Build To Order (BTO) BLAS Compiler

Oliver Ney

11th July, 2016

Overview

BLAS

- **B**asic **L**inear **A**lgebra **S**ubprograms
- Defined set of commonly used routines
- Highly optimized implementations on various platforms
  - e.g. Intel Math Kernel Library (IMKL), Armadillo
- Building blocks for higher level algorithms
- Split into 3 levels

BLAS - Levels

- **Level 1**
  - Vector routines
  - e.g. dot product, norms
- **Level 2**
  - Matrix-Vector routines
  - e.g. matrix-vector product, solve linear system with triangular matrix
- **Level 3**
  - Matrix routines
  - e.g. matrix product

Performance Constraints

Performance of BLAS execution is constrained by:

- Processing Power (FLOPS)
- Memory Size, Speed and Bandwidth

Performance Constraints

Performance of BLAS execution is constrained by:

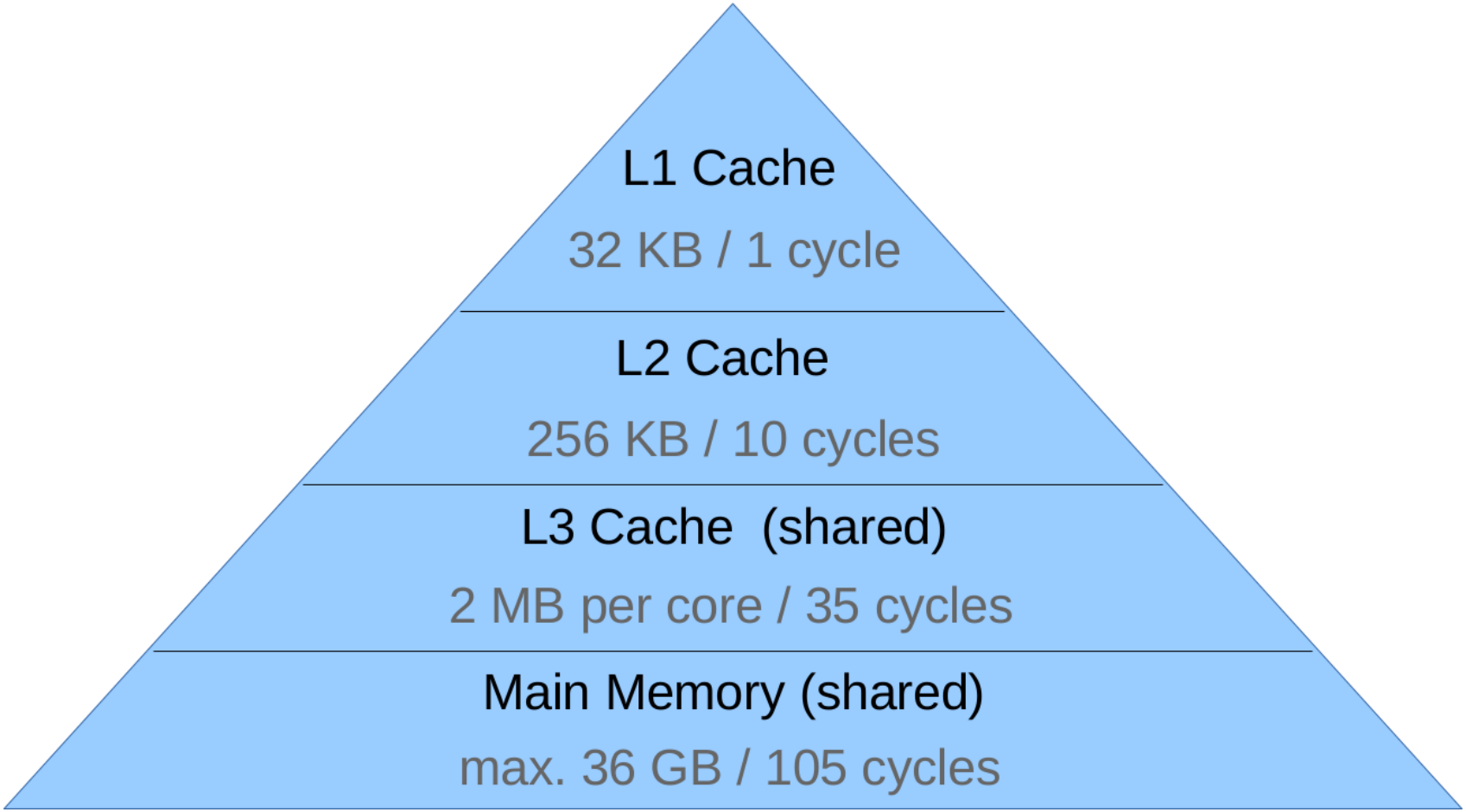
- Processing Power (FLOPS)
  - well optimized for by compilers
- Memory Size, Speed and Bandwidth

Performance Constraints

Performance of BLAS execution is constrained by:

- Processing Power (FLOPS)
  - well optimized for by compilers
- Memory Size, Speed and Bandwidth
  - requires in-depth analysis

Memory Hierarchy



based on Intel i7 Nehalem architecture

Consequences

During math kernel execution operands...

- ... need to stay in cache.
- ... should be local.

Common BLAS libraries...

- ... are well optimized for this on level 3.
- ... cannot optimize across routines.

Build To Order BLAS Compiler

- Developed at University of Colorado
- Compiles math kernels to custom BLAS routines
- Optimizes generated code for memory efficiency on host computer
  - Based on empirical results
  - Optional analytical model
- Allows for implicit parallelization using pthreads or OpenMP

Input and Output Format

Input:

- Name of kernel
- Input and output arguments
  - Scalars, Vectors, Matrices
  - Column-/Row-Major, Symmetric, Triangular, ...
- MATLAB-like body

Output:

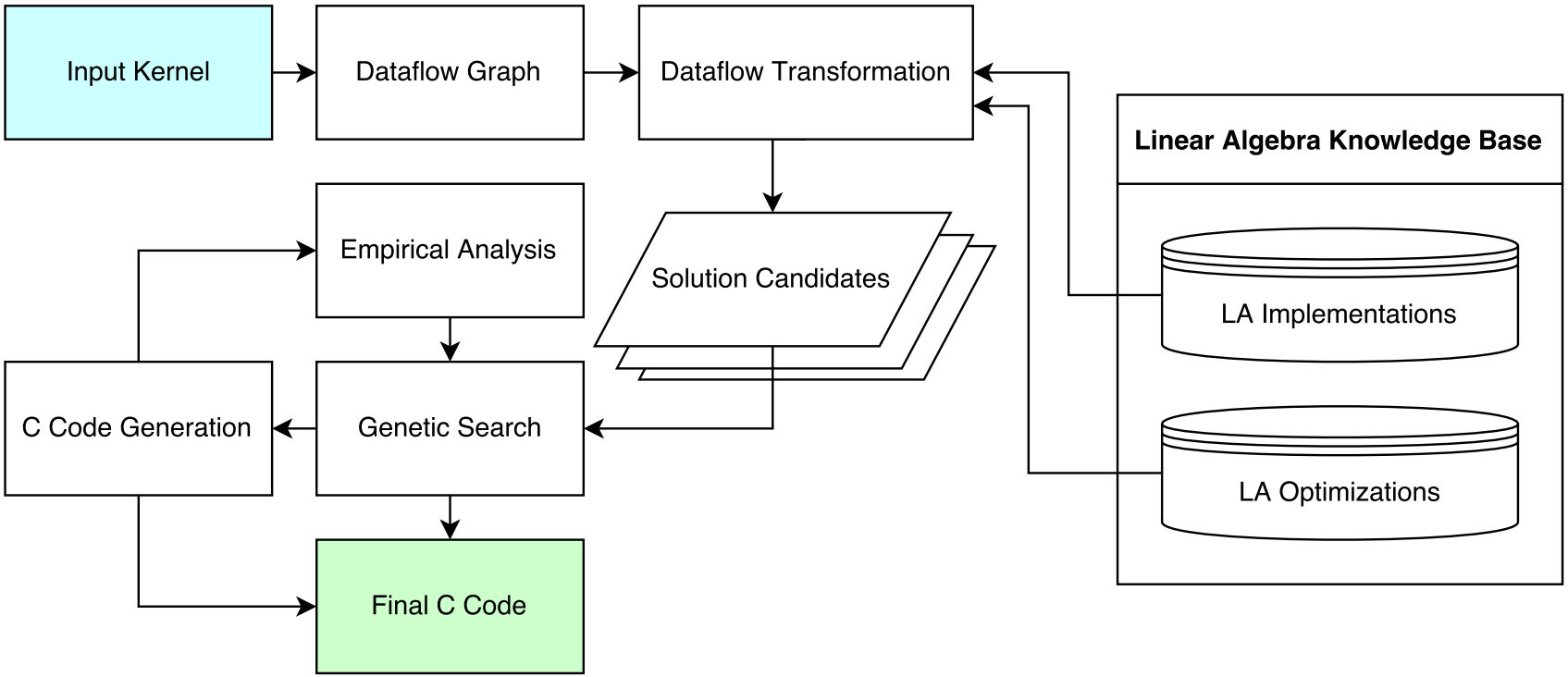
- Single C function named like the kernel
- Arguments matching input declaration (plus dimensions)

Input and Output Format

Input Example

```
1  GEMVER
2  in
3    A : matrix(column), u1 : vector(column), u2 : vector
         (column),
4    v1 : vector(column), v2 : vector(column),
5    a : scalar, b : scalar,
6    y : vector(column), z : vector(column)
7  out
8    B : matrix(column), x : vector(column), w : vector(
         column)
9  {
10   B = A + u1 * v1' + u2 * v2'
11   x = b * (B' * y) + z
12   w = a * (B * x)
13 }
```

Architecture Overview



Dataflow Graph

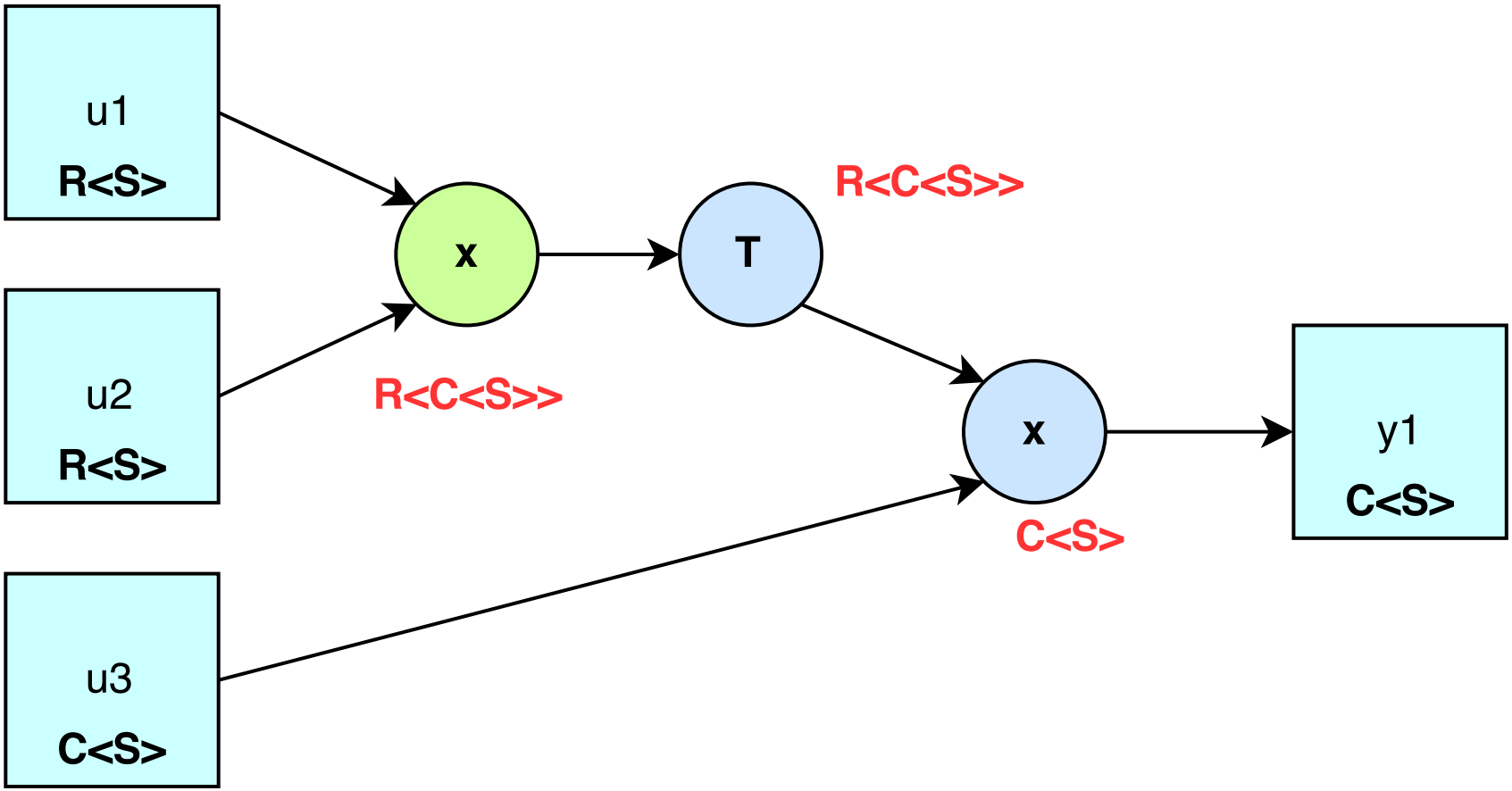
- Input file parsed into dataflow graph
- Arguments as roots/leaves, operators as knots

- Every knot is looked up in a linear algebra database
- Recursively applied until all knots work on scalars
  - generates iterating subgraphs

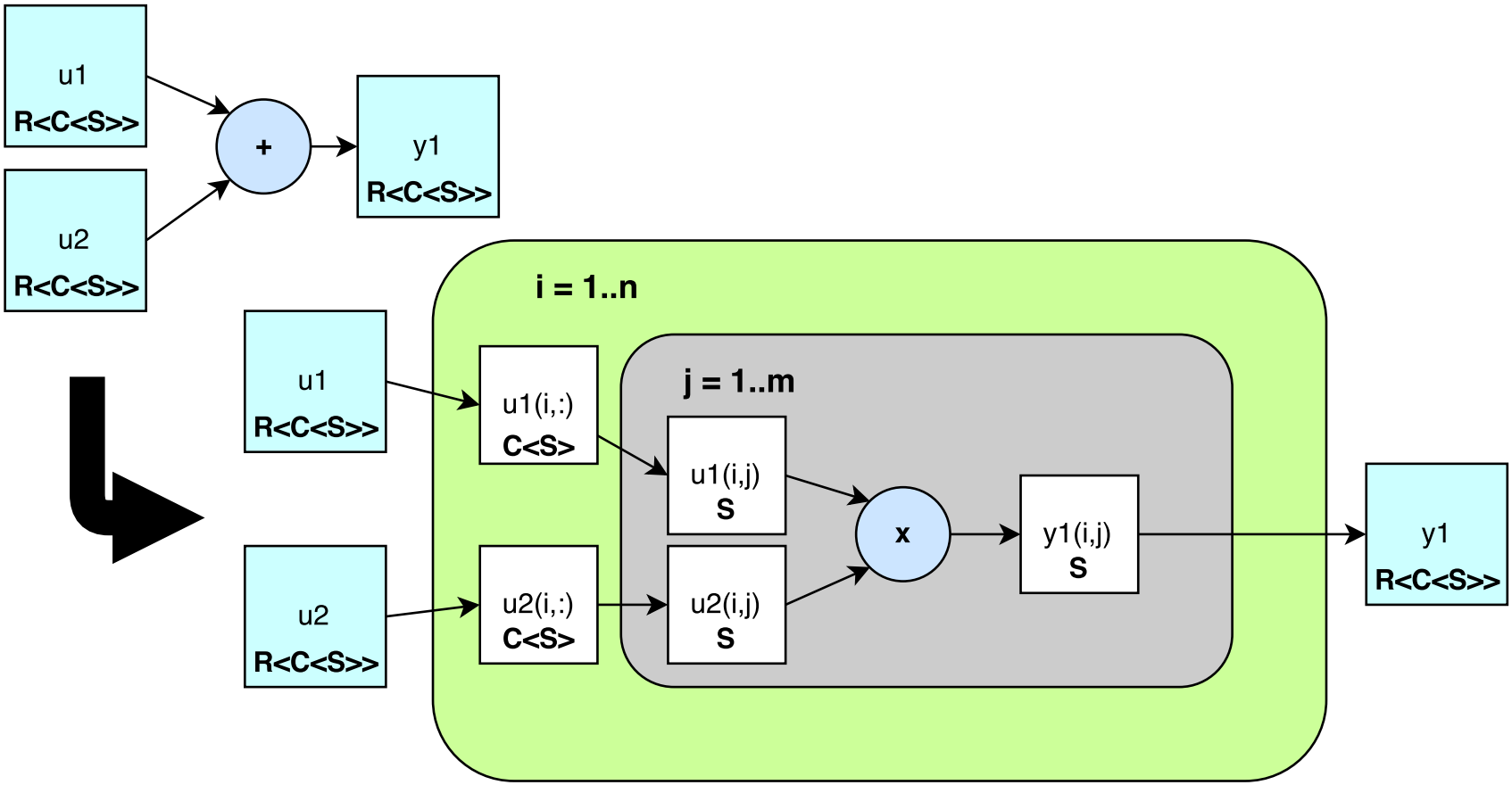
Linear Algebra Database Entry

`rr-mult`	$R < \tau_l > \times R < \tau_r >$	$R < R < \tau_l > \times \tau_r >$

Dataflow Graph



Dataflow Graph



Loop fusion

If two adjacent loops ...

- have the same indices and conditions
- and operate on the same values

... they can be **fusioned** into a single loop.

Loop fusion

```
1  mat a;
2
3  for(i = 1..n) {
4    x = a(i,:);
5    ...
6  }
7
8  for(j = 1..n) {
9    y = a(j,:);
10   ...
11 }
```

Loop fusion

```
1  mat a;
2
3  for(i = 1..n) {
4    x = a(i,:);
5    ...
6  }
7
8  for(j = 1..n) {
9    y = a(j,:);
10   ...
11 }
```

```
1  mat a;
2
3  for(i = 1..n) {
4    x = a(i,:);
5    /* y == x */
6    ...
7  }
```

Loop fusion

If two adjacent loops ...

- have the same indices and conditions
- and operate on the same values

... or a succeeding loop ...

- uses the same indices and conditions
- consumes the output of the previous loop

... they can be **fusioned** into a single loop.

Loop fusion

```
1  mat a, b;
2
3  for(i = 1..n) {
4    b(i,:) = a(i,:) * 2;
5    ...
6  }
7
8  for(j = 1..n) {
9    y = b(j,:);
10   ...
11 }
```

Loop fusion

```
1  mat a, b;
2
3  for(i = 1..n) {
4    b(i,:) = a(i,:) * 2;
5    ...
6  }
7
8  for(j = 1..n) {
9    y = b(j,:);
10   ...
11 }
```

```
1  mat a;
2
3  for(i = 1..n) {
4    y = a(i,:) * 2;
5    /* y == b(i,:) */
6    ...
7  }
```

- $\Rightarrow$ Improves localization of operands
- $\Rightarrow$ Reduces loop overhead

Loop tiling

- If a succeeding loop ...
  - has a high dependency on the previous loop
  - uses very large shared operands
- or if parallelism is wanted ...

... the loops can be **tiled** to improve performance.

Loop tiling

- If a succeeding loop ...
  - has a high dependency on the previous loop
  - uses very large shared operands
- or if parallelism is wanted ...

... the loops can be **tiled** to improve performance.

```
1  mat a, b;
2
3  for(i = 1..n) {
4    b(i,:) = a(i,:) * 2;
5    ..1..
6    b(i,:) *= 3;
7    ..2..
8  }
9  for(j = 1..n) {
10   y = a(j,:) * 2;
11   ..3..
12 }
```

Loop tiling

- If a succeeding loop ...
  - has a high dependency on the previous loop
  - uses very large shared operands
- or if parallelism is wanted ...

... the loops can be **tiled** to improve performance.

```
1  mat a, b;
2
3  for(i = 1..n) {
4    b(i,:) = a(i,:) * 2;
5    ..1..
6    b(i,:) *= 3;
7    ..2..
8  }
9  for(j = 1..n) {
10   y = a(j,:) * 2;
11   ..3..
12 }
```

```
1  mat a, b;
2
3  for(i = 1..n) {
4    b(i,:) = a(i,:) * 2;
5    ..1..
6    ..3..
7  }
8  for(j = 1..n) {
9    b(i,:) *= 3;
10   ..2..
11 }
```

Genetic Search

- Enormous search space for best performing solution
- Exhaustive search is inapplicable

- Explores possible candidates by decision mutation
- Every candidate evaluated by empirical analysis

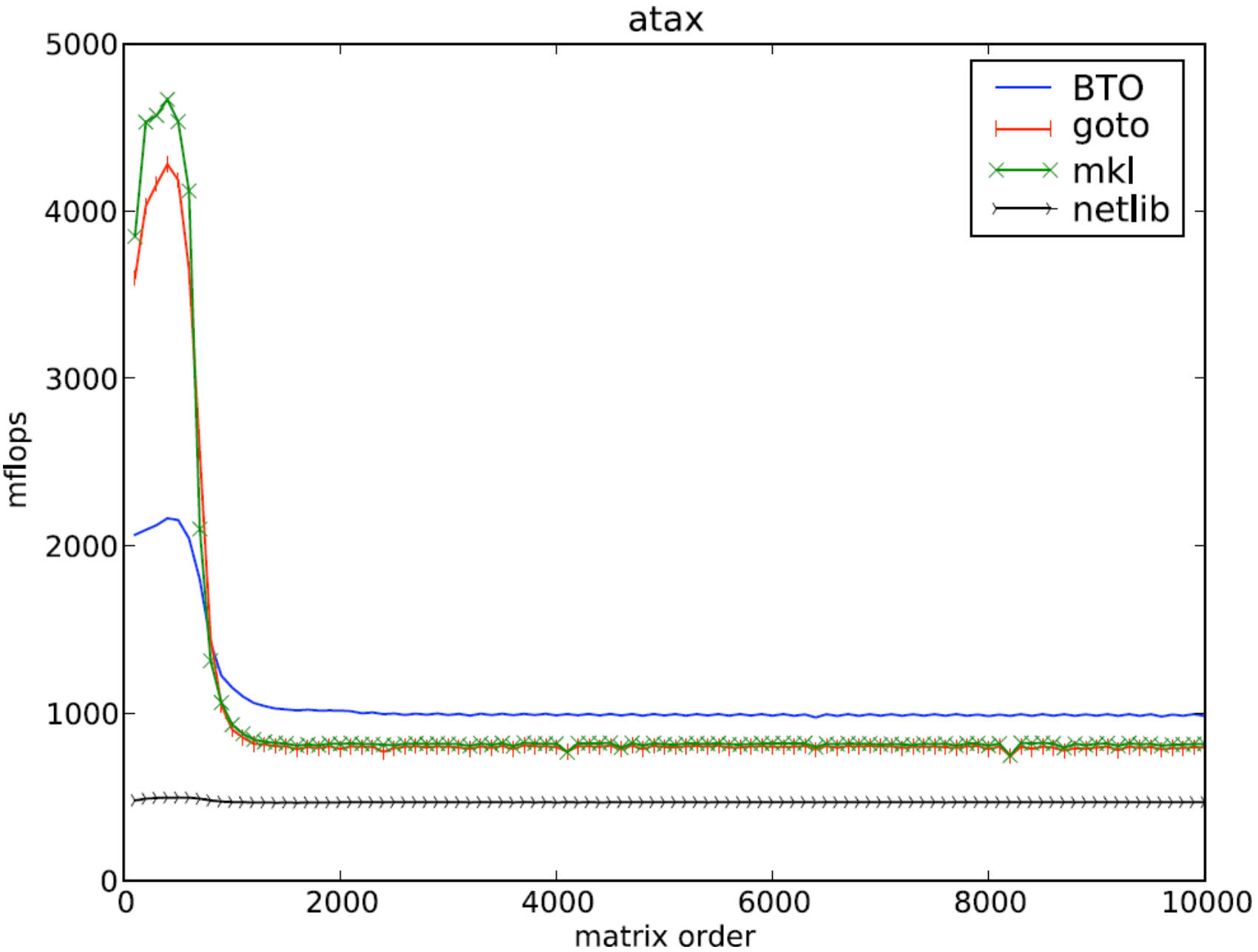
- No guarantee to find optimal solution
- Hard limit on number of generations

Empirical Analysis

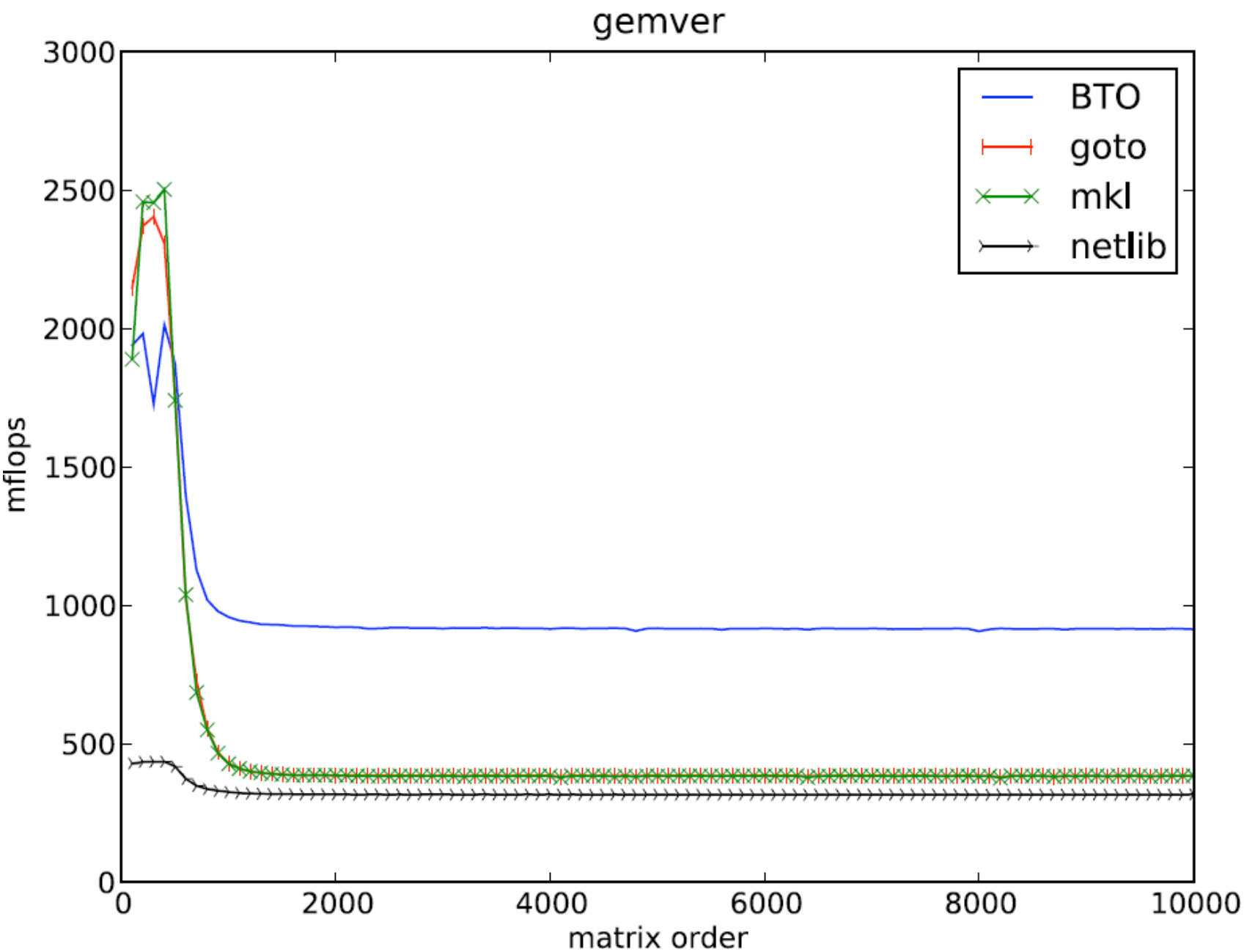
Every solution considered by the search is...

- compiled
- executed on a fixed input set
  - e.g. 3000x3000 matrices
- weighted by execution time

Results


atax
BTO
goto
mkl
netlib
mflops
matrix order
0
1000
2000
3000
4000
5000
0
2000
4000
6000
8000
10000

Results


gemver
BTO
goto
mkl
netlib
mflops
matrix order
0
500
1000
1500
2000
2500
3000
0
2000
4000
6000
8000
10000

Conclusions

Pros:

- Loop fusion/tiling strongly improves performance for chains of
  - vector-vector operations
  - matrix-vector operations
- Giant search space is a big hurdle
  - provides various methods to handle it
  - average compilation time still acceptable
- Automatic parallelization using pthreads

Cons:

- Project appears dead
- Errors on compilation

References I

[0] Belter, Geoffrey et al. (2009). "Automating the Generation of Composed Linear Algebra Kernels". In: *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*. SC '09. Portland, Oregon: ACM, 59:1–59:12. ISBN: 978-1-60558-744-8. DOI: 10.1145/1654059.1654119. URL: http://doi.acm.org/10.1145/1654059.1654119.

[0] Nelson, Thomas (2015). "DSLs and Search for Linear Algebra Perfromance Optimization". PhD thesis. UNIVERSITY OF COLORADO AT BOULDER.

[0] Siek, Jeremy G. (2009). *Build To Order BLAS*. CScADS Summer Workshop - Libraries and Autotuning for Petascale Applications. Lake Tahoe.

References II

[0] Siek, Jeremy G, Ian Karlin, and Elizabeth R Jessup (2008). "Build to order linear algebra kernels". In: *Parallel and Distributed Processing, 2008. IPDPS 2008. IEEE International Symposium on*. IEEE, pp. 1–8.