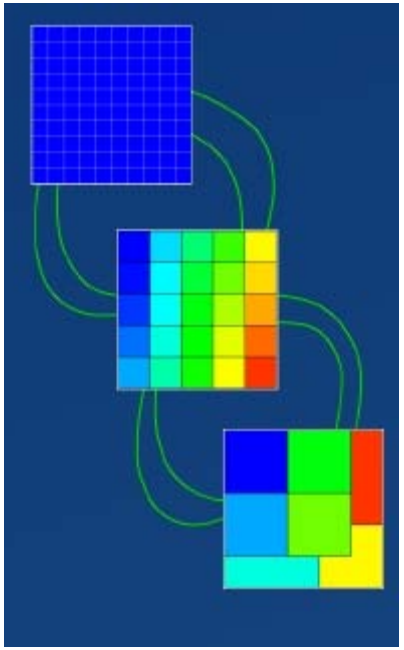
$$a_P \phi_P + \sum_{i=NB(P)} a_{Ni} \phi_{Ni} = b_P$$

$$\boldsymbol{A} \cdot \phi = b$$

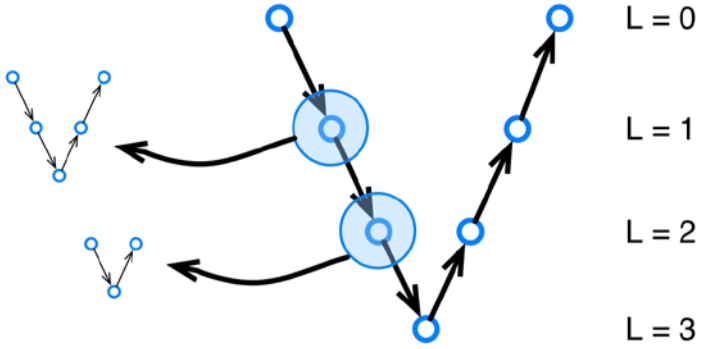


Iterative Solvers

- What are Iterative solvers?
- The Incomplete LU solver
- Multigrid Methods and the AMG solver



$$\begin{bmatrix} 0 & \alpha & 0 & 0 \\ 0 & 0 & \beta & 0 \\ 0 & \tau & 0 & 0 \\ 0 & 0 & 0 & \delta \end{bmatrix} \rightarrow \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & \lambda & 0 & 0 \\ 0 & 0 & 0 & \iota \end{bmatrix} \cdot \begin{bmatrix} 0 & \eta & 0 & 0 \\ 0 & 0 & \zeta & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \varphi \end{bmatrix}$$



Iterative Solvers

- Iterative solvers encompass a set of instructions to find a solution for a linear system from an initial estimate
- Why iteratively?
  - Coefficient matrix is nonlinear
  - Iterative solvers are computationally inexpensive
- Famous solvers: SOR, ILU
  - We use ILU because of its superior convergence rate

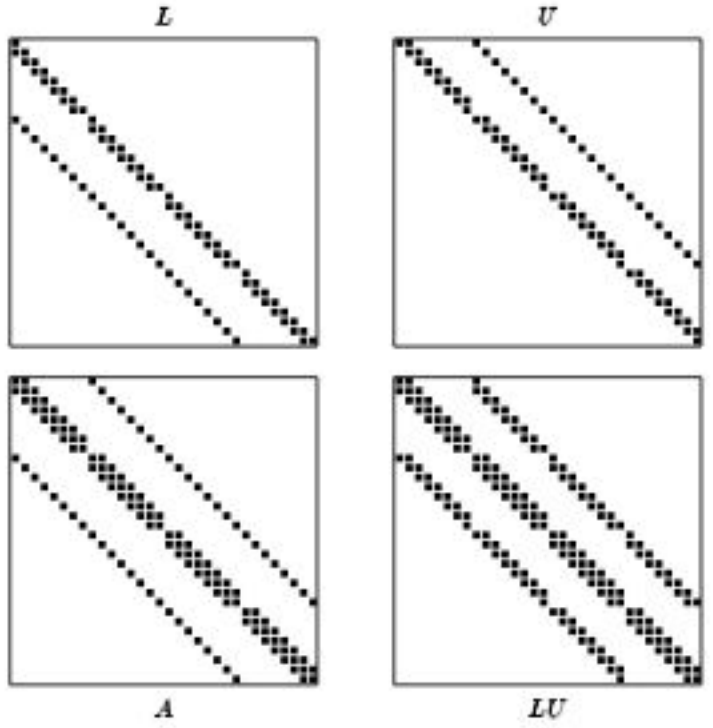
$$a_P \phi_P + \sum_{i=NB(P)} a_{Ni} \phi_{Ni} = b_P$$

$$\boldsymbol{A} \cdot \phi = b$$

Iterative Solvers: ILU

- Incomplete LU decomposition is similar to the standard LU decomposition
- Zero entries in A remain zeros during factorization
- The actual LU product is not equal to A

$$\begin{bmatrix} 0 & \alpha & 0 & 0 \\ 0 & 0 & \beta & 0 \\ 0 & \tau & 0 & 0 \\ 0 & 0 & 0 & \delta \end{bmatrix} \rightarrow \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & \lambda & 0 & 0 \\ 0 & 0 & 0 & \iota \end{bmatrix} \cdot \begin{bmatrix} 0 & \eta & 0 & 0 \\ 0 & 0 & \zeta & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \varphi \end{bmatrix}$$

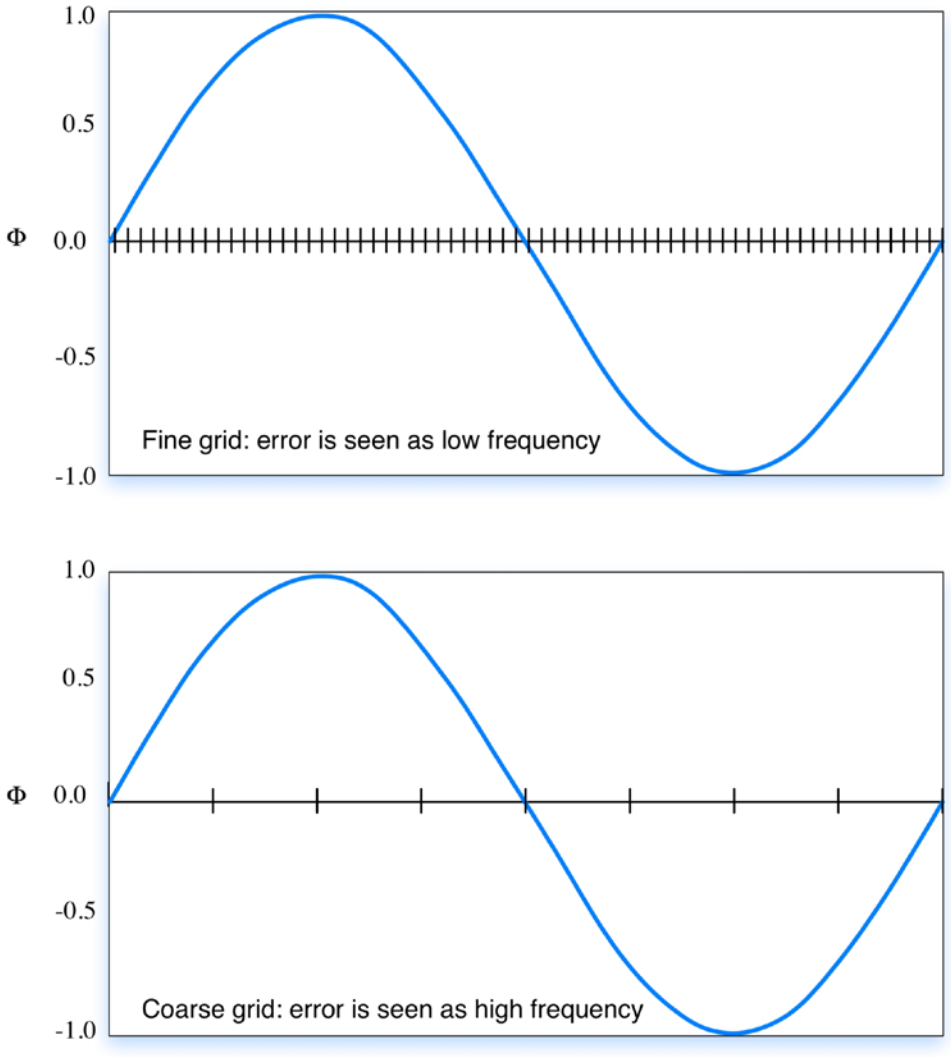


Iterative Solvers: Multigrid

- The Multigrid method is a technique used to accelerate the convergence of a given solver
- MG is NOT A SOLVER!
- The convergence rate of most solvers stalls after the first few iterations because the error becomes smooth
- Iterative solvers are efficient in smoothing out the high frequency error
- MG defines a set of “grid levels” on which the low frequency error is seen “relatively” as a high frequency one

Iterative Solvers: Multigrid

- The MG is based on the observation that low frequency errors on fine grids are seen as high frequency ones on coarse grids
- We thus define a set of corase grid levels derived from the original one



Iterative Solvers: Multigrid

$$A \cdot x = b$$

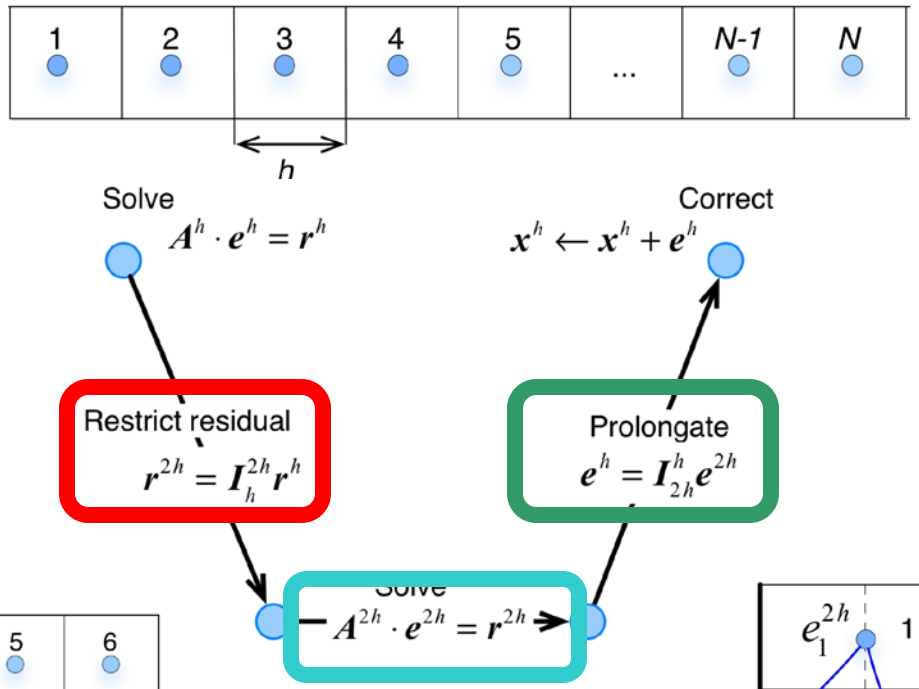
$$e^k = x - x^k$$

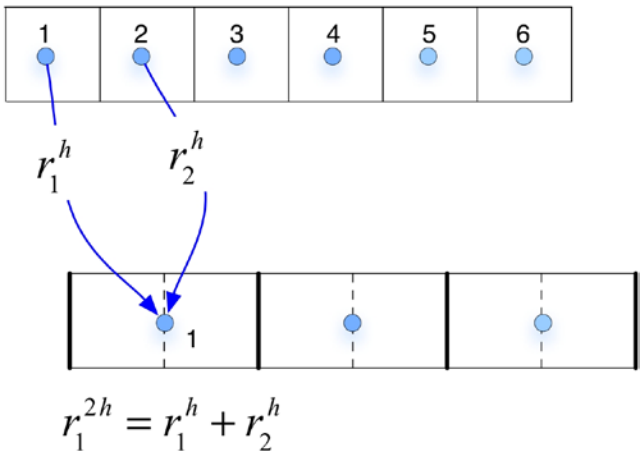
$$r^k = b - Ax^k$$

$$A \cdot x = b$$

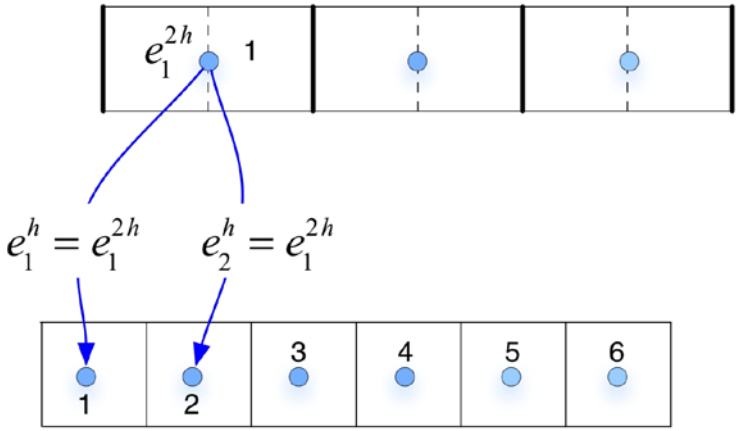
$$A \cdot \left(e^k + x^k\right) = r^k + Ax^k$$

$$A \cdot e^k = r^k$$





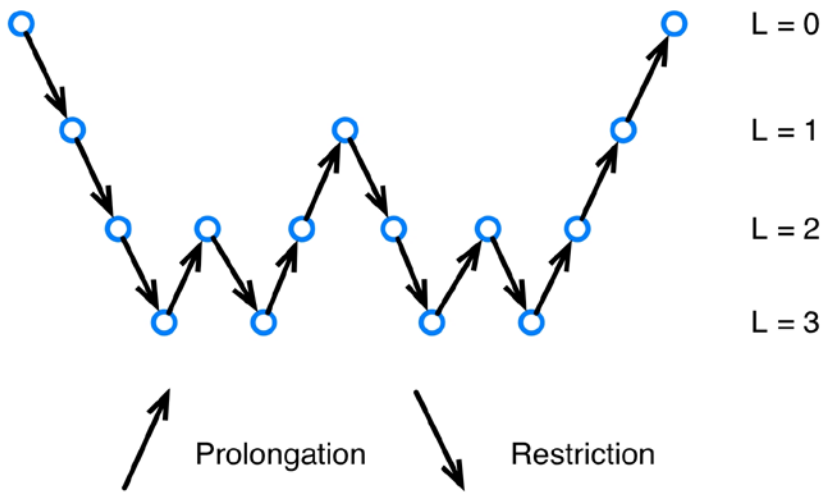
$$A_{I,J}^{(l+1)} = \sum_{i \in G_I} \sum_{j \in G_J} A_{i,j}^l$$

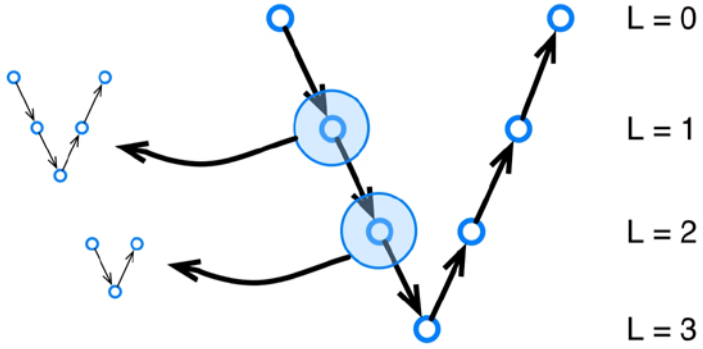


Iterative Solvers: Multigrid

AMG Cycles:

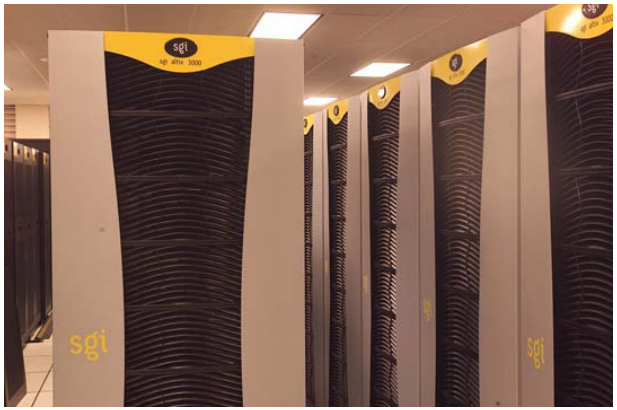
- V Cycle
- F Cycle
- W Cycle





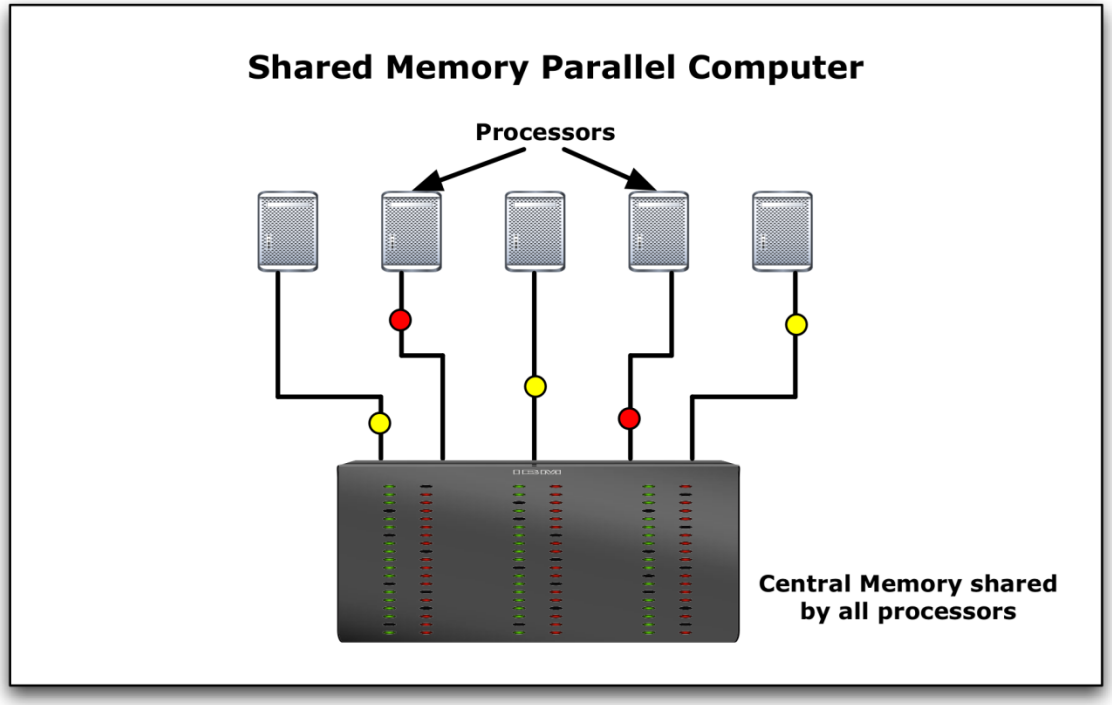
Parallel Computing

- Types of parallel computers
- Basics of Message Passing



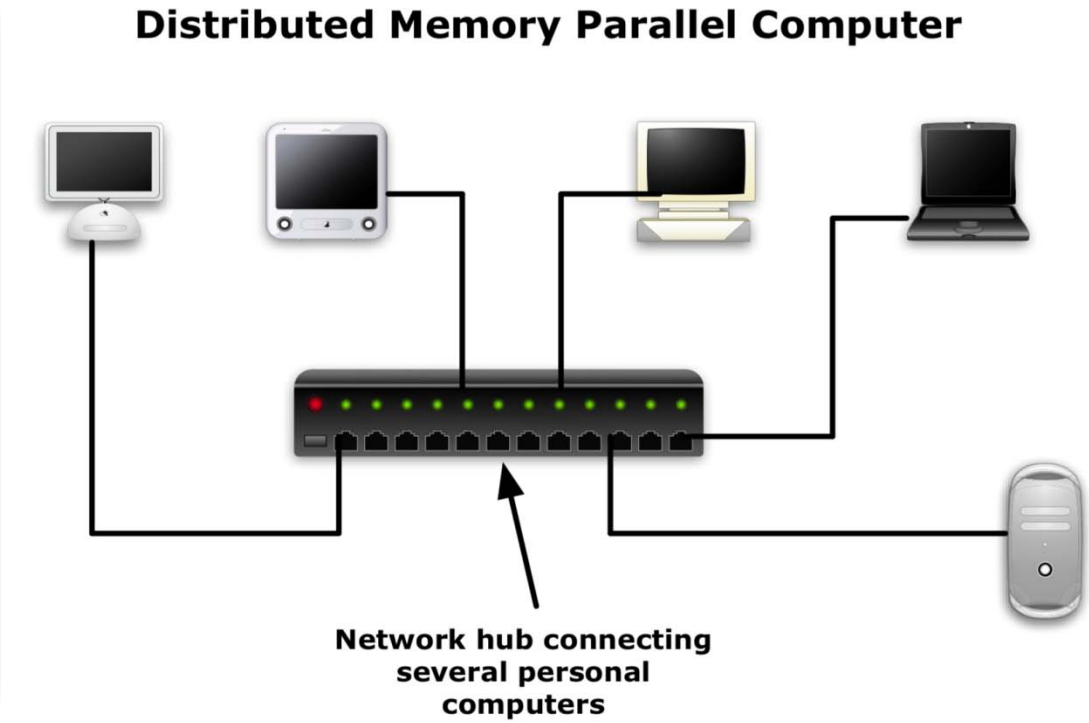
Parallel Computing

- Two types of parallel computers
  - Shared Memory Multiprocessor



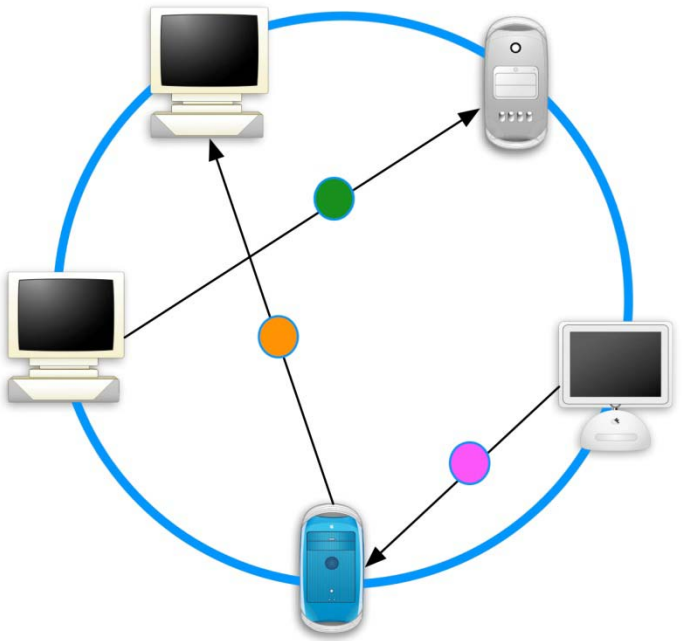
Parallel Computing

- Two types of parallel computers
  - Shared Memory Multiprocessor
  - Distributed Memory Computer



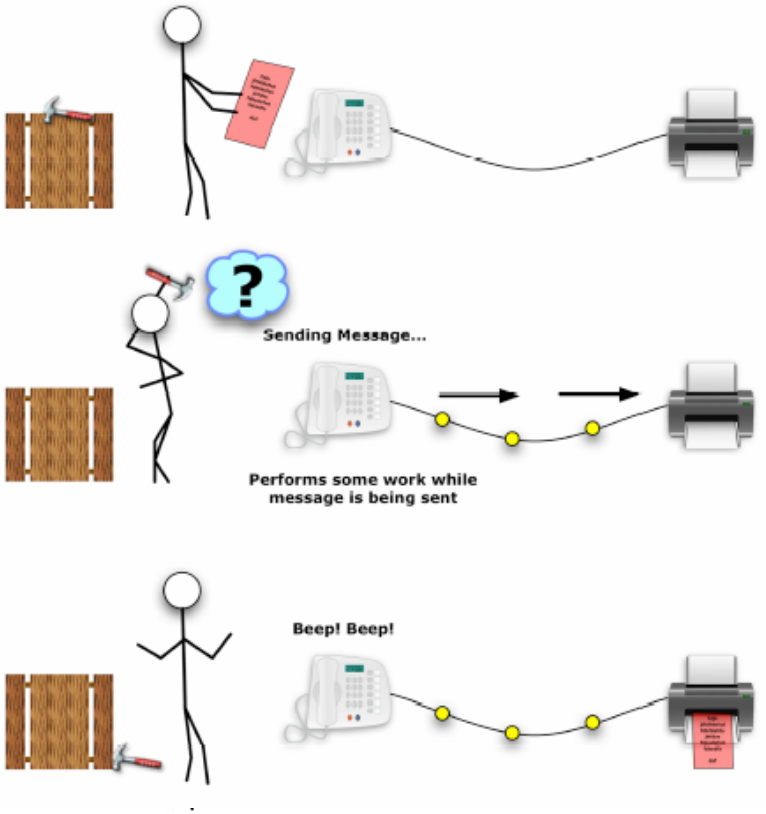
Parallel Computing: MPI

- In a distributed memory system, information need to be exchanged using messages
- We use the MPI standard
- There are two kinds of communication patterns
  - Peer-to-Peer communication
  - Collective communication



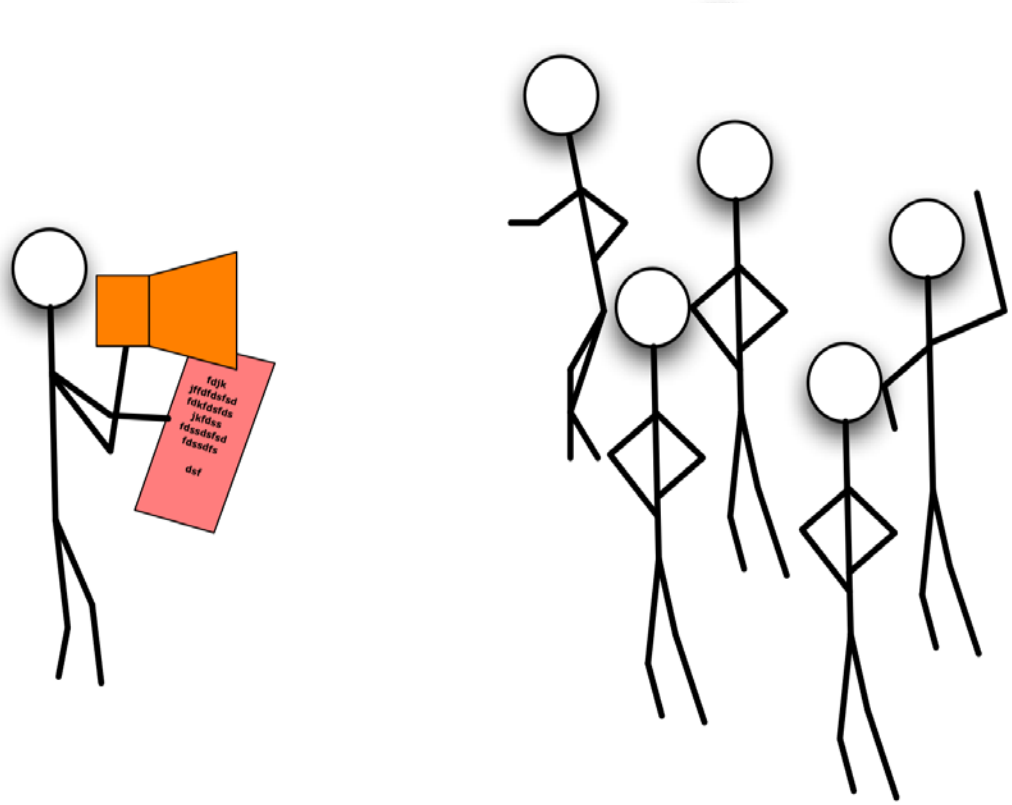
Parallel Computing: MPI

- P2P communication occurs when two processors communicate at one time. It occurs in two modes:
- Blocking
- Immediate



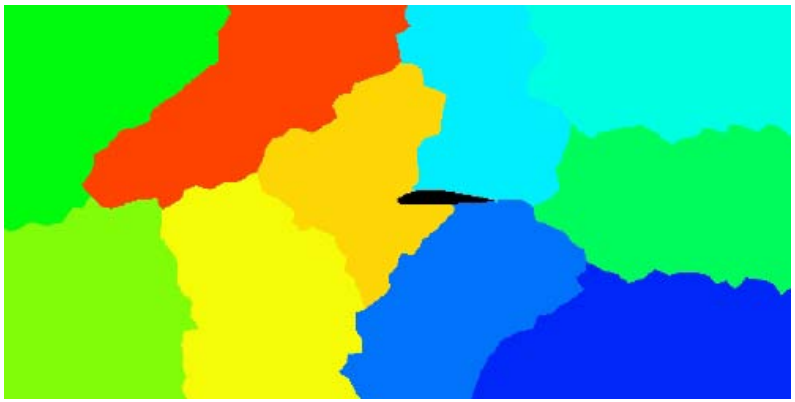
Parallel Computing: MPI

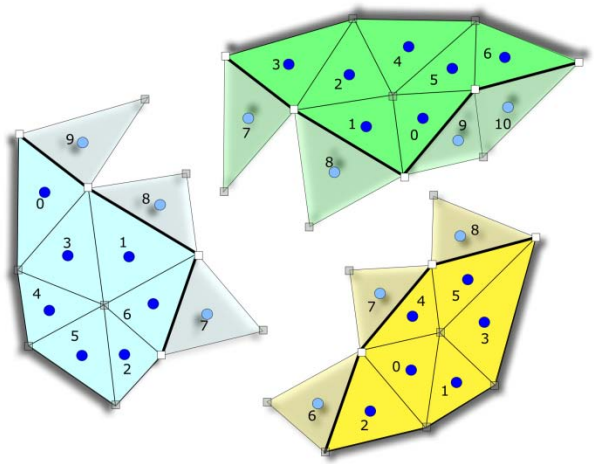
- Collective communication occurs when all processors participate in some sort of communication
- Reduce
- Broadcast



Implementation

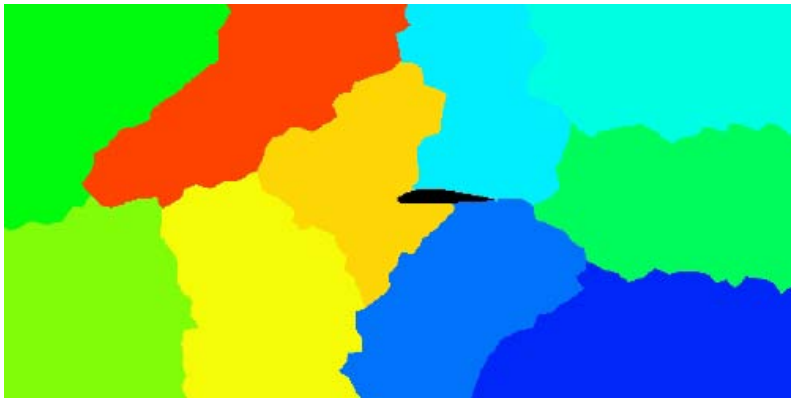
- Domain Partitioning
- Partition Renumbering
- Partition Distribution
- Parallelization of The Iterative Solver
- Parallelization of The AMG Solver

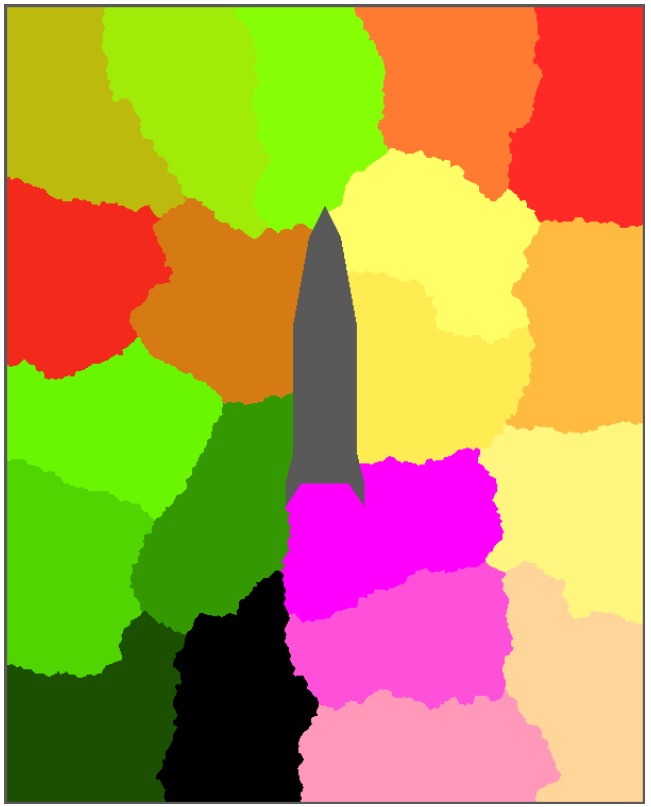




I- Domain Partitioning

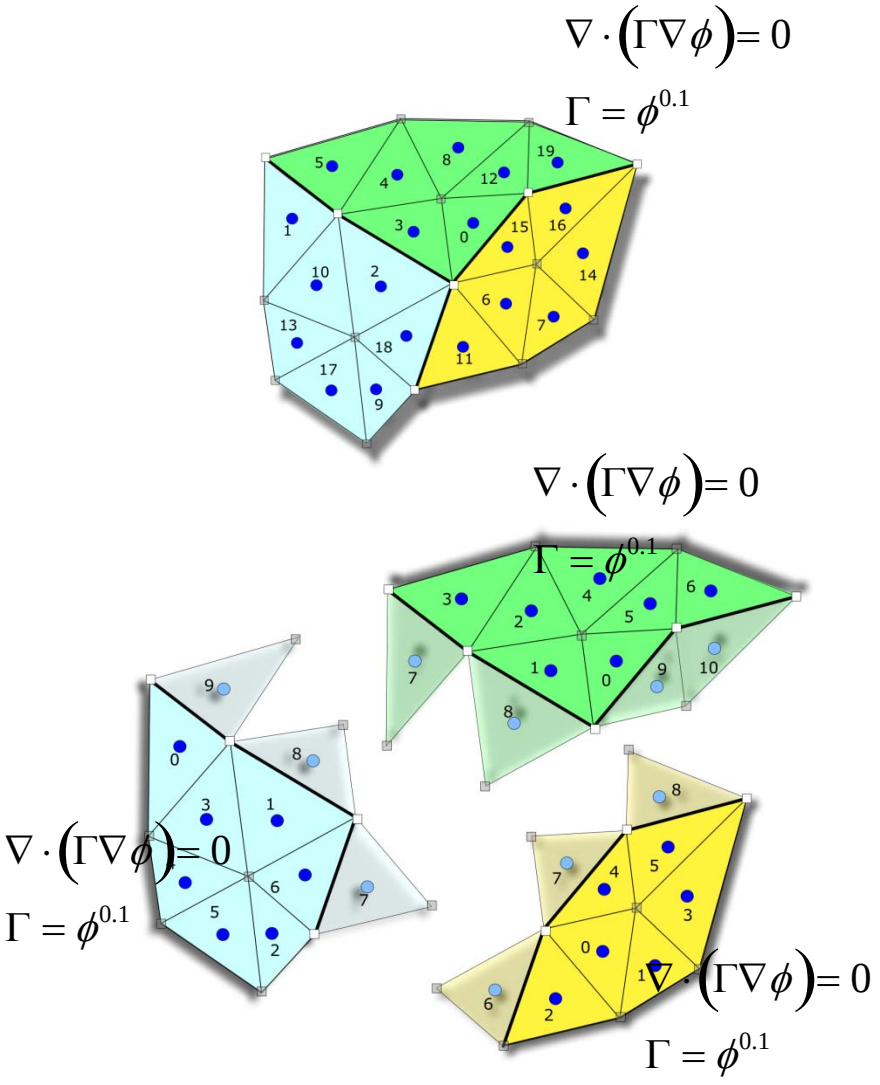
- As the name Designates, DP (DOMAIN PARTITIONING!) stands for the subdivision of a given **mesh** into **n** sub-meshes known as **partitions**.





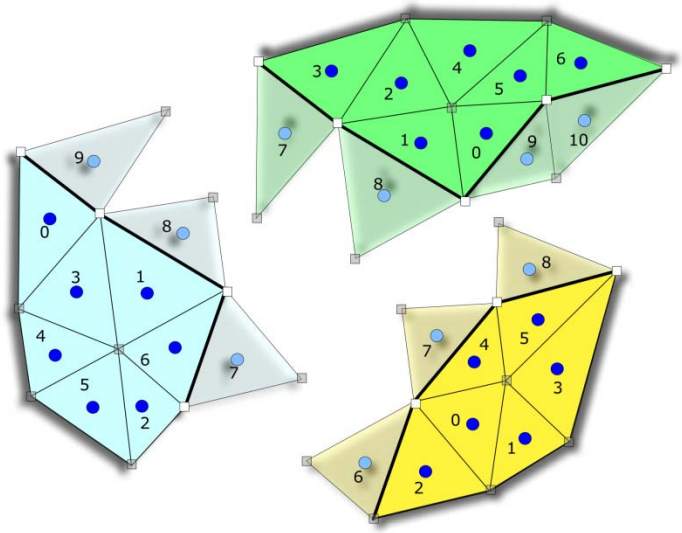
I- Domain Partitioning

- Each partition will “contain” the same physics (i.e. equations, BC) as the original mesh.
- Partitioning ensures load balancing
- Most famous method to partition is the recursive bisection method
  - start by dividing the domain into two, and then each of the sub-domains will be divided into two…and so on…



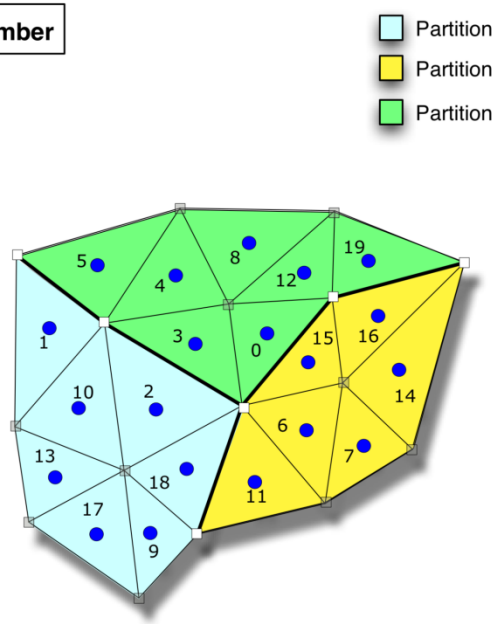
II– Partition Renumbering

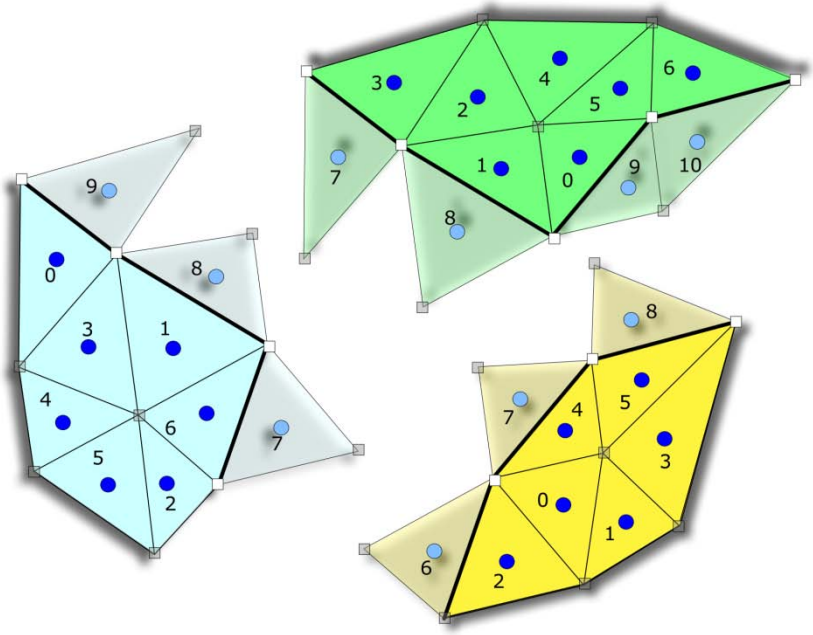
- Each partition will represent an independent mesh on which the problem is to be solved
- The entities representing the mesh should be locally renumbered
- The process of partition renumbering is taken care of by the FVPartitioner class object



II– Partition Renumbering

Global Cell	Partition Number
0	2
1	0
2	0
3	2
4	2
5	2
6	1
7	1
8	2
9	0
10	0
11	1
12	2
13	0
14	1
15	1
16	1
17	0
18	0
19	2





Partition 0

Core Local	Core Global
0	1
1	2
2	9
3	10
4	13
5	17
6	18

Shadow Local	Shadow Local	[Partition]
7	2	[1]
8	1	[2]
9	3	[2]

Partition 1

Core Local	Core Global
0	6
1	7
2	11
3	14
4	15
5	16

Shadow Local	Shadow Local	[Partition]
6	6	[0]
7	0	[2]
8	6	[2]

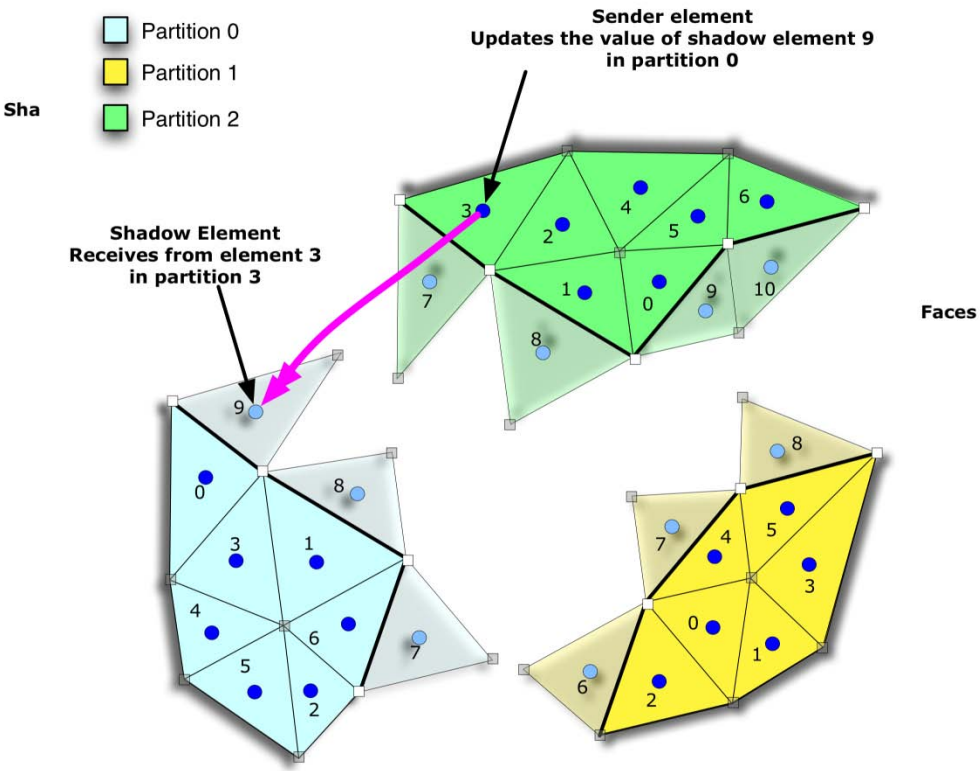
Partition 2

Core Local	Core Global
0	0
1	3
2	4
3	5
4	8
5	12
6	19

Shadow Local	Shadow Local	[Partition]
7	0	[0]
8	1	[0]
9	4	[1]
10	5	[1]

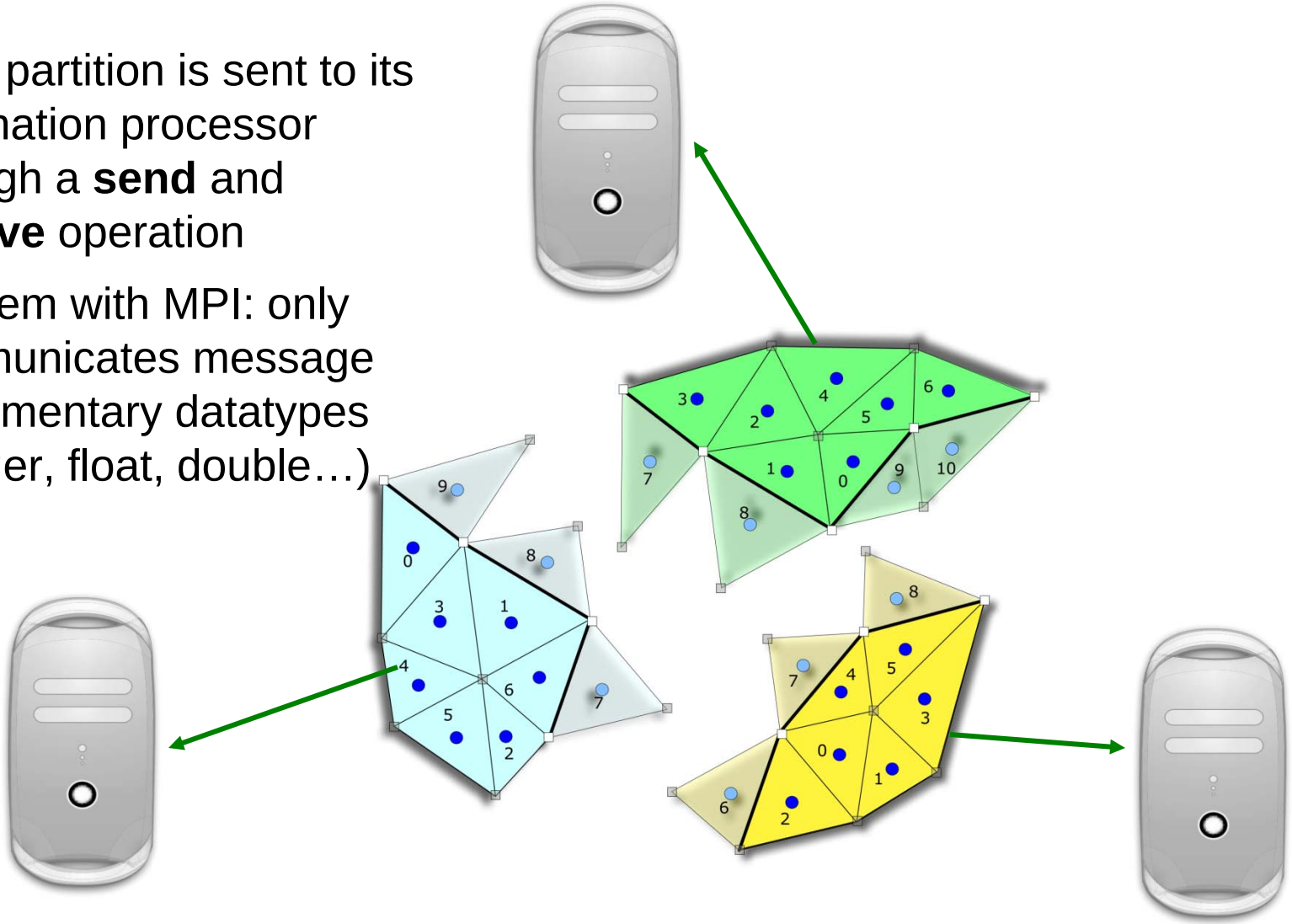
II– Partition Renumbering

- In order to maintain coupling throughout the whole domain, a special treatment is needed at th interface of any partition
- A **Shadow** entity is any topological entity that is not core to a given partition
- A **Sender** entity is any topological entity that updates a shadow entity



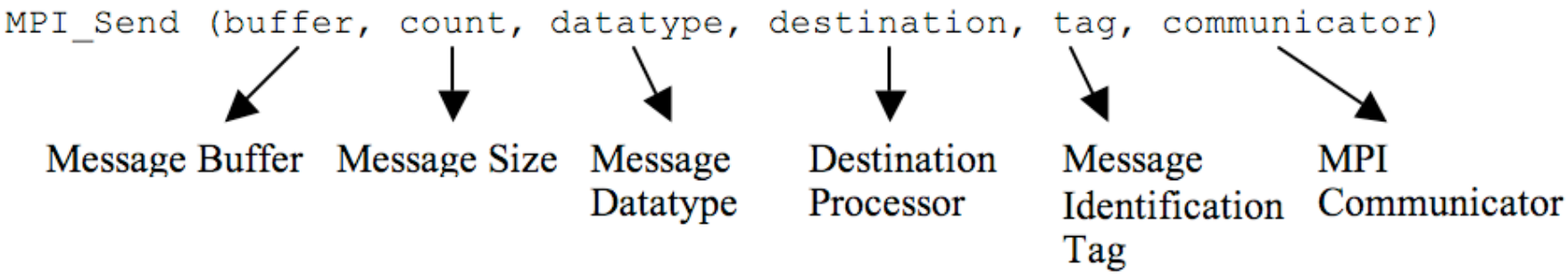
III– Partition Distribution

- Each partition is sent to its destination processor through a **send** and **receive** operation
- Problem with MPI: only communicates message of elementary datatypes (integer, float, double…)



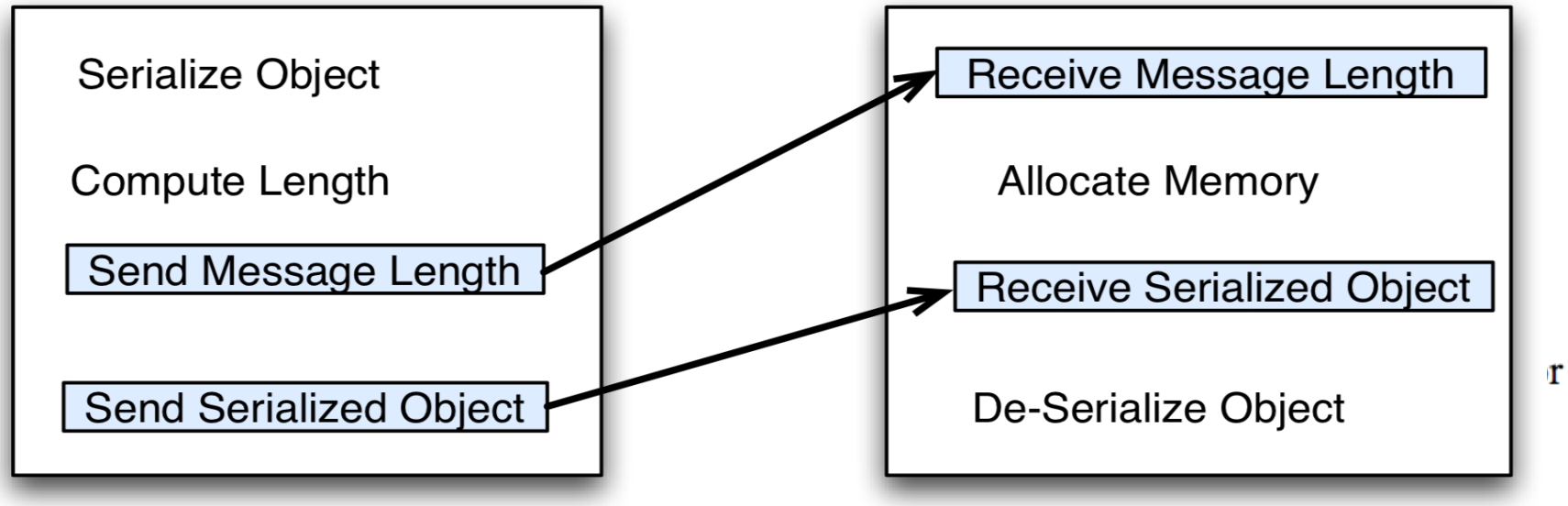
III– Partition Distribution

- MPI_Send()
- MPI_Receive()
- Solution to this problem is to use object serialization provided by Objective-C



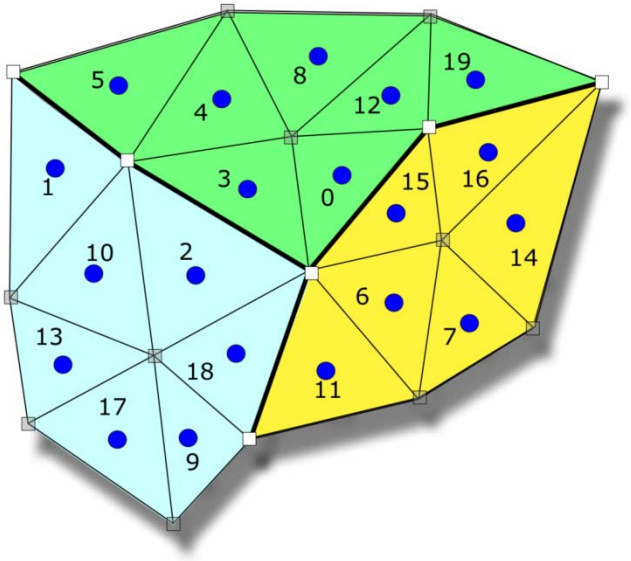
III– Partition Distribution

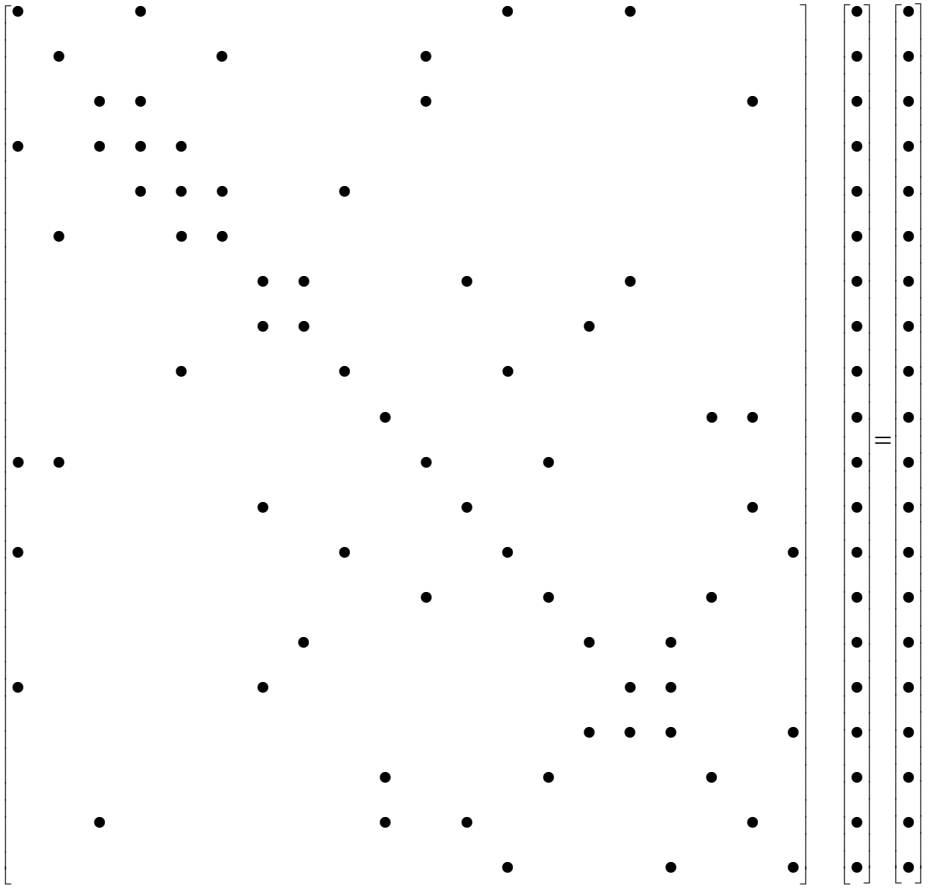
- MPI_Send()
- MPI_Receive()
- Solution to this problem is to use object serialization provided by Objective-C



IV- Parallel Solver

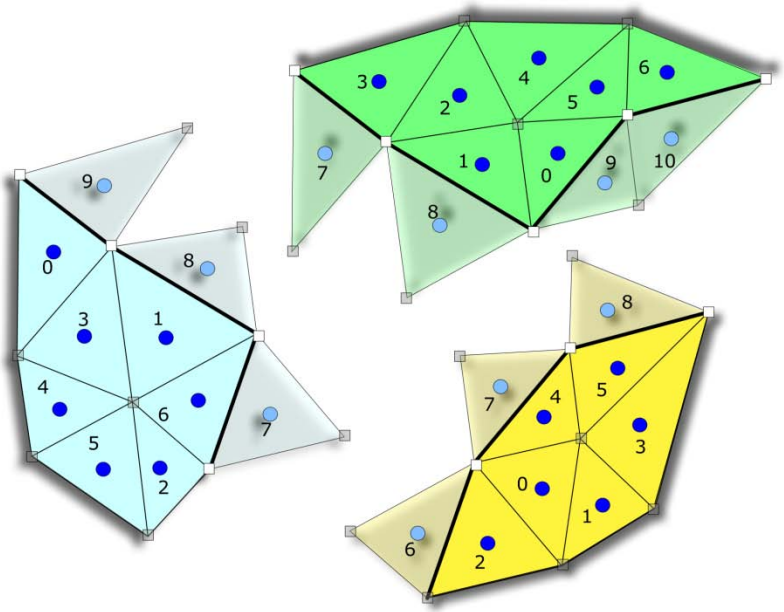
- The Iterative solver requires three steps
  - Solution
  - Communication
  - Decision (residual error)
- A linear system for a sequential run has the form

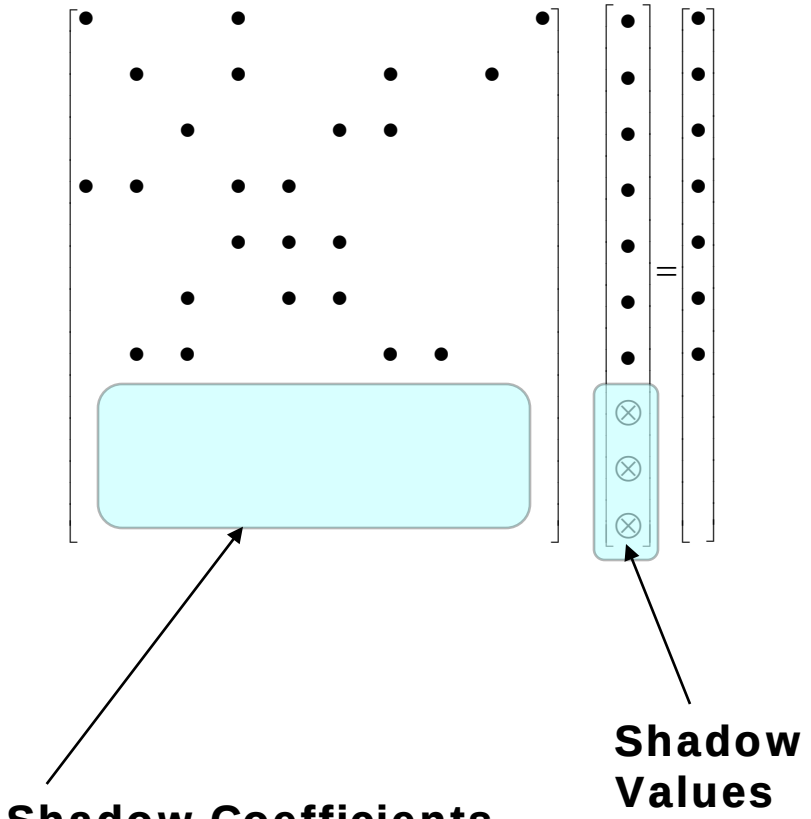




IV- Parallel Solver

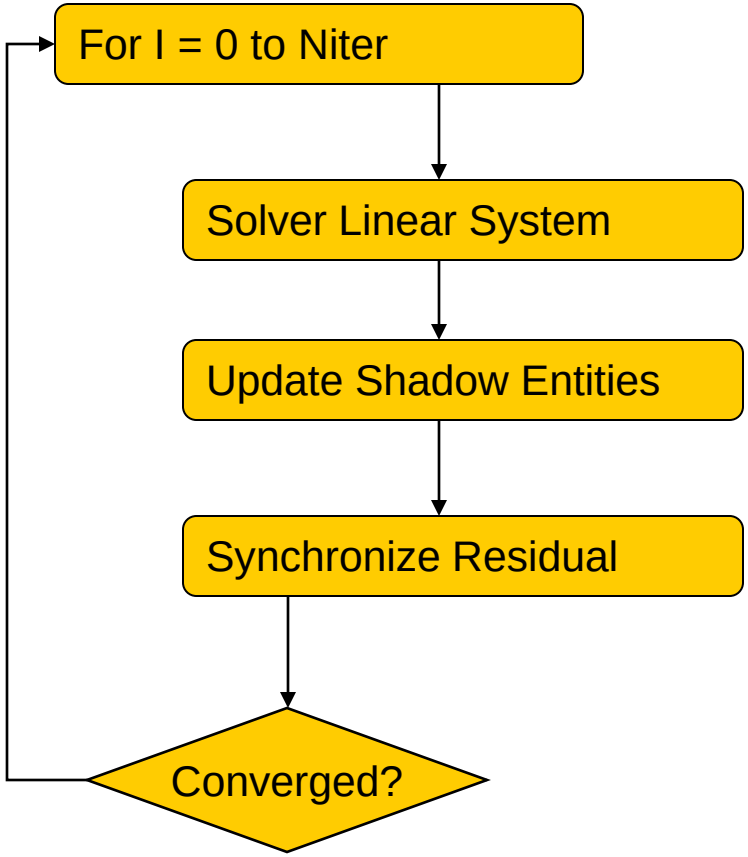
- The parallel linear solver proceeds in a manner similar to that of the sequential one
- Only one fundamental difference: The Shadow Elements





IV- Parallel Solver

- Solver Loop

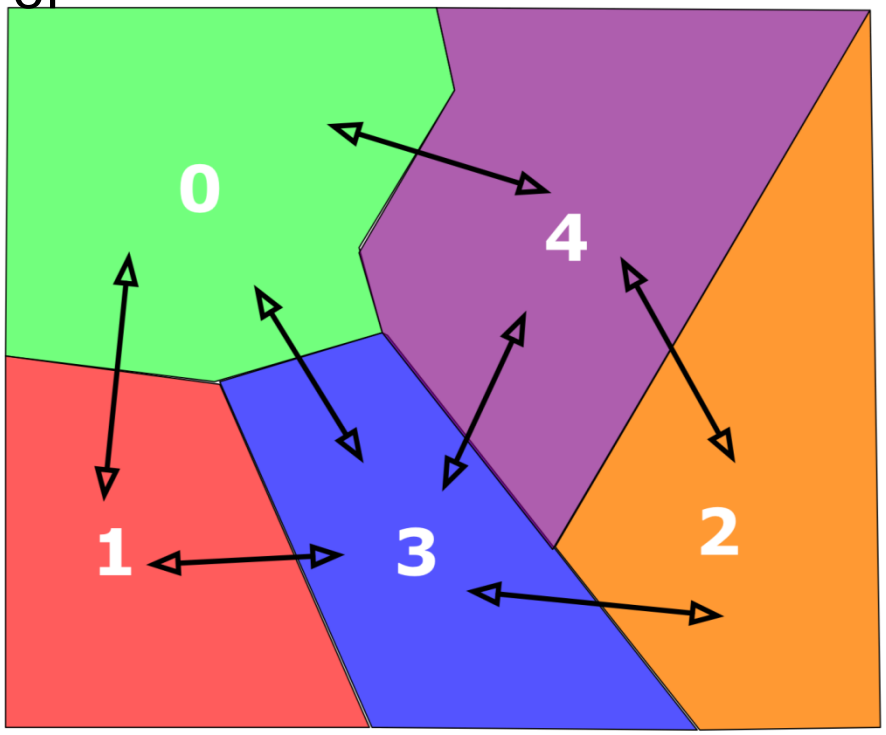


IV- Parallel Solver

- Interpartition communication is needed to exchange values at the interface
- This is problematic because of processor scheduling

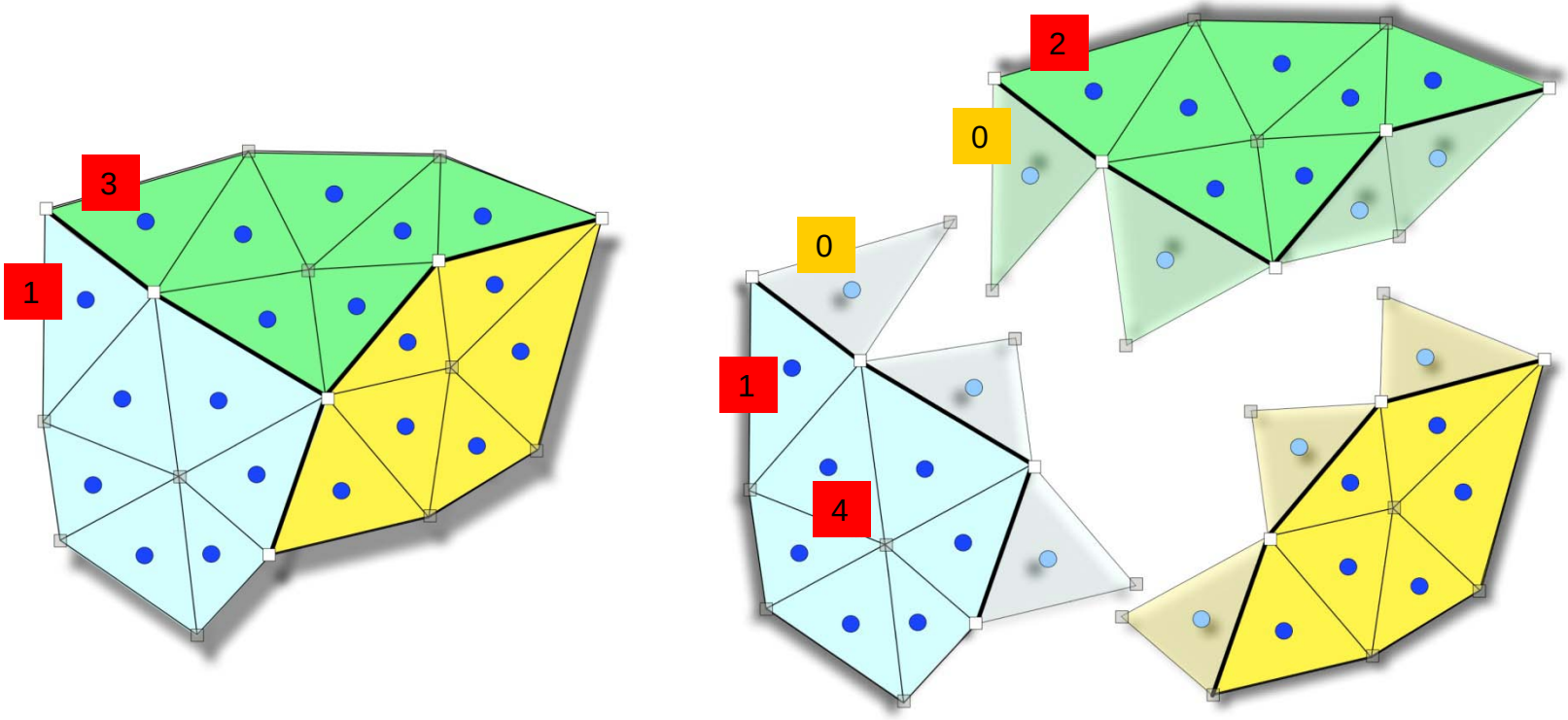
The need for scheduling arises from the fact that all processors are involved in the communication.

For example, while partition 0 is communicating with part. 1, part. 3 will be communicating w/t part. 0, 4 w/t 0,2 w/t 3 and so on…



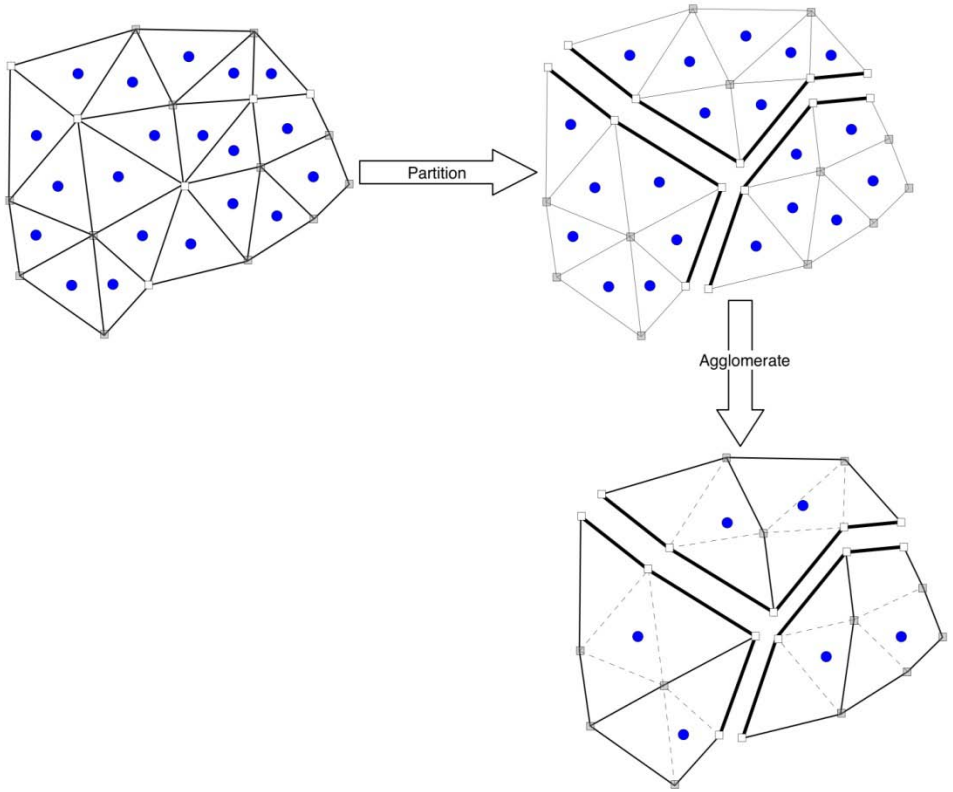
V- Parallel AMG Solver

- The parallel linear solver suffers from a fundamental weakness
- Values at the interface are always one iteration behind



V- Parallel AMG Solver

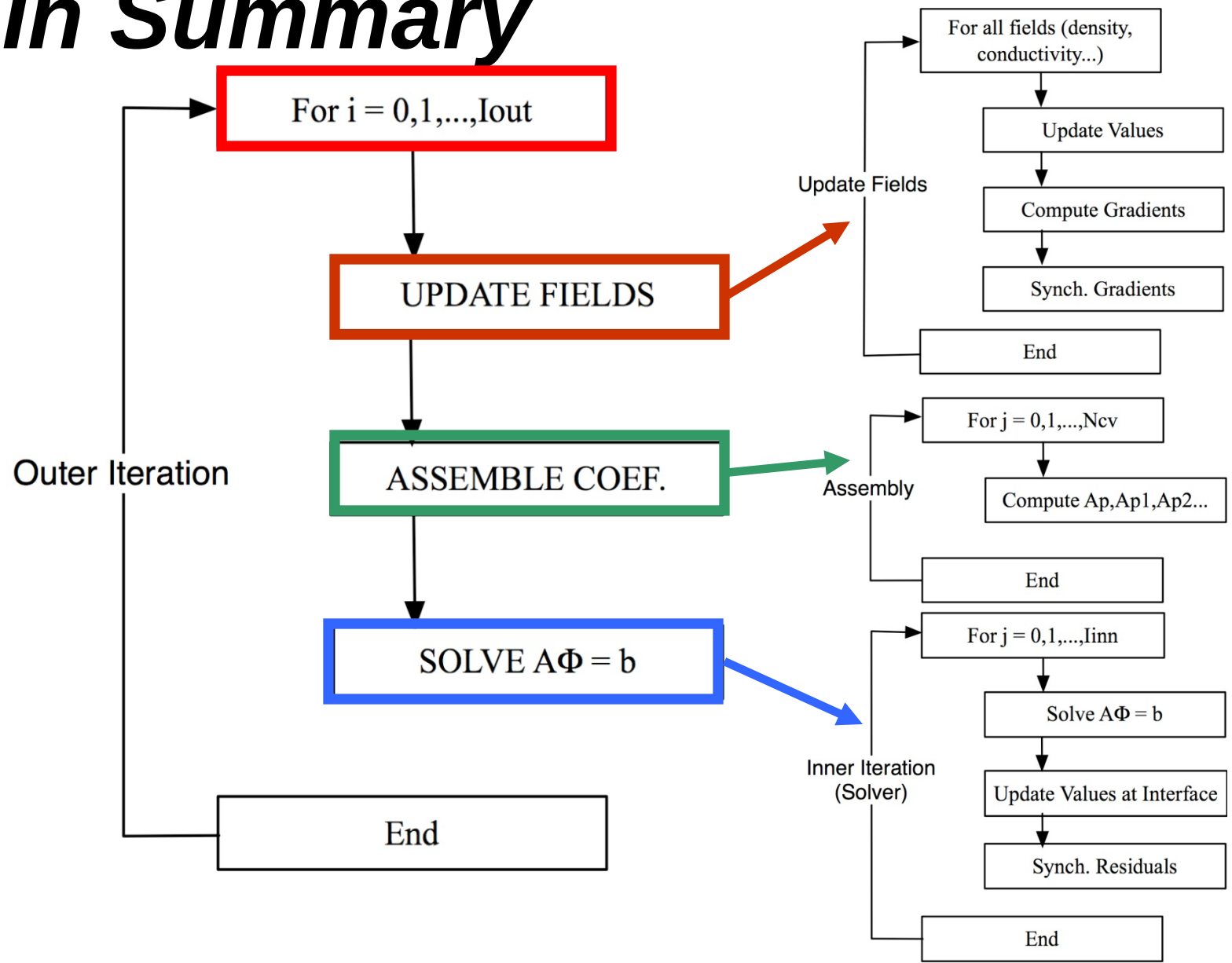
- The solution to this problem is to parallelize the AMG Solver
- There are two approaches:
  - Agglomerate & Partition
  - Partition & Agglomerate



V- Parallel AMG Solver

- At any given level, the shadow elements for a given partition are agglomerated according to the agglomeration of their corresponding sender elements in neighboring partitions
- The new shadow elements are then renumbered to conform with the current grid level
- Updates are made at the shadow elements on each level
- This makes sure that at the end of each outer iteration we have the same solution as that of the sequential

In Summary


For i = 0,1,...,Iout
UPDATE FIELDS
ASSEMBLE COEF.
SOLVE AΦ = b
End
Outer Iteration
Update Fields
For all fields (density, conductivity...)
Update Values
Compute Gradients
Synch. Gradients
End
Assembly
For j = 0,1,...,Ncv
Compute Ap,Ap1,Ap2...
End
Inner Iteration (Solver)
For j = 0,1,...,Iinn
Solve AΦ = b
Update Values at Interface
Synch. Residuals
End

Results

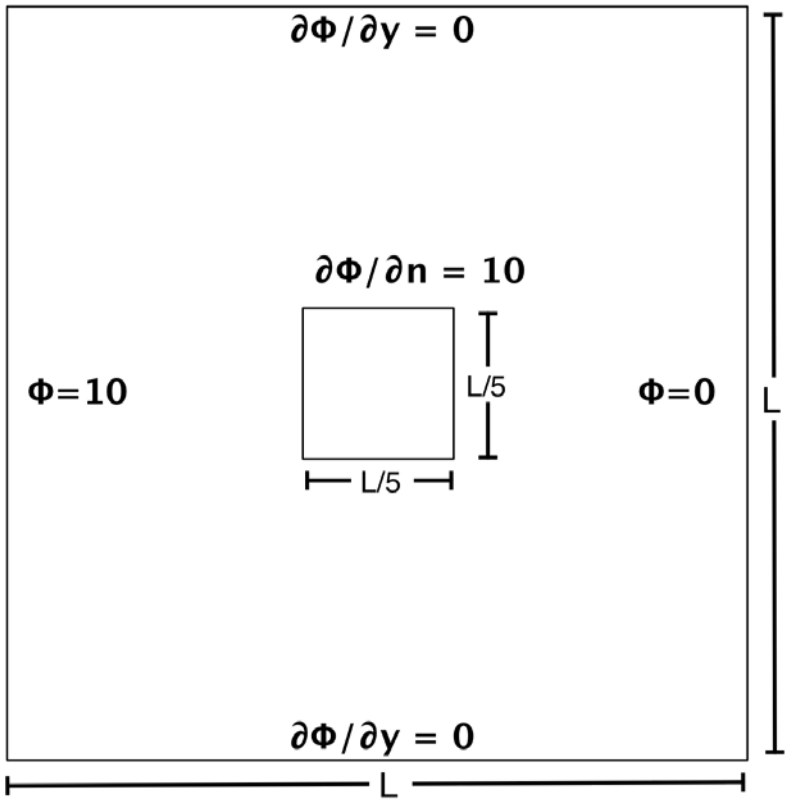
- Runs were made on the **AUB Mac** cluster
- It consists of 10 dual processor nodes with 512 MB each
- 100MB standard Ethernet switch

Results: Test Case 1

- V, F and W Cycles
- Meshes: 100,009; 200,025; 400,187; 600,245 and 1,001,430 elements

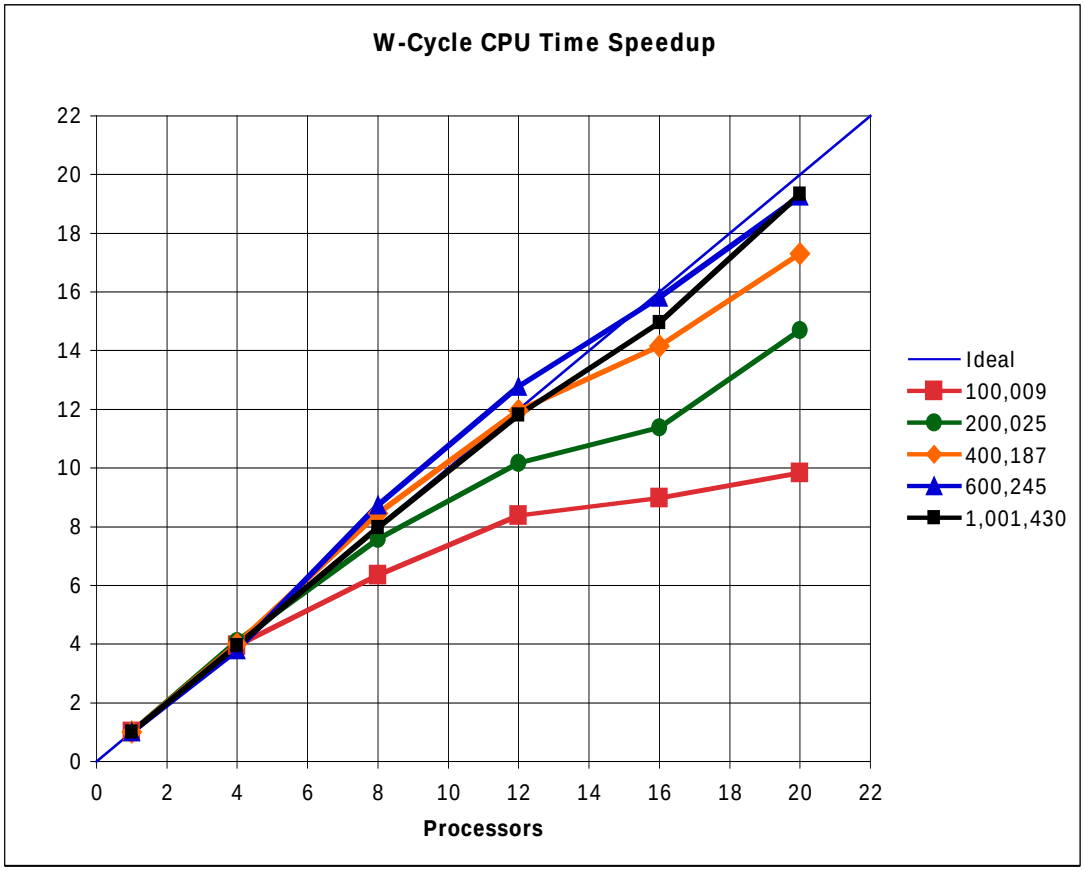
$$\nabla \cdot \left(\Gamma \nabla \phi \right) = 0$$

$$\Gamma = \phi^{0.1}$$



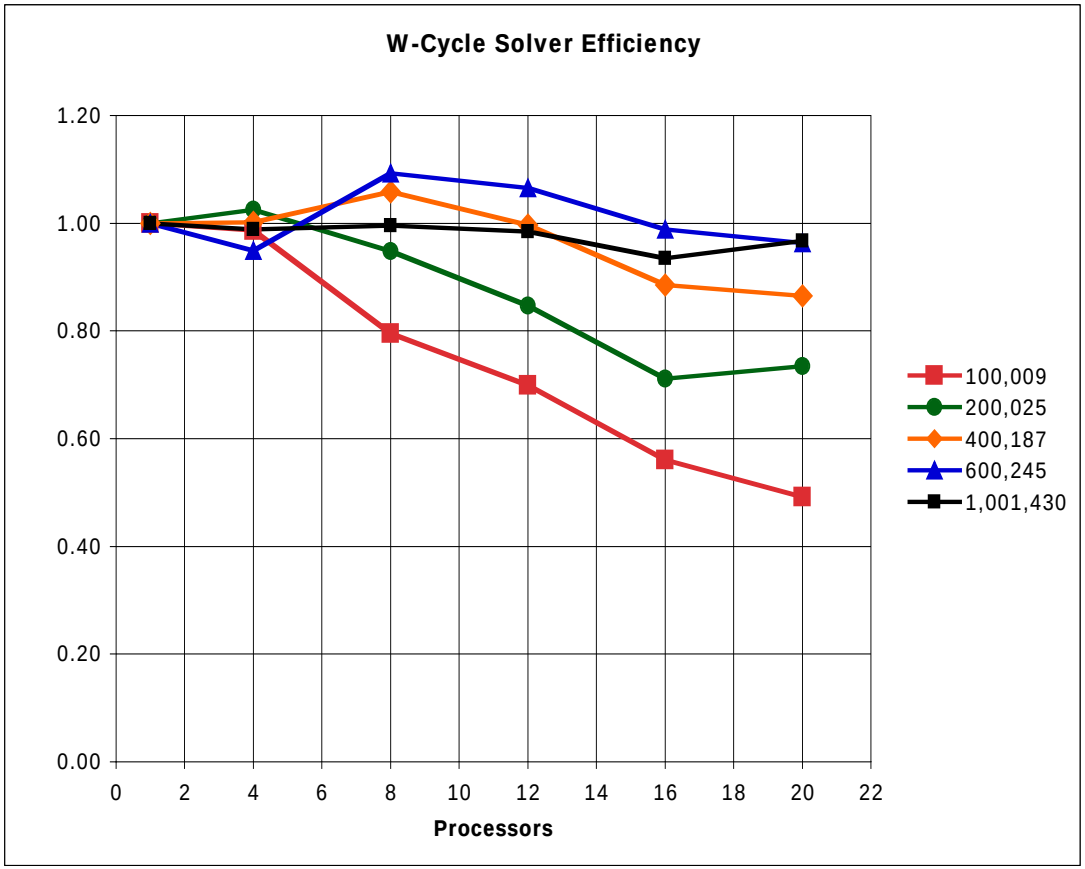
Results: Test Case 1

- Speedup



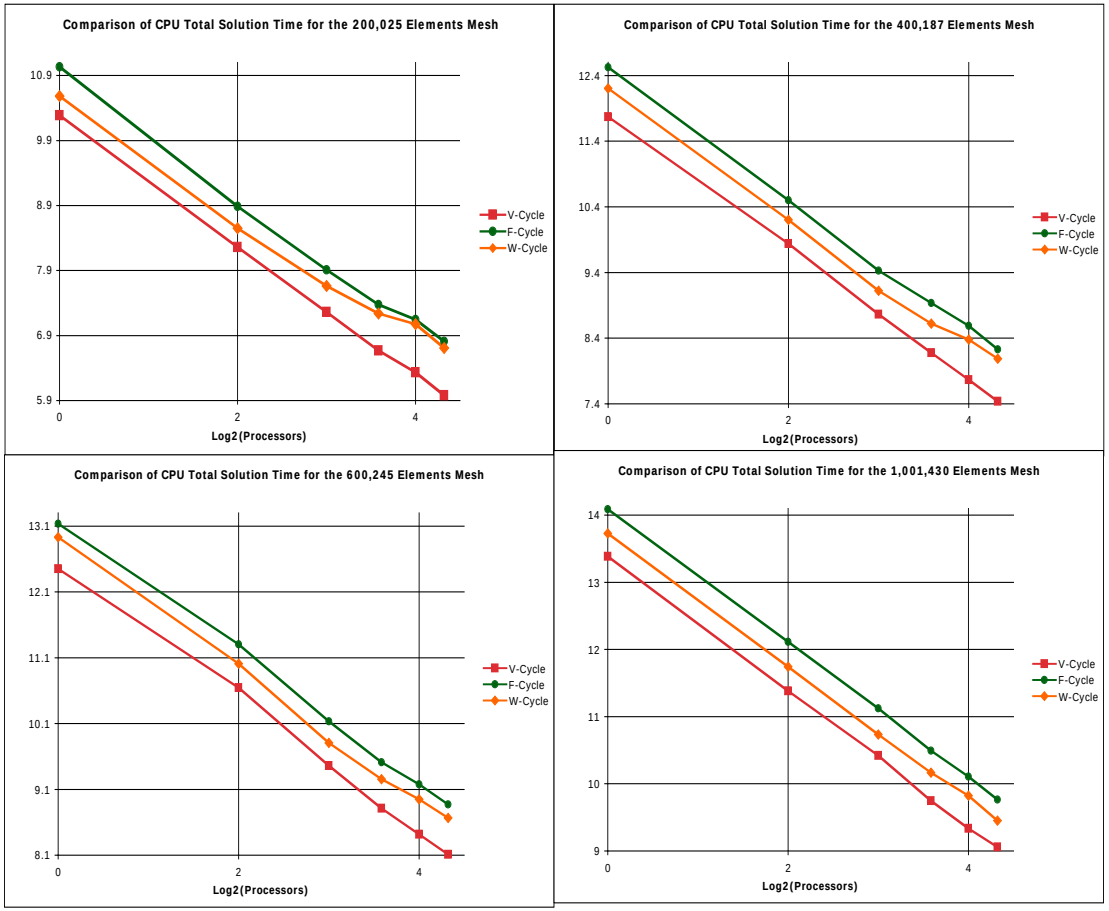
Results: Test Case 1

- Speedup
- Efficiency



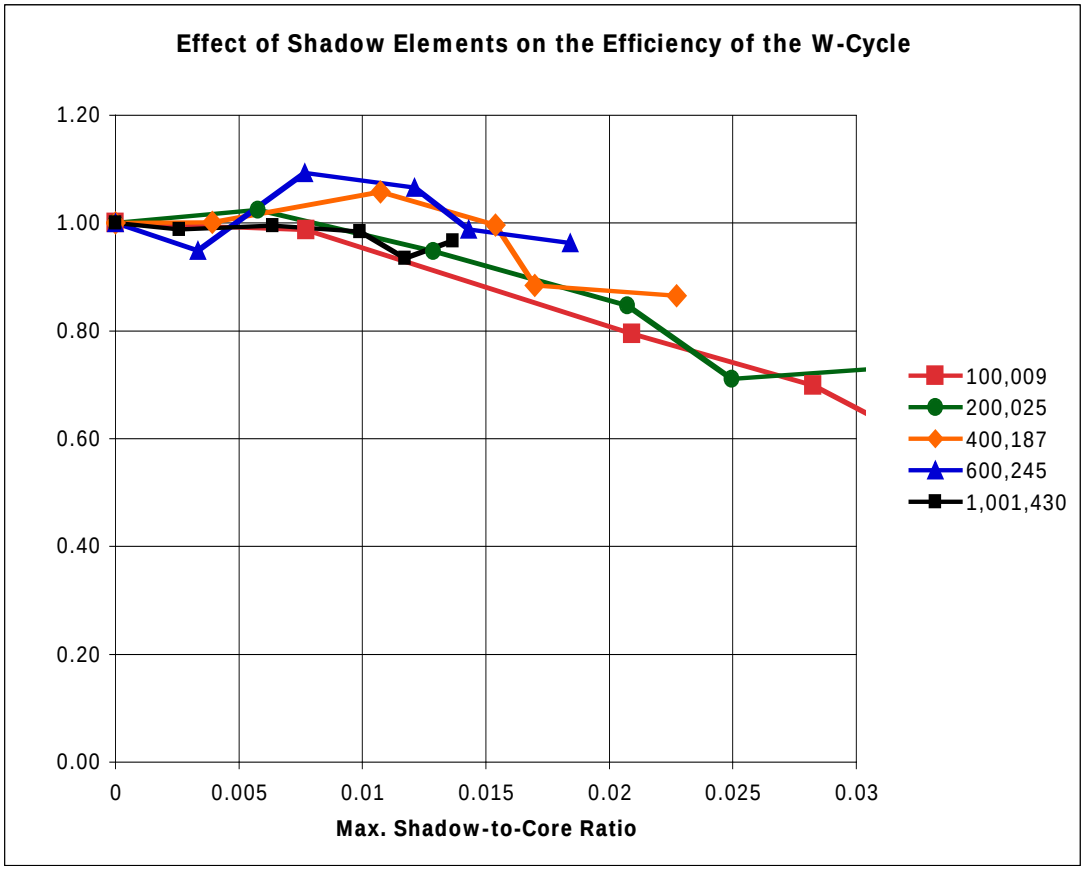
Results: Test Case 1

- Speedup
- Efficiency
- Solution Time



Results: Test Case 1

- Speedup
- Efficiency
- Solution Time
- Effect of shadow Elements

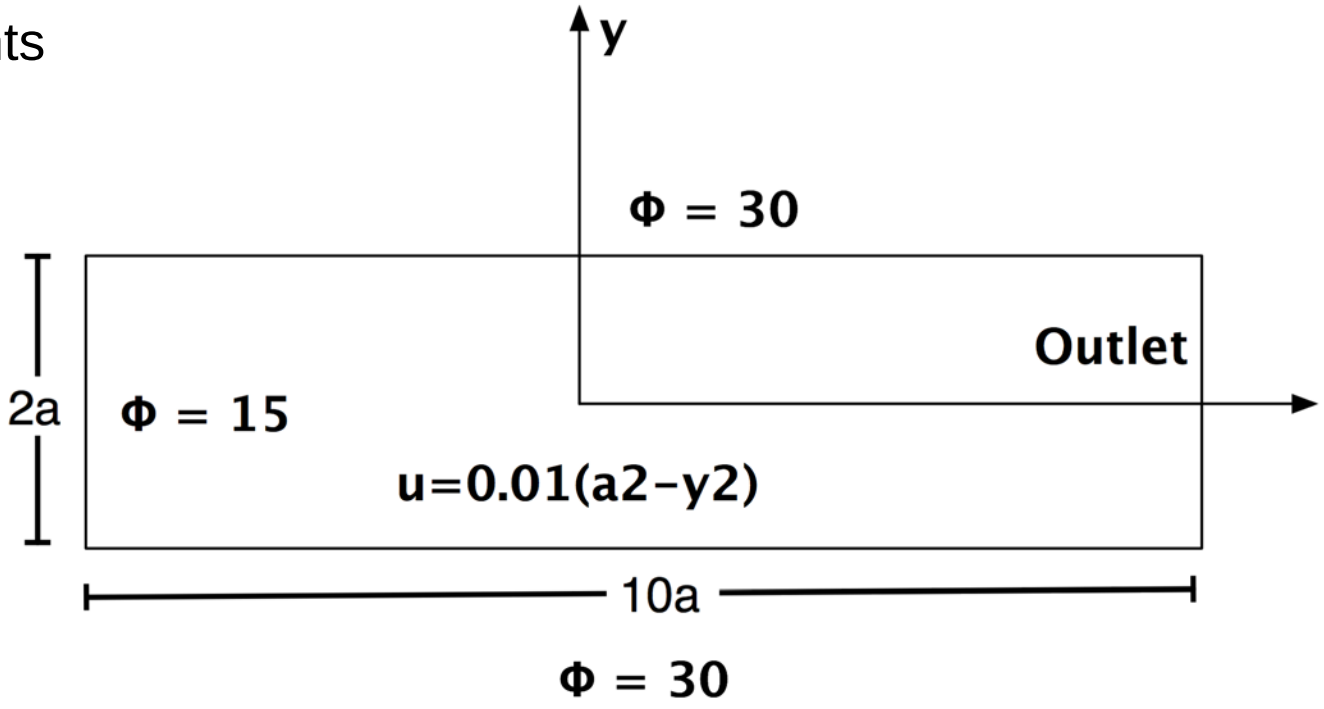


Results: Test Case 2

- V, F and W Cycles
- Meshes: 100,746, 201,584, 400,821, 600,347 and 1,000,643 elements

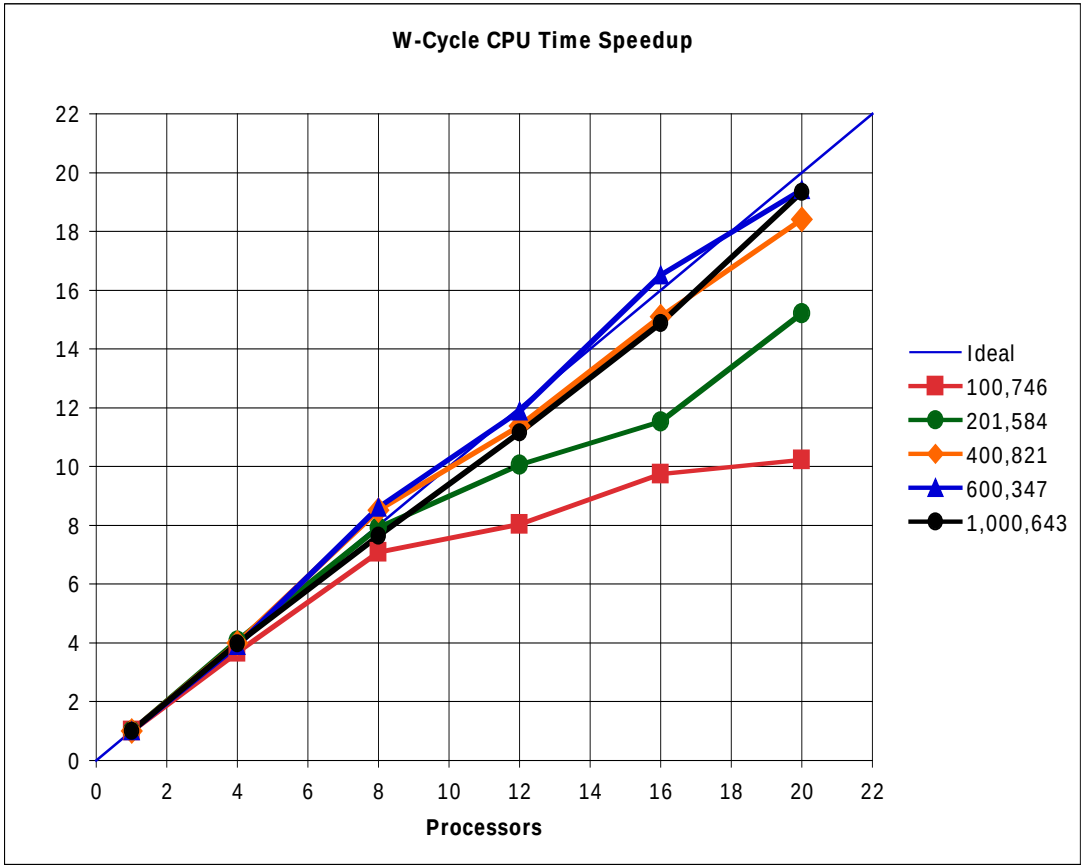
$$\nabla \cdot (\rho \boldsymbol{u}) = \nabla \cdot (\Gamma \nabla \phi)$$

$$\Gamma = \phi^{0.1}, \boldsymbol{u} = 0.01 \times (a^2 - y^2)$$



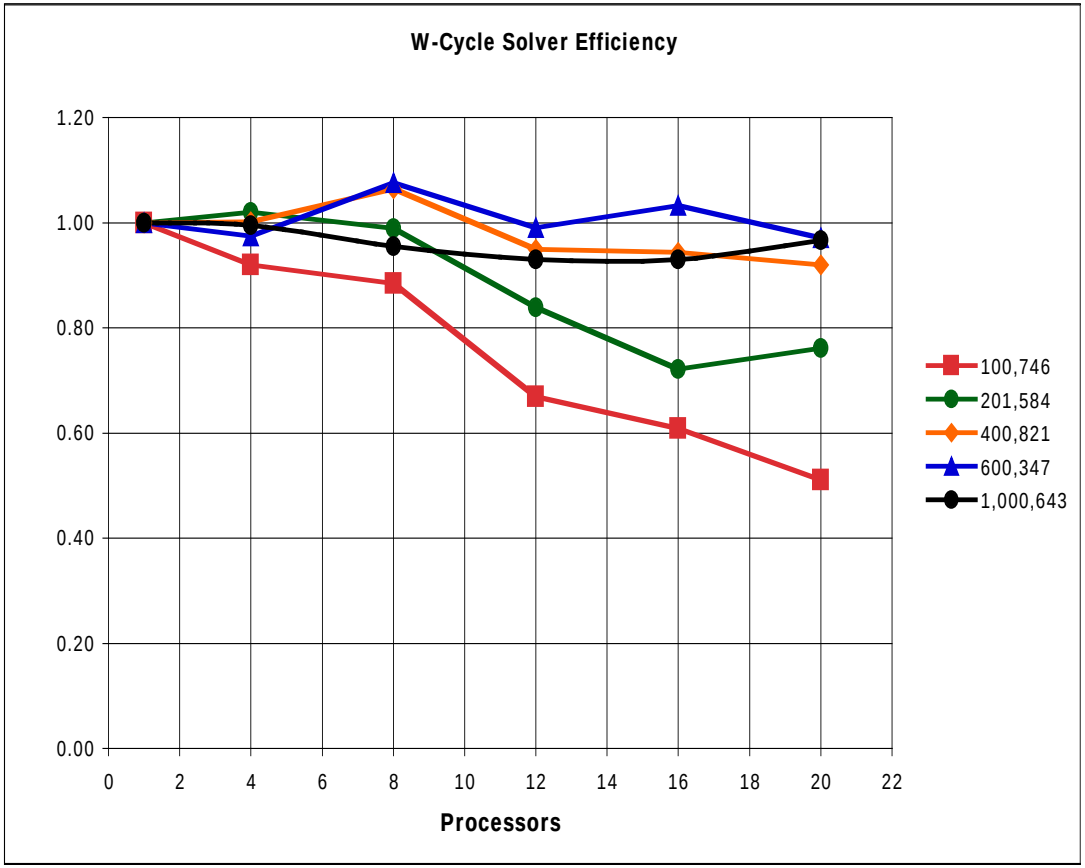
Results: Test Case 2

- Speedup



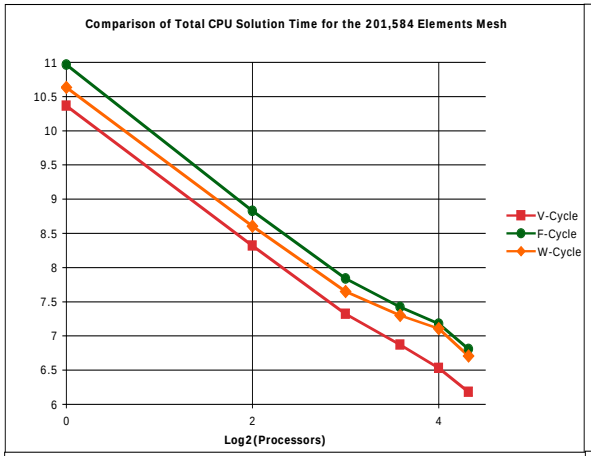
Results: Test Case 2

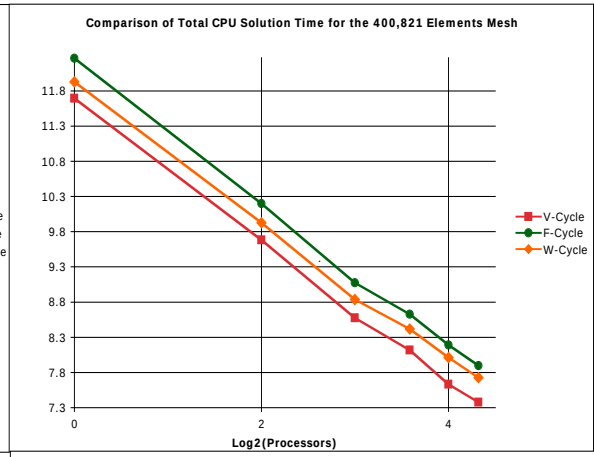
- Speedup
- Efficiency

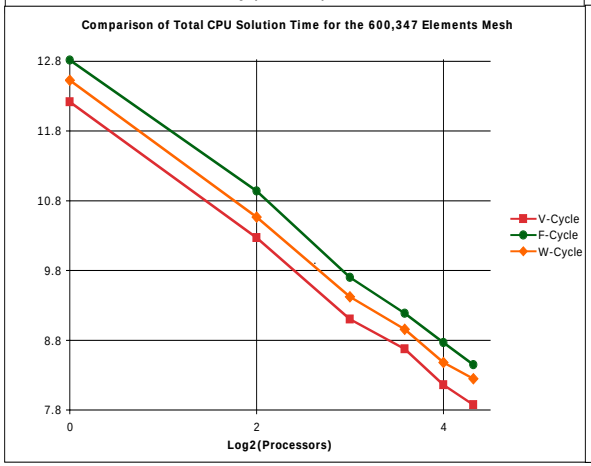


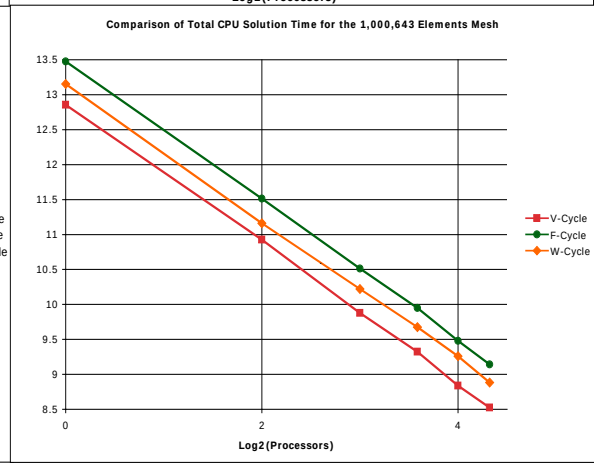
Results: Test Case 2

- Speedup
- Efficiency
- Solution Time



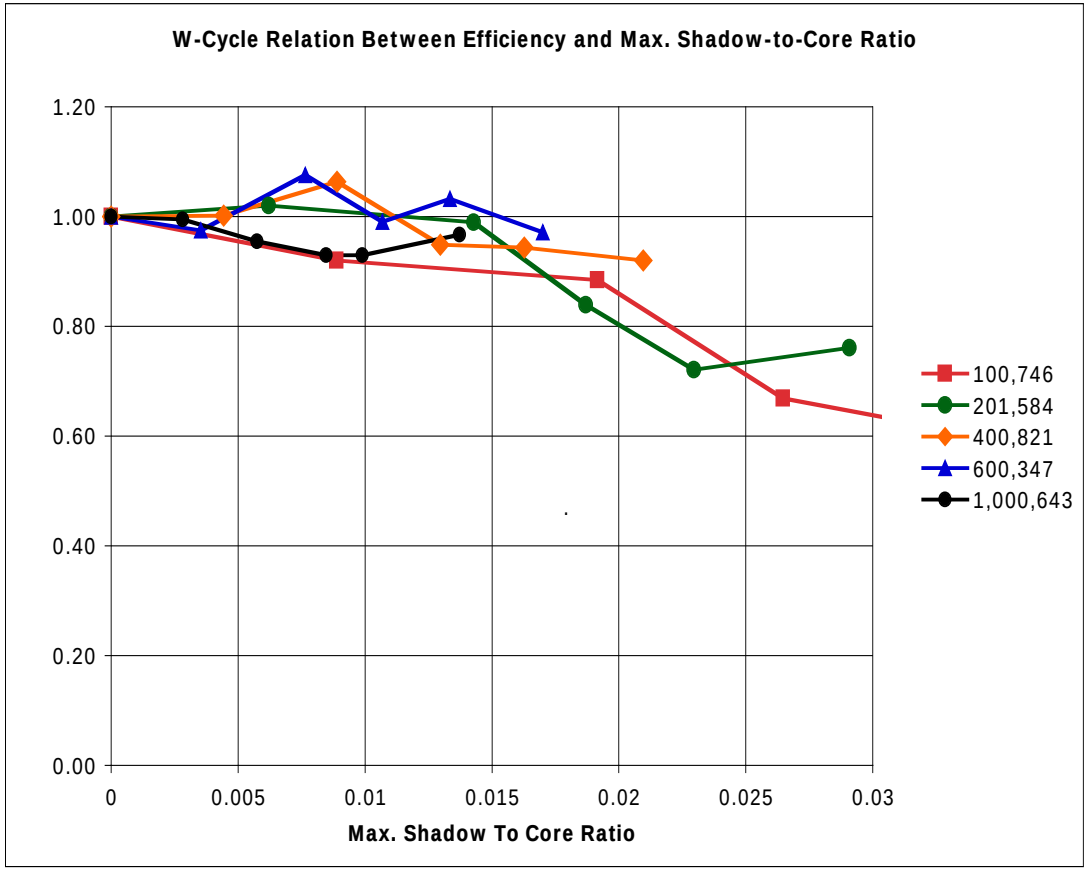






Results: Test Case 2

- Speedup
- Efficiency
- Solution Time
- Effect of shadow Elements



Conclusions

- Were the Objectives met?
  - ✓ Object oriented parallel CFD solver design
  - ✓ Highly Scalable AMG solver
  - ✓ Transparent code to easily accommodate additional models
- What next?
  - ✓ Pressure correction
  - ✓ Migration to 3D
  - ✓ Multiphase flow

THANK YOU FOR ATTENDING