Tensor Contractions?

\[
C := A_i B_j
\]

\[
C := \sum_i A[i] B[i]
\]

\[
C_a := A_{ai} B_i
\]

\[
\forall a. C[a] := \sum_i A[a,i] B[i]
\]

\[
C_{ab} := A_{ai} B_{ib}
\]

\[
\forall a, b. C[a,b] := \sum_i A[a,i] B[i,b]
\]

\[
C_{abc} := A_{ai} B_{ibc}
\]

\[
\forall a, b, c. C[a,b,c] := \sum_i A[a,i] B[i,b,c]
\]

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

\[
\forall a, b, c. C[a,b,c] := \sum_{i,j} A[i,j,a] B[j,b,i,c]
\]

free indices

contracted indices

Elmar Peise
Use BLAS!

\[ C_{ab} := A_{ai} B_{ib} \quad \text{for } b = 1 : b \]
\[ C[:,b] = A[:,,:) B[:,b] \]

\[ C_{abc} := A_{ai} B_{ibc} \quad \text{for } b = 1 : b \]
\[ C[:,b,:) = A[:,,:] B[:,b,:] \]
\[ C_{abc} := A_{aij} B_{jbic} \]

Total: 176 Algorithms!

Goals

- Generate algorithms
- Predict their performance
Outline

1 Algorithm Generation

2 Performance Prediction
   1. Repeated Execution
   2. Cache Setup
   3. Prefetching
   4. Prefetching Failures
   5. First Iterations

3 Results
   \[ C_a := A_{iaj} B_{ji} \]
   \[ C_{abc} := A_{aij} B_{jbic} \]
   Multithreading
   Efficiency
Algorithm Generation

- Select kernel
- Match kernel indices to tensor indices
- Cast remaining tensor indices as for-loops
- Assemble algorithm (AST, C-code)

Example

\[ C_\alpha := A_\alpha L \rightarrow C_{abc} = A_{ai} B_{ibc} \]

\[ \begin{align*}
\text{for } a = 1:a \\
\text{for } b = 1:b
\end{align*} \]

\[ C[a,b,:] = A[a,:] \ B[:,b,:] \]
Algorithms for $C_{abc} := A_{ai} B_{ibc}$

- **BLAS-1**
  - 6 dot-based: $(C := A_l B_l)$
    - $abc$-dot $acb$-dot $bac$-dot $bca$-dot $cab$-dot $cba$-dot
  - 18 axpy-based: $(C_\alpha := AB_\alpha)$
    - $ibc$-axpy $icb$-axpy $bic$-axpy $bci$-axpy $cib$-axpy $cbi$-axpy $iac$-axpy $ica$-axpy $aic$-axpy
    - $aci$-axpy $cia$-axpy $cai$-axpy $iab$-axpy $iba$-axpy $aib$-axpy $abi$-axpy $bia$-axpy $bai$-axpy

- **BLAS-2**
  - 6 gemv-based: $(C_\alpha := A_\alpha B_l)$
    - $bc$-gemv $cb$-gemv $ac$-gemv $ca$-gemv $ab$-gemv $ba$-gemv
  - 4 ger-based: $(C_{\alpha\beta} := A_\alpha B_\beta)$
    - $ic$-ger $ci$-ger $ib$-ger $bi$-ger

- **BLAS-3**
  - 2 gemm-based: $(C_{\alpha\beta} := A_\alpha B_{l\beta})$
    - $c$-gemm $b$-gemm
Outline

1 Algorithm Generation

2 Performance Prediction
   1. Repeated Execution
   2. Cache Setup
   3. Prefetching
   4. Prefetching Failures
   5. First Iterations

3 Results

\[ C_a := A_{iaj} B_{ji} \]
\[ C_{abc} := A_{aij} B_{jbic} \]

Multithreading
Efficiency
What are we predicting?

- $bc$-gemv
- $cb$-gemv
- $ac$-gemv
- $ca$-gemv
- $ab$-gemv
- $ba$-gemv
- $ci$-ger
- $ic$-ger
- $bi$-ger
- $ib$-ger
- $c$-gemm
- $b$-gemm

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

- Intel Penryn E5450 (Harpertown)
- Single-threaded OPENBLAS
Micro-Benchmarks

\[ C_{abc} := A_{ai}B_{ibc} \]

\[
\begin{align*}
\text{for } b &= 1:b \\
\text{for } a &= 1:a \\
C[a,b,:] &= A[a,:] B[:,b,:] 
\end{align*}
\]

estimate

\[ a \cdot b \cdot (\text{median time}) \]

Micro-benchmark

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

time 10×
1. Repeated Execution

Problem: general overestimation

measurements

predictions

Elmar Peise
1. Repeated Execution

Problem: general overestimation
2. Caching!

**Problem:** general overestimation

**Cause:** Cache locality not accounted for (micro-benchmark works in-cache)

**Solution Approach**

Recreate cache precondition within micro-benchmark

**Assumption:** Fully associative Least Recently Used (LRU) cache replacement policy

⇒ Cache state is defined by the order of memory accesses.
Access Distance

\[ C_{abc} := A_{ai} B_{ibc} \quad \text{ca-gemv} \]

\[
\begin{align*}
\text{for} \ c &= 1 : c \\
\text{for} \ a &= 1 : a \\
C[a,:,:c] &= A[a,:] B[:,:,c]
\end{align*}
\]

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

\[ C_{abc} := A_{ai} B_{ibc} \]

\[ \text{for } c = 1 : c \]
\[ \text{for } a = 1 : a \]
\[ C[a,:,c] = A[a,:) B[:,:,c] \]

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

\[ B[:, :, c] \text{ doesn’t vary in for } a \]
\[ d(B[:, :, c]) = 0 \text{ doubles} \]

\[ A[a,:] \]

\[ C[a,:,c] \]
### Access Distance

**\( C_{abc} := A_{ai} B_{ibc} \) - ca-gemv**

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

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

- \( B[:,:,c] \) doesn’t vary in `for a`
  - \( d(B[:,:,c]) = 0 \) doubles
- \( A[a,:] \) varies in `for a`; doesn’t vary in `for c`
  - \( d(A[a,:]) = \text{size(\textit{all operands in for a})} \)
    - \( = \text{size}(C[:,:,c]) + \text{size}(A[:,:]) + \text{size}(B[:,:,c]) \)
    - \( = a \cdot b + a \cdot i + i \cdot b \)
- \( C[a,:,c] \)
Access Distance

\[ C_{abc} := A_{ai} B_{ibc} \]

\[
\begin{align*}
\text{for } c &= 1 : c \\
\text{for } a &= 1 : a \\
C[a,:,c] &= A[a,:] B[:, :, c]
\end{align*}
\]

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

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

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

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

Access Distances

\[
\begin{align*}
\left( B[:, :, c] \right) &= 0 = 0 \text{ doubles} \\
\left( A[a, :] \right) &= a \cdot b + a \cdot i + i \cdot b = 166,400 \text{ doubles} \\
\left( C[a, :, c] \right) &= a \cdot b \cdot c + a \cdot i + i \cdot b \cdot c = 65,283,200 \text{ doubles}
\end{align*}
\]

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

**Micro-benchmark**

\[
\begin{align*}
C_{abc} := A_{ai} B_{ibc} \\
\text{for } c = 1 : c \\
\quad \text{for } a = 1 : a \\
\quad \quad C[a, :, c] = A[a, :] B[:, :, c]
\end{align*}
\]

Limit at \( \frac{5}{4} \text{size(cache)} = \frac{5}{4} \cdot 6\text{MB} = 983,040 \text{ doubles} \)
2. Estimates with Cache Setup

\[
\begin{align*}
bc &- \text{gemv} & cb &- \text{gemv} & ac &- \text{gemv} & ca &- \text{gemv} & ab &- \text{gemv} & ba &- \text{gemv} \\
ci &- \text{ger} & ic &- \text{ger} & bi &- \text{ger} & ib &- \text{ger} & c &- \text{gemm} & b &- \text{gemm}
\end{align*}
\]

Problem: underestimation

\[
\begin{align*}
a = b = c & \quad (i = 8) \\
\text{flops/cycle}
\end{align*}
\]
2. Estimates with Cache Setup

\[ a = b = c \quad (i = 8) \]

**Problem:** underestimation
3. Prefetching!

**Problem:** selective underestimation

**Cause:** Prefetching not accounted for

**Solution Approach**

Mimick prefetching

Prefetching:

- access
- prefetch
Mimicking the Prefetching

\[ C_{abc} := A_{ai} B_{ibc} \]

\[
\text{for } b = 1:b \\
\quad \text{for } i = 1:i \\
\quad \quad C[:,b,:) = A[:,i] B[i,b,:] \]

Micro-benchmark

- touch \( A[:,i] \)
- flush 5,992
- touch \( A[:,8,i] \)
- touch \( B[i,b,:] \)
- touch \( C[:,b,:] \)
- \( C[:,b,:] = A[:,i] B[i,b,:] \)
3. Estimates with Prefetching

![Graphs showing flops/cycle vs. iteration count for different operations with different parameters.](image)

**Problem:** Some incorrect prefetching
3. Estimates with Prefetching

### Diagram

- **bc-gemv**
- **cb-gemv**
- **ac-gemv**
- **ca-gemv**
- **ab-gemv**
- **ba-gemv**
- **ci-ger**
- **ic-ger**
- **bi-ger**
- **ib-ger**
- **c-gemm**
- **b-gemm**

**Problem:** some incorrect prefetching
4. Prefetching Failures

**Problem:** selective prefetching failure

**Cause:** No Prefetching along 1st dimension across cache-lines. (every 8th iteration is not prefetched)

**Solution Approach**
Separate micro-benchmarks with and without prefetching

| Micro-benchmark (pre)                      | Micro-benchmark (no pre)                      |
|-------------------------------------------|-----------------------------------------------|
| touch $A[:,i]$                            | flush 816,240                                |
| flush 5,992                               | touch $A[:,i]$                               |
| touch $A[:,8,i]$                          | flush 5,992                                  |
| touch $B[i,b,:]$                          | touch $A[:,8,i]$                             |
| touch $C[:,b,:]$                          | touch $C[:,b,:]$                             |
| $C[:,b,:] = A[:,i] B[i,b,:]$              | $C[:,b,:] = A[:,i] B[i,b,:]$                 |

\[\text{time } 10\times\]

\[\text{estimate} = \frac{1}{8}(7\text{median} + 1\text{median})\]
4. Estimates with Prefetching Failures

- $bc$-gemv
- $cb$-gemv
- $ac$-gemv
- $ca$-gemv
- $ab$-gemv
- $ba$-gemv
- $ci$-ger
- $ic$-ger
- $bi$-ger
- $ib$-ger
- $c$-gemm
- $b$-gemm

Problem: selective overestimation

Elmar Peise
4. Estimates with Prefetching Failures

**Problem**: selective overestimation
5. Small Loops’ First Iterations

**Problem:** selective overestimation

**Cause:** Innermost loop dimension too small (first iteration of innermost loop differs)
5. Small Loops’ First Iterations

**Problem**: selective overestimation

**Cause**: Innermost loop dimension too small (first iteration of innermost loop differs)

\[ C_{abc} := A_{ai} B_{ibc} \]

```
for b = 1:b
    for i = 1:i
        C[:,b,:) = A[:,i] * B[i,b,:]
```

**Solution Approach**

Separate micro-benchmarks for first iteration of small loops
First Iteration Benchmark

- **Access Distance:**

\[
C_{abc} := A_{ai}B_{ibc}
\]

\[
\begin{align*}
&\text{for } b = 1:b \\
&\text{for } i = 1:i \\
&\quad C[::,b,:] = A[::,i] \ B[i,b,:]
\end{align*}
\]

Find last access to \(A[::,i], B[i,b,:),\) and \(C[::,b,:]\) within for \(i\)

Find last access to \(A[::,:], B[:,b,:],\) and \(C[:,b,:]\) within for \(c\)

- **Prefetching:**
5. Final Estimates

\begin{align*}
\text{flops/cycle} = \frac{\text{number of floating point operations}}{\text{number of cycles}}
\end{align*}

\begin{align*}
a = b = c \quad (i = 8)
\end{align*}
Outline

1 Algorithm Generation

2 Performance Prediction
   1. Repeated Execution
   2. Cache Setup
   3. Prefetching
   4. Prefetching Failures
   5. First Iterations

3 Results

\[ C_a := A_{iaj} B_{ji} \]
\[ C_{abc} := A_{aij} B_{jbic} \]
Multithreading
Efficiency

Elmar Peise
\[ C_a := A_{iaj} B_{ji} \quad \text{— Only BLAS-1 and BLAS-2} \]

- \( C_a := \)

8 algorithms:

- 4 dot-based:
  - \( aj \)-dot \( ja \)-dot \( ai \)-dot \( ia \)-dot
- 2 gemv-based:

\[
\begin{align*}
C_a := A_{iaj} B_{ji} & \quad j\text{-gemv} \\
\text{for } j = 1:j \\
C[:,:] & += A[:,:,:j] B[j,:] \\

C_a := A_{iaj} B_{ji} & \quad i'\text{-gemv} \\
\text{for } i = 1:i \\
\tilde{A}[:,:] & = A[i,:,:] \\
C[:] & += \tilde{A}[:,:] B[:,:] \\
\end{align*}
\]

- 2 axpy-based:
  - \( ij \)-axpy \( ji \)-axpy

- \( a = i = j = 8 \ldots 1,000 \)
$C_a := A_{iaj} B_{ji}$ — Only BLAS-1 and BLAS-2

Results

![Graphs showing performance results for different operations.](image)

- $aj$-$dot$, $ja$-$dot$, $ai$-$dot$, $ia$-$dot$
- $ij$-$axpy$, $ji$-$axpy$, $j$-$gemv$, $i'$-$gemv$

FLOPS/cycle for $a = i = j$ in the range of 0 to 1,000.
\[ C_{abc} := A_{aij} B_{jbic} \] — Challenging Contraction

- \[ C_{abc} := A_{aij} B_{jbic} \]
- 176 Algorithms:
  - 48 dot-based
  - 72 axpy-based
  - 36 gemv-based
  - 12 ger-based
  - 8 gemm-based:
    - \( cj'-\text{gemm} \)
    - \( jc'-\text{gemm} \)
    - \( ci'-\text{gemm} \)
    - \( i'c-\text{gemm} \)
    - \( bj'-\text{gemm} \)
    - \( jb'-\text{gemm} \)
    - \( bi'-\text{gemm} \)
    - \( i'b-\text{gemm} \)
- \( i = j = 8, \ a = b = c = 8 \ldots 1,000 \)
- Intel Ivy Bridge E5-2680 v2
- Single-threaded OPENBLAS
\[ C_{abc} := A_{aij} B_{jbic} \] — Challenging Contraction

Results

\begin{itemize}
  \item \textit{cj'}-gemm
  \item \textit{jc'}-gemm
  \item \textit{ci'}-gemm
  \item \textit{i'}-\textit{c}-gemm
  \item \textit{bj'}-gemm
  \item \textit{jb'}-gemm
  \item \textit{bi'}-gemm
  \item \textit{i'}-\textit{b}-gemm
  \item \textit{dot}-based
  \item \textit{axpy}-based
  \item \textit{gemv}-based
  \item \textit{ger}-based
\end{itemize}

\begin{figure}
\centering
\begin{subfigure}{0.45\textwidth}
\includegraphics[width=\textwidth]{chart1}
\caption{Flops/cycle for different operations.}
\end{subfigure}
\begin{subfigure}{0.45\textwidth}
\includegraphics[width=\textwidth]{chart2}
\caption{Flops/cycle for different operations.}
\end{subfigure}
\end{figure}

a = b = c \quad (i = j = 8)
\[ C_{abc} := A_{aij}B_{jbic} \quad \text{— 10 Threads} \]

- \( C_{abc} := A_{aij}B_{jbic} \)
- \( i = j = 32, \ a = b = c = 8 \ldots 1,000 \)
- Intel Ivy Bridge E5-2680 v2
- OPENBLAS, 10 threads
\( C_{abc} := A_{aij} B_{jbi} \) — 10 Threads

Results

\[
\begin{align*}
&c_j'\text{-gemm} & jc'\text{-gemm} & c_{i'}\text{-gemm} & i'c\text{-gemm} \\
&bj'\text{-gemm} & jb'\text{-gemm} & bi'\text{-gemm} & i'b\text{-gemm} \\
\text{dot-based} & \text{axpy-based} & \text{gemv-based} & \text{ger-based}
\end{align*}
\]

\[
\begin{array}{c}
a = b = c \\
(i = j = 32)
\end{array}
\]

Elmar Peise
Prediction Efficiency

\[ C_{abc} := A_{ai} B_{ibc} \]

kernel: \( \text{dot}, \text{axpy}, \text{gemv}, \text{ger}, \text{gemm} \)

\[
time(\text{alg}) / time(\text{benchmark})
\]

\[
a = b = c \quad (i = 8)
\]
On the Performance Prediction of BLAS-based Tensor Contractions

- BLAS-based algorithm generation
- Micro-benchmarks with careful cache setup
- Applicable to wide range of challenging scenarios

Funding from Deutsche Forschungsgemeinschaft and Deutsche Telekom Stiftung is gratefully acknowledged.

Deutsche Forschungsgemeinschaft

Deutsche Telekom Stiftung