Improving High Performance Sparse Libraries using Compiler-Assisted Specialization

A PETSc Case Study

By Shreyas Ramalingam
Outline

- Motivation
- PETSc and Specializations
- Specialization Approach
- Result
- Summary
Motivation

Performance depends on execution context

Specializing code to execution context can improve performance significantly.
Motivation

Specialization is not a new concept. Why can’t a “joe” programmer do this by himself?
Specialization Dilemma

I heard specialization can improve performance significantly.. Neat 😊 !!

Yeah, but you need to somehow figure out all those frequent use cases.

Optimistic Joe  Pessimistic Joe
Specialization Dilemma

That shouldn’t be that hard!! I just need to profile the code and instrument it a bit.

Hmm, what if you break something when you modify or insert code.

Optimistic Joe  Pessimistic Joe
Specialization Dilemma

Hmm, so I will study the application and maybe the library and then maybe I can figure it out.

Wait, did you not listen to the first slide?! Performance is a complicated relationship between data, application, algorithm, data structures, compiler, architecture …
Specialization Dilemma

Angry Joe

Arghhhh !!!

Pessimistic Joe

I told you to stick to being an average Joe
Specialization Dilemma

Hey, I got all the experience!! All I need is access to your code and execution environment 😊
Problem

Amused Joe

Optimistic Stephanie

Compiler Based

Transformation Recipes
Libraries
Experience

Execution Environment
Specialized Library
Problem

Execution Environment → Specialized Library

Transformation Recipes → Libraries → Experience

Compiler Based

Optimistic Joe

Optimistic Stephanie
Outline

- Motivation
- PETSc and Specializations
- Approach
- Result
- Conclusion
Specializing PETSc

What is it?

• Suite of matrix representations and routines.
• Higher level linear and non linear solvers that call lower level BLAS where applicable.

Why is it Important?

• It is used by more than 200 applications.
• Library already supports specialization.
• Actively developed by Argonne National Labs
Brief Overview of Sparse Linear Algebra

Matrix A

Row 1: 22 30
Row 2: 72
Row 3: 85
Row 4

PETSc Name - AIJ

Compressed Sparse Row Representation
Brief Overview of Sparse Linear Algebra

Blocked Sparse Row Representation
PETSc Name - BAIJ

Each square is a block of \(mxm\) elements

- Blocked Row 1
  - B1
  - B2

- Blocked Row 2
  - B1
  - B2

- Blocked Row 3
  - B2
  - B3

- Blocked Row 4
  - B3
  - B4

Layout in memory
# Specializations for Block Sizes in PETSc

<table>
<thead>
<tr>
<th>Matrix Type</th>
<th>Description</th>
<th>No of Specialized Functions</th>
<th>Total No of Manually Written Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>AIJ</td>
<td>Compressed Sparse Row</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>BAIJ</td>
<td>Blocked Sparse Row</td>
<td>15</td>
<td>115</td>
</tr>
<tr>
<td>MAIJ</td>
<td>Matrix used for restriction and interpolation for multi component problems</td>
<td>4</td>
<td>52</td>
</tr>
<tr>
<td>SBAIJ</td>
<td>Symmetric Blocked Sparse Row</td>
<td>10</td>
<td>75</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td></td>
<td><strong>29</strong></td>
<td><strong>242</strong></td>
</tr>
</tbody>
</table>
Optimizing Sparse Computations
Challenges for “Stephanie”

- Dynamic Loop Bounds – Non Zeros per row
- Small and odd Loop Bounds
- Indirect Indexing Expressions
- Limited Cache Optimizations
Optimizing Sparse Computations

What “stephanie” must focus on

- Improving Instruction Scheduling
- Increasing Instruction Level Parallelism
- Improve register usage and reuse.
- Reduce Cache misses
- Generate SIMD instructions
Specialization Approach

Library Specific

- Post Processing
- Source to Source Transformations (CHiLL Framework)

- Auto tuning

- Select Code Variants
- Profiling

- Optimized Code

Execution Context

- Input Data
- Target Architecture
- Target Compiler
- Application
Profiling Applications

- **PFLOTRAN**
  - Los Alamos
  - Subsurface simulation to model ground water to study the effects of geological sequestration and contaminants.

- **Uintah**
  - U of U
  - Framework to solve wide variety of simulations.
  - Created to model interactions between hydrocarbon fibers, structures and explosives

- **UNIC**
  - ANL
  - 3D unstructured neutronic code to obtain highly detailed description of the nuclear reactor core.
# Profiling – Step 1

## Identifying PETSc Functions

<table>
<thead>
<tr>
<th>Application</th>
<th>Matrix Type</th>
<th>PETSc Function</th>
<th>% Exec Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>PFLOTRAN</td>
<td>BAIJ</td>
<td>MatLUFactorNumeric_SeqBAIJ_N</td>
<td>10%</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MatSolve_SeqBAIJ_N</td>
<td>9.8%</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MatMult _SeqBAIJ N</td>
<td>9.8%</td>
</tr>
<tr>
<td>Uintah</td>
<td>AIJ</td>
<td>MatMult _SeqAIJ (PetscSolve)</td>
<td>23%</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MatMult _SeqAIJ (ApplyFilter)</td>
<td>3%</td>
</tr>
<tr>
<td>UNIC</td>
<td>SBAIJ</td>
<td>MatMult _SeqSBAIJ_1</td>
<td>39.7%</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MatRelax _SeqBAIJ</td>
<td>46.9%</td>
</tr>
</tbody>
</table>
Visualizing Banded Matrices

Profiling – Step 2
Frequency of Non Zeros in Input Data

PFLOTRAN - MatMult Matrix
- 60% 5%
- 35%
- 4
- 6
- 7

PFLOTRAN MatSolve Matrix
- 80% 19%
- 1%
- 1
- 2
- 3

Uintah Matrix
- 1% 13%
- 86%
- 5

UNIC Matrix
- 27 25%
- 25 9%
- 18 8%
- 28 21%
- 49 20%
- 32

Visualizing Banded Matrices
Optimizing Sparse Computations

Alternate Code Variants
- Matrix Representation
- Gather Operation

Higher Level Transformations
- Loop Unrolling
- Loop Splitting

Post Processing
- Prefetching
- Scalar Replacement
-Pragma Vector Always

Specialized using “known” Command in CHiLL
Optimizing Sparse Computations
Alternate Code Variants – Avoiding dynamic loop bounds

Visualizing Banded Matrices

Visualizing CSR for Banded Matrices

Visualizing ELL for Banded Matrices
Optimizing Sparse Computations
Alternate Code Variants – Avoiding indirection

Each square is a block of $mxm$ elements

Matrix A

Matrix Multiplication in Sparse Matrices

• Memory Level – Block Sparse Matrices.
• Register Level – Compressed Sparse Row Matrices
CHiLL
Framework for composing High Level Loop Transformation

• Source to source optimizer
• Exposes a programming interface (scripts) to the user
  – Unroll
  – Unroll and Jam
  – Loop splitting
  – Loop distribution and fusion
  – Datacopy
Source to Source Transformations

Uintah

Alternated code Variant: ELL Format + Register level gather

CHiLL Script

```
value_n = 7
known(n>value_n-1)
known(n<value_n+1)
original()
```

Register level gather

```
Matrix Multiplication for ELL format.
```

```cpp
for(t2=0;t2<m;t2++){
    //Load into temporary arrays
    //n is specialized for 7
    for(t4=0;t4<7;t4++){
        col[t4] = aj[t2*7+t4];
        colVal[t4] = x[col[t4]];
        y[t2] += aa[t2*7+t4]*colVal[t4];
    }
}
```
Source to Source Transformations
Example – Gather and Loop Splitting

value_n = 7
known(n > value_n - 1)
known(n < value_n + 1)
original()

// Distribute for better control
distribute([0,1,2],2)

// Unroll the gather statement
unroll(0,2,7)
unroll(1,2,7)

// Split the loop by powers of 2
split(2,2,L2<6)
split(2,2,L2<4)

for(t2 = 0; t2 < m; t2++){
  // Load into temporary arrays
  *col = aj[t2 * 7 + 0];
  col[0 + 1] = aj[t2 * 7 + 0 + 1];
  <!---- Rest of statements are hidden ---!>
  *colVal = x[*col];
  colVal[0 + 1] = x[col[0 + 1]];
  <!---- Rest of statements are hidden ---!>
  for (t4 = 0; t4 <= 3; t4++)
    y[t2] = y[t2] + aa[t2 * 7 + t4] * colVal[t4];
  for (t4 = 4; t4 <= 5; t4++)
    y[t2] = y[t2] + aa[t2 * 7 + t4] * colVal[t4];
}
Post Processing
Example – Prefetching and vectorizing

for(t2=0; t2<m; t2++){

    // Insert Prefetching
    _mm_prefetch((char*)&aa[(t2*7)+40], _MM_HINT_T0);
    _mm_prefetch((char*)&aj[(t2*7)+80], _MM_HINT_T0);

    <! – Register Level Gather statements --! >

    #pragma vector always
    for (t4 = 0; t4 <= 3; t4++)
        y[t2] = y[t2] + aa[t2 * 7 + t4] * colVal[t4];

    #pragma vector always
    for (t4 = 4; t4 <= 5; t4++)
        y[t2] = y[t2] + aa[t2 * 7 + t4] * colVal[t4];
}

Example of Specialized Function

```c
for (i=0; i<mbs; i++) {
//Initializations
switch (n) {
    case 27:
        //specialized code for n=27
        *z_temp = z[i];
        for (t4 = 1; t4 <= 22; t4 += 3) {
            col[t4] = ib[t4];
            col[t4 + 1] = ib[t4 + 1];
            col[t4 + 2] = ib[t4 + 2];
            colVal[t4] = x[col[t4]];
            colVal[t4 + 1] = x[col[t4 + 1]];
            colVal[t4 + 2] = x[col[t4 + 2]];
            z[col[t4]] += v[t4] * x1;
            z[col[t4 + 1]] += v[t4 + 1] * x1;
            z[col[t4 + 2]] += v[t4 + 2] * x1;
            *z_temp += v[t4] * colVal[t4];
            *z_temp += v[t4+1] * colVal[t4+1];
            *z_temp += v[t4+2] * colVal[t4+2];
        }
        break;
    }
    default:
        for (j=jmin; j<n; j++) {
            cval = *ib;
            z[cval] += *v * x1;
            z[i] += *v++ * x[*ib++];
        }
        break;
    }
}
```
Outline

- Motivation
- PETSc and Specializations
- Specialization Approach
- Result
- Summary
PFLOTRAN on Hopper XE6 System

PFLOTRAN - PGI Compiler
Speedup over BLAS

PFLOTRAN - Intel
Speedup Over BLAS

- Hand Tuned
- Original + Unroll
- Original + Unroll + Scalar Rep.
- Original + Unroll + Scalar Rep. + Prefetch
- Gather + Unroll
- Gather + Unroll + Scalar Rep
- Gather + Unroll + Scalar Rep. + Prefetch
Uintah on Hopper XE6 System

Uintah
Speedup over Original Code

MatMult_SeqAIJ

- PGI – Unrolling
- PGI – Unrolling + Prefetching
- Intel – Unrolling
- Intel – Loop Splitting
- Intel – Loop Splitting + Prefetching
UNIC on Hopper XE6 System

UNIC - PGI
Speedup over Original Code

UNIC - Intel
Speedup over Original Code
Application Speedups

PFLOTRAN
120x240x40
Speedup

Uintah - Methane Fire
Container 140x140x140
Speedup

UNIC
Speedup

Specialized Library – PGI
Specialized Library – Intel
Specialized Library – Intel
Outline

Motivation

PETSc and Specializations

Specialization Approach

Result

Summary
## Degree of Automation

<table>
<thead>
<tr>
<th>Stage</th>
<th>State of Automation</th>
<th>Future Work</th>
</tr>
</thead>
<tbody>
<tr>
<td>Profiling – Finding PETSc Functions</td>
<td>Manual - Used HPCToolkit</td>
<td>Integrate with TAU</td>
</tr>
<tr>
<td>Profiling – Identifying the non zeros</td>
<td>Manual – Used HPCToolkit</td>
<td>Integrate with TAU</td>
</tr>
<tr>
<td>Selecting code variants</td>
<td>Manual</td>
<td>Auto tuning framework – Active Harmony</td>
</tr>
<tr>
<td>Optimizing code</td>
<td>Automated</td>
<td>Automated</td>
</tr>
<tr>
<td>Post Processing</td>
<td>Manual</td>
<td>CHiLL</td>
</tr>
<tr>
<td>Finding optimal solution</td>
<td>Exhaustive search using Scripts</td>
<td>Auto tuning framework – Active Harmony</td>
</tr>
</tbody>
</table>
Summary
Specializing to Execution Context

• Approach to specialize sparse linear algebra libraries.
• Application Context
  • Uintah and PFLOTRAN are specialized for all problem sizes.
  • UNIC is specialized only for the single problem size.
• Compiler Context
  • Different Strategies are required for different compilers
• Architecture Context
  • Optimization parameters such as unroll factors were auto tuned the hopper XE6 architecture
• Performance
  • Performance improvement from 1.12X to 1.87X on individual functions.
  • Overall application performance from 1.1X to 1.25X.
Related Work

Optimizing Sparse Matrices
• Herrero, J. R., and Navarro, J. J. Improving performance of hypermatrix Cholesky factorization.

Auto tuning and specialization
• Shin, J., Hall, M. W., Chame, J., Chen, C., Fischer, P. F., and Hovland, P. D. Speeding up nek5000 with autotuning and specialization.