Performance Strategies for Parallel Mathematical Libraries Based on Historical Knowledgebase

CScADS workshop 2009

Eduardo Cesar, Anna Morajko, Ihab Salawdeh
Universitat Autònoma de Barcelona
• Objective
• Mathematical Library (PETSc)
• Performance Methodology
  – Knowledgebase
  – Data Analysis
  – Load balancing
  – Memory pre-allocation
• Summary
• What we have ... would like
Design a methodology, based on input data, that can automatically choose between existing solving parameters to improve PETSc application’s performance.
PETSc

Level of abstraction

Data Structure
- Matrices
- Vectors
- Index set

Mathematical Algorithms
- KSP
- PC

Other higher Level solvers
- PDE Solvers
- SNES
- SLES
- TS (Time Stepping)

Application Code

BLAS
LAPACK
MPI
- 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
Mathematical Algorithms

KSP

PC

Data Structure

Matrices

Vectors

Index set

Level of abstraction

Other higher Level solvers

PDE Solvers

SNES

TS

Time Stepping

SLES

Draw

Application Code

BLAS

LAPACK

MPI
• 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

Pattern recognition
Data Mining
Load balancing
Memory preallocation

Knowledgebase
Mathematical Constrains

Solve problem
Control Solution

Load balancing
Memory preallocation

Mathematical Constrains
Knowledgebase

Pattern recognition
Data Mining
**PETSC SOLVING PROCESS**

- **Application Initialization**
- **Initialize $A$ and $b$**
- **Post-Processing**

**Main Routine**

- **PETSc**
  - **Solve $Ax = b$**

**Linear Solvers (KSP)**

**PC**

**User code**

**PETSc code**
**PETSC PERFORMANCE METHODOLOGY**

- **Application Initialization**
  - Initialize $A$ and $b$

- **Main Routine**
  - Solve $Ax = b$
  - Pattern
  - Data
  - Knowledgebase
  - Solver Parameters Decision
  - PC

- **PETSc**
  - User code
  - PETSc code

- **Post-Processing**
  - User code
  - PETSc code
  - Tuning Code
• 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
<table>
<thead>
<tr>
<th>Input Type</th>
<th>Input Size</th>
<th>No. Processors</th>
<th>Memory Representation</th>
<th>KSP</th>
<th>PC</th>
<th>Total Time (s)</th>
<th>Matrix Pattern</th>
</tr>
</thead>
<tbody>
<tr>
<td>Around-Diagonal</td>
<td>9152X9152</td>
<td>8</td>
<td>mpiaij</td>
<td>bcgs</td>
<td>asm</td>
<td>103,648252</td>
<td></td>
</tr>
<tr>
<td>Around-Diagonal</td>
<td>9152X9152</td>
<td>8</td>
<td>mpiaij</td>
<td>bcgs</td>
<td>bjacobi</td>
<td>215,159341</td>
<td></td>
</tr>
<tr>
<td>Around-Diagonal</td>
<td>9152X9152</td>
<td>8</td>
<td>mpiaij</td>
<td>bcgs</td>
<td>jacobi</td>
<td>35,788464</td>
<td></td>
</tr>
<tr>
<td>Around-Diagonal</td>
<td>9152X9152</td>
<td>8</td>
<td>mpiaij</td>
<td>bcgsl</td>
<td>asm</td>
<td>222,055694</td>
<td></td>
</tr>
<tr>
<td>Around-Diagonal</td>
<td>9152X9152</td>
<td>8</td>
<td>mpiaij</td>
<td>bcgsl</td>
<td>bjacobi</td>
<td>189,236794</td>
<td></td>
</tr>
<tr>
<td>Around-Diagonal</td>
<td>9152X9152</td>
<td>8</td>
<td>mpiaij</td>
<td>bcgs</td>
<td>asm</td>
<td>103,648252</td>
<td></td>
</tr>
<tr>
<td>Distributed</td>
<td>19242X19242</td>
<td>16</td>
<td>mpiaij</td>
<td>bcgs</td>
<td>asm</td>
<td>522,155023</td>
<td></td>
</tr>
<tr>
<td>Distributed</td>
<td>19242X19242</td>
<td>16</td>
<td>mpiaij</td>
<td>bcgs</td>
<td>bjacobi</td>
<td>455,43445</td>
<td></td>
</tr>
<tr>
<td>Distributed</td>
<td>19242X19242</td>
<td>16</td>
<td>mpiaij</td>
<td>bcgs</td>
<td>jacobi</td>
<td>459,629514</td>
<td></td>
</tr>
<tr>
<td>Distributed</td>
<td>19242X19242</td>
<td>16</td>
<td>mpiaij</td>
<td>bcgsl</td>
<td>asm</td>
<td>457,210476</td>
<td></td>
</tr>
<tr>
<td>Distributed</td>
<td>19242X19242</td>
<td>16</td>
<td>mpiaij</td>
<td>bcgsl</td>
<td>bjacobi</td>
<td>456,518158</td>
<td></td>
</tr>
<tr>
<td>Distributed</td>
<td>19242X19242</td>
<td>16</td>
<td>mpiaij</td>
<td>bcgsl</td>
<td>jacobi</td>
<td>455,395112</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
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

= 25

50% Non-zero Diagonal
18 Non-zero Blocks

Diagonal distance = 50%
### Data Analysis

**Matrix Similarity (Fitness)**

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

**Similarity**

\[
\text{Similarity}_{AB} = \frac{|A \cap B|}{|A \cup B|}
\]

<table>
<thead>
<tr>
<th>Matrix</th>
<th>No. Similar Entries (intersection)</th>
<th>No. of Different Entries</th>
<th>Union</th>
<th>Matrix Similarity (Fitness)</th>
<th>City Block (Manhattan) distance</th>
<th>Diagonal Density Distance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mat1</td>
<td>39</td>
<td>25</td>
<td>89</td>
<td>0.4382</td>
<td>25</td>
<td>50%</td>
</tr>
<tr>
<td>Mat2</td>
<td>44</td>
<td>20</td>
<td>84</td>
<td>0.5238</td>
<td>20</td>
<td>60%</td>
</tr>
</tbody>
</table>

1 - Fitness \(\frac{\text{Distance}}{\text{Max Distance (64)}}\)
### Data Analysis

**Similarity**

\[
\text{Similarity}_{AB} = \frac{|A \cap B|}{|A \cup B|}
\]

**Matrix similarity (Fitness)**

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

<table>
<thead>
<tr>
<th>Matrix</th>
<th>No. Similar Entries (intersection)</th>
<th>No. of Different Entries</th>
<th>Union</th>
<th>Matrix Similarity (Fitness) N.</th>
<th>City Block (Manhattan) distance N.</th>
<th>Diagonal Density Distance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mat1</td>
<td></td>
<td>1.4524</td>
<td></td>
<td>0.5618</td>
<td>0.3906</td>
<td>0.5</td>
</tr>
<tr>
<td>Mat2</td>
<td></td>
<td>1.3887</td>
<td></td>
<td>0.4762</td>
<td>0.3125</td>
<td>0.6</td>
</tr>
</tbody>
</table>
PERFORMANCE METHODOLOGY

Pattern recognition
Data Mining
Load balancing
Memory preallocation

Knowledgebase
Mathematical Constrains

Solve problem
Control Solution

Load balancing
Memory preallocation
LOAD BALANCING

Uniform Matrix Distribution using PETSC_DECIDE

Non uniform Matrix Distribution using PETSC_DECIDE

Non uniform Matrix Distribution using Load Balancing Module
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);
Load Balancing (III)

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

<table>
<thead>
<tr>
<th>Proc</th>
<th>Number of Lines</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2406</td>
<td>1356012</td>
</tr>
<tr>
<td>1</td>
<td>2406</td>
<td>608378</td>
</tr>
<tr>
<td>2</td>
<td>2406</td>
<td>121242</td>
</tr>
<tr>
<td>3</td>
<td>2406</td>
<td>232133</td>
</tr>
<tr>
<td>4</td>
<td>2406</td>
<td>918270</td>
</tr>
<tr>
<td>5</td>
<td>2406</td>
<td>826023</td>
</tr>
<tr>
<td>6</td>
<td>2406</td>
<td>314930</td>
</tr>
<tr>
<td>7</td>
<td>2406</td>
<td>294349</td>
</tr>
<tr>
<td>Total</td>
<td>19242</td>
<td>4671337</td>
</tr>
</tbody>
</table>
Value-based distribution for a 19242x 19242 matrix

<table>
<thead>
<tr>
<th>Proc</th>
<th>Number of Lines</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>915</td>
<td>584199</td>
</tr>
<tr>
<td>1</td>
<td>1084</td>
<td>583943</td>
</tr>
<tr>
<td>2</td>
<td>1421</td>
<td>583936</td>
</tr>
<tr>
<td>3</td>
<td>6410</td>
<td>584083</td>
</tr>
<tr>
<td>4</td>
<td>1568</td>
<td>584209</td>
</tr>
<tr>
<td>5</td>
<td>1250</td>
<td>584252</td>
</tr>
<tr>
<td>6</td>
<td>1912</td>
<td>584061</td>
</tr>
<tr>
<td>7</td>
<td>4682</td>
<td>584654</td>
</tr>
<tr>
<td>Total</td>
<td>19242</td>
<td>4671337</td>
</tr>
</tbody>
</table>
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

Performance

Execution time (s)

- Create
- Setting
- Reading
- Assembly
- Solve
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.
• 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?)