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

THE UNIVERSITY OF UTAH

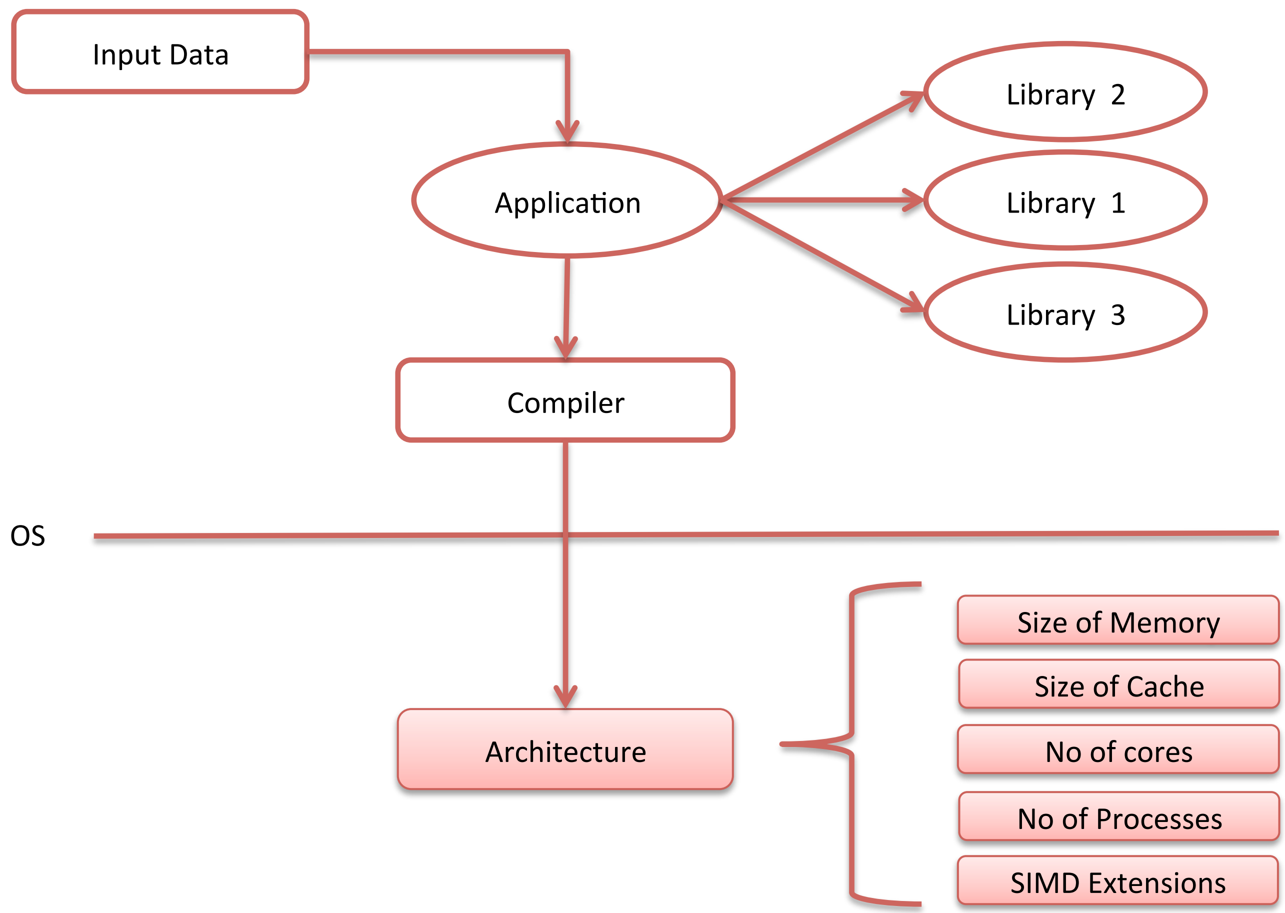
Outline

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

THE UNIVERSITY OF UTAH

Motivation

Performance depends on execution context

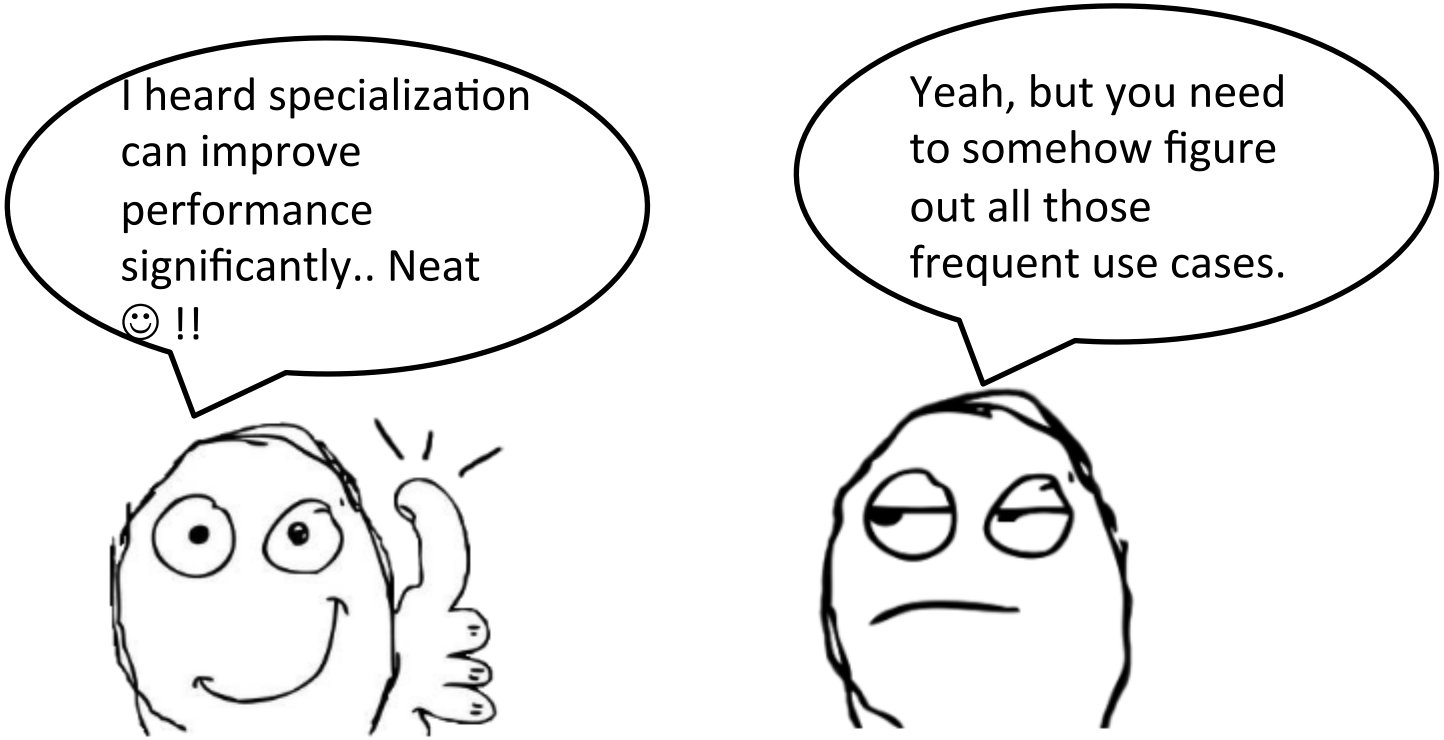


Specializing code to execution context can improve performance significantly.

Motivation

Specialization is not a new concept. Why can't a "joe" programmer do this by himself ?

Specialization Dilemma

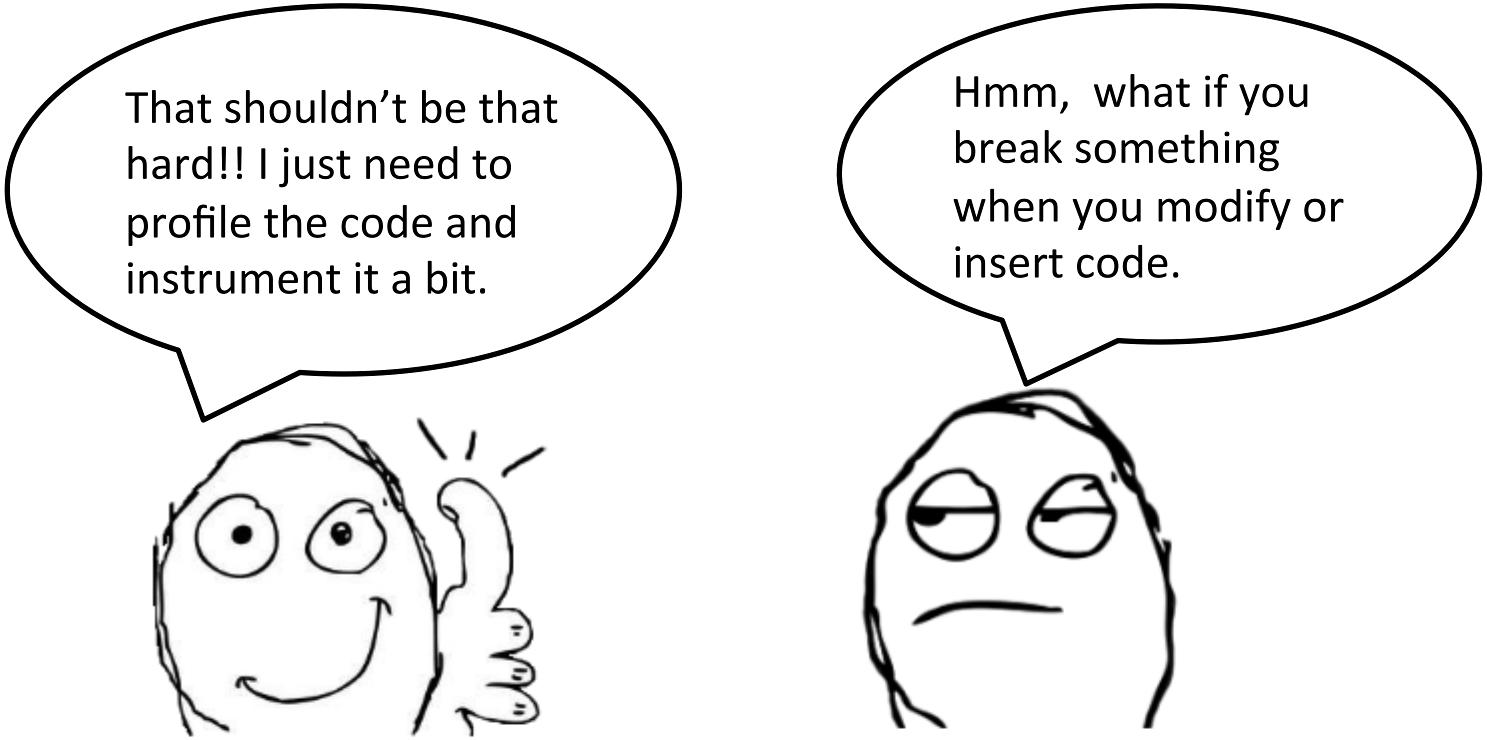


Optimistic Joe

Pessimistic Joe

THE UNIVERSITY OF UTAH

Specialization Dilemma

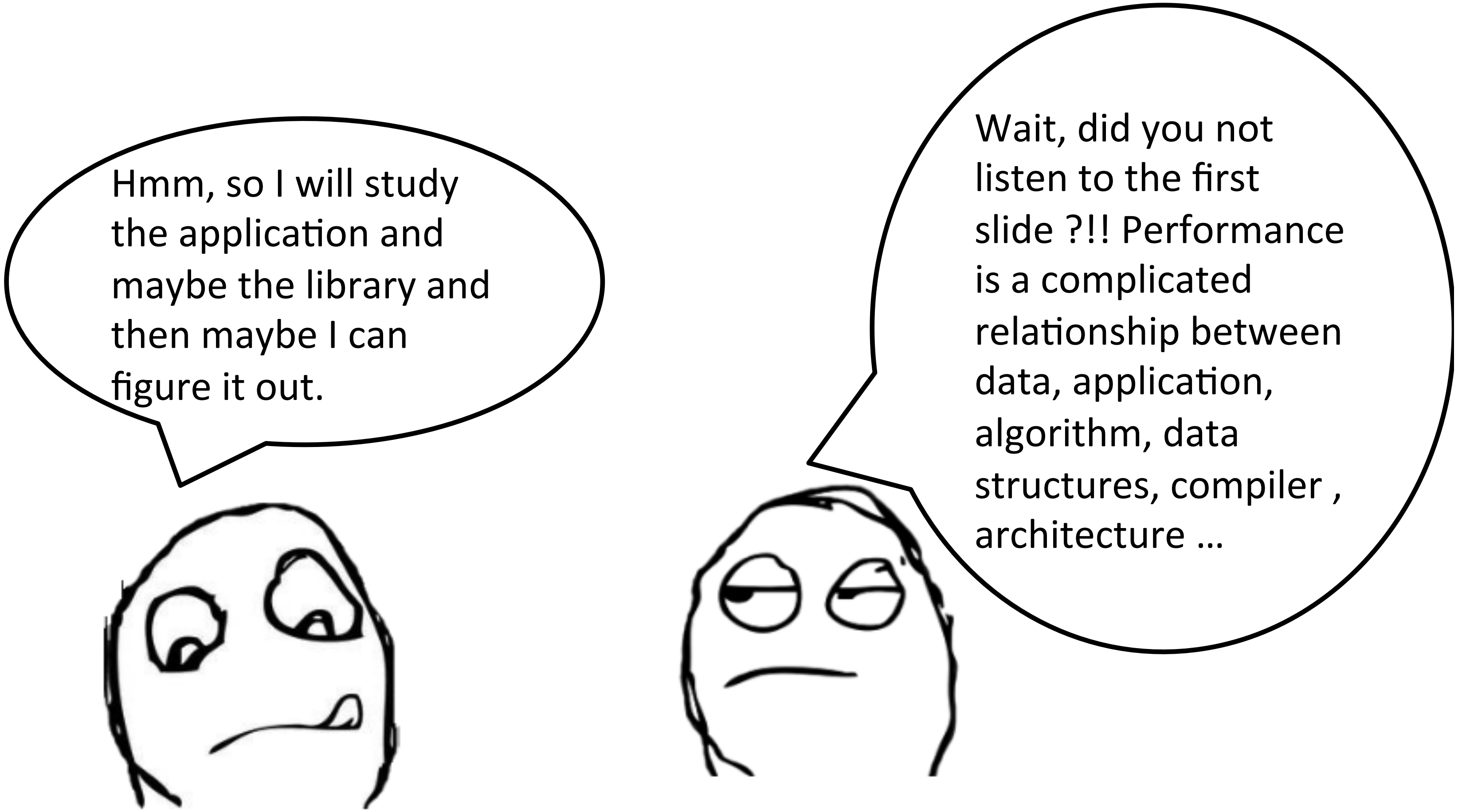


Optimistic Joe

Pessimistic Joe

THE UNIVERSITY OF UTAH

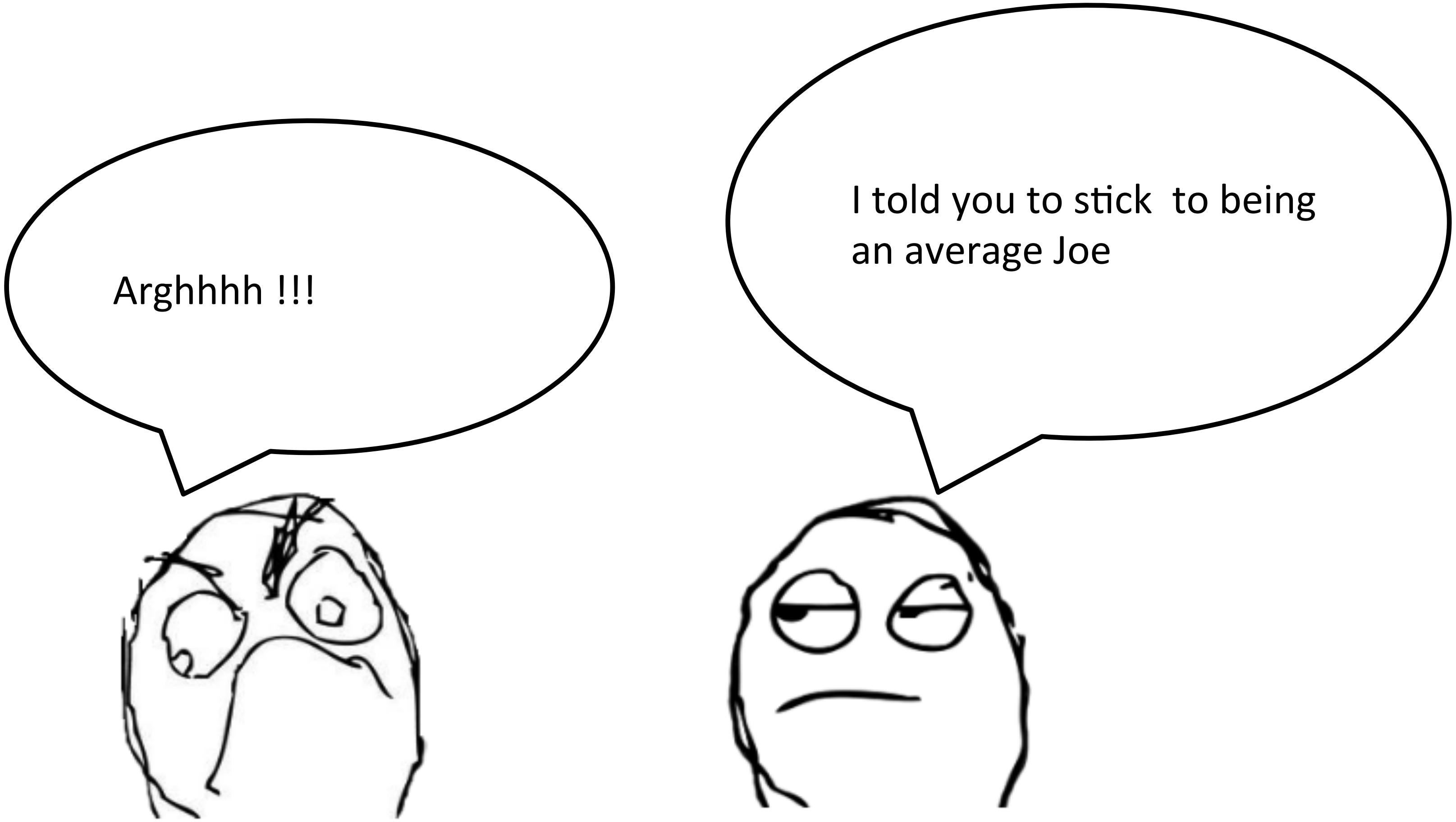
Specialization Dilemma



Self Doubting Joe

Pessimistic Joe

Specialization Dilemma



Angry Joe

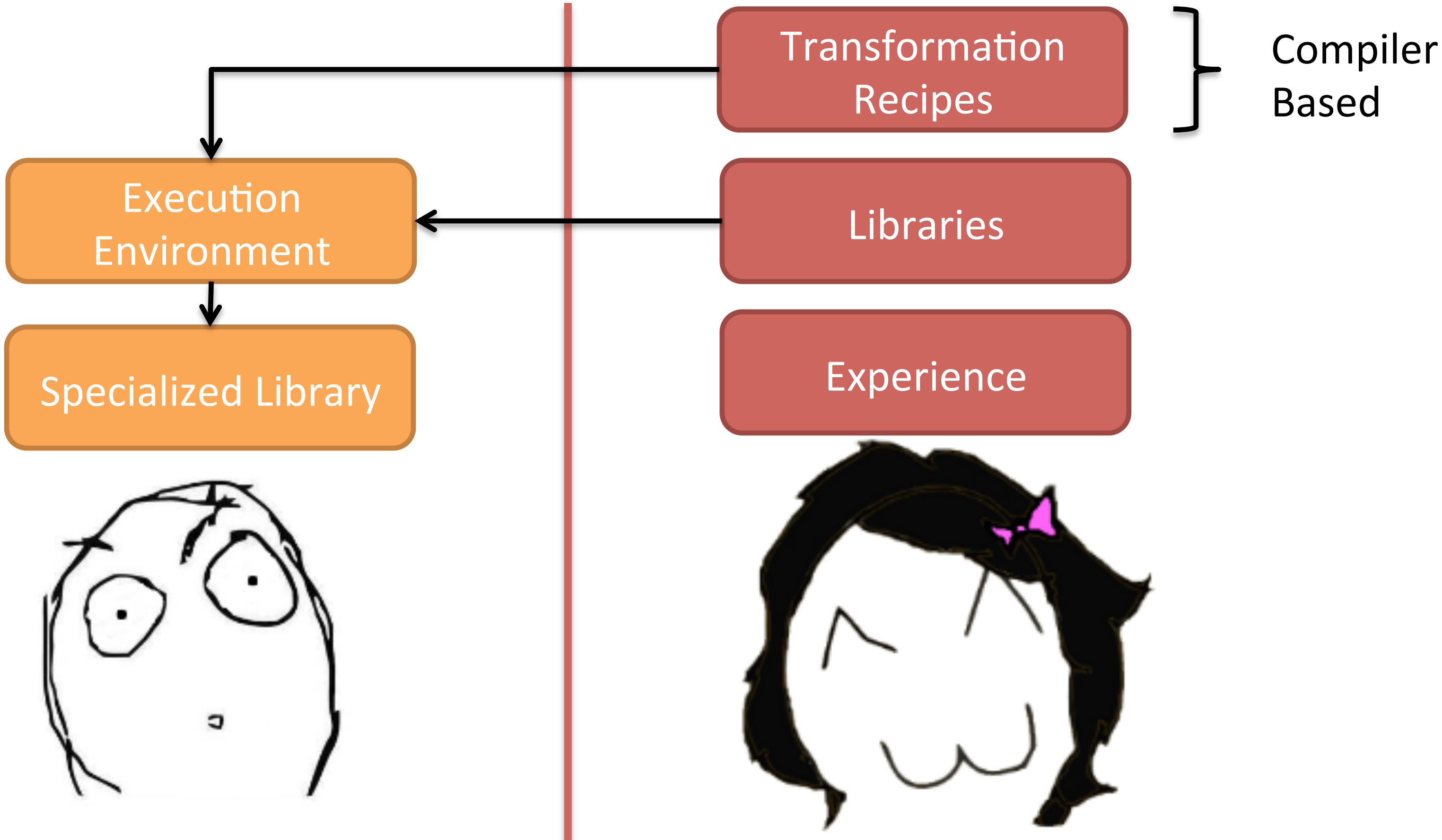
Pessimistic Joe

Specialization Dilemma

Amused Joe

Optimistic Stephanie

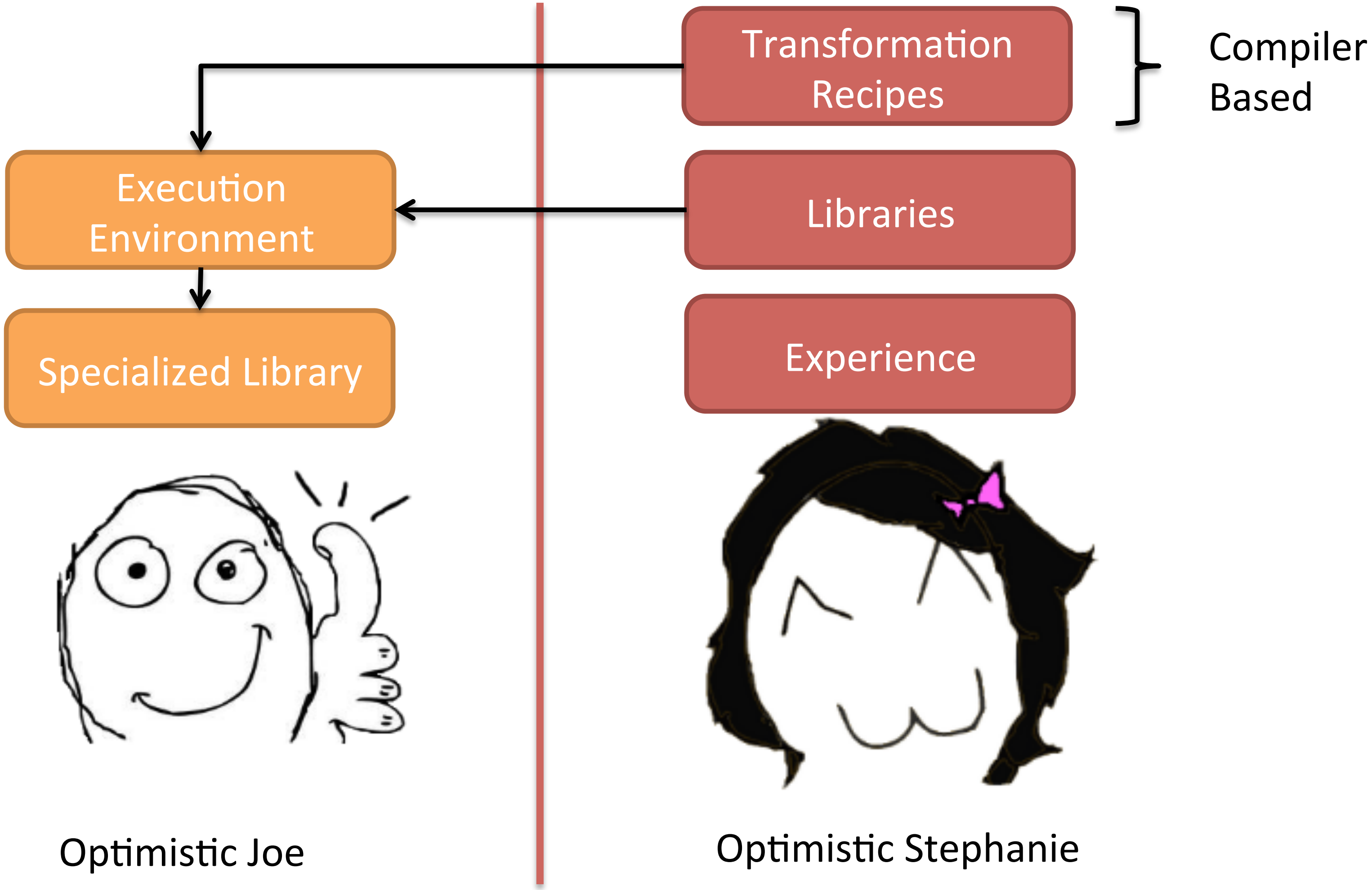
Problem



Amused Joe

Optimistic Stephanie

Problem


Transformation Recipes
Compiler Based
Execution Environment
Libraries
Specialized Library
Experience
Optimistic Joe
Optimistic Stephanie

Outline

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

THE UNIVERSITY OF UTAH

Specializing PETSc

What is it?

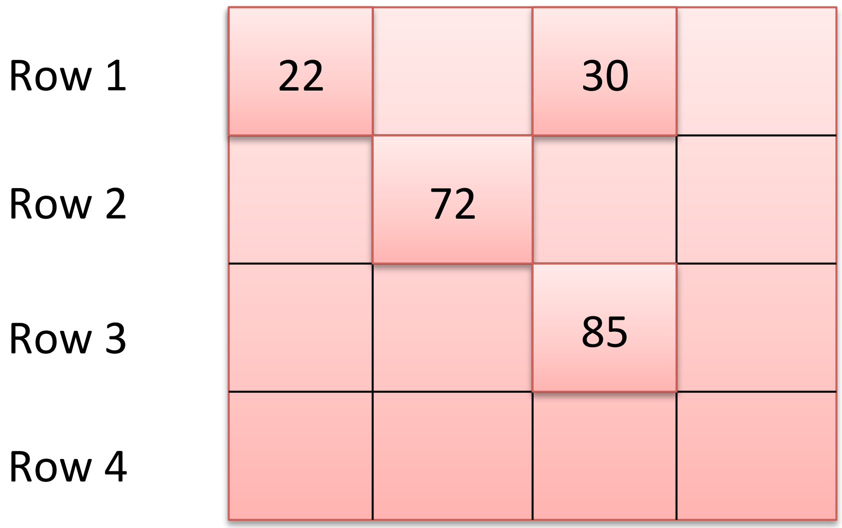
- Portable, Extensible Toolkit for Scientific Computing.
- Suite of matrix representations and routines.
- Higher level linear and non linear solvers that call lower level BLAS where applicable.

Why is it Important?

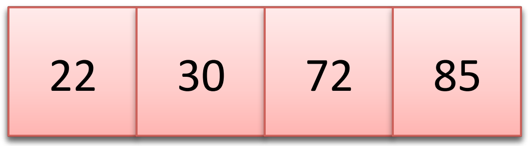
- It is used by more than 200 applications.
- Library already supports specialization.
- Actively developed by Argonne National Labs

THE UNIVERSITY OF UTAH

Brief Overview of Sparse Linear Algebra



Matrix A



Layout in memory

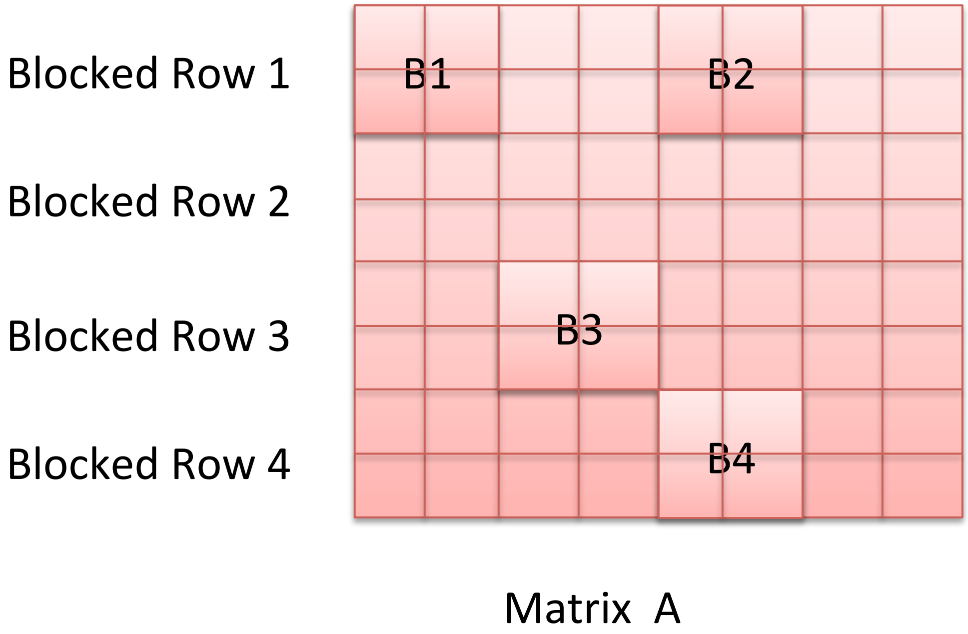
Compressed Sparse Row Representation

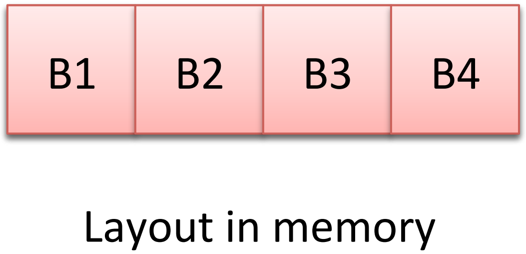
PETSc Name - AIJ

THE UNIVERSITY OF UTAH

Brief Overview of Sparse Linear Algebra

Each square is a block of mxm elements





Blocked Sparse Row Representation

PETSc Name - BAIJ

Specializations for Block Sizes in PETSc

Matrix Type	Description	No of Specialized Functions	Total No of Manually Written Functions
AIJ	Compressed Sparse Row	0	0
BAIJ	Blocked Sparse Row	15	115
MAIJ	Matrix used for restriction and interpolation for multi component problems	4	52
SBAIJ	Symmetric Blocked Sparse Row	10	75
Total		29	242

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

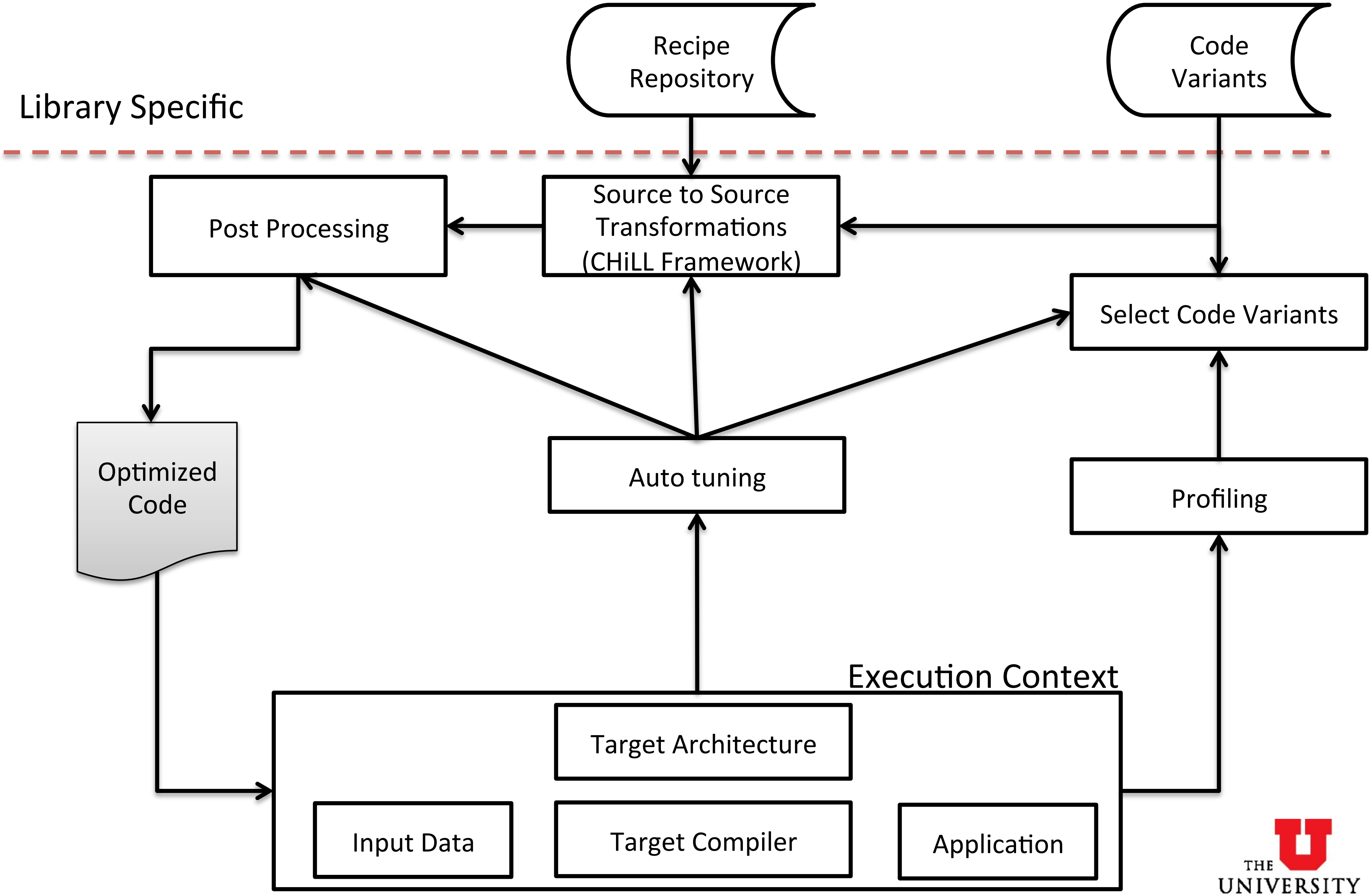
- Improving Instruction Scheduling
- Increasing Instruction Level Parallelism
- Improve register usage and reuse.
- Reduce Cache misses
- Generate SIMD instructions

Outline

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

THE UNIVERSITY OF UTAH

Specialization Approach



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 hydro carbon fibers, structures and explosives
UNIC ANL	• 3D unstructured neutronic code to obtain higly detailed description of the neuclear reactor core.

THE UNIVERSITY OF UTAH

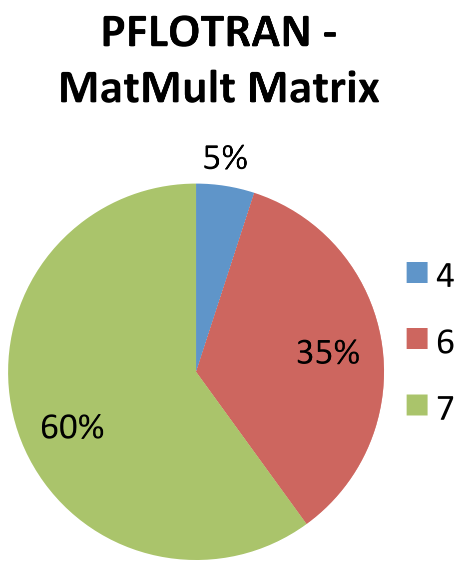
Profiling – Step 1
Identifying PETSc Functions

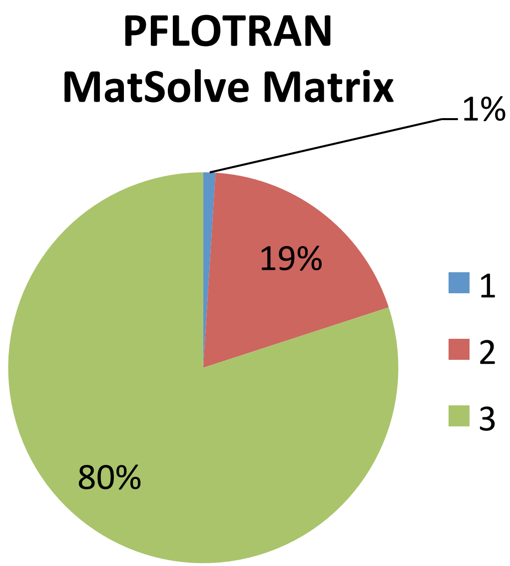
Application	Matrix Type	PETSc Function	% Exec Time
PFLOTRAN	BAIJ	MatLUFactorNumeric_SeqBAIJ_N MatSolve_SeqBAIJ_N MatMult _SeqBAIJ N	10% 9.8% 9.8%
Uintah	AIJ	MatMult _SeqAIJ (PetscSolve) MatMult _SeqAIJ (ApplyFilter)	23% 3%
UNIC	SBAIJ	MatMult _SeqSBAIJ_1 MatRelax _SeqBAIJ	39.7 % 46.9%

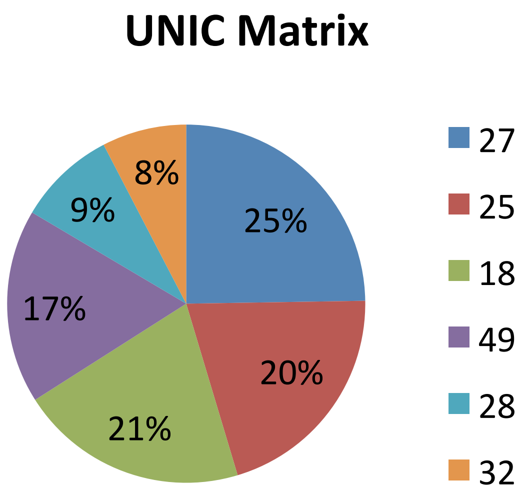
THE UNIVERSITY OF UTAH

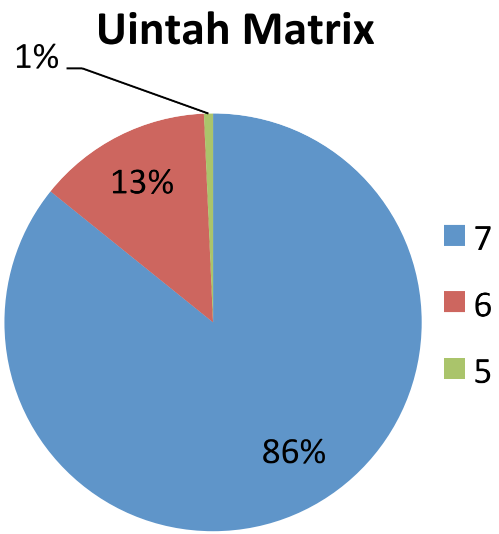
Profiling – Step 2

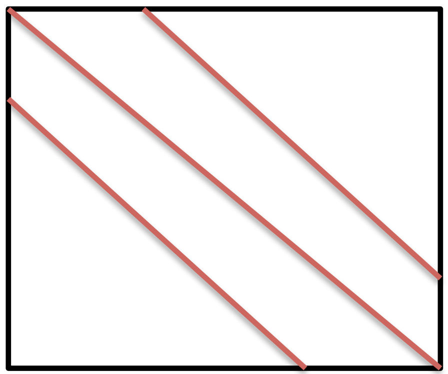
Frequency of Non Zeros in Input Data











Visualizing Banded Matrices

Optimizing Sparse Computations

Alternate Code Variants

- Matrix Representation
- Gather Operation

Higher Level Transformations

- Loop Unrolling
- Loop Splitting

Specialized using “known” Command in CHiLL

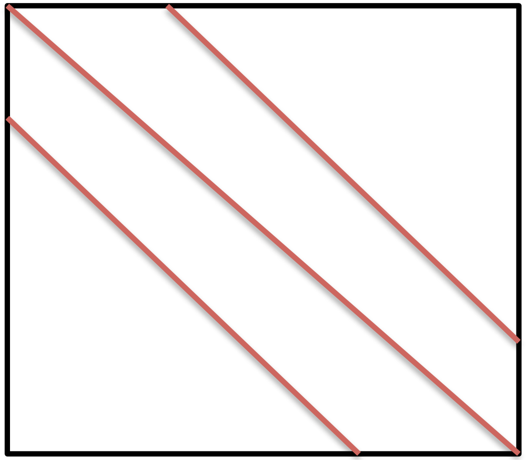
Post Processing

- Prefetching
- Scalar Replacement
- Pragma Vector Always

THE UNIVERSITY OF UTAH

Optimizing Sparse Computations

Alternate Code Variants – Avoiding dynamic loop bounds



Visualizing Banded Matrices

```
 X X X X X X X X X X X
X X X X X X X X X X X
X X X X X X X X X X X X X X
X X X X X X X X X X X X X X
X X X X X X X X X X X
X X X X X X X X X X X
X X X X X X X X X X X
```

Visualizing CSR for Banded Matrices

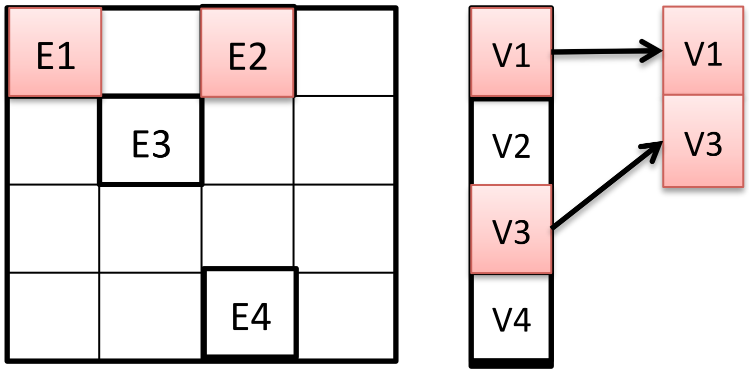
```
X X X X X X X X X X X 0 0 0
X X X X X X X X X X X 0 0 0
X X X X X X X X X X X X X X
X X X X X X X X X X X X X X
X X X X X X X X X X X 0 0 0
X X X X X X X X X X X 0 0 0
X X X X X X X X X X X 0 0 0
```

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

THE UNIVERSITY OF UTAH

Source to Source Transformations Uintah

Alternated code Variant : ELL Format + Register level gather

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

CHiLL Script

```
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];
    }
}
```

Register level gather
Matrix Multiplication for ELL format.

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];
      y[t2] = y[t2] + aa[t2 * 7 + 6] * colVal[6];
}
```

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];
     y[t2] = y[t2] + aa[t2 * 7 + 6] * colVal[6];
}
```

Example of Specialized Function

```
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];
        }
        <-------------------------------------------->
        Specialized Code for 5 more cases
        <-------------------------------------------->
        default:
        for (j=jmin; j<n; j++) {
                cval = *ib;
                z[cval] += *v * x1;
                z[i] += *v++ * x[*ib++];
                }
        break;
        }
}
```

Outline

Motivation

PETSc and Specializations

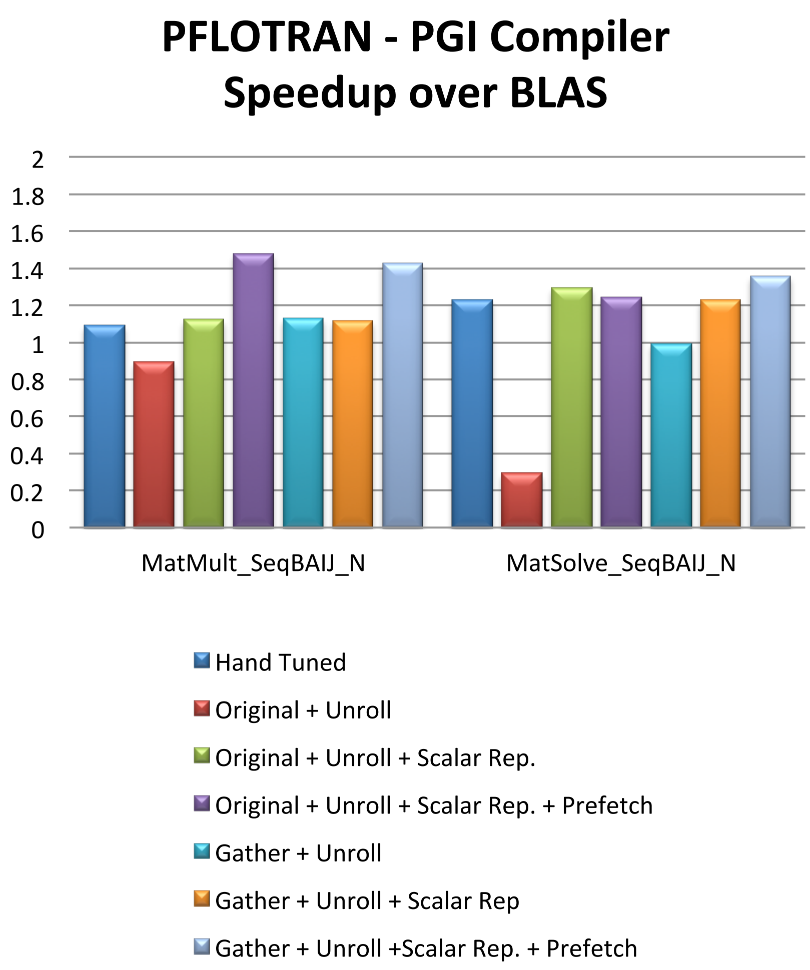
Specialization Approach

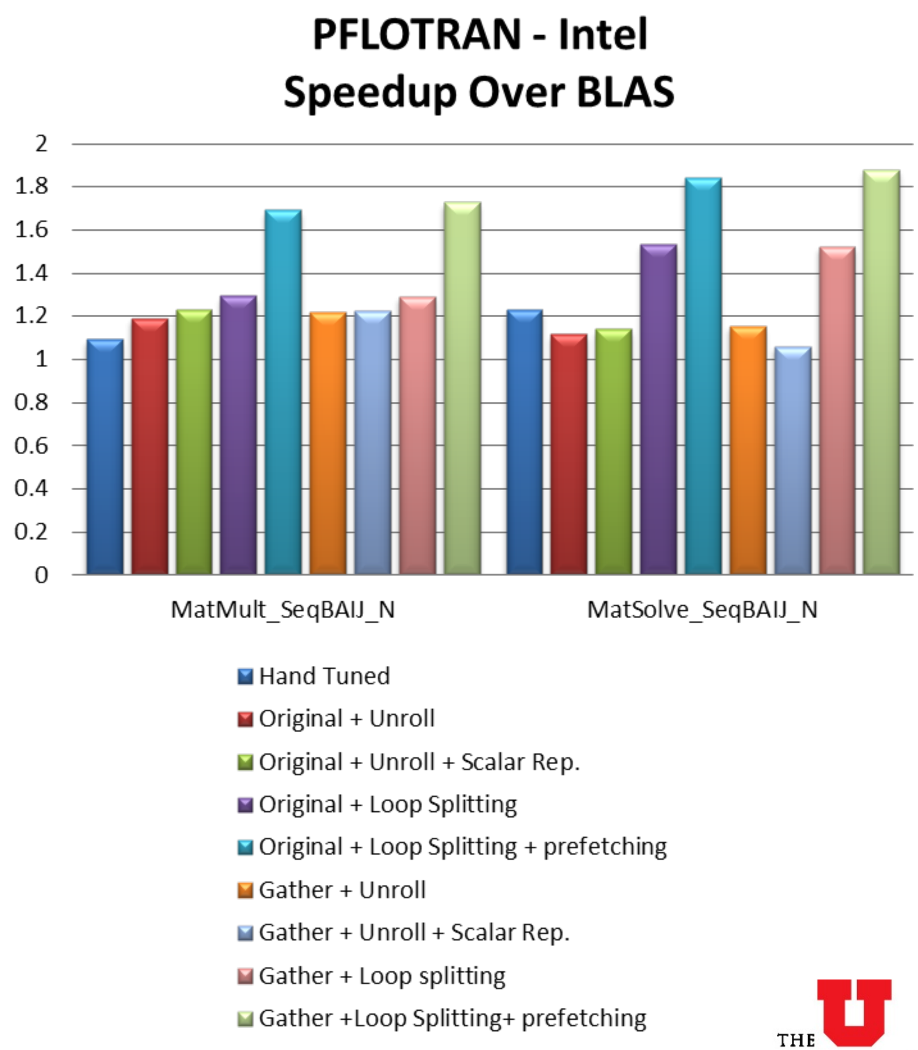
Result

Summary

THE UNIVERSITY OF UTAH

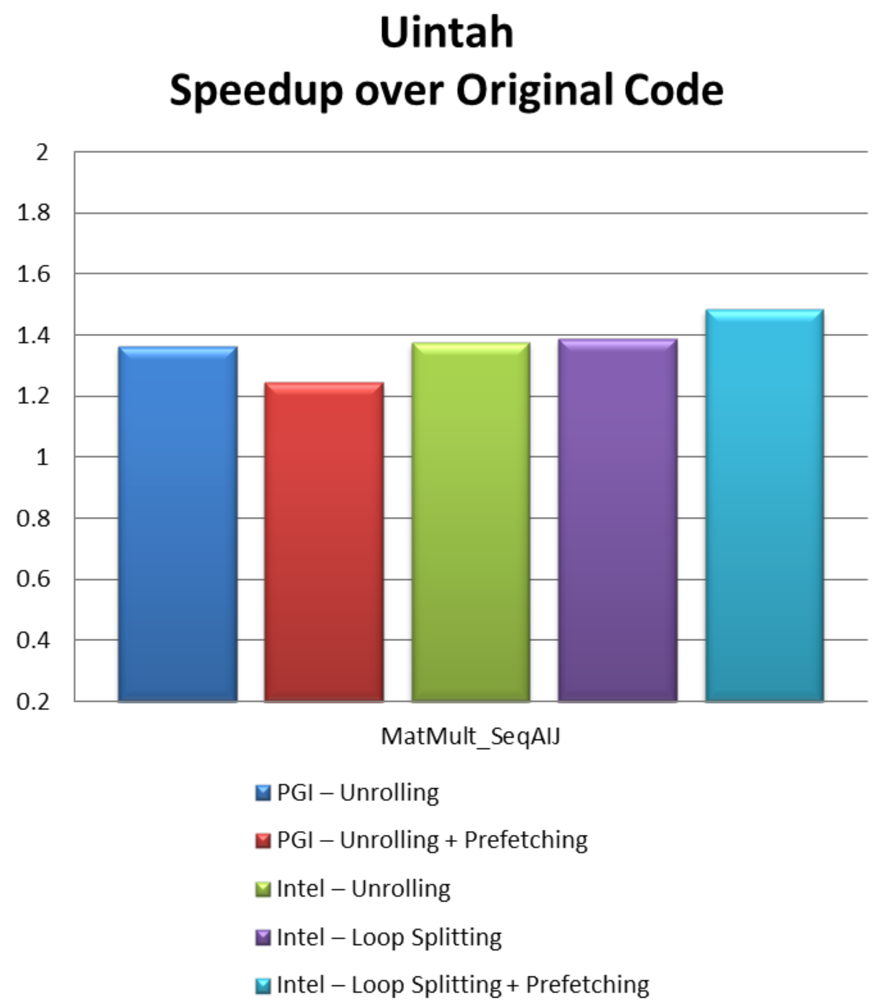
PFLOTRAN on Hopper XE6 System



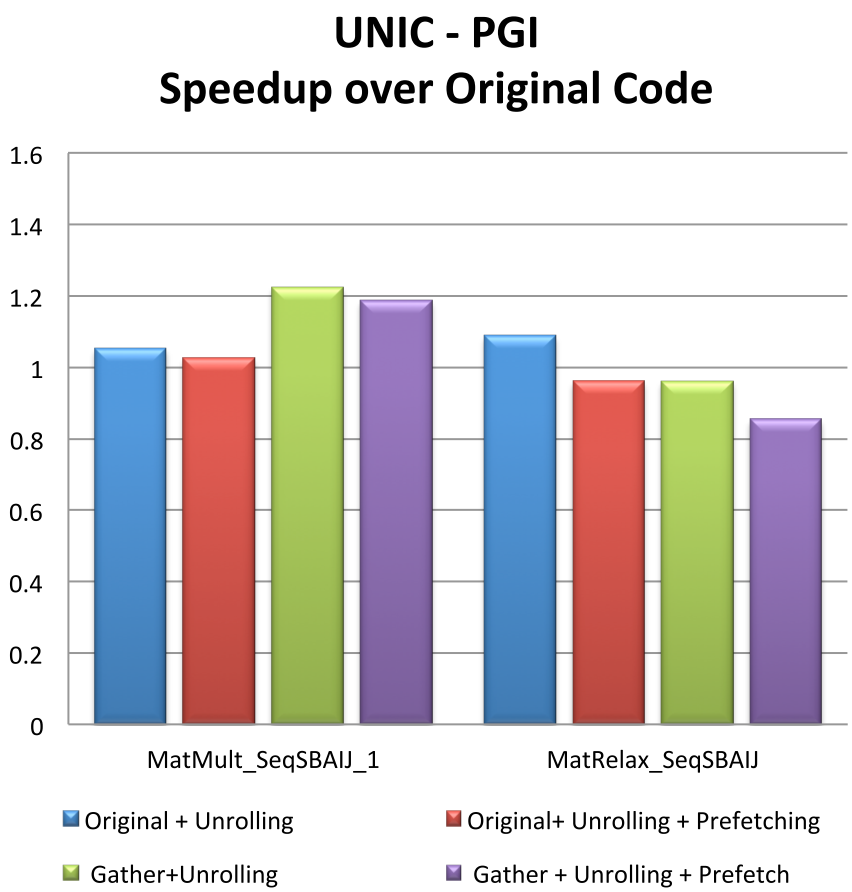


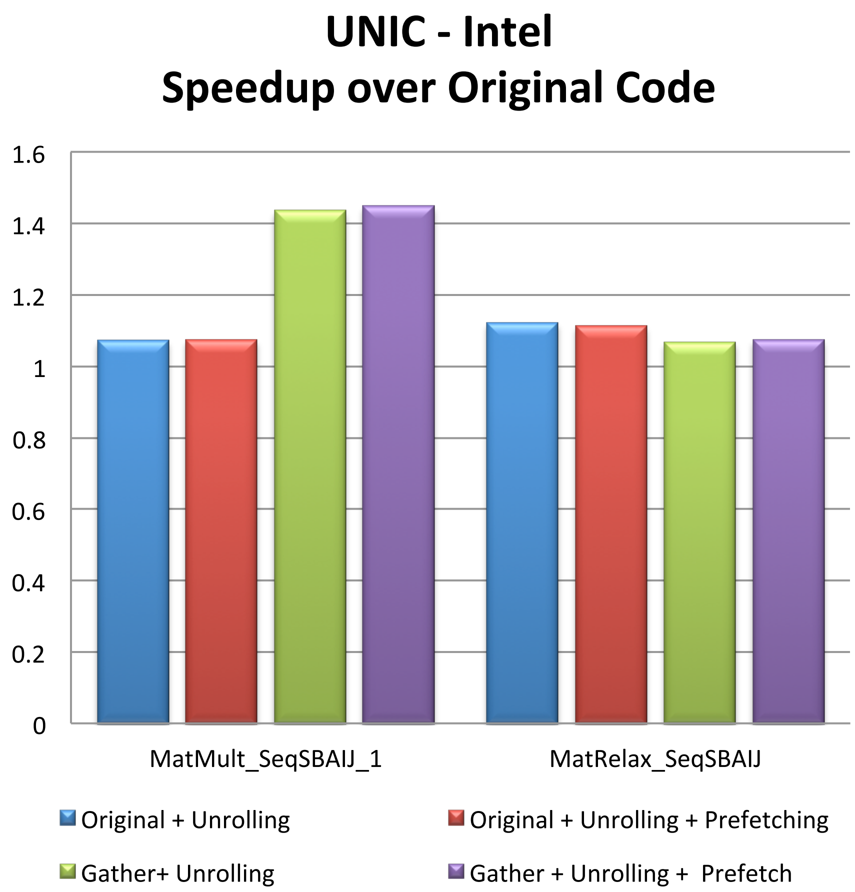
THE UNIVERSITY OF UTAH

Uintah on Hopper XE6 System

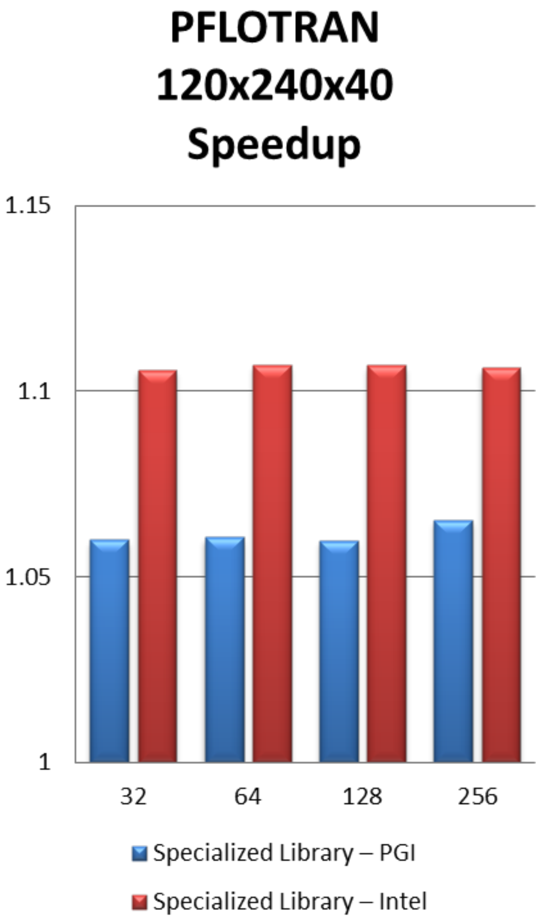


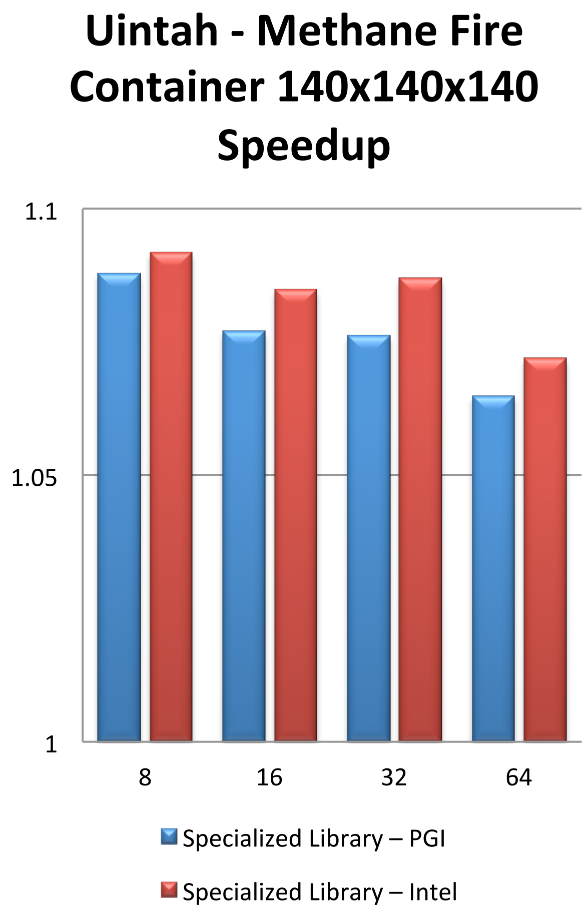
UNIC on Hopper XE6 System

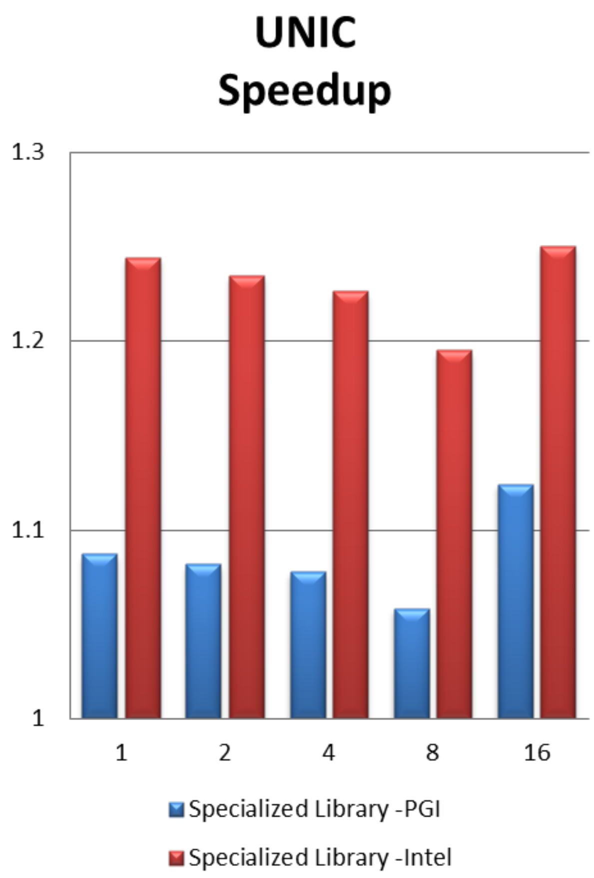




Application Speedups







Outline

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

THE UNIVERSITY OF UTAH

Degree of Automation

Stage	State of Automation	Future Work
Profiling – Finding PETSc Functions	Manual - Used HPCToolkit	Integrate with TAU
Profiling – Identifying the non zeros	Manual – Used HPCToolkit	Integrate with TAU
Selecting code variants	Manual	Auto tuning framework – Active Harmony
Optimizing code	Automated	Automated
Post Processing	Manual	CHiLL
Finding optimal solution	Exhaustive search using Scripts	Auto tuning framework – Active Harmony

THE UNIVERSITY OF UTAH

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.

THE UNIVERSITY OF UTAH

Related Work

Optimizing Sparse Matrices

- John Mellor-Crummey and John Garvin. 2004. Optimizing Sparse Matrix-Vector Product Computations Using Unroll and Jam
- Herrero, J. R., and Navarro, J. J. Improving performance of hypermatrix Cholesky factorization.

Auto tuning and specialization

- Tiwari, A., Hollingsworth, J. K., Chen, C., Hall, M., Liao, C., Quinlan,D. J., and Chame, J. Auto-tuning full applications: A case study.
- Shin, J., Hall, M. W., Chame, J., Chen, C., Fischer, P. F., and Hovland,P. D. Speeding up nek5000 with autotuning and specialization.