Dense Linear Algebra

Basic Linear Algebra Subprograms (BLAS)

- $C := A \ B + C$
- $B := B \ A^T$
- $C := A \ A^T + C$
- $C := B \ A^T + A \ B^T + C$
- $B := A^{-1} \ B$
- ... 

Implementations:
80% – 95% of peak

Use BLAS
different algorithms:
20% – 90% of peak

peak (performance): all floating-point units fully utilized
$A := A^{-1}$

**Algorithm 1**

**Algorithm 2**

**Algorithm 3**

**Algorithm 4**

**Setup**

- size $n$
- parameter $b$
- # threads
- kernel lib
- CPU

**Goal:** Maximize performance for any given scenario quickly by making optimal choices automatically.

**Problem:**
- Without measurement even experts struggle.
- Measurement for all scenarios and choices takes days.

---

$n = 760, \ b = 128, \ 1 \ \text{thread}, \ \text{OpenBLAS, } \text{Haswell-EP E5-2680 v3}$
Algorithm 1
Algorithm 2
Algorithm 3
Algorithm 4

Setup
- size \( n \)
- parameter \( b \)
- \# threads
- kernel lib
- CPU

Algorithm 1
Algorithm 2
Algorithm 3
Algorithm 4

100 % of peak

performance [GFLOPs/s]

Performance = \#(elementary floating-point operations) / runtime

\( n = 760, b = 128, 1 \) thread, OpenBLAS, Haswell-EP E5-2680 v3
\[ A := A^{-1} \]

### Algorithm 1
- Size \( n \)
- Parameter \( b \)
- \# threads
- Kernel lib
- CPU

### Setup

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Setup</th>
<th>Performance [GFLOPs/s]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Algorithm 1</td>
<td>size ( n )</td>
<td>100</td>
</tr>
<tr>
<td>Algorithm 2</td>
<td>parameter ( b )</td>
<td>200</td>
</tr>
<tr>
<td>Algorithm 3</td>
<td># threads</td>
<td>300</td>
</tr>
<tr>
<td>Algorithm 4</td>
<td>kernel lib</td>
<td>400</td>
</tr>
<tr>
<td>Setup</td>
<td>CPU</td>
<td>500</td>
</tr>
</tbody>
</table>

**Goal:** Maximize performance for any given scenario quickly by making optimal choices automatically.

**Problem:** Without measurements even experts struggle.

**Problem:** Measurements for all scenarios and choices take days.

Performance = \#(elementary floating-point operations) / runtime

\( n = 4024, \ b = 128, \ 12 \) threads, OpenBLAS, Haswell-EP E5-2680 v3
Algorithm 1
Algorithm 2
Algorithm 3
Algorithm 4

Setup
Performance Prediction
accumulate
analyze

\[
A := A^{-1}
\]

dtrmm dtrmm dtrmm
dtrsm dtrsm dtrsm
dtrti2 dtrti2 dtrti2
dtrmm dtrmm dtrmm
dtrsm dtrsm dtrsm
dtrti2 dtrti2 dtrti2
dtrmm dtrmm dtrmm
dtrsm dtrsm dtrsm
dtrti2 dtrti2 dtrti2
Summary

Objective: Develop methods and tools that
- Maximize performance of dense linear algebra operations through
  - Algorithm selection
  - Parameter selection
- Find solutions fast

Constraints: The methods and tools should not
- Execute the full target operation
- Rely on low-level knowledge of kernels or hardware
- Be limited to specific operations
- Require much manual effort
Blocked Algorithms

- $A := A^{-1}$
- $Q \ R := A$

Tensor Contractions

- $C_{abc} := A_{ai} B_{ibc}$
- $C_{abcd} := A_{ija} B_{jbic}$

Performance Prediction

Performance models

Results

micro-benchmarks
**Blocked Algorithms**

- $A := A^{-1}$
- $QR := A$

**Tensor Contractions**

- $C_{abc} := A_{ai} B_{ibc}$
- $C_{abcd} := A_{ija} B_{jbic}$

Performance models

Micro-benchmarks

Performance Prediction

Results
Example: $A := A^{-1}$

Algorithm 1

Algorithm 2

Algorithm 3

Algorithm 4

Traverse $A$ along $\downarrow$:

$A_{21} := -A_{22}^{-1} A_{21}$

$A_{20} := A_{20} - A_{21} A_{10}$

$A_{10} := A_{10} A_{00}$

$A_{11} := A_{11}^{-1}$
Blocked Algorithms

Features
- Few kernels: \( \approx 2 - 5 \)
- Many invocations: \( \approx 10 - 100 \)
- Varying sizes: \( \approx 10 - 1000 \)
  - Traversal
  - Problem size
  - Block size

Algorithm 1

traverse \( A \) along \( \searrow \):

\[
\begin{align*}
A_{10} & := A_{10} A_{00} \\
A_{10} & := -A_{11}^{-1} A_{10} \\
A_{11} & := A_{11}^{-1}
\end{align*}
\]

Solution:

Performance Models
- Generate offline (slow)
- Evaluate online (fast)
Blocked Algorithms

- \( A := A^{-1} \)
- \( QR := A \)

Tensor Contractions

- \( C_{abc} := A_{ai} B_{ibc} \)
- \( C_{abcd} := A_{ija} B_{jbc} \)

Performance Prediction

Results

performance models

micro-benchmarks
Related Work: BLAS Performance Modeling

BLAS Tuning
- No measurements
- BLAS knowledge
- No predictions

Analytic Models
- Accurate
- BLAS and CPU knowledge
- Manual

BLAS-specific
- Piecewise polynomials
- Fairly accurate
- BLAS knowledge
- 1D

BLAS-independent
- No BLAS knowledge
- Multiple 1D
- Error: $\approx 10\%$
Performance Models

dtrsm(side, uplo, transA, diag, m, n, alpha, A, ldA, B, ldB)

<table>
<thead>
<tr>
<th>flags</th>
</tr>
</thead>
</table>

\[
\begin{align*}
B &:= A^{-1} B \\
B &:= A^{-T} B \\
B &:= B A^{-1} \\
B &:= B A^{-T}
\end{align*}
\]
Performance Models

dtrsm(side, uplo, transA, diag, \( m, n \), alpha, A, ldA, B, ldB)

\[
\begin{bmatrix}
B & := & A^{-1} & B \\
B & := & A^{-1} & B
\end{bmatrix}
\]

\[
\begin{bmatrix}
B & := & A^{-1} & B \\
B & := & A^{-1} & B
\end{bmatrix}
\]

\[
\begin{bmatrix}
B & \in & \mathbb{R}^{m \times n}
\end{bmatrix}
\]
Performance Models

dtrsm(side, uplo, transA, diag, m, n, alpha, A, ldA, B, ldB)

\[ B \triangleq \alpha A^{-1} B \]

Most frequent:

\[ B \triangleq -1 A^{-1} B \]
\[ B \triangleq 1 A^{-1} B \]
Performance Models

dtrsm(side, uplo, transA, diag, m, n, alpha, A, ldA, B, ldB)
dtrsm(side, uplo, transA, diag, m, n, alpha, A, \text{ldA}, B, \text{ldB})

\[ B := A^{-1} B \]
dtrsm(side, uplo, transA, diag, m, n, alpha, A, ldA, B, ldB)

Cases
- L, L, N, N, 1
- R, L, T, N, 1
- R, L, N, N, -1
- L, L, N, U, 1

Branching

≈ Polynomial

Negligible

Caching

Piecewise Polynomial

m

n

Runtime
Piecewise Polynomials (1D example)

\[ B := A^{-1} B \in \mathbb{R}^{n \times n} \]

\( (n^3 \text{ FLOPs}) \)

\[ \begin{array}{c}
\text{runtime [ms]} \\
\hline
0 & 5 & 10 \\
\hline
0 & 200 & 400 \\
\end{array} \]

\[ \begin{array}{c}
\text{error [%]} \\
\hline
0 & 10 \\
\hline
0 & 200 & 400 \\
\end{array} \]

Sandy Bridge-EP E5-2670, 1 thread, median of 100 repetitions

fit 1 polynomials
Piecewise Polynomials (1D example)

\[ B := A^{-1}B \in \mathbb{R}^{n \times n} \quad (n^3 \text{ FLOPs}) \]

Run time [ms]

Fit 2 polynomials

Sandy Bridge-EP E5-2670, 1 thread, median of 100 repetitions

average: 1.12 %
Adaptive Refinement Framework

(2D example)

\[ B := A^{-1}B \in \mathbb{R}^{m \times n} \]
Adaptive Refinement Framework

(2D example)

\[ B := A^{-1} B \in \mathbb{R}^{m \times n} \]

- Sample
Adaptive Refinement Framework

(2D example)

\[ B := A^{-1} B \in \mathbb{R}^{m \times n} \]

- Sample
- Fit polynomial
Adaptive Refinement Framework

(2D example)

\[ B := A^{-1} B \in \mathbb{R}^{m \times n} \]

Sample

Fit polynomial

Compute error

error: 4.21 %

\( m \)

\( n \)

Sandy Bridge-EP E5-2670, 1 thread, OpenBLAS
Adaptive Refinement Framework

(2D example)

\[ B := A^{-1} B \in \mathbb{R}^{m \times n} \]

\[ n \]

\[ \begin{array}{c}
24 \\
536
\end{array} \]

\[ \begin{array}{c}
24 \\
2088
\end{array} \]

\[ \begin{array}{c}
2088 \\
4152
\end{array} \]

\[ \begin{array}{c}
24 \\
536
\end{array} \]

\[ \begin{array}{c}
0.59 \% \\
3.36 \%
\end{array} \]

Repeat:

- Sample
- Fit polynomial
- Compute error
- Refine

Sandy Bridge-EP E5-2670, 1 thread, OpenBLAS
Adaptive Refinement Framework

(2D example)

\[ B := A^{-1} B \in \mathbb{R}^{m \times n} \]

<table>
<thead>
<tr>
<th>( n )</th>
<th>24</th>
<th>1056</th>
<th>2088</th>
<th>4152</th>
</tr>
</thead>
<tbody>
<tr>
<td>( m \times n )</td>
<td>536</td>
<td>1.13%</td>
<td>0.83%</td>
<td>0.59%</td>
</tr>
</tbody>
</table>

Repeat:

- Sample
- Fit polynomial
- Compute error
- Refine where:
  - error \( > \) threshold\(_e\)
  - size \( > \) threshold\(_s\)
Adaptive Refinement Framework

(2D example)

\[ B := A^{-1} \quad B \in \mathbb{R}^{m \times n} \]

Repeat:

- Sample
- Fit polynomial
- Compute error
- Refine where:
  - error > threshold\(_e\)
  - size > threshold\(_s\)
Adaptive Refinement Framework

(2D example)

\[ B := A^{-1} B \in \mathbb{R}^{m \times n} \]

Repeat:
- Sample
- Fit polynomial
- Compute error
- Refine where:
  - error > threshold_e
  - size > threshold_s

8 tuning parameters

Sandy Bridge-EP E5-2670, 1 thread, OpenBLAS
Summary: Performance Models

I developed a performance modeling framework that

- Builds separate models for each
  - Kernel
  - Kernel implementation (i.e., BLAS library)
  - Processor architecture
  - Thread count
- Builds separate sub-models for different flag and scalar values
- Builds piecewise multivariate polynomials in terms of operand sizes
- Adaptively refines the polynomial pieces to get accurate but cheap models
- Offers statistical runtime estimates:
  - minimum, median, average, standard deviation, ...
Blocked Algorithms

- $A := A^{-1}$
- $QR := A$

Tensor Contractions

- $C_{abc} := A_{ai} B_{ibc}$
- $C_{abcd} := A_{ija} B_{jibc}$

Performance Prediction

- Performance models

Results

- micro-benchmarks
Performance Prediction

Blocked Algorithm
- Problem size $n$
- Block size $b$

analyze

dtrmm  dtrmm  dtrmm
dtrsm  dtrsm  dtrsm
dtrti2 dtrti2 dtrti2
dtrmm  dtrmm  dtrmm
dtrsm  dtrsm  dtrsm
dtrti2 dtrti2 dtrti2
dtrmm  dtrmm  dtrmm
dtrsm  dtrsm  dtrsm
dtrti2 dtrti2 dtrti2

Prediction
- Runtime
- Performance

accumulate

Models
- Processor
- #threads
- BLAS

<table>
<thead>
<tr>
<th>Processor</th>
<th>#threads</th>
<th>BLAS</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.00 µs</td>
<td>509 µs</td>
<td>1.84 ms</td>
</tr>
<tr>
<td>0.00 µs</td>
<td>315 µs</td>
<td>628 µs</td>
</tr>
<tr>
<td>113 µs</td>
<td>113 µs</td>
<td>113 µs</td>
</tr>
<tr>
<td>71.8 µs</td>
<td>872 µs</td>
<td>2.45 ms</td>
</tr>
<tr>
<td>108 µs</td>
<td>421 µs</td>
<td>730 µs</td>
</tr>
<tr>
<td>113 µs</td>
<td>113 µs</td>
<td>113 µs</td>
</tr>
<tr>
<td>252 µs</td>
<td>1.30 ms</td>
<td>3.16 ms</td>
</tr>
<tr>
<td>213 µs</td>
<td>524 µs</td>
<td>831 µs</td>
</tr>
<tr>
<td>113 µs</td>
<td>113 µs</td>
<td>113 µs</td>
</tr>
</tbody>
</table>
Prediction: \( L L^T := A \)

(Algorithm 3 of 3)

\[ \begin{align*}
\text{performance prediction [GFLOPs/s]} \\
\text{error [%]} \\
\text{problem size } n
\end{align*} \]

\( b = 128, \text{ Sandy Bridge-EP E5-2670, OpenBLAS, median } \)
Blocked Algorithms

- \( A := A^{-1} \)
- \( Q \quad R := A \)

Tensor Contractions

- \( C_{abc} := A_{ai} B_{ibc} \)
- \( C_{abcd} := A_{ija} B_{jbic} \)

Performance models

Performance Prediction

Results
Algorithm Selection: $A := A^{-1}$

Prediction and Measurements graphs showing performance [GFLOPs/s] vs. problem size $n$.

$b = 128$, Haswell-EP E5-2680 v3, 1 thread, OpenBLAS.
Algorithm Selection: \[ A := A^{-1} \]

**Prediction**

<table>
<thead>
<tr>
<th>problem size $n$</th>
<th>performance [GFLOPs/s]</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1000</td>
<td>100</td>
</tr>
<tr>
<td>2000</td>
<td>200</td>
</tr>
<tr>
<td>3000</td>
<td>300</td>
</tr>
<tr>
<td>4000</td>
<td>400</td>
</tr>
</tbody>
</table>

**Measurements**

<table>
<thead>
<tr>
<th>problem size $n$</th>
<th>performance [GFLOPs/s]</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1000</td>
<td>100</td>
</tr>
<tr>
<td>2000</td>
<td>200</td>
</tr>
<tr>
<td>3000</td>
<td>300</td>
</tr>
<tr>
<td>4000</td>
<td>400</td>
</tr>
</tbody>
</table>

$b = 128$, Haswell-EP E5-2680 v3, **12 threads**, OpenBLAS
Block Size Selection

\[
A := L^{-1} A L^{-T}
\]

\[
Q R := A
\]

- LAPACK
- optimal
- predicted optimum

Haswell-EP E5-2680 v3, 1 thread, OpenBLAS
Block Size Selection

\[ A := L^{-1} A L^{-T} \]

\[ Q R := A \]

Performance [GFLOPs/s] vs. matrix size \( n \):

Haswell-EP E5-2680 v3, 12 threads, OpenBLAS
Summary: Model-Based Performance Predictions

I introduced runtime and performance predictions that
- Are obtained by summing runtime estimates for its kernel invocations
- Are virtually instantaneous
- Are accurate for different
  - Operations
  - Algorithms
  - Problem sizes
  - Block sizes
  - Thread counts
  - Data types
- Correctly identify the fastest algorithm for an operation
- Determine block sizes that yield near-optimal performance
Want to know more?

Performance Measurements

- Dissertation: 2.2 Measurements and Experiments: ELAPS

Modeling and Prediction

- Dissertation: 3 Performance Modeling
- Dissertation: 4 Model-Based Predictions for Blocked Algorithms

Caching

- Dissertation: 5 Cache Modeling and Prediction

Recursive Algorithms

- *Algorithm 979: Recursive Algorithms for Dense Linear Algebra—The ReLAPACK Collection.* ACM TOMS.
Blocked Algorithms

- $A := A^{-1}$
- $QR := A$

Tensor Contractions

- $C_{abc} := A_{ai} B_{ibc}$
- $C_{abcd} := A_{ija} B_{jbc}$

performance models

Performance Prediction

Results

micro-benchmarks
Tensor Contractions?

\[ C := A_i B_i \quad C := \sum_i A[i] B[i] \]

\[ C_a := A_{ai} B_i \quad \forall a. C[a] := \sum_i A[a, i] B[i] \]

\[ C_{ab} := A_{ai} B_{ib} \quad \forall a, b. C[a, b] := \sum_i A[a, i] B[i, b] \]

\[ C_{abc} := A_{ai} B_{ibc} \quad \forall a, b, c. C[a, b, c] := \sum_i A[a, i] B[i, b, c] \]

\[ C_{abc} := A_{ija} B_{j bic} \quad \forall a, b, c. C[a, b, c] := \sum_{i, j} A[i, j, a] B[j, b, i, c] \]

free indices \quad \text{contracted indices}
$C_{ab} := A_{ai} B_{ib}$

Algorithm

for $b = 1:b$

$C[:,b] = A[:,,:] B[:,b]$

$C_{abc} := A_{ai} B_{ibc}$

Algorithm

for $b = 1:b$

$C[:,b,:] = A[:,,:] B[:,b,:]$
Blocked Algorithms

Algorithm

\[ C_{\alpha\beta\gamma} := A_{\alpha i} B_{\beta\gamma} \]

for \( b = 1 : b \)

\[ C[:,b,:) = A[:,,:] B[:,b,:]\]

Features

- One kernel
- Many invocations: \( \approx 10 - 1000 \)
- Fixed size
- Skewed sizes

Solution:
Micro-Benchmarks

- Execute fraction of each algorithm online
Blocked Algorithms

- \( A := A^{-1} \)
- \( QR := A \)

Tensor Contractions

- \( C_{abc} := A_{ai} B_{ibc} \)
- \( C_{abcd} := A_{ija} B_{jbc} \)

performance models

Performance Prediction

micro-benchmarks

Results
Related Work: Tensor Contractions

Tensor Frameworks
- Fast for “square” contractions
- One algorithm per contraction

Algorithm Generation
- Our starting point

BLAS-level Algorithms
- Ongoing PhD project
- Benchmarks and ranking
- BLAS-like performance
\[ C_{abc} := A_{ai} B_{ibc} \]

with \( i = 8, a = b = c = 8, \ldots, 1000 \)

<table>
<thead>
<tr>
<th>Kernel</th>
<th>Algorithms</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \alpha ) ( += ) ( x^T y )</td>
<td>6</td>
</tr>
<tr>
<td>( y ) ( += ) ( \alpha x )</td>
<td>18</td>
</tr>
<tr>
<td>( y ) ( += ) ( A x )</td>
<td>4</td>
</tr>
<tr>
<td>( A ) ( += ) ( x y^T )</td>
<td>6</td>
</tr>
<tr>
<td>( C ) ( += ) ( A B )</td>
<td>2</td>
</tr>
</tbody>
</table>

Harpertown E5450, 1 thread, OpenBLAS, median of 10 repetitions
Approach: Micro-Benchmarks

Algorithm

for a = 1:a
  for b = 1:b
    \( C[a,b,:]=A[a,:]\ B[(:,b,:)] \)

estimate

Micro-Benchmark

\( C[a,b,:]=A[a,:]\ B[(:,b,:)] \)

time \( 10\times \)

\( a \cdot b \cdot (\text{median time}) \)
Prediction

\[ C_{abc} := A_{ai}B_{ibc}, \quad i = 8, \text{ Harpertown E5450, 1 thread, OpenBLAS, median of 10 repetitions} \]
Problem: Overestimation

Cause: Micro-benchmarks use cached “warm” data

Solution: Emulate caching

\[ C_{abc} := A_{ai}B_{ibc}, \ i = 8, \ \text{Harpertown E5450, 1 thread, OpenBLAS, median of 10 repetitions} \]
Cache Emulation: Access Distance

Algorithm

for c = 1:
c
   for a = 1:
a
      $C[a,:,c] = A[a,:] \cdot B[:,:,c]$

Access Distance $d(M) = \text{size(all data used since the last access to } M)$

- $B[:,:,c]$ doesn’t vary in for a
  $d(B[:,:,c]) = 0$ doubles

- $A[a,:]$ varies in for a; doesn’t vary in for c
  $d(A[a,:]) = \text{size(all operands in for a)}$
  $= \text{size}(C[:,:,c]) + \text{size}(A[:,:,c]) + \text{size}(B[:,:,c])$
  $= a \cdot b + a \cdot i + i \cdot b$ doubles

- $C[a,:,c]$ varies in for a; varies in for c
  $d(C[a,:,c]) = \text{size(all operands in for c)}$
  $= \text{size}(C[:,:,c]) + \text{size}(A[:,:,c]) + \text{size}(B[:,:,c])$
  $= a \cdot b \cdot c + a \cdot i + i \cdot b \cdot c$ doubles
Cache Emulation: Micro-Benchmark

Access Distances

<table>
<thead>
<tr>
<th>Expression</th>
<th>Distance</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>(d(B[:, :, c]))</td>
<td>(0)</td>
<td>(0) doubles</td>
</tr>
<tr>
<td>(d(A[a, :]))</td>
<td>(a \cdot b + a \cdot i + i \cdot b)</td>
<td>(166,400) doubles</td>
</tr>
<tr>
<td>(d(C[a, :, c]))</td>
<td>(a \cdot b \cdot c + a \cdot i + i \cdot b \cdot c)</td>
<td>(65,283,200) doubles</td>
</tr>
</tbody>
</table>

Sizes: \(a = b = c = 400\), \(i = 8\)

Algorithm

```plaintext
for c = 1:c
    for a = 1:a
        C[a, :, c] = A[a, :] \cdot B[:, :, c]
```

Micro-Benchmark

- Flush \(816\,632\) doubles
- Load \(A[a, :]\)
- Flush \(163\,200\) doubles
- Load \(B[:, :, c]\)
- \(C[a, :, c] = A[a, :] \cdot B[:, :, c]\)

Limit at \(\frac{5}{4}\) \(\text{size(cache)}\) = \(\frac{5}{4} \cdot 6\,\text{MB} = 983,040\) doubles

Harpertown E5450
Emulate Caching

\[ C_{abc} := A_{ai} B_{ibc}, \quad i = 8, \quad \text{Harpertown E5450, 1 thread, OpenBLAS, median of 10 repetitions} \]
Emulate Caching

\[ C_{abc} := A_{ai} B_{ibc}, \ i = 8, \ \text{Harpertown E5450, 1 thread, OpenBLAS, median of 10 repetitions} \]
Emulate Prefetching

\[ C_{abc} := A_{ai}B_{ibc}, \quad i = 8, \quad \text{Harpertown E5450, 1 thread, OpenBLAS, median of 10 repetitions} \]
Correct Prefetching

\[ C_{abc} := A_{ai}B_{ibc}, \quad i = 8, \text{ Harpertown E5450, 1 thread, OpenBLAS, median of 10 repetitions} \]
First Iterations

\[ C_{abc} := A_{ai}B_{ibc}, \quad i = 8, \text{ Harpertown E5450, 1 thread, OpenBLAS, median of 10 repetitions} \]
Blocked Algorithms

- $A := A^{-1}$
- $Q R := A$

Tensor Contractions

- $C_{abc} := A_{ai} B_{ibc}$
- $C_{abcd} := A_{ija} B_{jibc}$

Performance Prediction

Results

performance models

micro-benchmarks
\[ C_{abc} := A_{ija}B_{jbic} \]

\[ i = j = 8, \ a = b = c = 8, \ldots, 1000 \]

<table>
<thead>
<tr>
<th>Kernel</th>
<th>Algorithms</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \alpha ) ( += ) ( x^T y )</td>
<td>48</td>
</tr>
<tr>
<td>( y ) ( += ) ( \alpha x )</td>
<td>72</td>
</tr>
<tr>
<td>( y ) ( += ) ( A x )</td>
<td>36</td>
</tr>
<tr>
<td>( A ) ( += ) ( x y^T )</td>
<td>12</td>
</tr>
<tr>
<td>( C ) ( += ) ( A B )</td>
<td>8</td>
</tr>
</tbody>
</table>

Ivy Bridge-EP E5-2680 v2, 1 thread, OpenBLAS, median of 10 repetitions

Elmar Peise
\[ C_{abc} := A_{ija} B_{jbic} \quad \text{— Prediction} \]

\[ i = j = 8, \text{Ivy Bridge-EP E5-2680 v2, 1 thread, OpenBLAS, median of 10 repetitions} \]
$C_{abc} := A_{ija}B_{jbic}$ — Prediction

Prediction

Measurements

\[ \text{performance [GFLOPs/s]} \]

\[ \text{tensor size } a = b = c \]

\[ \text{performance [GFLOPs/s]} \]

\[ \text{tensor size } a = b = c \]

\[ i = j = 32, \text{ Ivy Bridge-EP E5-2680 v2, 10 threads, OpenBLAS, median of 10 repetitions} \]
I developed micro-benchmarks that

- Predict the performance of BLAS-based tensor contractions
- Execute only a fraction of a algorithm’s computation
- Are fully automated
- Yield good rankings for different contractions and setups

Want to know more?

- Dissertation: 6 Micro-Benchmarks for Tensor Contractions
Blocked Algorithms

- $A := A^{-1}$
- $Q R := A$

Tensor Contractions

- $C_{abc} := A_{ai} B_{ibc}$
- $C_{abcd} := A_{ija} B_{jbic}$

Performance Models

Performance Prediction

Results

Micro-benchmarks
Contributions

- Accurate automatically generated performance models
- Execution-less optimization of blocked algorithms
- Cache-aware micro-benchmarks
- Fast selection of tensor contraction algorithms


This talk: http://hpac.rwth-aachen.de/~peise/downloads/defense.pdf

Publications:  http://hpac.rwth-aachen.de/publications/author/Peise

Talks:  http://hpac.rwth-aachen.de/talks/author/Peise

Software:  http://github.com/elmar-peise

Contact:  peise@aices.rwth-aachen.de