Performance Modeling for Ranking Blocked Algorithms

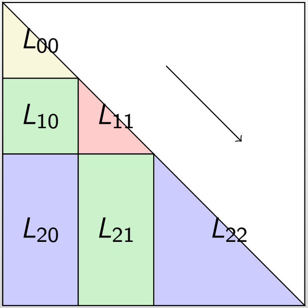
Elmar Peise

Aachen Institute for Advanced Study in Computational Engineering Science

27.4.2012

RWTH AACHEN UNIVERSITY

Inversion of a triangular matrix $L \leftarrow L^{-1} \in \mathbb{R}^{n \times n}$



Variant 1

$L_{10} \leftarrow L_{10} L_{00}$
$L_{10} \leftarrow -L_{11}^{-1} L_{10}$
$L_{11} \leftarrow L_{11}^{-1}$

Variant 2

$L_{21} \leftarrow L_{22}^{-1} L_{21}$
$L_{21} \leftarrow -L_{21} L_{11}^{-1}$
$L_{11} \leftarrow L_{11}^{-1}$

Variant 3

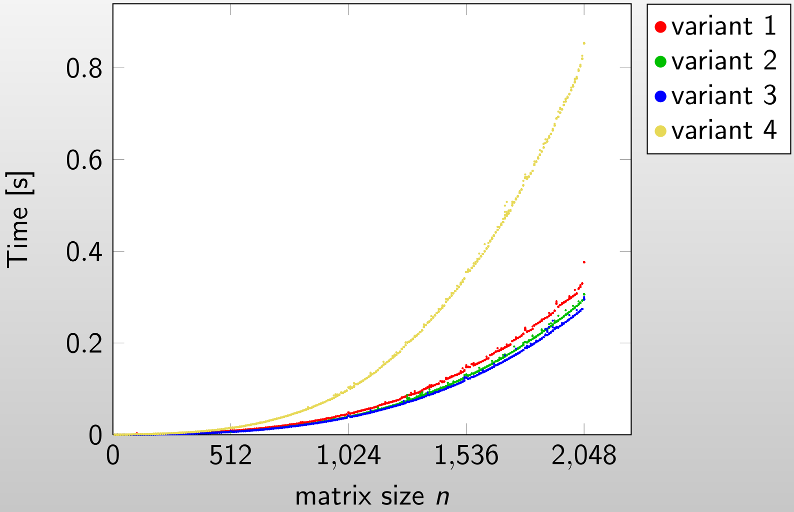
$L_{21} \leftarrow -L_{21} L_{11}^{-1}$
$L_{20} \leftarrow L_{20} + L_{21} L_{10}$
$L_{10} \leftarrow L_{11}^{-1} L_{10}$
$L_{11} \leftarrow L_{11}^{-1}$

Variant 4

$L_{21} \leftarrow -L_{22}^{-1} L_{21}$
$L_{20} \leftarrow L_{20} - L_{21} L_{10}$
$L_{10} \leftarrow L_{10} L_{00}$
$L_{11} \leftarrow L_{11}^{-1}$

Execution time

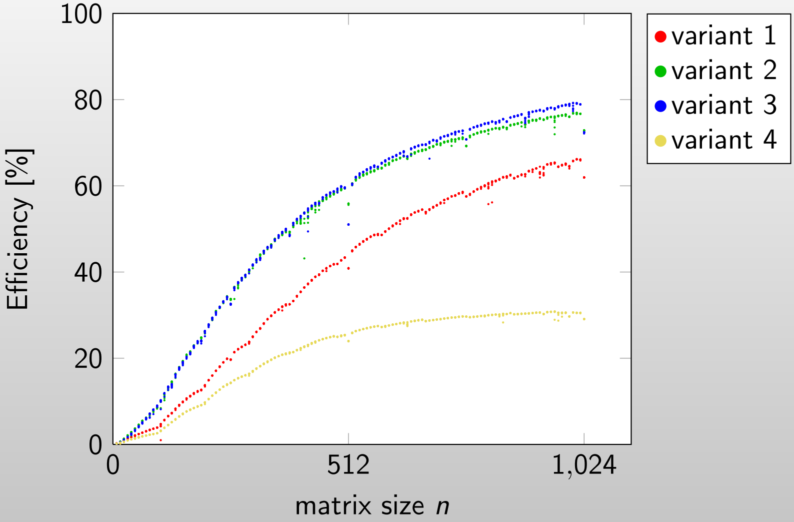
Inversion of a triangular matrix $L \leftarrow L^{-1} \in \mathbb{R}^{n \times n}$



Intel Harpertown E5450 @ 2.99 GHz — 1 core — Intel MKL BLAS

Efficiency

Inversion of a triangular matrix $L \leftarrow L^{-1} \in \mathbb{R}^{n \times n}$



Intel Harpertown E5450 @ 2.99 GHz — 1 core — Intel MKL BLAS

3rd (●)	2nd (●)	1st (●)	4th (●)
Variant 1	**Variant 2**	**Variant 3**	**Variant 4**
$L_{10} \leftarrow L_{10} L_{00}$	$L_{21} \leftarrow L_{22}^{-1} L_{21}$	$L_{21} \leftarrow -L_{21} L_{11}^{-1}$	$L_{21} \leftarrow -L_{22}^{-1} L_{21}$
$L_{10} \leftarrow -L_{11}^{-1} L_{10}$	$L_{21} \leftarrow -L_{21} L_{11}^{-1}$	$L_{20} \leftarrow L_{20} + L_{21} L_{10}$	$L_{20} \leftarrow L_{20} - L_{21} L_{10}$
$L_{11} \leftarrow L_{11}^{-1}$	$L_{11} \leftarrow L_{11}^{-1}$	$L_{10} \leftarrow L_{11}^{-1} L_{10}$	$L_{10} \leftarrow L_{10} L_{00}$
		$L_{11} \leftarrow L_{11}^{-1}$	$L_{11} \leftarrow L_{11}^{-1}$

Can we rank the algorithms based on their update statements?

No!

Outline

1. Sampling
2. Modeling
3. Prediction and Ranking

$$B \leftarrow 0.37 \, B \, L^{-1} \quad \text{with} \quad B \in \mathbb{R}^{128\times 96}, \; L \in \mathbb{R}^{96\times 96}$$

Routine call

dtrsm(R, L, N, U, 128, 96, 0.37, *L*, 128, *B*, 128)

↓ Sampling ↓

Performance counters (PAPI)

ticks	*flops*	*L1misses*	…
887491	595968	4	

Routine call

`dtrsm(R, L, N, U, 128, 96, 0.37, L, 128, B, 128)`

Performance is independent of L and B.
$\Rightarrow$ Assign (random) memory chunk of sufficient size.

Input

$(\texttt{dtrsm}, \texttt{R}, \texttt{L}, \texttt{N}, \texttt{U}, 128, 96, 0.37, 128 \times 96, 128, 128 \times 96, 128)$

Multiple samples

RWTH AACHEN UNIVERSITY

Input

(dtrsm, R, L, N, U, 128, 96, 0.37, 12288, 128, 12288, 128)
(dtrsm, R, L, N, U, 128, 96, 0.37, 12288, 128, 12288, 128)
(dtrsm, R, L, N, U, 128, 96, 0.37, 12288, 128, 12288, 128)
(dtrsm, R, L, N, U, 128, 96, 0.37, 12288, 128, 12288, 128)

⋮

↓ Sampling ↓

Performance counters

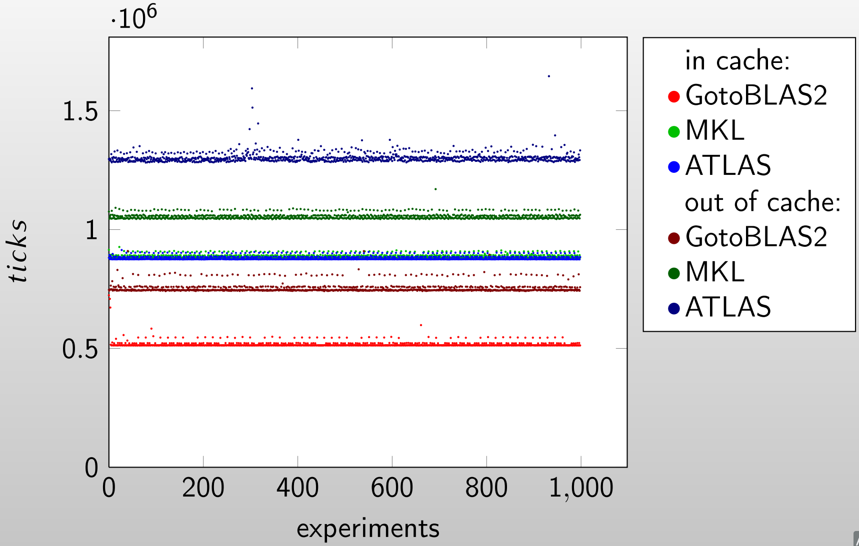
ticks	*flops*	*L1misses*	...
12755926	595976	5172	
926324	595968	27	
887491	595968	4	
882572	595968	1	

⋮

AICES

Influence of caching

`dtrsm(R, L, N, U, 128, 96, 0.37, L, 128, B, 128)`

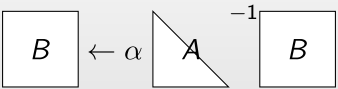


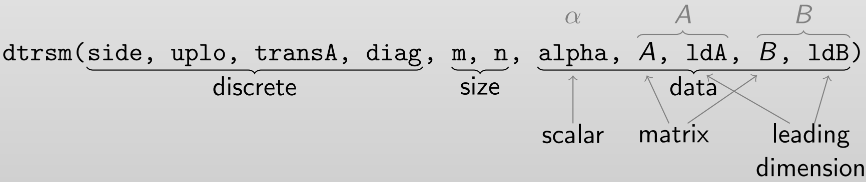
AMD Opteron 8356 (Barcelona) @ 2.3 GHz — 1 core

Outline

1. Sampling
2. Modeling
3. Prediction and Ranking

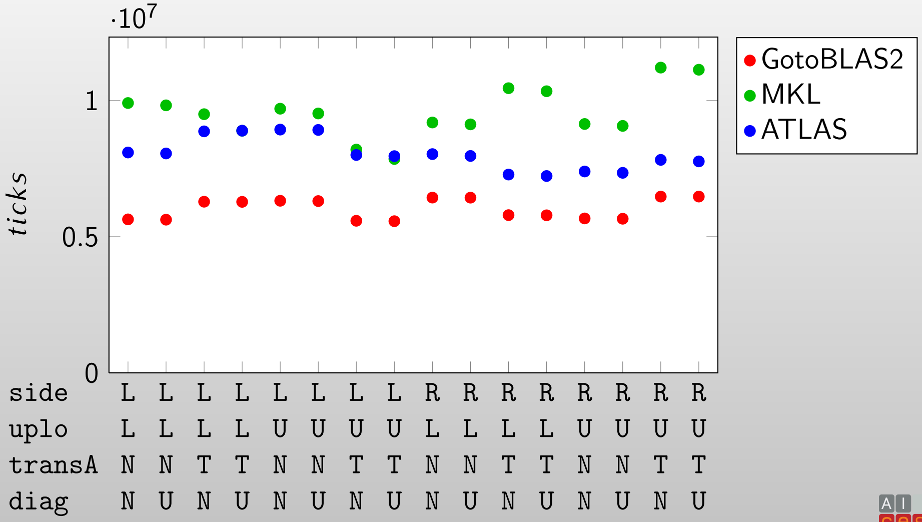
- Understanding performance
- Model structure
- Model generation
- Modeling results


B
← α
A
−1
B


α
A
B
dtrsm(side, uplo, transA, diag, m, n, alpha, A, ldA, B, ldB)
discrete
size
data
scalar
matrix
leading dimension

RWTH AACHEN UNIVERSITY

`dtrsm(side, uplo, transA, diag, 256, 256, 0.5, A, 256, B, 256)`

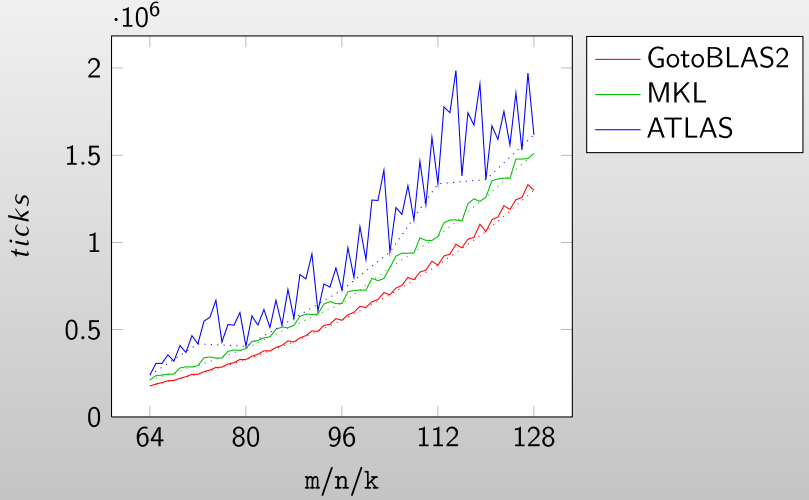


AMD Opteron 8356 (Barcelona) @ 2.3 GHz — 1 core

RWTH AACHEN UNIVERSITY

Small scale

`dgemm(N, N, m, n, k, 0.5, A, m, B, k, 0.5, C, m)`

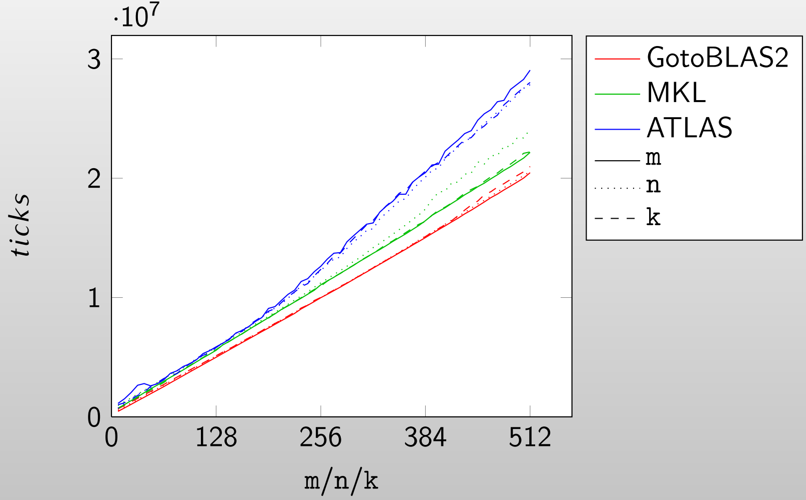


AMD Opteron 8356 (Barcelona) @ 2.3 GHz — 1 core

AI ces

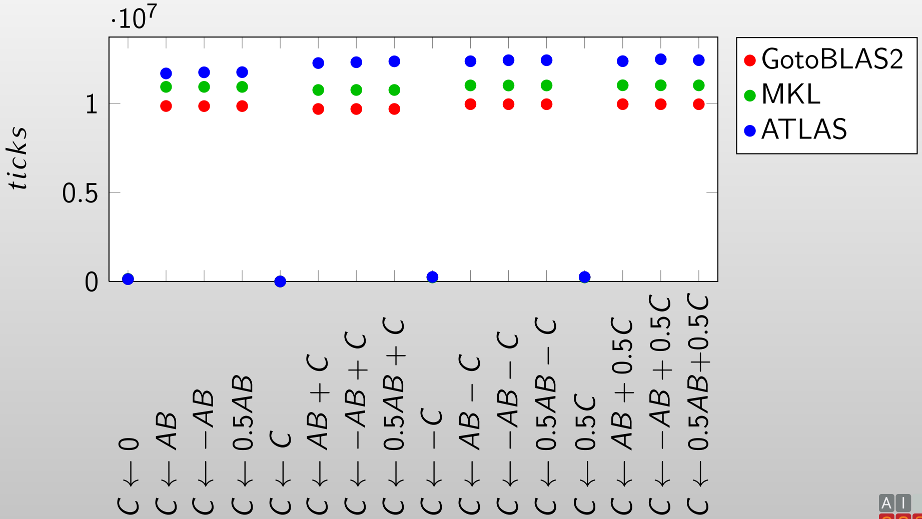
Large scale

`dgemm(N, N, m, n, k, 0.5, A, m, B, k, 0.5, C, m)`



AMD Opteron 8356 (Barcelona) @ 2.3 GHz — 1 core

dgemm(N, N, 256, 256, 256, alpha, A, 256, B, 256, beta, C, 256)



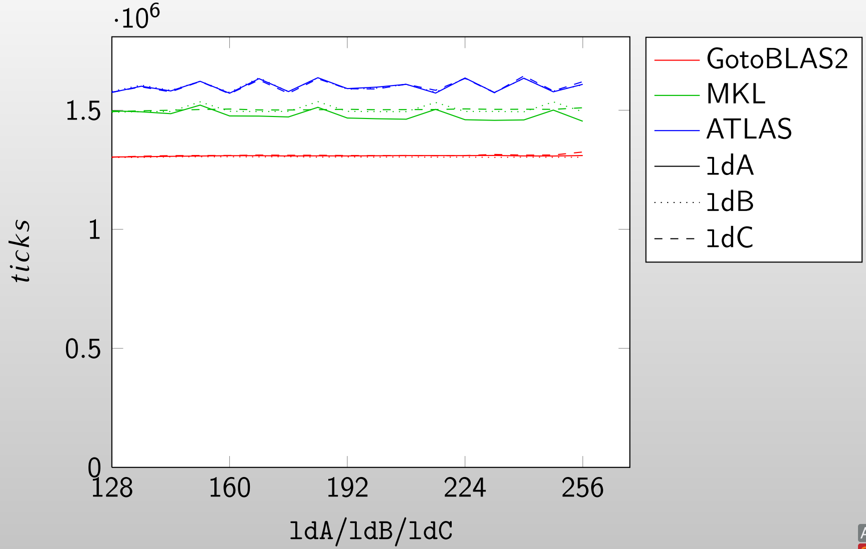
AMD Opteron 8356 (Barcelona) @ 2.3 GHz — 1 core

Matrix arguments

```
dgemm(N, N, 256, 256, 256, 0.5, A, 256, B, 256, 0.5, C, 256)
```

No performance dependency

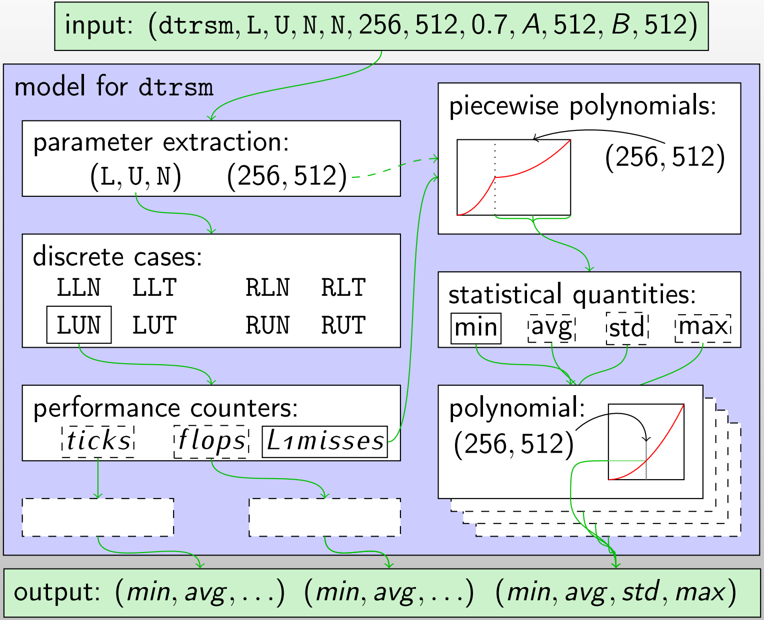
dgemm(N, N, 128, 128, 128, 0.5, *A*, lda, *B*, ldB, 0.5, *C*, ldC)



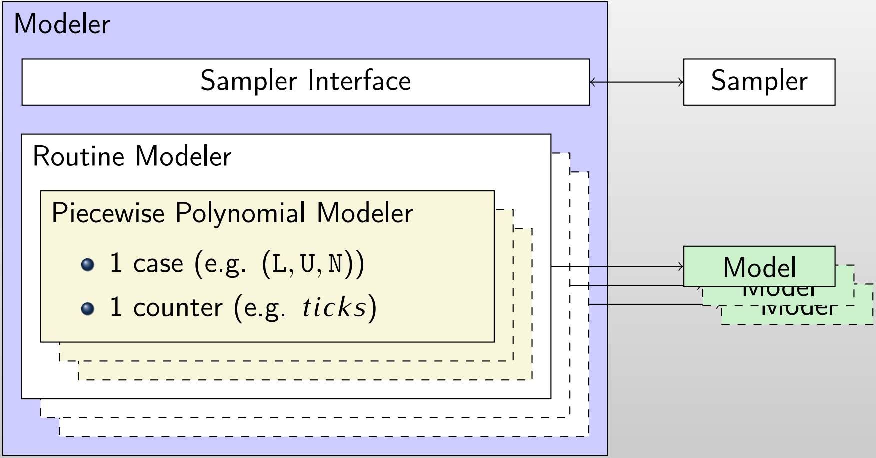
AMD Opteron 8356 (Barcelona) @ 2.3 GHz — 1 core

- Understanding performance
- Model structure
- Model generation
- Modeling results

Model structure



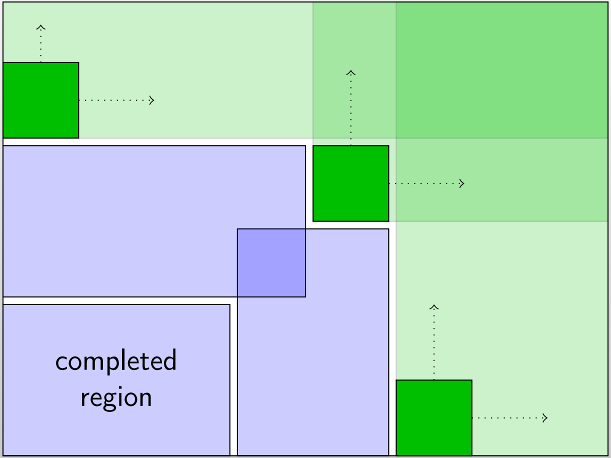
- Understanding performance
- Model structure
- Model generation
- Modeling results


Modeler
Sampler Interface
Sampler
Routine Modeler
Piecewise Polynomial Modeler
• 1 case (e.g. (L, U, N))
• 1 counter (e.g. *ticks*)
Model
Model
Model

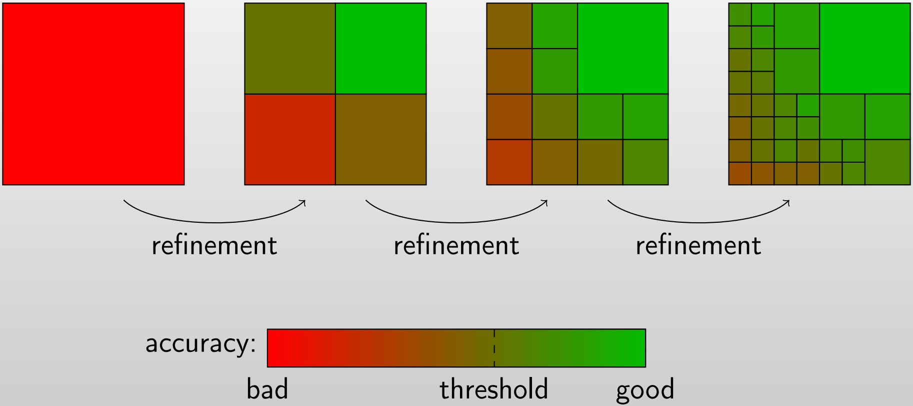
- Model Expansion

- Adaptive Refinement

Model Expansion



Adaptive Refinement

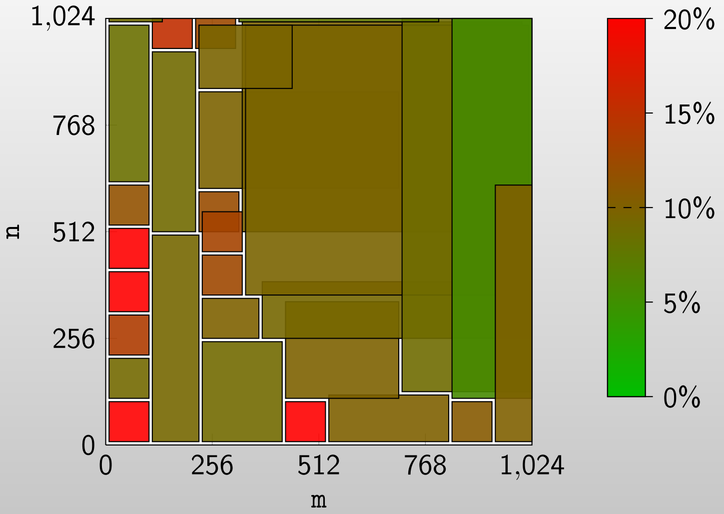


- Understanding performance
- Model structure
- Model generation
- Modeling results

Model Expansion for *ticks*

RWTH AACHEN UNIVERSITY

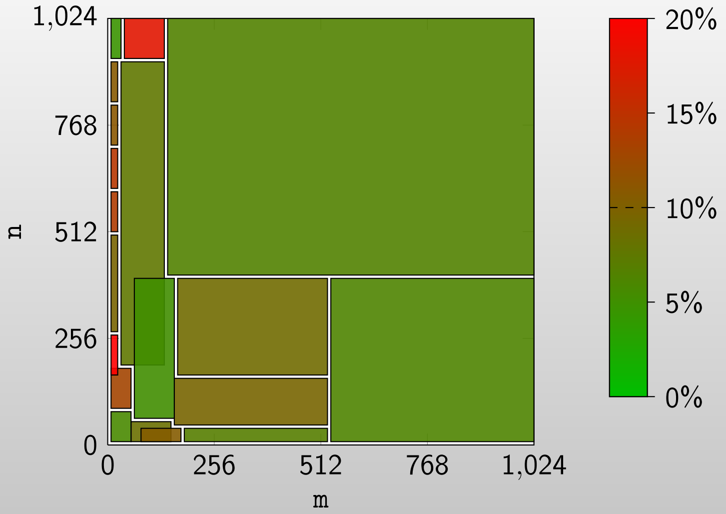
`dtrsm(L, L, N, N, m, n, .5, L, 2500, B, 2500)`



AMD Opteron 8356 (Barcelona) @ 2.3 GHz — 1 core — GotoBLAS2

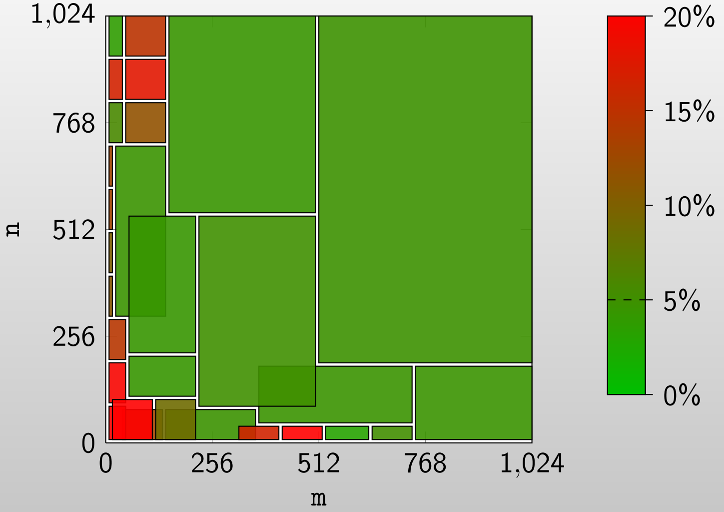
Opposite direction: ↙

dtrsm(L, L, N, N, m, n, .5, *L*, 2500, *B*, 2500)



AMD Opteron 8356 (Barcelona) @ 2.3 GHz — 1 core — GotoBLAS2

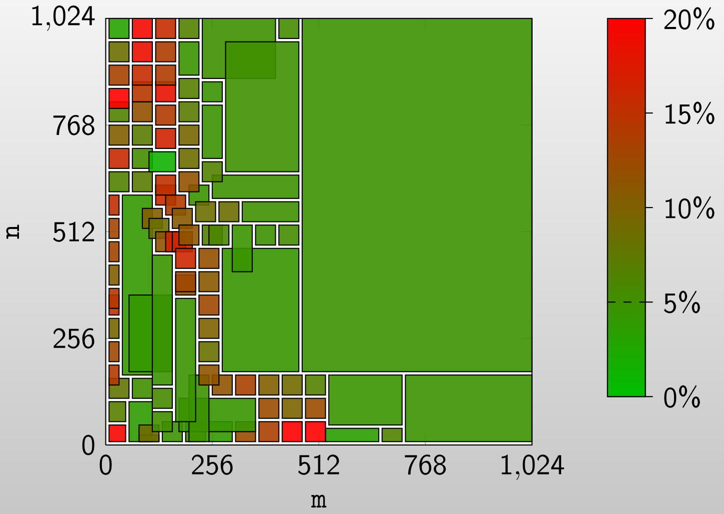
`dtrsm(L, L, N, N, m, n, .5, L, 2500, B, 2500)`



AMD Opteron 8356 (Barcelona) @ 2.3 GHz — 1 core — GotoBLAS2

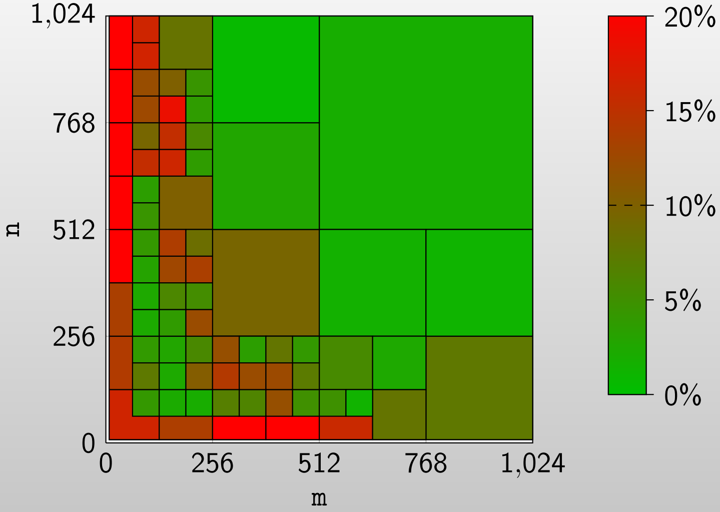
Smaller models

`dtrsm(L, L, N, N, m, n, .5, L, 2500, B, 2500)`



AMD Opteron 8356 (Barcelona) @ 2.3 GHz — 1 core — GotoBLAS2

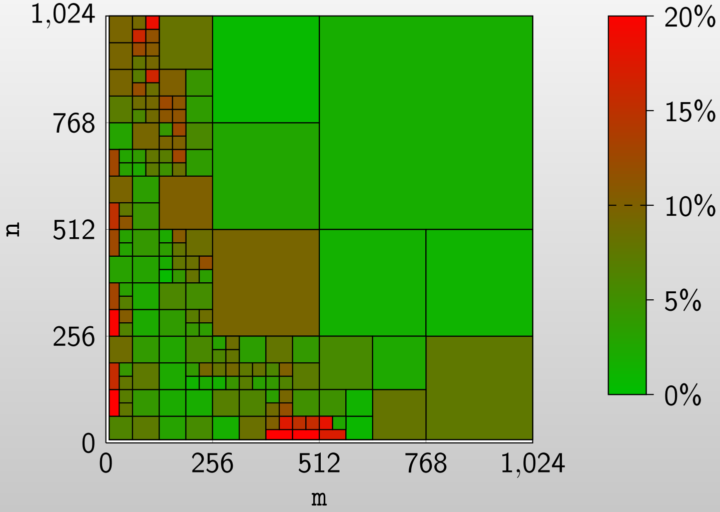
dtrsm(L, L, N, N, m, n, .5, *L*, 2500, *B*, 2500)



AMD Opteron 8356 (Barcelona) @ 2.3 GHz — 1 core — GotoBLAS2

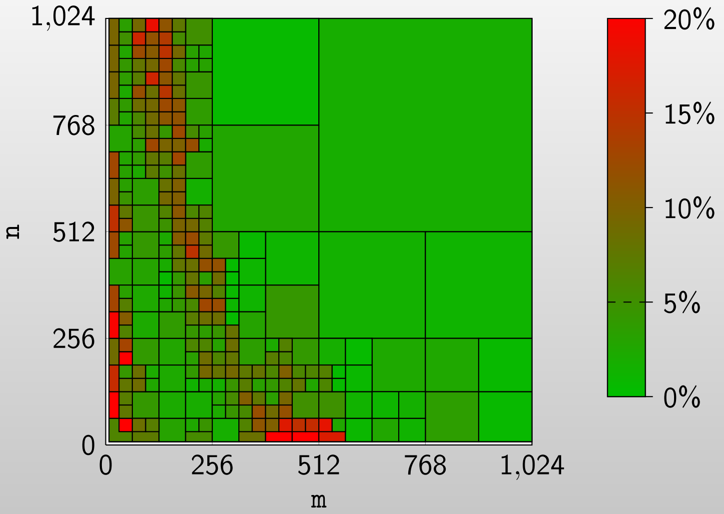
Smaller models

`dtrsm(L, L, N, N, m, n, .5, L, 2500, B, 2500)`



AMD Opteron 8356 (Barcelona) @ 2.3 GHz — 1 core — GotoBLAS2

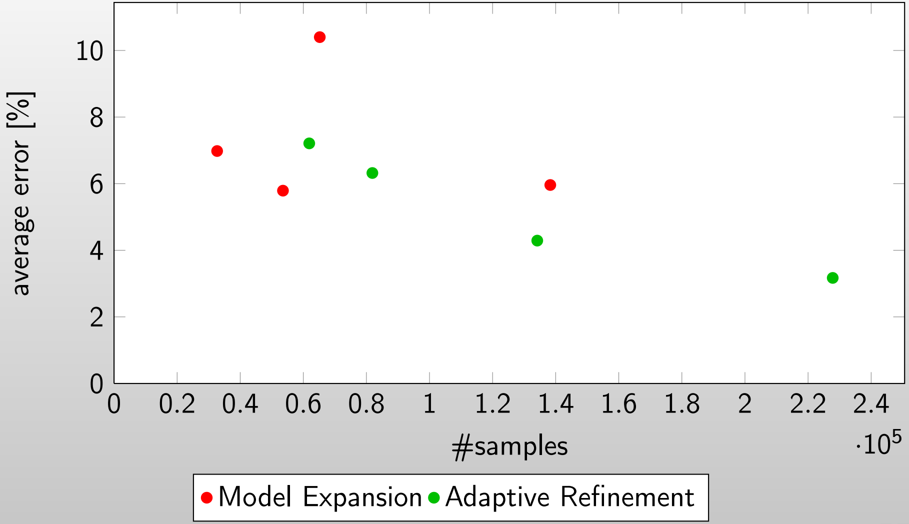
`dtrsm(L, L, N, N, m, n, .5, L, 2500, B, 2500)`



AMD Opteron 8356 (Barcelona) @ 2.3 GHz — 1 core — GotoBLAS2

RWTH AACHEN UNIVERSITY

`dtrsm(L, L, N, N, m, n, .5, L, 2500, B, 2500)`

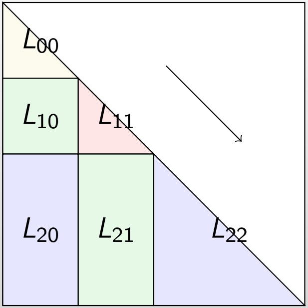


AMD Opteron 8356 (Barcelona) @ 2.3 GHz — 1 core — GotoBLAS2

Outline

1. Sampling
2. Modeling
3. Prediction and Ranking

Blocked algorithm revisited: $L \leftarrow L^{-1}$



Variant 1

$L_{10} \leftarrow L_{10} L_{00}$
$L_{10} \leftarrow -L_{11}^{-1} L_{10}$
$L_{11} \leftarrow L_{11}^{-1}$

Variant 2

$L_{21} \leftarrow L_{22}^{-1} L_{21}$
$L_{21} \leftarrow -L_{21} L_{11}^{-1}$
$L_{11} \leftarrow L_{11}^{-1}$

Variant 3

$L_{21} \leftarrow -L_{21} L_{11}^{-1}$
$L_{20} \leftarrow L_{20} + L_{21} L_{10}$
$L_{10} \leftarrow L_{11}^{-1} L_{10}$
$L_{11} \leftarrow L_{11}^{-1}$

Variant 4

$L_{21} \leftarrow -L_{22}^{-1} L_{21}$
$L_{20} \leftarrow L_{20} - L_{21} L_{10}$
$L_{10} \leftarrow L_{10} L_{00}$
$L_{11} \leftarrow L_{11}^{-1}$

C implementation: Variant 1

```c
int trinv1_(char* diag, int* n, double* A, int* ldA, int* bsize) {
  if (*n == 1) {
    if (diag[0] == 'N')
      *A = 1 / *A;
    return 0;
  }

  int ione = 1; double one = 1; double mone = -1;

  for (int p = 0; p < *n; p += *bsize) {
    int b = *bsize;
    if (p + b > *n)
      b = *n - p;
#define A00 (A)
#define A10 (A + p)
#define A11 (A + *ldA * p + p)

    dtrmm_("R", "L", "N", diag, &b, &p, &one, A00, ldA, A10, ldA);
    dtrsm_("L", "L", "N", diag, &b, &p, &mone, A11, ldA, A10, ldA);
    trinv1_(diag, &b, A11, ldA, &ione);
  }
  return 0;
}
```

Variant 1

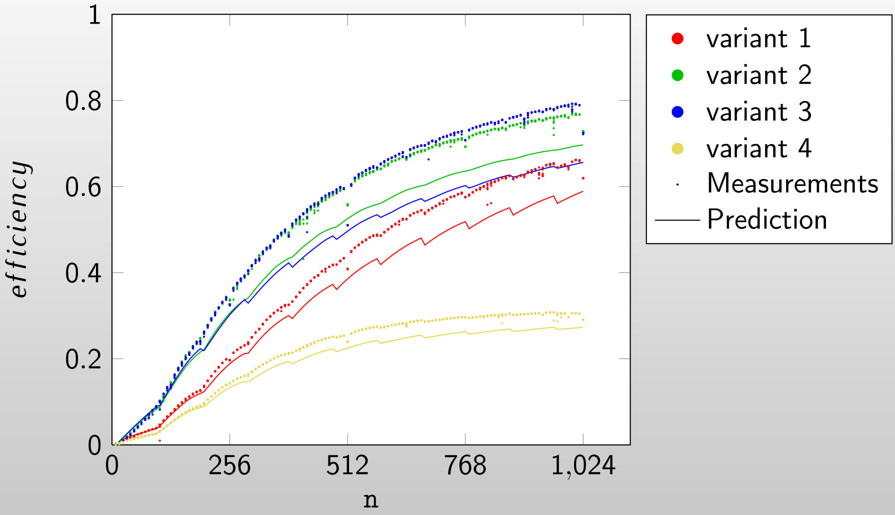
$L_{10} \leftarrow L_{10} L_{00}$
$L_{10} \leftarrow -L_{11}^{-1} L_{10}$
$L_{11} \leftarrow L_{11}^{-1}$

Prediction

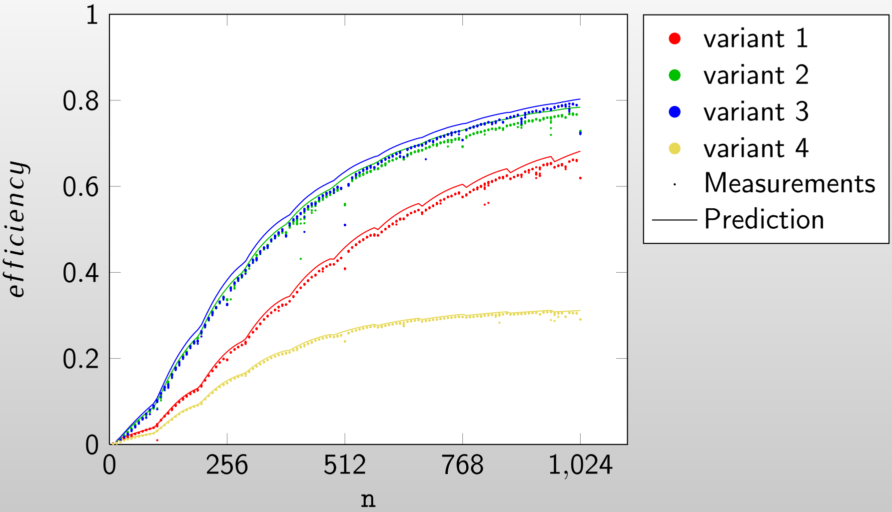
- Input: `trinv1(N, 300, A, 300, 100)`
- Compute routine invocations

update	routine invocation
$L_{10} \leftarrow L_{10} L_{00}$	$(\mathtt{dtrmm}, \mathtt{R}, \mathtt{L}, \mathtt{N}, \mathtt{N}, 100, 0, 1, \cdot, 300, \cdot, 300)$
$L_{10} \leftarrow -L_{11}^{-1} L_{10}$	$(\mathtt{dtrsm}, \mathtt{L}, \mathtt{L}, \mathtt{N}, \mathtt{N}, 100, 0, -1, \cdot, 300, \cdot, 300)$
$L_{11} \leftarrow L_{11}^{-1}$	$(\mathtt{trinv1}, \mathtt{N}, 100, \cdot, 300, 1)$
$L_{10} \leftarrow L_{10} L_{00}$	$(\mathtt{dtrmm}, \mathtt{R}, \mathtt{L}, \mathtt{N}, \mathtt{N}, 100, 100, 1, \cdot, 300, \cdot, 300)$
$L_{10} \leftarrow -L_{11}^{-1} L_{10}$	$(\mathtt{dtrsm}, \mathtt{L}, \mathtt{L}, \mathtt{N}, \mathtt{N}, 100, 100, -1, \cdot, 300, \cdot, 300)$
$L_{11} \leftarrow L_{11}^{-1}$	$(\mathtt{trinv1}, \mathtt{N}, 100, \cdot, 300, 1)$
$L_{10} \leftarrow L_{10} L_{00}$	$(\mathtt{dtrmm}, \mathtt{R}, \mathtt{L}, \mathtt{N}, \mathtt{N}, 100, 200, 1, \cdot, 300, \cdot, 300)$
$L_{10} \leftarrow -L_{11}^{-1} L_{10}$	$(\mathtt{dtrsm}, \mathtt{L}, \mathtt{L}, \mathtt{N}, \mathtt{N}, 100, 200, -1, \cdot, 300, \cdot, 300)$
$L_{11} \leftarrow L_{11}^{-1}$	$(\mathtt{trinv1}, \mathtt{N}, 100, \cdot, 300, 1)$

- Evaluate performance Models
- Accumulate *ticks*

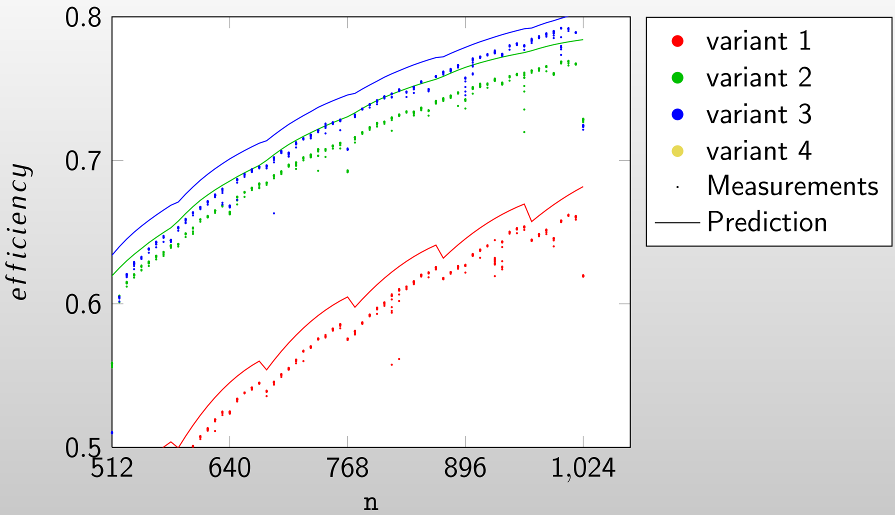


Intel Harpertown E5450 @ 2.99 GHz — 1 core — Intel MKL BLAS



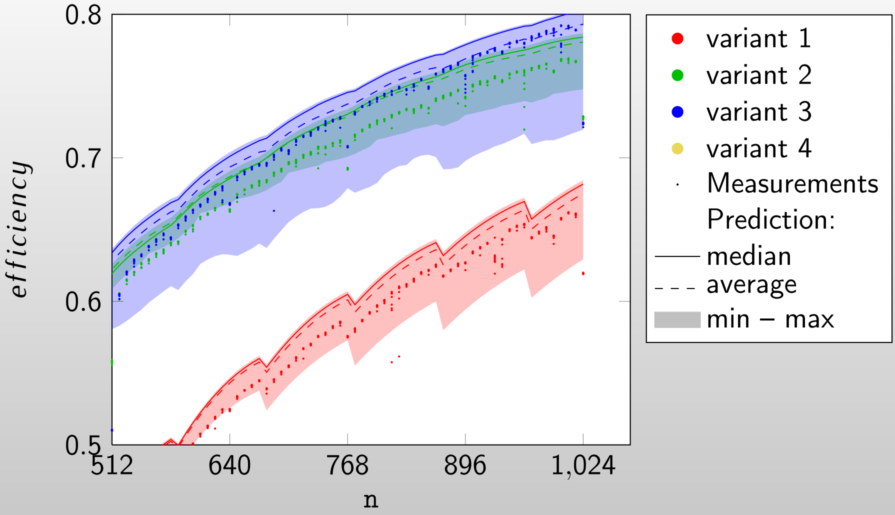
Intel Harpertown E5450 @ 2.99 GHz — 1 core — Intel MKL BLAS

Results: in-cache



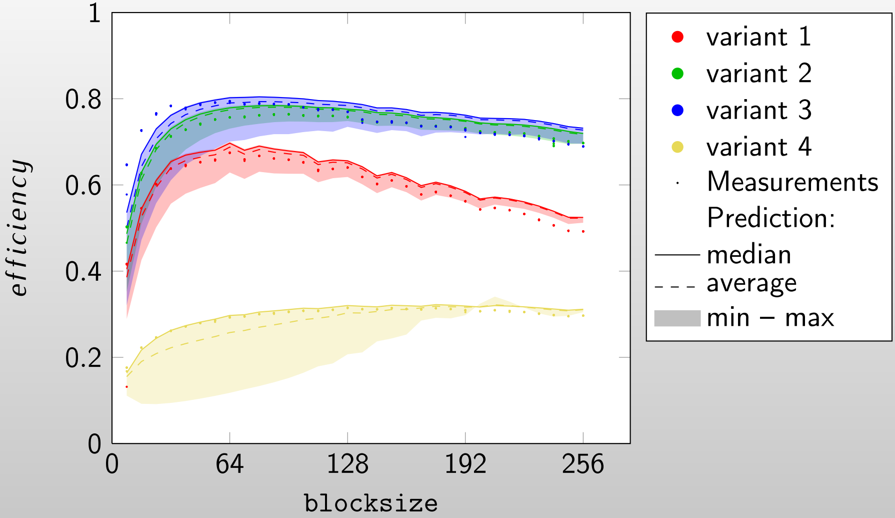
Intel Harpertown E5450 @ 2.99 GHz — 1 core — Intel MKL BLAS

Results: probabilistic prediction



Intel Harpertown E5450 @ 2.99 GHz — 1 core — Intel MKL BLAS

Optimal block-size



Intel Harpertown E5450 @ 2.99 GHz — 1 core — Intel MKL BLAS

Conclusion

1. Sampling
   - Performance measurement tool
   - Applicable to various DLA routines
2. Modeling
   - Automatic modeling system
   - Flexible and accurate
3. Prediction and Ranking
   - Accurate predictions
   - Correct ranking

Questions?

Funding from DFG is gratefully acknowledged

Deutsche
Forschungsgemeinschaft
DFG