Performance Strategies for Parallel Mathematical Libraries Based on Historical Knowledgebase

CScADS workshop 2009

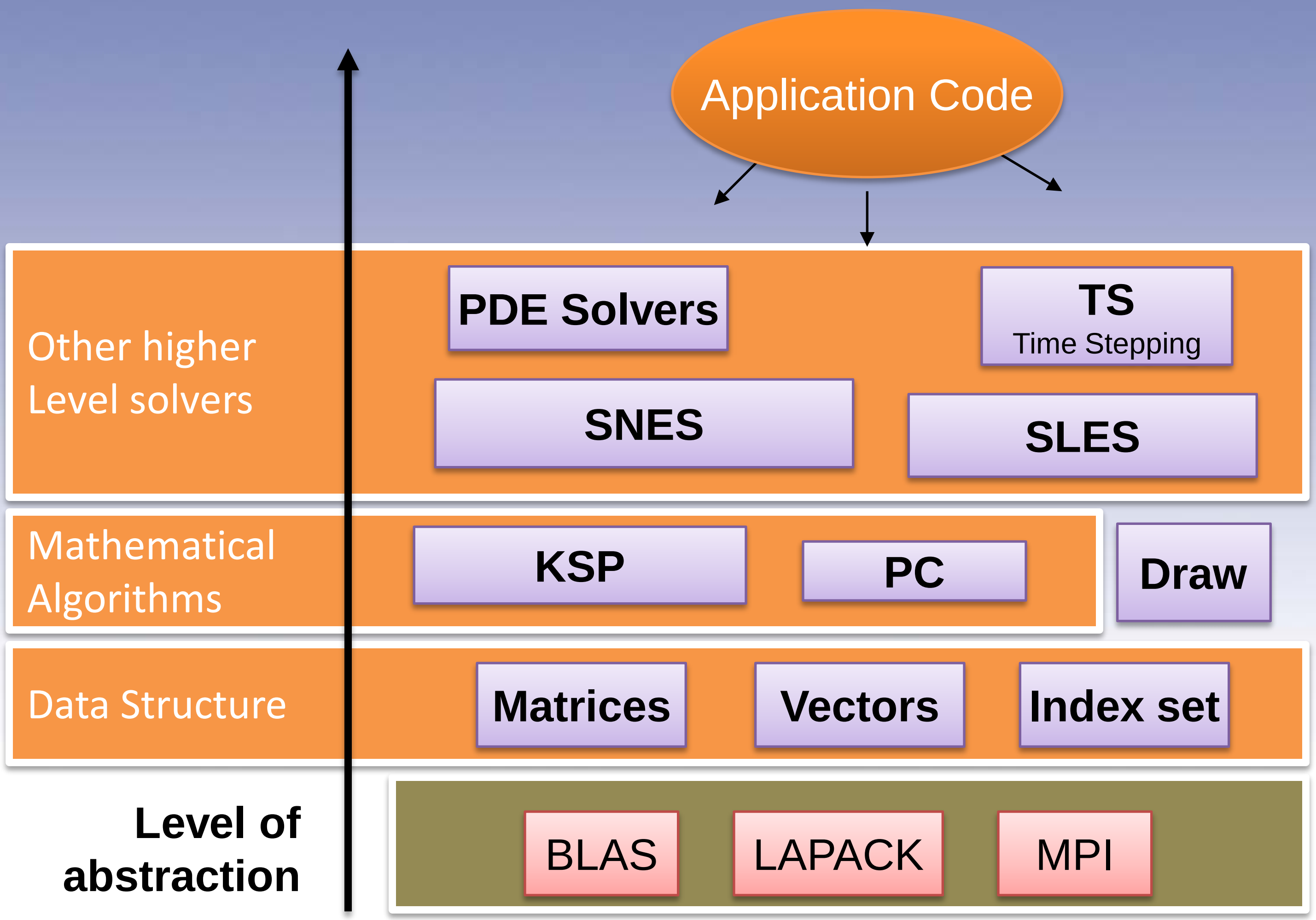
Eduardo Cesar, Anna Morajko, Ihab Salawdeh
Universitat Autònoma de Barcelona

TABLE OF CONTENTS

OBJECTIVE

Design a methodology, based on input data, that can automatically choose between existing solving parameters to improve PETSc application's performance

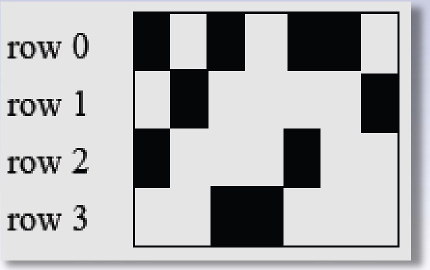
PETSC

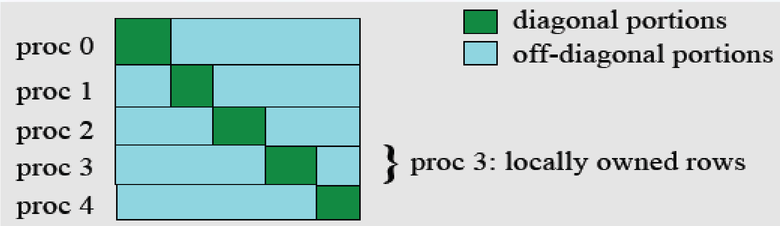

Application Code
Other higher
Level solvers
PDE Solvers
TS
Time Stepping
SNES
SLES
Mathematical
Algorithms
KSP
PC
Draw
Data Structure
Matrices
Vectors
Index set
Level of
abstraction
BLAS
LAPACK
MPI

PETSC

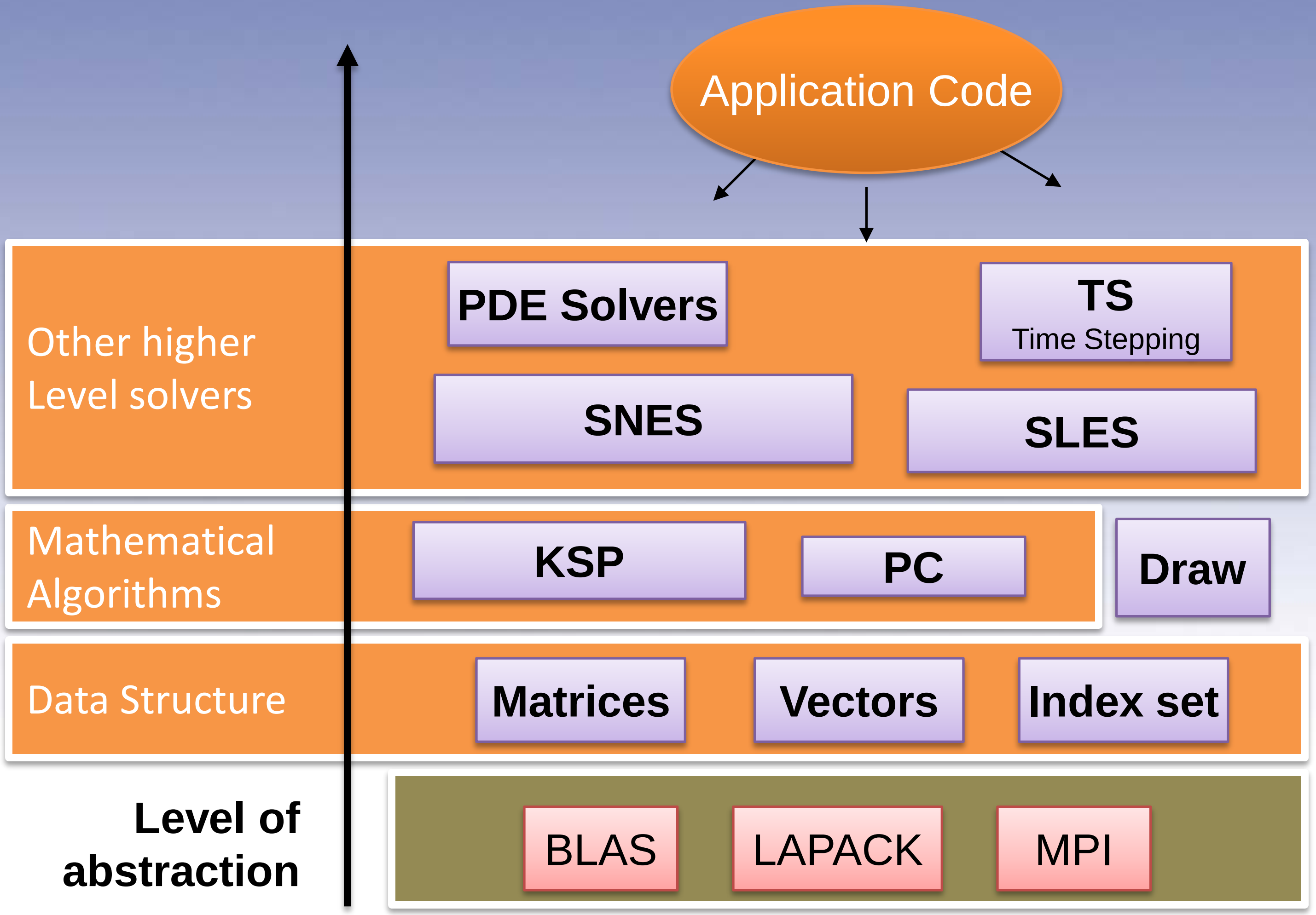
Data Structure | **Matrices** | **Vectors** | **Index set**

- Fundamental objects for storing linear operators
- There is no data structure appropriate for all problems.
- PETSc supports:
  - dense storage where each process stores its entries in the usual Array style.
  - Compressed sparse row storage, where only the nonzero values be stored.
  - Block compressed Sparse row Storage
  - Block Diagonal Storage





PETSC

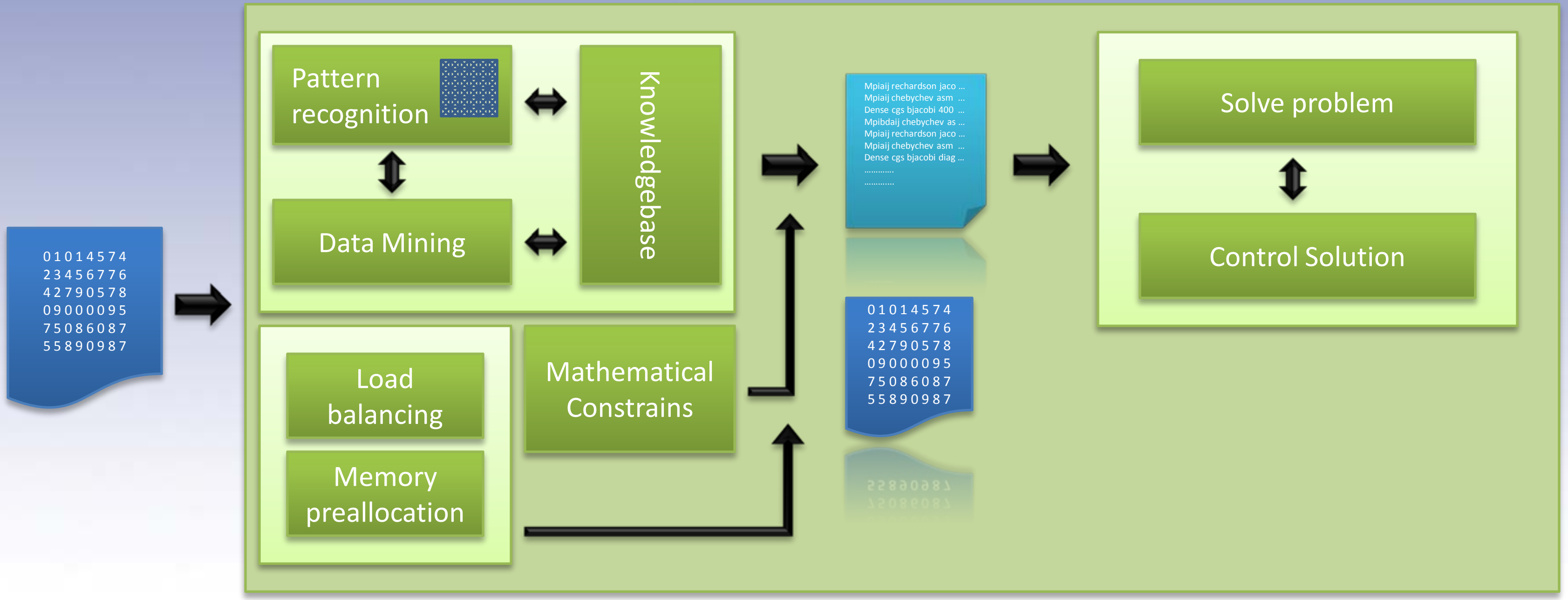

Application Code
Other higher
Level solvers
PDE Solvers
TS
Time Stepping
SNES
SLES
Mathematical
Algorithms
KSP
PC
Draw
Data Structure
Matrices
Vectors
Index set
Level of
abstraction
BLAS
LAPACK
MPI

PETSC

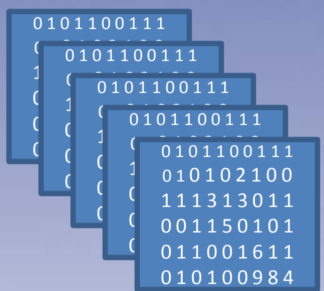
Mathematical Algorithms | KSP | PC

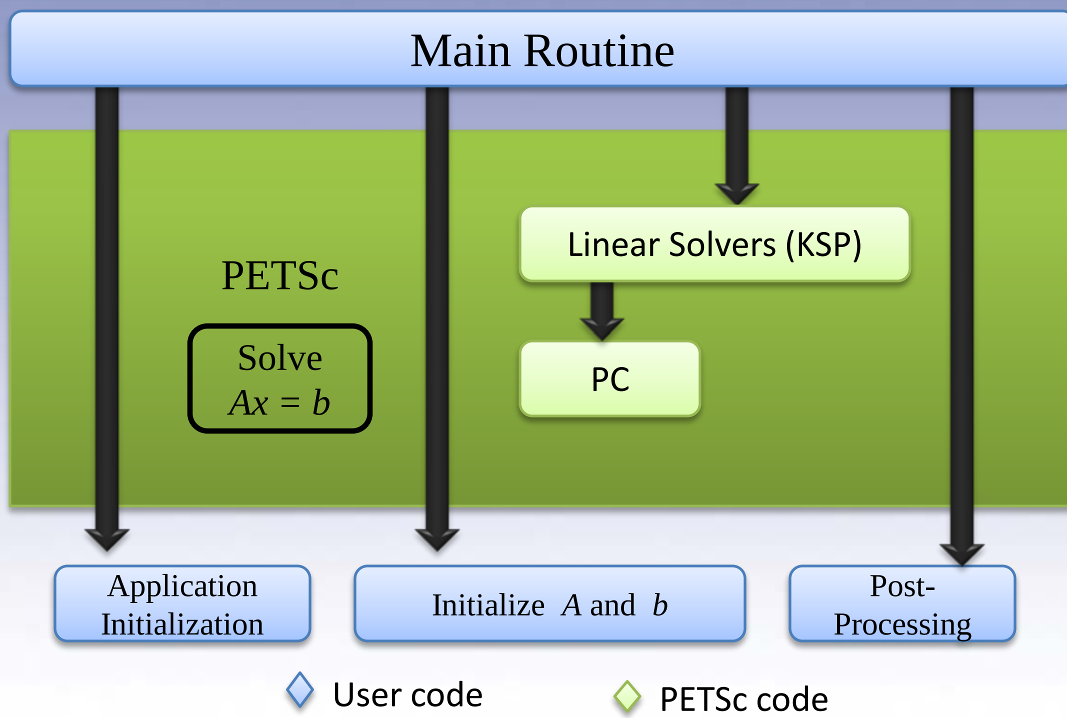
- To solve a linear system you need to calculate the matrix's preconditioner (PC) then to apply the Krylov Subspace method (KSP)
- Different mathematical algorithms exist either to calculate the preconditioner (jacobi, LU, etc.) or for the KSP (GMRES, CG, etc.)

PERFORMANCE METHODOLOGY

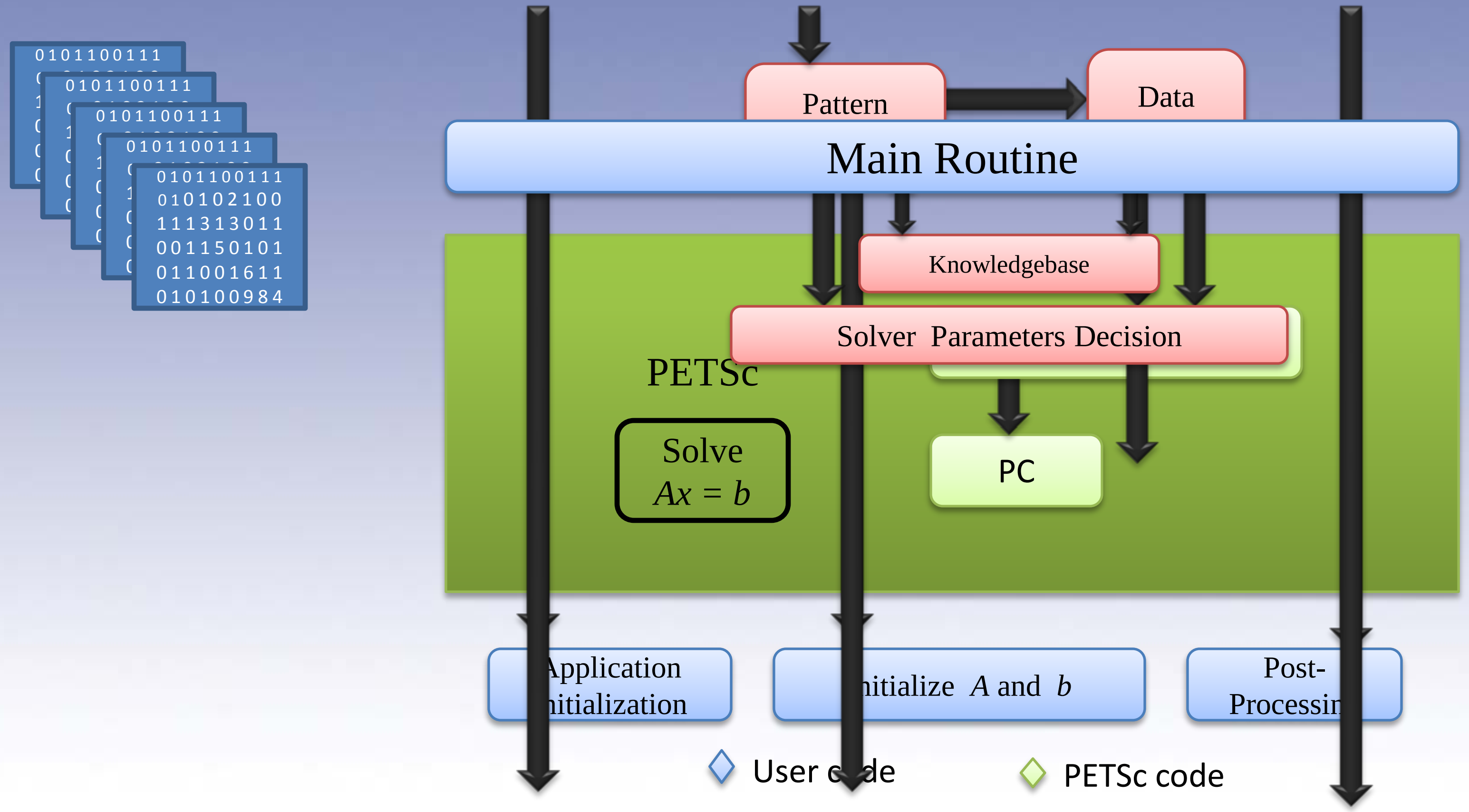


PETSC SOLVING PROCESS



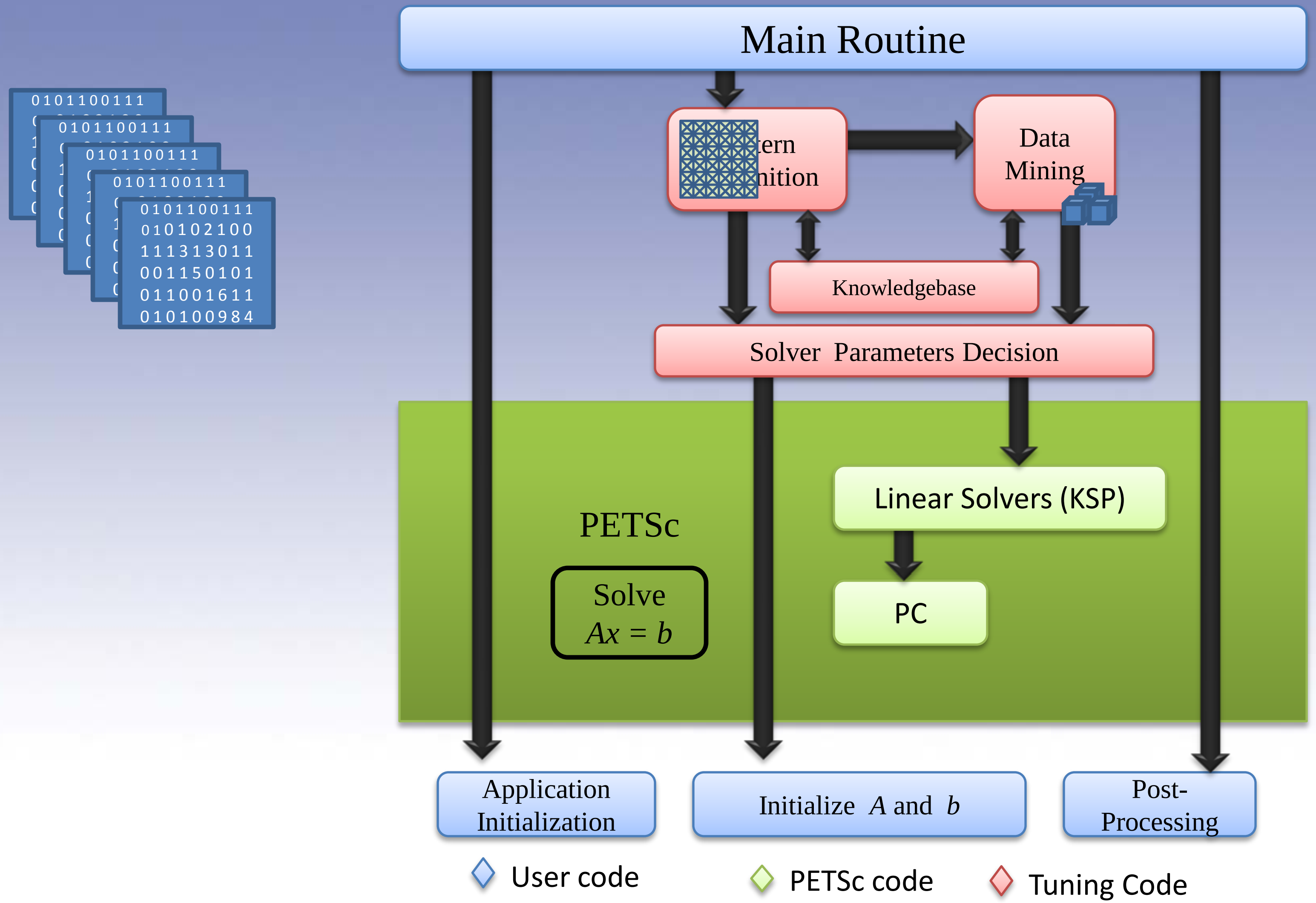


PETSC PERFORMANCE METHODOLOGY



User code PETSc code Tuning Code

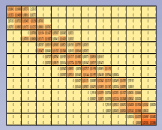
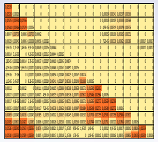
PETSC PERFORMANCE METHODOLOGY



KNOWLEDGEBASE

- Contains performance characterization of historical executions for different input data and hardware configurations
- It is organized according to the most common matrix patterns and the matrix size

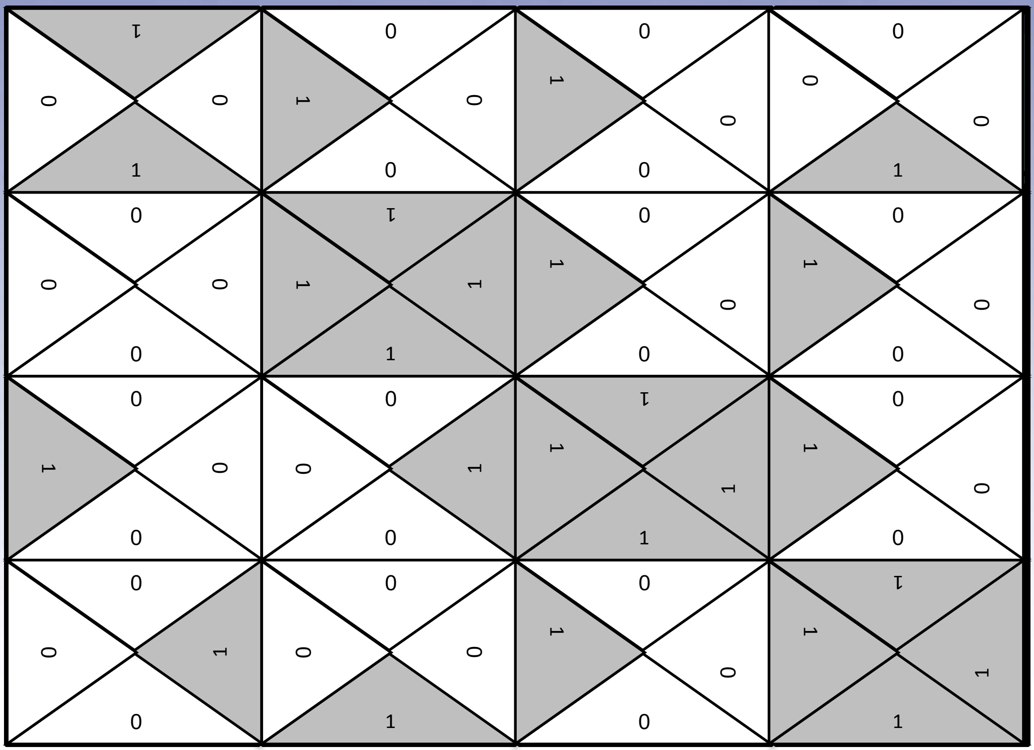
KNOWLEDGEBASE

Input Type	Input Size	No. Processors	Memory Representation	KSP	PC	Total Time (s)	Matrix Pattern
Around-Diagonal	9152X9152	8	mpiaij	bcgs	asm	103,648252	
Around-Diagonal	9152X9152	8	mpiaij	bcgs	bjacobi	215,159341	
Around-Diagonal	9152X9152	8	mpiaij	bcgs	jacobi	35,788464	
Around-Diagonal	9152X9152	8	mpiaij	bcgsl	asm	222,055694	
Around-Diagonal	9152X9152	8	mpiaij	bcgsl	bjacobi	189,236794	
Around-Diagonal	9152X9152	8	mpiaij	bcgs	asm	103,648252	
…	…	…	…	…	…	…	
Distributed	19242X19242	16	mpiaij	bcgs	asm	522,155023	
Distributed	19242X19242	16	mpiaij	bcgs	bjacobi	455,43445	
Distributed	19242X19242	16	mpiaij	bcgs	jacobi	459,629514	
Distributed	19242X19242	16	mpiaij	bcgsl	asm	457,210476	
Distributed	19242X19242	16	mpiaij	bcgsl	bjacobi	456,518158	
Distributed	19242X19242	16	mpiaij	bcgsl	jacobi	455,395112	
…	…	…	…	…	…	…	

DATA ANALYSIS

25 Non-zero Blocks

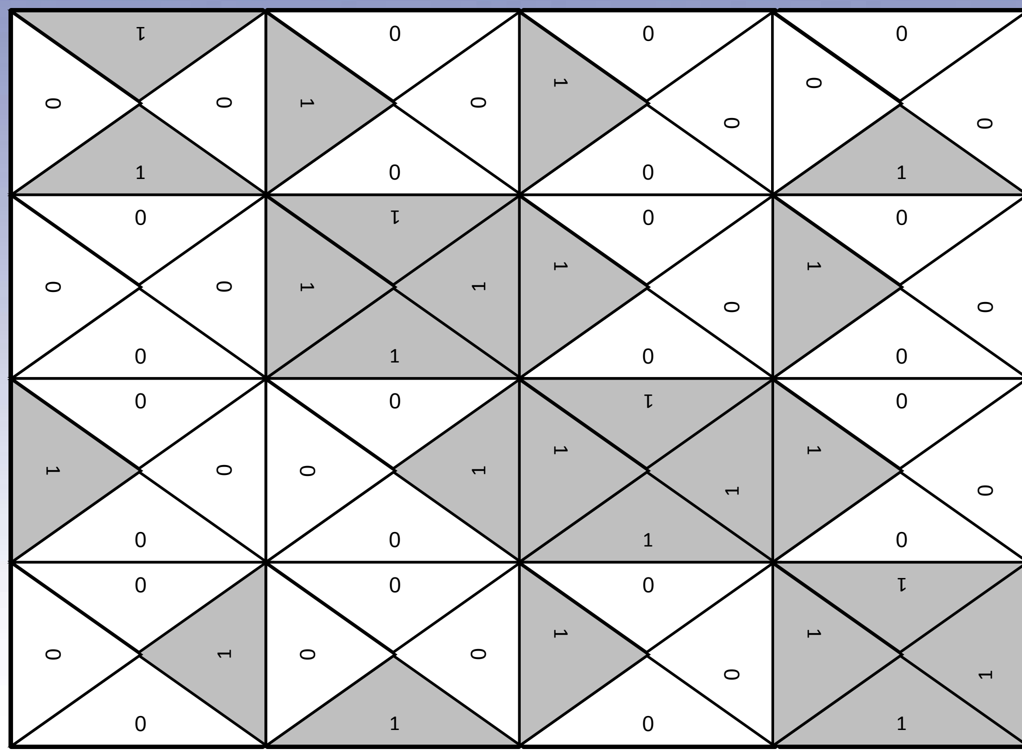
100% Non-zero Diagonal



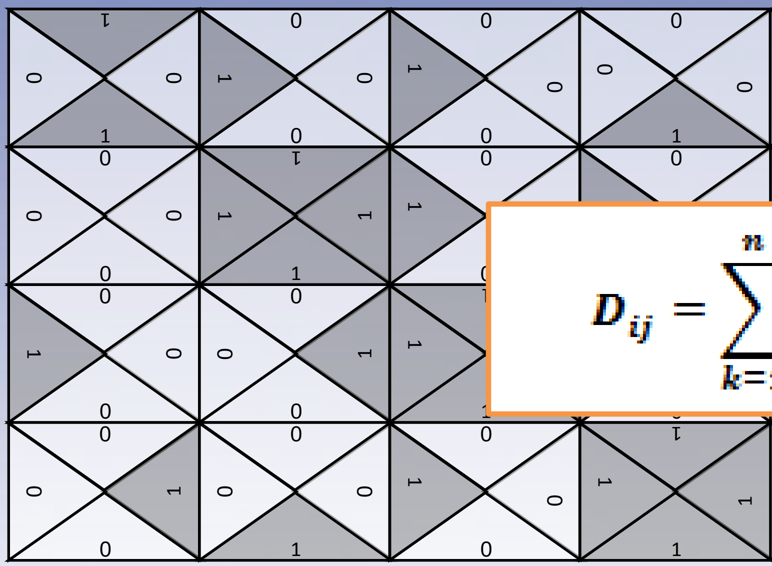
DATA ANALYSIS

25 Non-zero Blocks

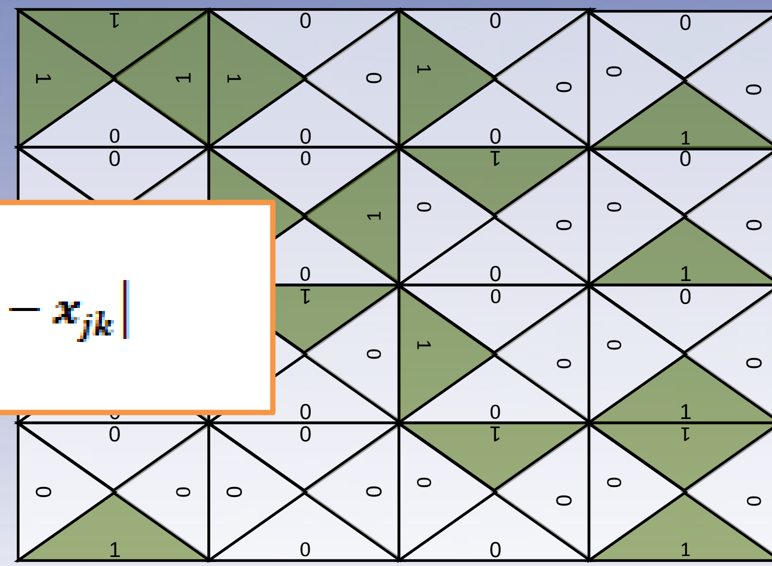
100% Non-zero Diagonal



DATA ANALYSIS



$$D_{ij} = \sum_{k=1}^{n} |x_{ik} - x_{jk}|$$



100% Non-zero Diagonal

25 Non-zero Blocks

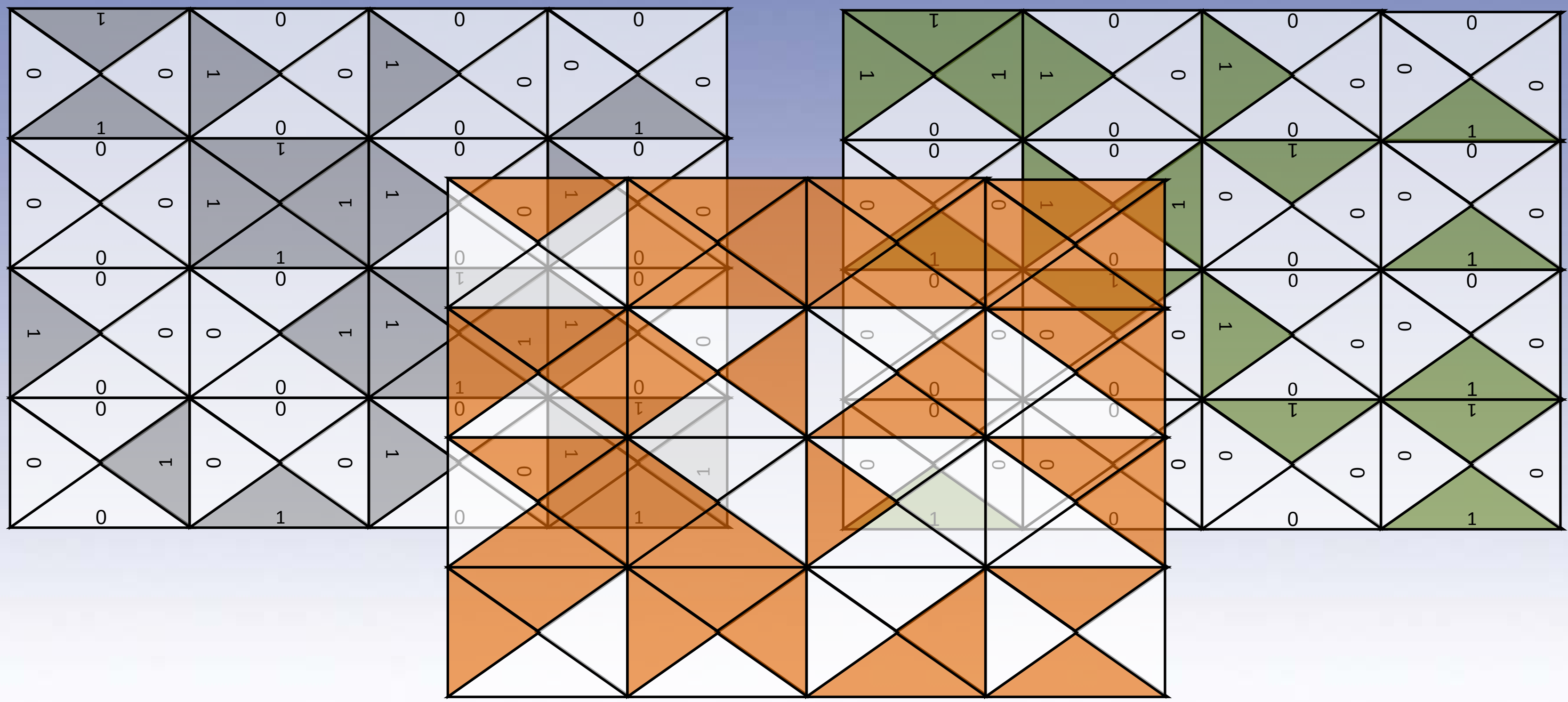
50% Non-zero Diagonal

18 Non-zero Blocks

= 25

Diagonal distance = 50%

DATA ANALYSIS



- Number of Similar Entries = 39
- Number of Different Entries = 25

DATA ANALYSIS

$$Similarity_{AB} = \frac{|A \cap B|}{|A \cup B|}$$

$$Mat.similarity(Fitness) = \frac{number\ of\ ones\ in\ the\ masked\ matrix}{64 + 64 - \ number\ of\ ones\ in\ the\ masked\ matrix}$$

Matrix	No. Similar Entries (intersection)	No. of Different Entries	Union	Matrix Similarity (Fitness)	City Block (Manhattan) distance	Diagonal Density Distance
Mat1	39	25	89	0.4382	25	50%
Mat2	44	20	84	0.5238	20	60%
				1 - Fitness	$\frac{\text{Distance}}{\text{Max Distance (64)}}$	

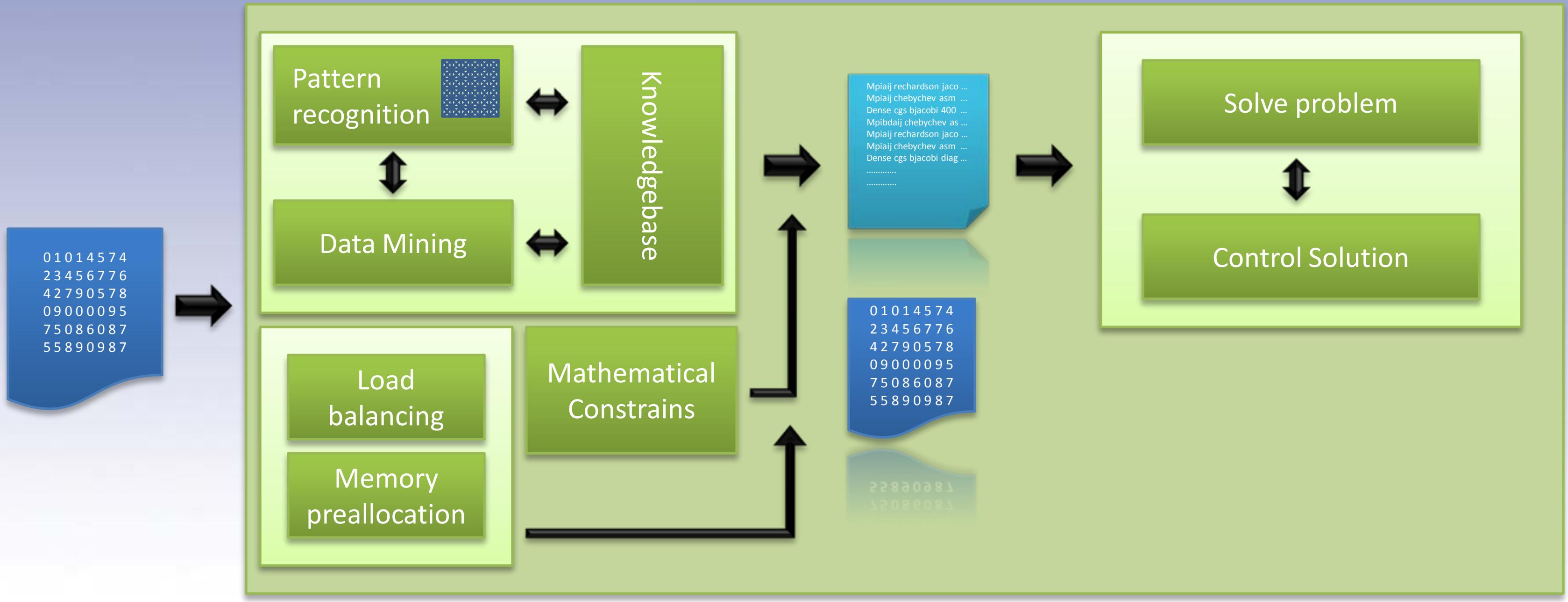
DATA ANALYSIS

$$Similarity_{AB} = \frac{|A \cap B|}{|A \cup B|}$$

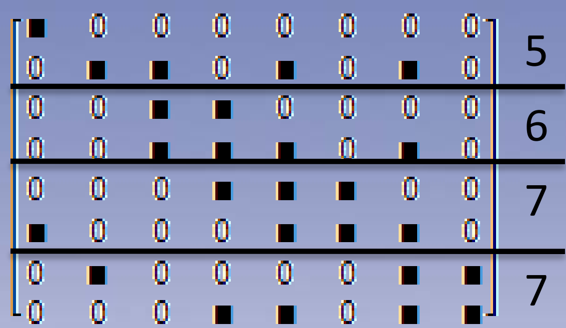
$$Mat.similarity(Fitness) = \frac{number\ of\ ones\ in\ the\ masked\ matrix}{64 + 64 - \ number\ of\ ones\ in\ the\ masked\ matrix}$$

Matrix	No. Similar Entries (intersection)	No. of Different Entries	Union	Matrix Similarity (Fitness) N.	City Block (Manhattan) distance N.	Diagonal Density Distance
Mat1	1.4524			0.5618	0.3906	0.5
Mat2	1.3887			0.4762	0.3125	0.6

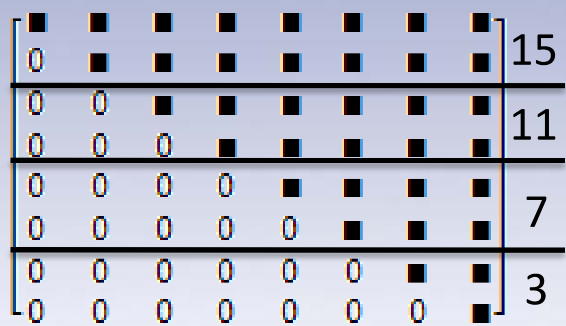
PERFORMANCE METHODOLOGY



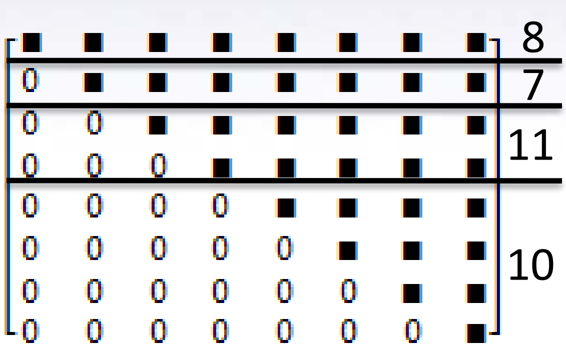
LOAD BALANCING



Uniform Matrix Distribution using PETSC_DECIDE



Non uniform Matrix Distribution using PETSC_DECIDE



Non uniform Matrix Distribution using Load Balancing Module

LOAD BALANCING (II)

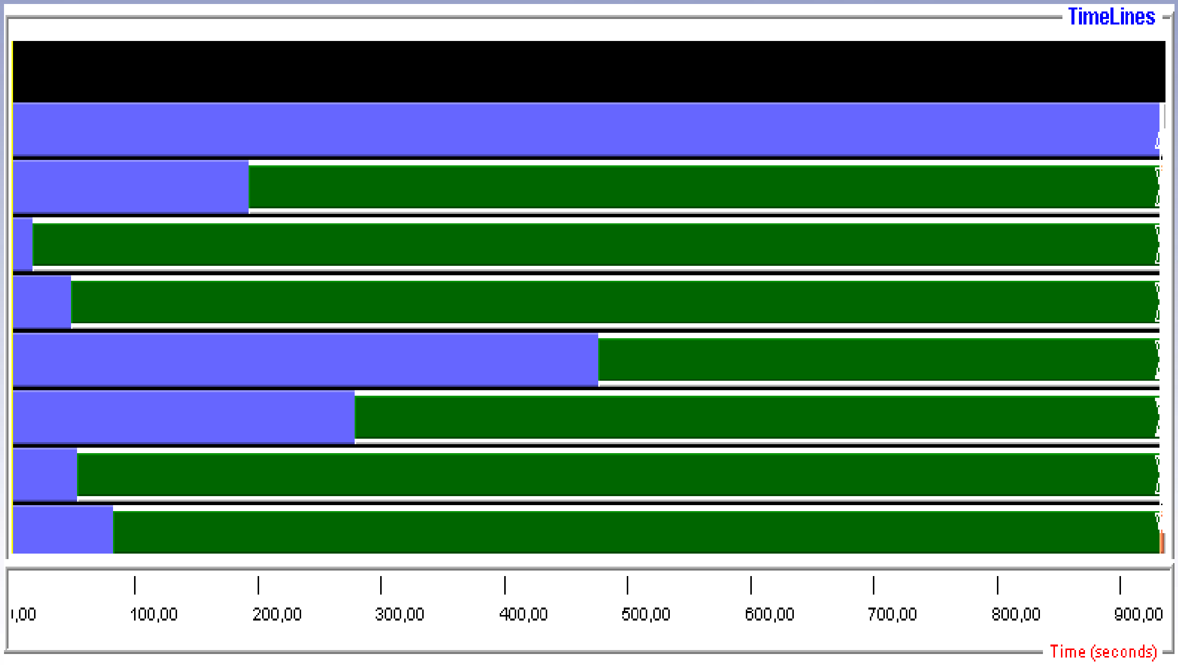
```
Mat A;
Int column[3],i;
double value[3];
…
MatCreate (PETSC_COMM_WORLD,&A);
MatSetSizes (A, PETSC_DECIDE, PETSC_DECIDE, N, N);
MatSetFromOption(A);

value[0] = -1.0; value[1] = 2.0; value[2] = -1.0;
if (rank == 0) {
	for (i=1; i<N-2; i++) {
		column[0] = i-1; column[1] = i; column[2] = i+1;
		MatSetValues(A,1,&i,3,column,value,INSERT_VALUES);
	}
}
//   Must set also the 0 and N-1 values

MatAssemblyBegin(A,MAT_FINAL_ASSEMBLY);
MatAssemblyEnd(A,MAT_FINAL_ASSEMBLY);
```

Matrix global index

PETSc determine the matrix cross-processor distribution

LOAD BALANCING (III)

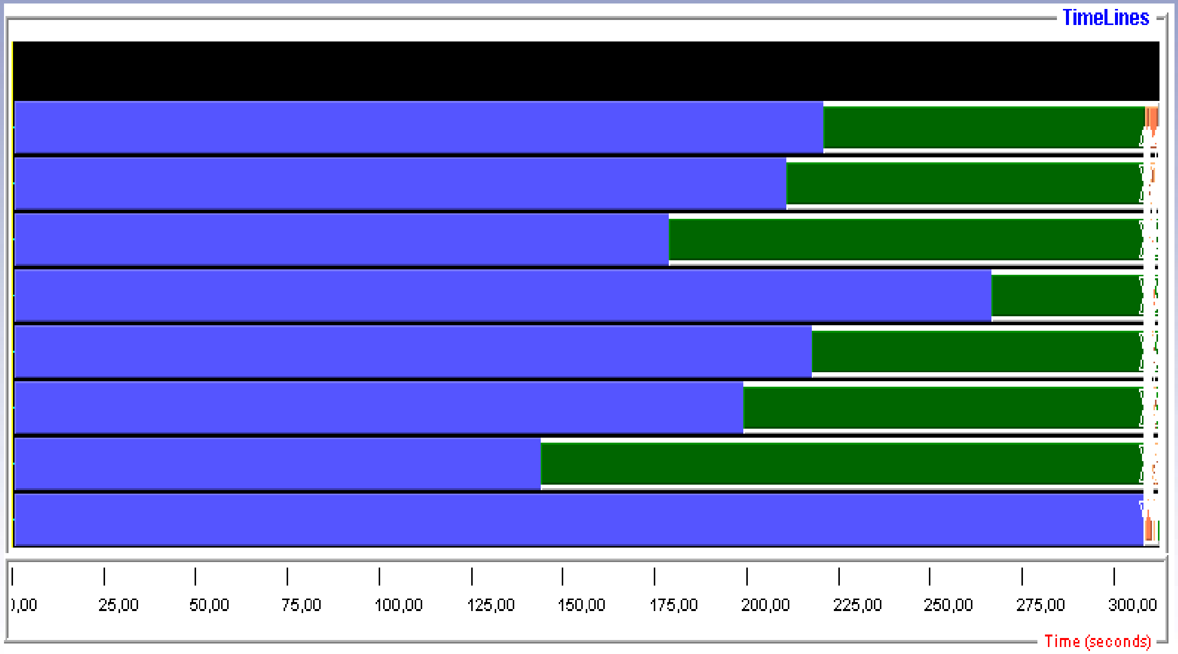
Line-based distribution for a 19242x 19242 matrix (PETSC_DECIDE)



Proc.	Number of Lines	Values
0	2406	1356012
1	2406	608378
2	2406	121242
3	2406	232133
4	2406	918270
5	2406	826023
6	2406	314930
7	2406	294349
Total	19242	4671337

LOAD BALANCING (III)

Value-based distribution for a 19242x 19242 matrix

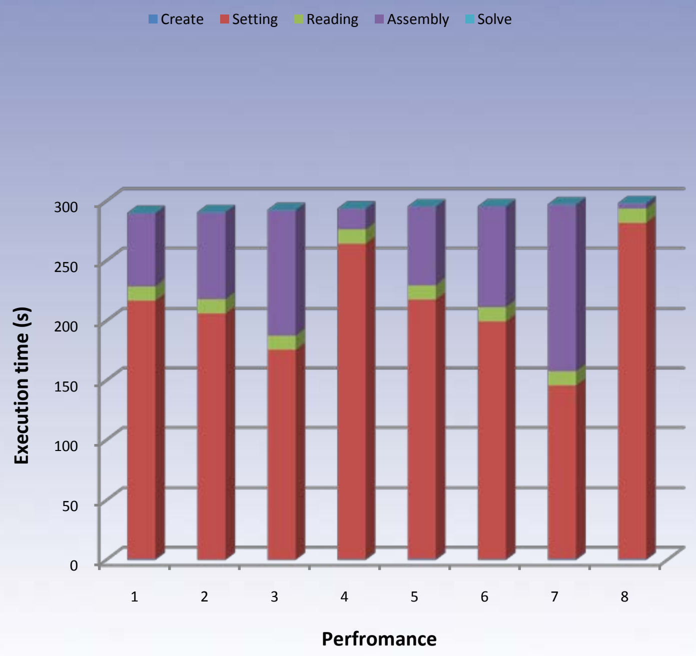


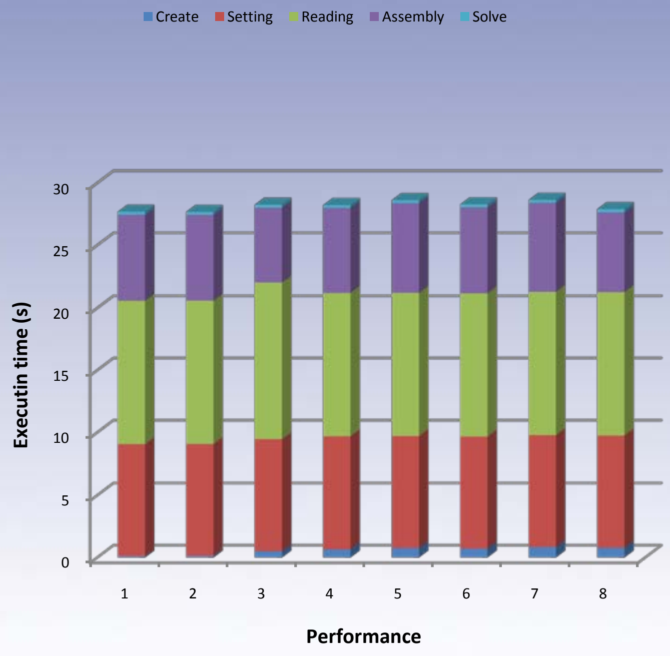
Proc.	Number of Lines	Values
0	915	584199
1	1084	583943
2	1421	583936
3	6410	584083
4	1568	584209
5	1250	584252
6	1912	584061
7	4682	584654
Total	19242	4671337

MEMORY PRE-ALLOCATION

- After specifying the data structure properties and the way it will be distributed, it is needed to allocate the memory.
  - Automatic: each SET_VALUE call PETSc allocates the memory that corresponds to this value. (nonzero values malloc)
  - Each processor allocate a memory of the size of its corresponding rows. (1 malloc)
  - Allocate the memory for the local sub matrix depending on the most dense row. (1 malloc)
  - Calculate the number of nonzero values per row. And allocate the memory for the row. (No. rows malloc)
- Each local processor loads its own data from the file

MEMORY PRE-ALLOCATION





SUMMARY

- We have designed a methodology based on input data that can automatically choose between existing solving parameters to improve PETSc application's performance
- Our methodology follows data mining techniques and is based on the comparison of the input matrix with a set of predefined patterns that are stored in a knowledgebase
- The load balancing method automatically hands data out to all processes based on the non zero-value distribution across the matrix
- Choosing the suitable memory pre-allocation method improves memory utilization and reduces the cost of memory allocation

WHAT WE HAVE… WOULD LIKE

- Components for analyzing the input matrix
- Database with hundreds of executions
- Components for load balancing and memory preallocation
- We can easily produce a wrapper for PETSc calls

- A more detailed characterization of the hardware (cache, memory, FLOPS, #cores?)