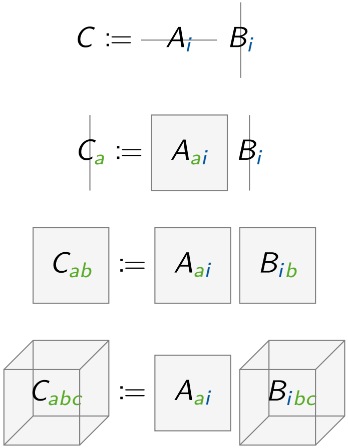
On the Performance Prediction of BLAS-based Tensor Contractions

Elmar Peise Diego Fabregat-Traver Paolo Bientinesi

Aachen Institute for Advanced Study in
Computational Engineering Science
RWTH Aachen University

November 16th 2014 — PMBS14

Tensor Contractions?



$$C := \sum_{\mathtt{i}} A[\mathtt{i}]\, B[\mathtt{i}]$$

$$\forall \mathtt{a}.\, C[\mathtt{a}] := \sum_{\mathtt{i}} A[\mathtt{a},\mathtt{i}]\, B[\mathtt{i}]$$

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

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

$$C_{abc} := A_{ija} B_{jbic}$$

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

free indices

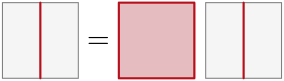
contracted indices

Use BLAS!

$C_{ab} := A_{ai} B_{ib} \rightarrow$

$C_{ab} := A_{ai} B_{ib}$ b-gemv

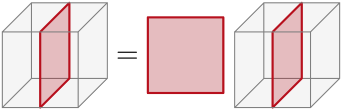
```
for b = 1:b
    C[:,b] = A[:,:] B[:,b]
```



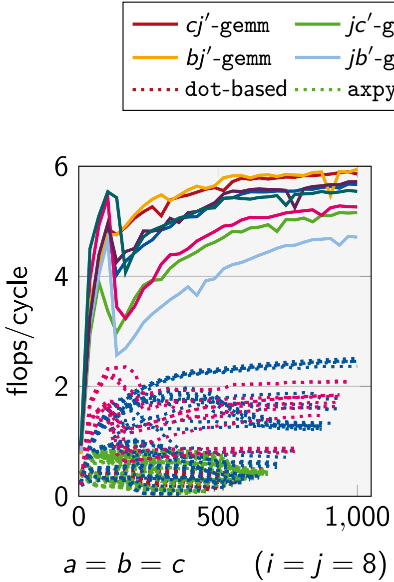
$C_{abc} := A_{ai} B_{ibc} \rightarrow$

$C_{abc} := A_{ai} B_{ibc}$ b-gemm

```
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

Algorithm Generation

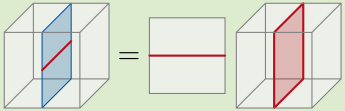
- Select kernel
- Match kernel indices to tensor indices
- Cast remaining tensor indices as **for**-loops
- Assemble algorithm (AST, C-code)

Example

$$C_\alpha := A_{\alpha\iota} B_\iota \quad \rightarrow \quad C_{abc} = A_{ai} B_{ibc}$$

$$\iota \quad \rightarrow \quad i$$

$$\alpha \quad \rightarrow \quad c$$



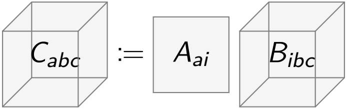
for $\leftarrow$ b

for $\leftarrow$ a

$C_{abc} := A_{ai} B_{ibc}$ — ba-gemv

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

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



- BLAS-1
  - 6 dot-based: $(C := A_\iota B_\iota)$

    abc-dot *acb*-dot *bac*-dot *bca*-dot *cab*-dot *cba*-dot
  - 18 axpy-based: $(C_\alpha := A B_\alpha)$

    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: $(C_\alpha := A_{\alpha\iota} B_\iota)$

    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: $(C_{\alpha\beta} := A_{\alpha\iota} B_{\iota\beta})$

    c-gemm *b*-gemm

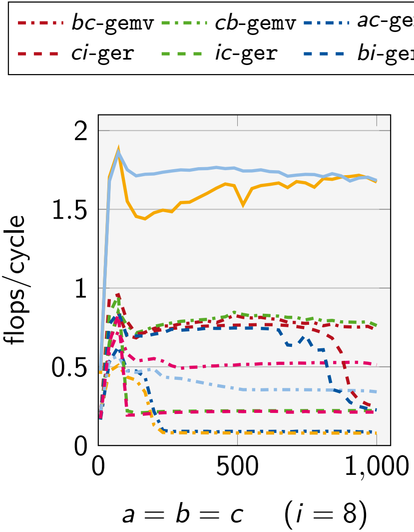
Outline

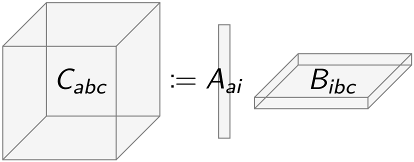
1. Algorithm Generation

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?





- Intel Penryn E5450 (Harpertown)
- Single-threaded OpenBLAS

Micro-Benchmarks

$C_{abc} \coloneqq A_{ai} B_{ibc}$ — *ba*-gemv

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

Micro-benchmark

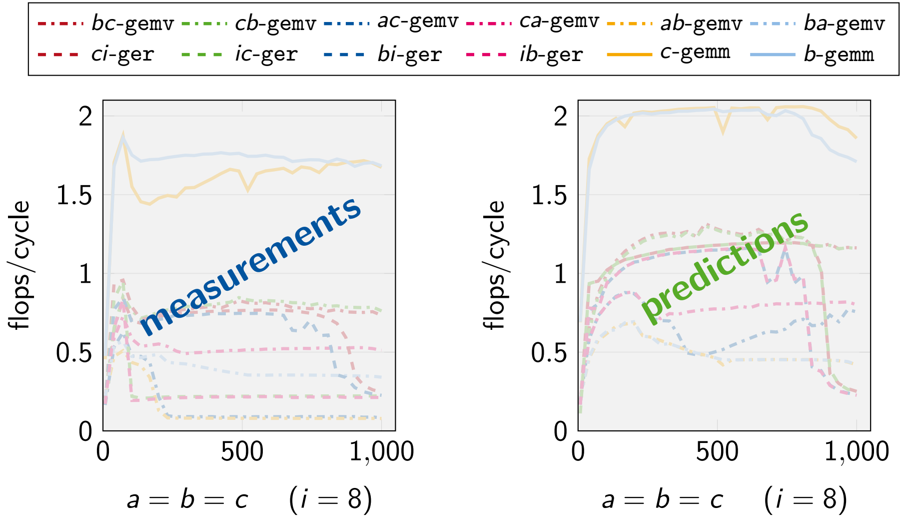
```
C[a,b,:] = A[a,:] B[:,b,:]
```

estimate

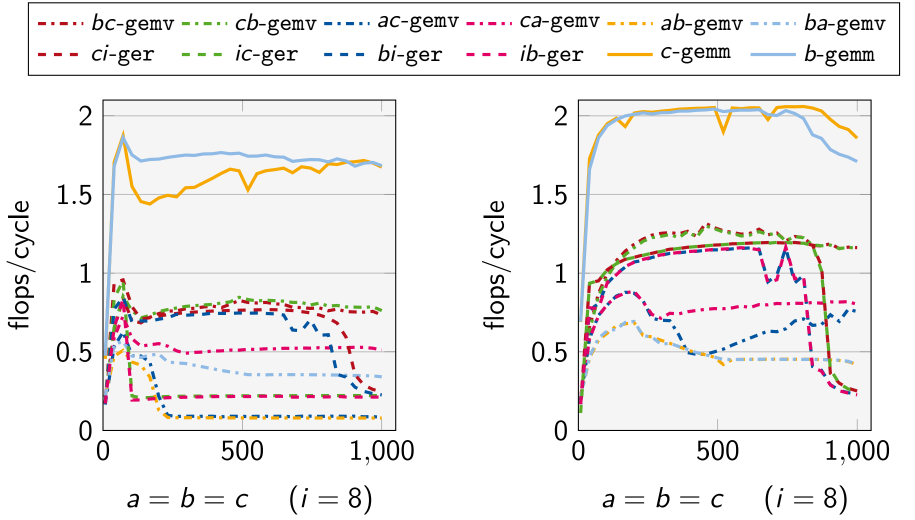
time $10\times$

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

1. Repeated Execution



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

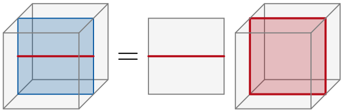
Assumption: Fully associative Least Recently Used (LRU) cache replacement policy

$\Rightarrow$ Cache state is defined by the order of memory accesses.

Access Distance

$C_{abc} := A_{ai} B_{ibc}$ *ca*-gemv

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

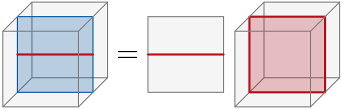


Access Distance $d(M) = \text{size}(\textit{all data used since the last access to } M)$

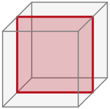
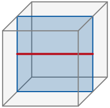
Access Distance

$C_{abc} := A_{ai} B_{ibc}$ *ca*-`gemv`

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



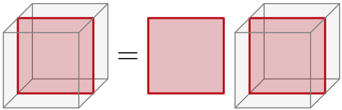
Access Distance $d(M) = \text{size}(\textit{all data used since the last access to } M)$

-  B`[:,:,c]` doesn't vary in **for** `a`
  $d(B$`[:,:,c]`$) = 0$ doubles
-  A`[a,:]`
-  C`[a,:,c]`

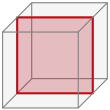
Access Distance

$C_{abc} := A_{ai} B_{ibc}$ *ca*-gemv

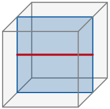
```
for c = 1:c
    for a = 1:a
        C[a,:,c] = A[a,:] B[:,:,c]
```



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

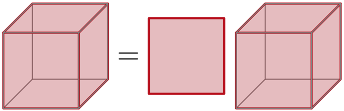
- 
  `A[a,:]` varies in **for** `a`; doesn't vary in **for** `c`
  $d($`A[a,:]`$) = \text{size}(\textit{all operands in}$ **for** `a`$)$
  $= \text{size}(C[:,:,c]) + \text{size}(A[:,:]) + \text{size}(B[:,:,c])$
  $= a \cdot b + a \cdot i + i \cdot b$

- 
  `C[a,:,c]`

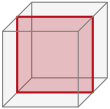
Access Distance

$C_{abc} := A_{ai} B_{ibc}$	*ca*-`gemv`

```
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`

  $d(\texttt{B[:,:,c]}) = 0$ doubles

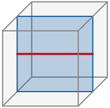
- 

  `A[a,:]` varies in **for** `a`; doesn't vary in **for** `c`

  $d(\texttt{A[a,:]}) = \text{size}(\textit{all operands in } \textbf{for}\ \texttt{a})$

  $= \text{size}(\texttt{C[:,:,c]}) + \text{size}(\texttt{A[:,:]}) + \text{size}(\texttt{B[:,:,c]})$

  $= \quad a \cdot b \quad + \quad a \cdot i \quad + \quad i \cdot b$

- 

  `C[a,:,c]` varies in **for** `a`; varies in **for** `c`

  $d(\texttt{C[a,:,c]}) = \text{size}(\textit{all operands in } \textbf{for}\ \texttt{c})$

  $= \text{size}(\texttt{C[:,:,:]}) + \text{size}(\texttt{A[:,:]}) + \text{size}(\texttt{B[:,:,:]})$

  $= \quad a \cdot b \cdot c \quad + \quad a \cdot i \quad + \quad i \cdot b \cdot c$

Setup

Access Distances

$d(B\texttt{[:,:,c]})$	$= 0$	$= 0$	doubles
$d(A\texttt{[a,:]})$	$= a \cdot b + a \cdot i + i \cdot b$	$= 166{,}400$	doubles
$d(C\texttt{[a,:,c]})$	$= a \cdot b \cdot c + a \cdot i + i \cdot b \cdot c$	$= 65{,}283{,}200$	doubles

Sizes: $a = b = c = 400$, $i = 8$.

$C_{abc} := A_{ai} B_{ibc}$ *ca*-gemv

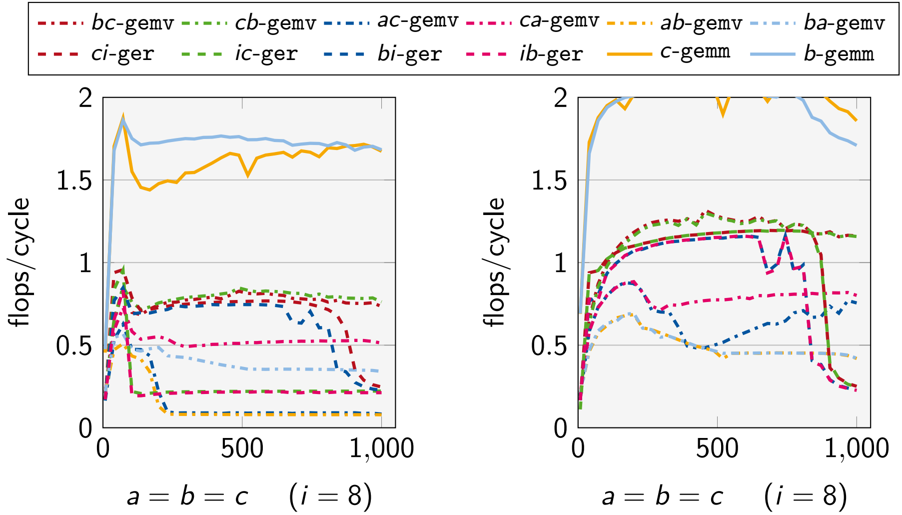
```
for c = 1:c
    for a = 1:a
        C[a,:,c] = A[a,:] B[:,:,c]
```

Micro-benchmark

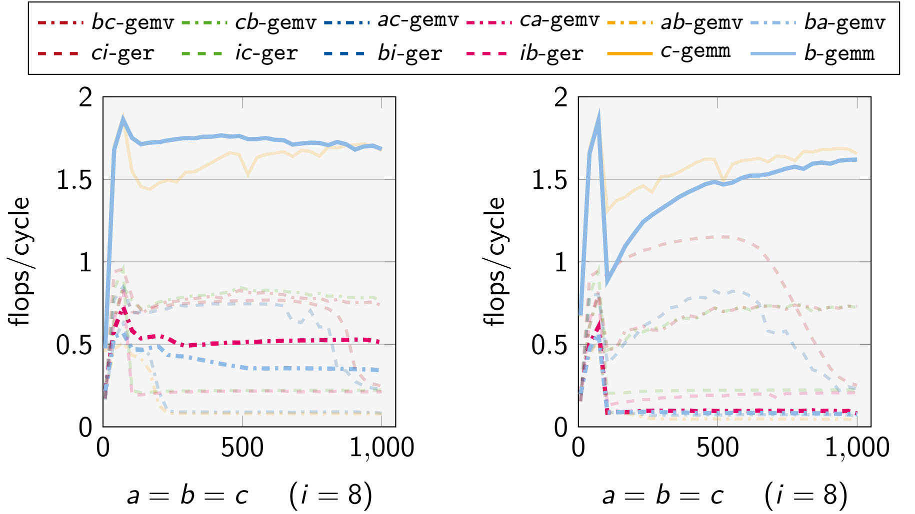
```
flush 816,632
touch A[a,:]
flush 163,200
touch B[:,:,c]
C[a,:,c] = A[a,:] B[:,:,c]
```

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

2. Estimates with Cache Setup



2. Estimates with Cache Setup



Problem: underestimation

3. Prefetching!

Problem: selective underestimation

Cause: Prefetching not accounted for

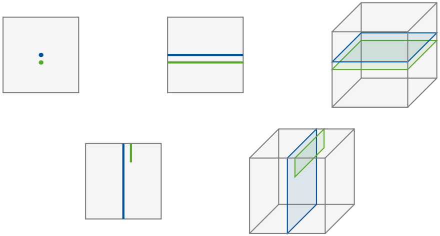
Solution Approach

Mimick prefetching

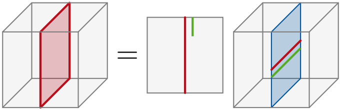
Prefetching:

access

prefetch



Mimicking the Prefetching



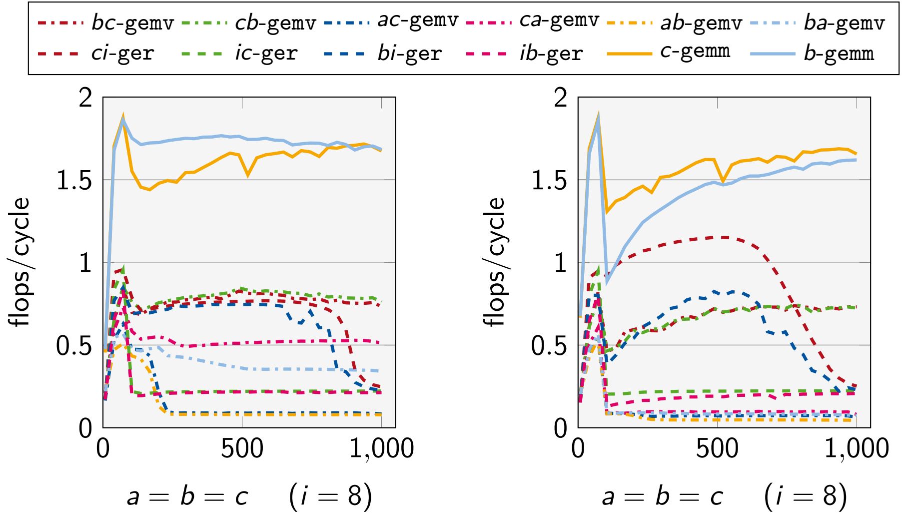
$C_{abc} := A_{ai} B_{ibc}$ *bi*-ger

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

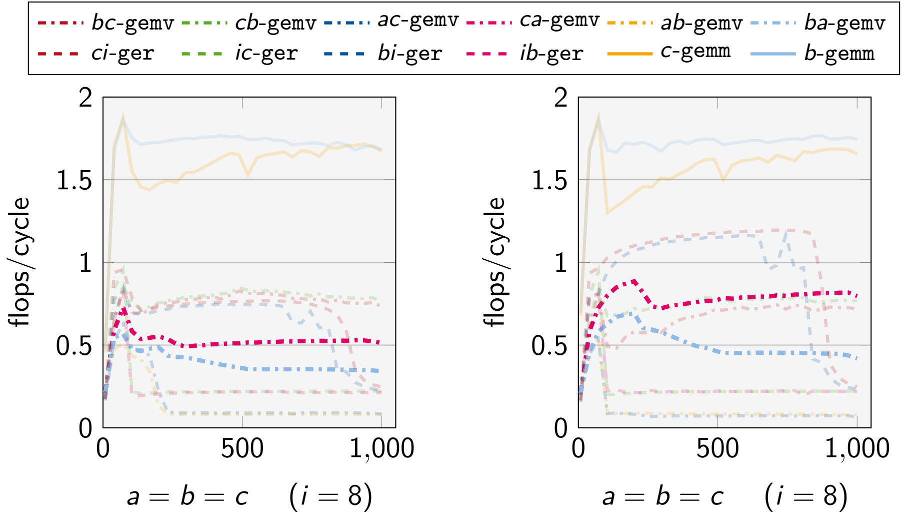
Micro-benchmark

```
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



3. Estimates with Prefetching



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)

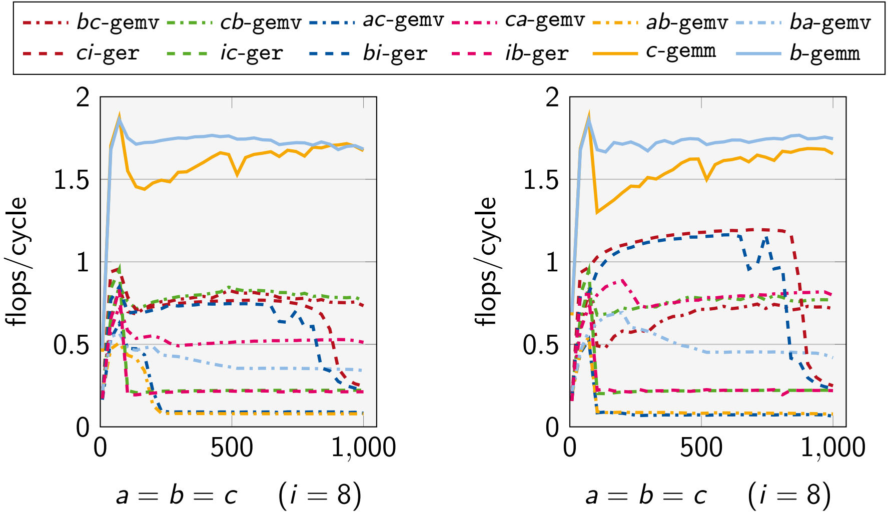
$\underbrace{\begin{array}{l}\texttt{touch } A\texttt{[:,i]} \\ \texttt{flush } 5{,}992 \\ \texttt{touch } A\texttt{[:8,i]} \\ \texttt{touch } B\texttt{[i,b,:]} \\ \texttt{touch } C\texttt{[:,b,:]} \\ C\texttt{[:,b,:]} = A\texttt{[:,i]}\ B\texttt{[i,b,:]}\end{array}}_{\text{time } 10\times}$

Micro-benchmark (no pre)

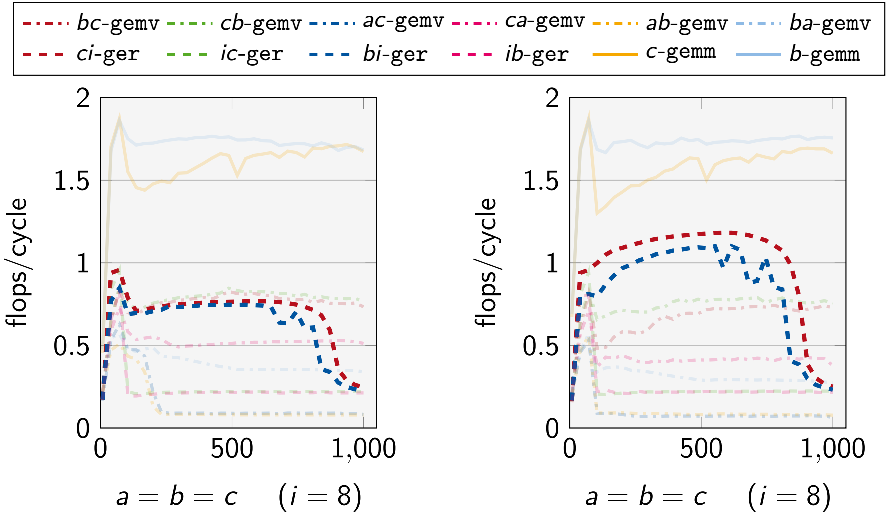
$\underbrace{\begin{array}{l}\texttt{flush } 816{,}240 \\ \texttt{touch } A\texttt{[:,i]} \\ \texttt{flush } 5{,}992 \\ \texttt{touch } A\texttt{[:8,i]} \\ \texttt{touch } C\texttt{[:,b,:]} \\ C\texttt{[:,b,:]} = A\texttt{[:,i]}\ B\texttt{[i,b,:]}\end{array}}_{\text{time } 10\times}$

$$\text{estimate} = \frac{1}{8}(7\text{median} + 1\text{median})$$

4. Estimates with Prefetching Failures



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)

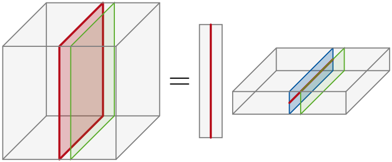
5. Small Loops' First Iterations

Problem: selective overestimation

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

$C_{abc} := A_{ai} B_{ibc}$ *bi*-ger

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



Solution Approach

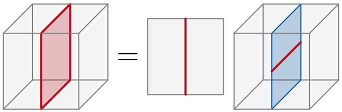
Separate micro-benchmarks for first iteration of small loops

First Iteration Benchmark

- **Access Distance**:

$C_{abc} := A_{ai} B_{ibc}$ *bi*-ger

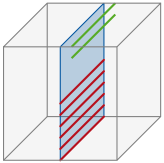
```
for b = 1:b
    for i = 1:i
        C[:,b,:] = A[:,i] B[i,b,:]
```



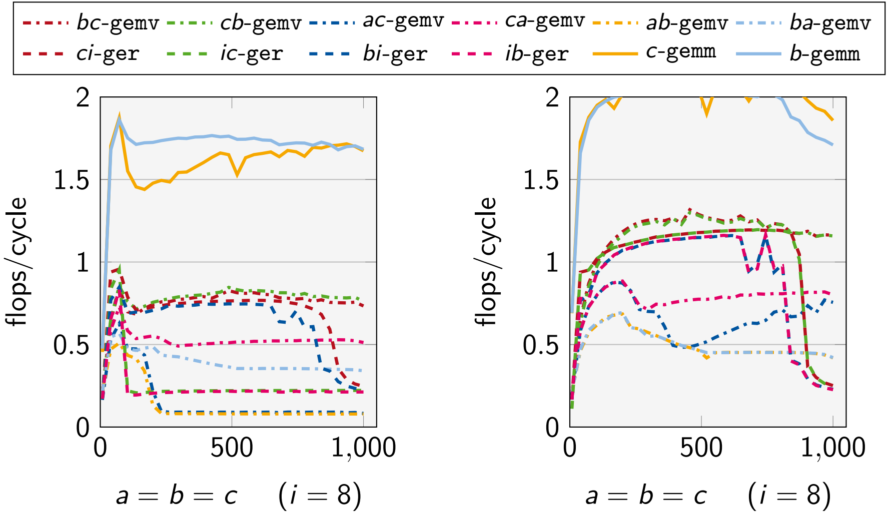
~~Find last access to A[:,i], B[i,b,:], and C[:,b,:] within **for** i~~

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

- **Prefetching**:



5. Final Estimates



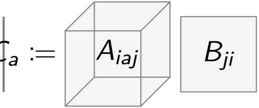
Outline

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

$C_a := A_{iaj} B_{ji}$ — Only BLAS-1 and BLAS-2

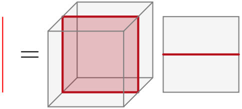
- $C_a :=$ 

  8 algorithms:
  - 4 dot-based:

    aj-dot *ja*-dot *ai*-dot *ia*-dot
  - 2 gemv-based:

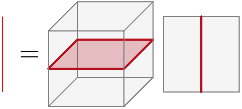
$C_a := A_{iaj} B_{ji}$ *j*-gemv

```
for j = 1:j
    C[:] += A[:,:,j] B[j,:]
```



$C_a := A_{iaj} B_{ji}$ *i'*-gemv

```
for i = 1:i
    Ã[:,:] = A[i,:,:]
    C[:] += Ã[:,:] B[:,i]
```



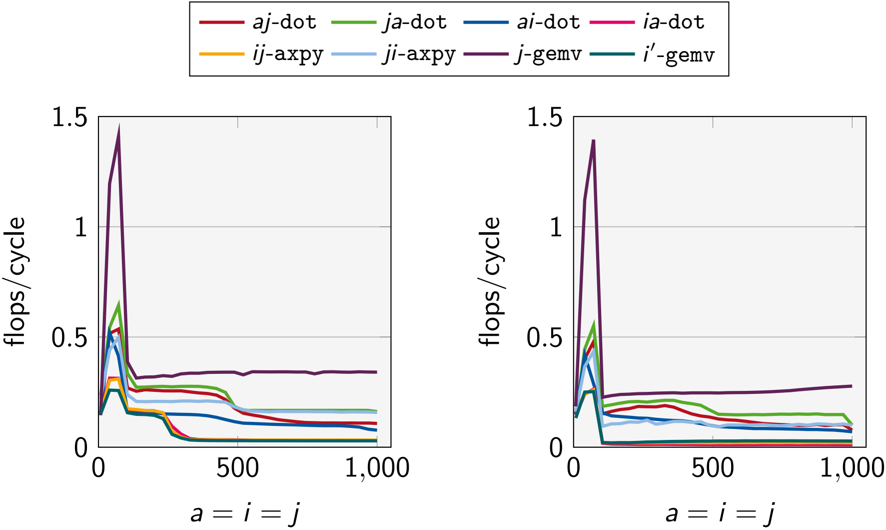
  - 2 axpy-based:

    ij-axpy *ji*-axpy

- $a = i = j = 8 \dots 1{,}000$

$C_a := A_{iaj} B_{ji}$ — Only BLAS-1 and BLAS-2

Results

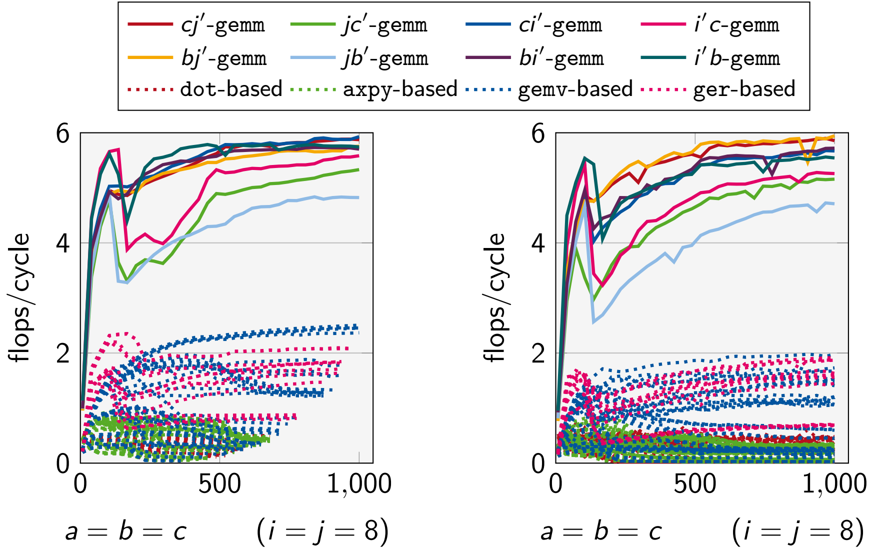


$C_{abc} := A_{aij} B_{jbic}$ — Challenging Contraction

- $C_{abc} := A_{aij} B_{jbic}$
  176 Algorithms:
  - 48 `dot`-based
  - 72 `axpy`-based
  - 36 `gemv`-based
  - 12 `ger`-based
  - 8 `gemm`-based:
    cj'-`gemm` jc'-`gemm` ci'-`gemm` $i'c$-`gemm`
    bj'-`gemm` jb'-`gemm` bi'-`gemm` $i'b$-`gemm`
- $i = j = 8$, $a = b = c = 8 \dots 1{,}000$
- Intel Ivy Bridge E5-2680 v2
- Single-threaded OpenBLAS

$C_{abc} := A_{aij} B_{jbic}$ — Challenging Contraction

Results

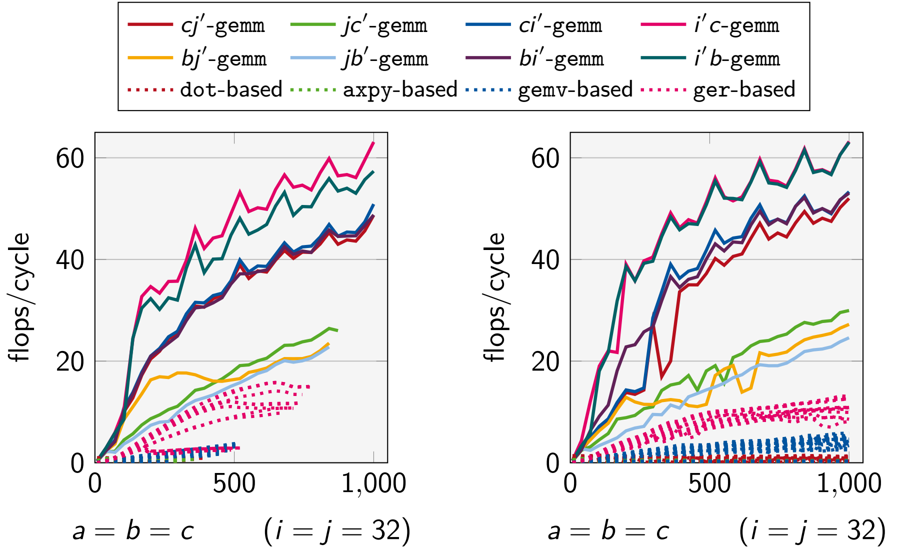


$C_{abc} := A_{aij} B_{jbic}$ — 10 Threads

- $C_{abc} := A_{aij} B_{jbic}$
- $i = j = \mathbf{32}$, $a = b = c = 8 \dots 1{,}000$
- Intel Ivy Bridge E5-2680 v2
- OpenBLAS, 10 threads

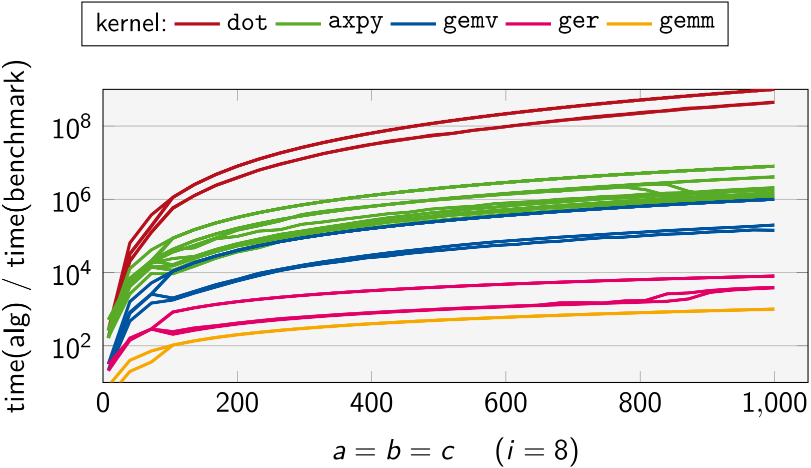
$C_{abc} := A_{aij} B_{jbic}$ — 10 Threads

Results



Prediction Efficiency

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



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.