Estimating the Efficiency of BLAS-based Tensor Contractions

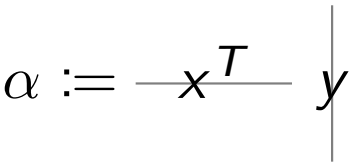
2nd Annual Report

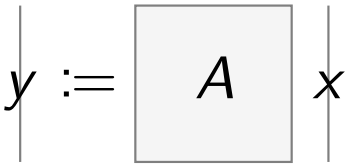
Elmar Peise

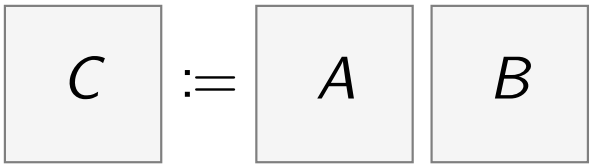
Aachen Institute for Advanced Study in
Computational Engineering Science
RWTH Aachen University

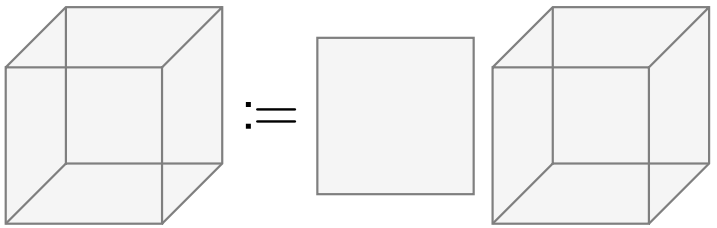
November 6th 2014

Tensor Contractions?

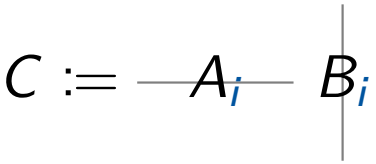
$\alpha := x^T y$  ✓ (dot)

$y := Ax$  ✓ (gemv)

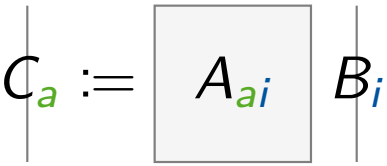
$C := AB$  ✓ (gemm)

???  ✗

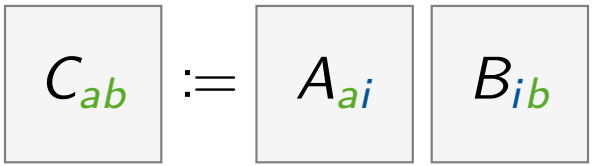
Tensor Contractions!



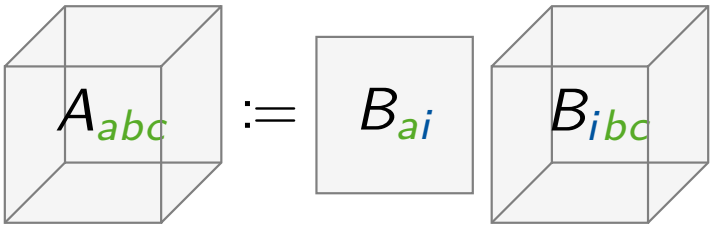
$$C := \sum_{\mathtt{i}} A[\mathtt{i}]\, B[\mathtt{i}]$$



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



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



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

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

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

free indices

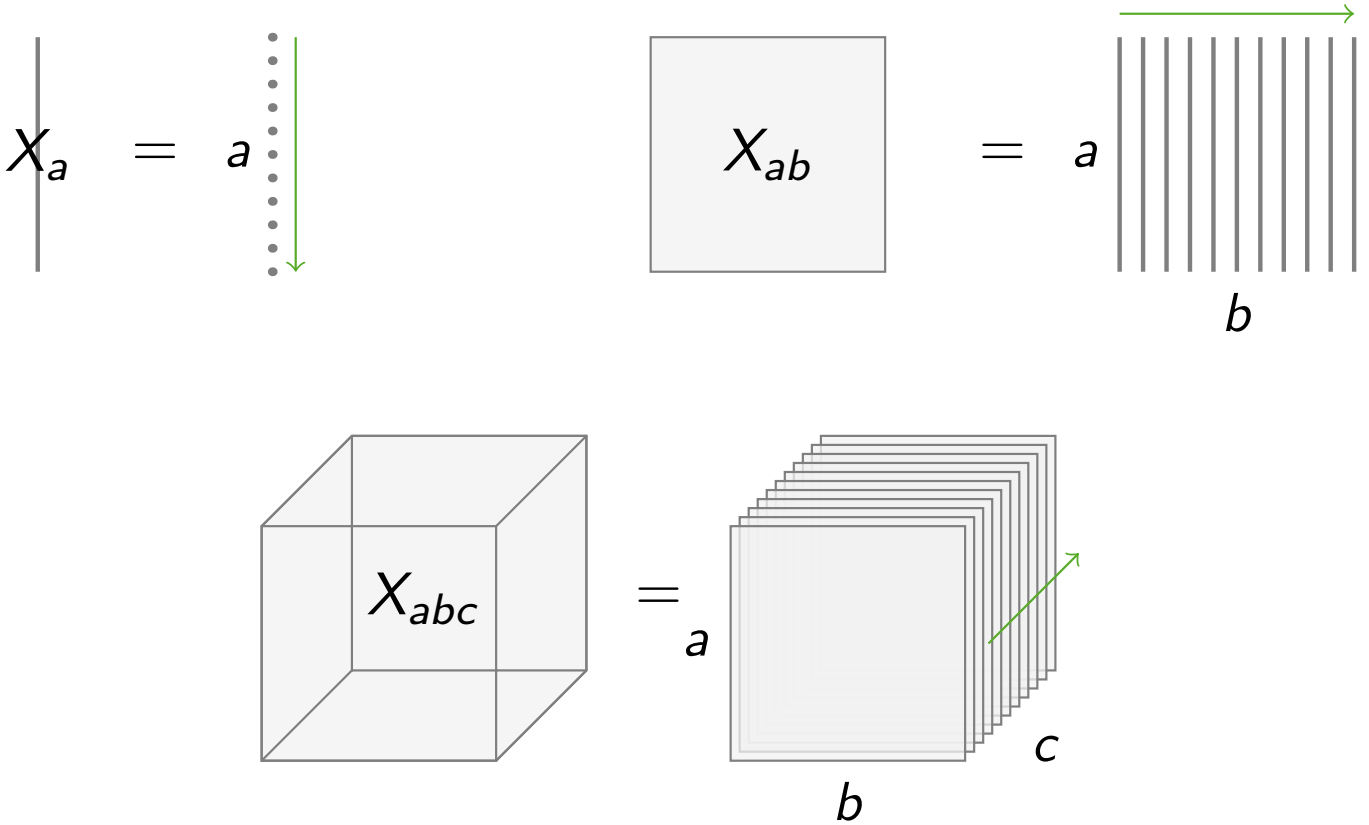
contracted indices

Tensor Storage

- Contiguous
- First index varies the fastest (Fortran style)

Tensor Storage

- Contiguous
- First index varies the fastest (Fortran style)

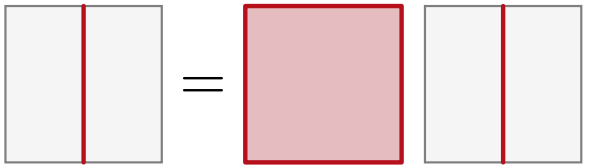


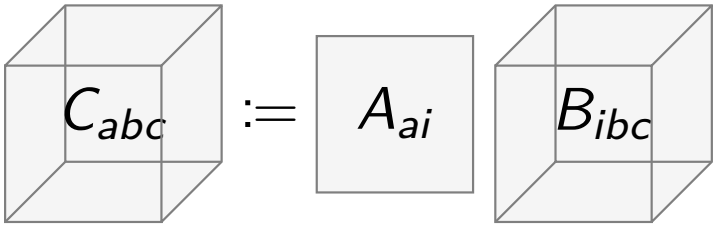
Outline

Casting Computation in Terms of BLAS

$C_{ab} := A_{ai} B_{ib}$ — *b*-gemv

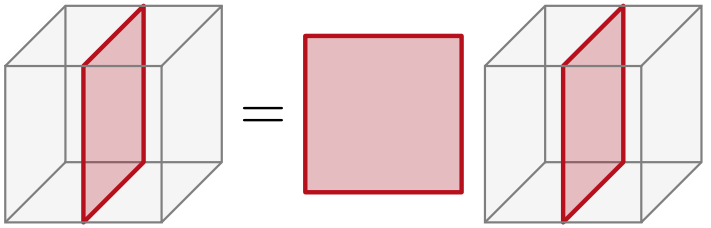
```
for b = 1:b
    C[:,b] = A[:,:] B[:,b]
```



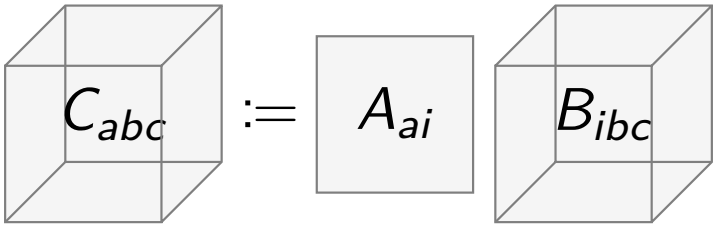


$C_{abc} := A_{ai} B_{ibc}$ — *b*-gemm

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

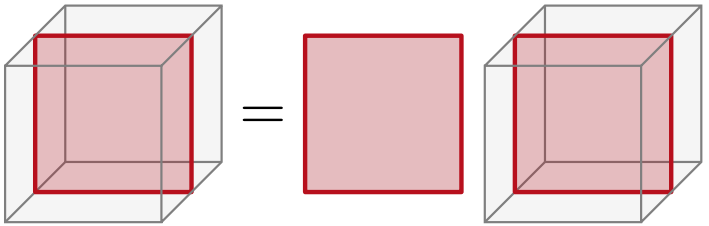


More Examples: $C_{abc} := A_{ai} B_{ibc}$



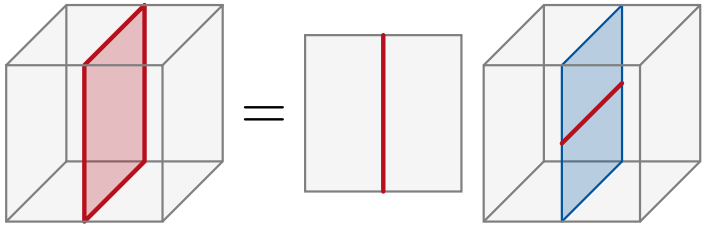
$C_{abc} := A_{ai} B_{ibc}$ *c*-gemm

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



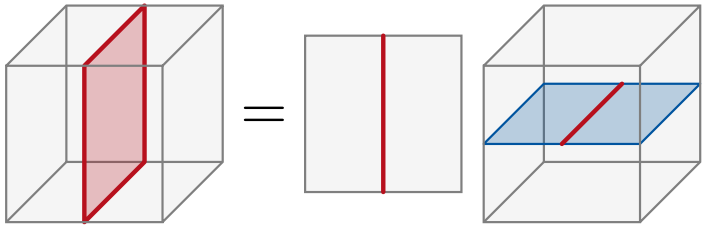
$C_{abc} := A_{ai} B_{ibc}$ *bi*-ger

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



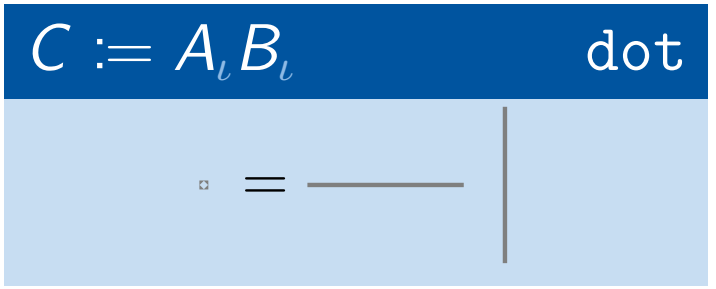
$C_{abc} := A_{ai} B_{ibc}$ *ib*-ger

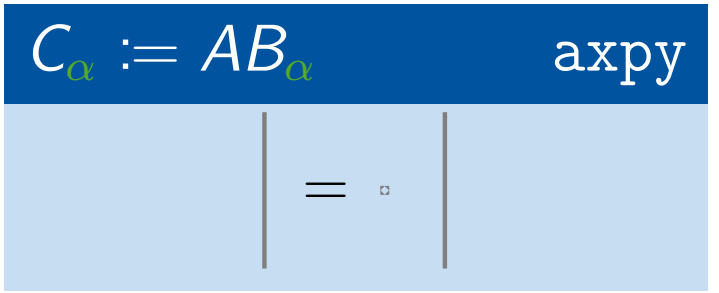
```
for i = 1:i
    for b = 1:b
        C[:,b,:] = A[:,i] B[i,b,:]
```



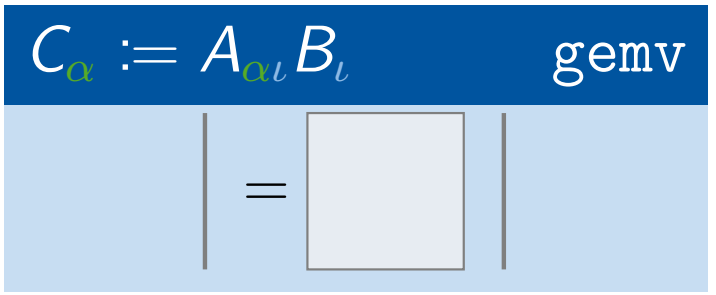
BLAS Kernels

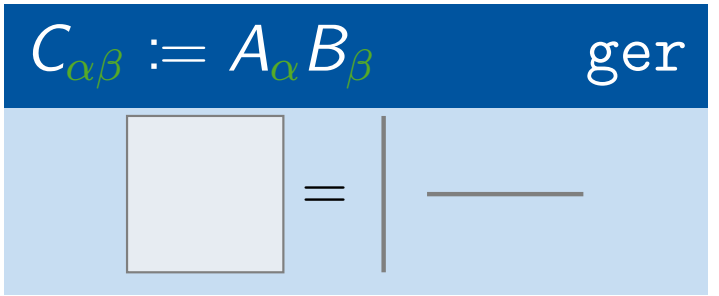
- BLAS-1:



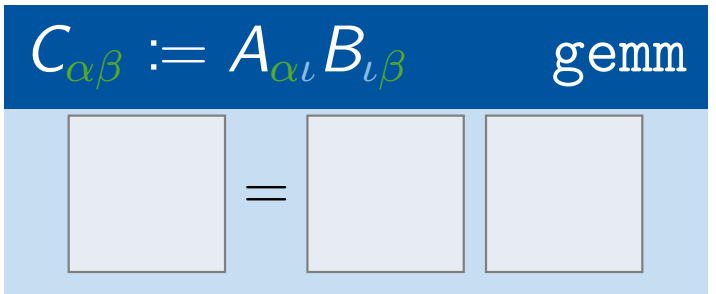


- BLAS-2:





- BLAS-3:



free indices contracted indices

Algorithm Generation

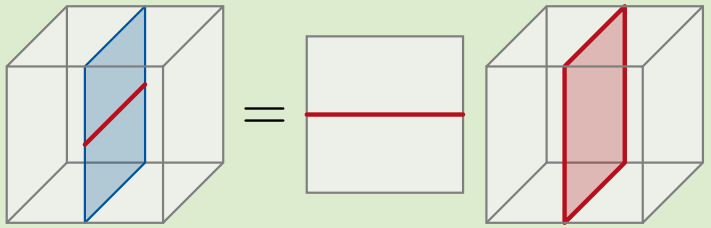
- Select kernel
- Match kernel indices to tensor indices
- Cast remaining tensor indices as **for**-loops
- Assemble algorithm (AST, C-code)

Example

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

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

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



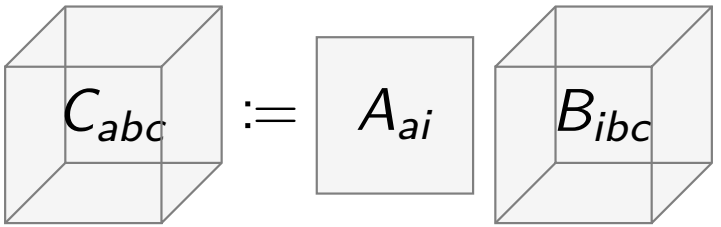
for $\leftarrow$ b

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,:]
```

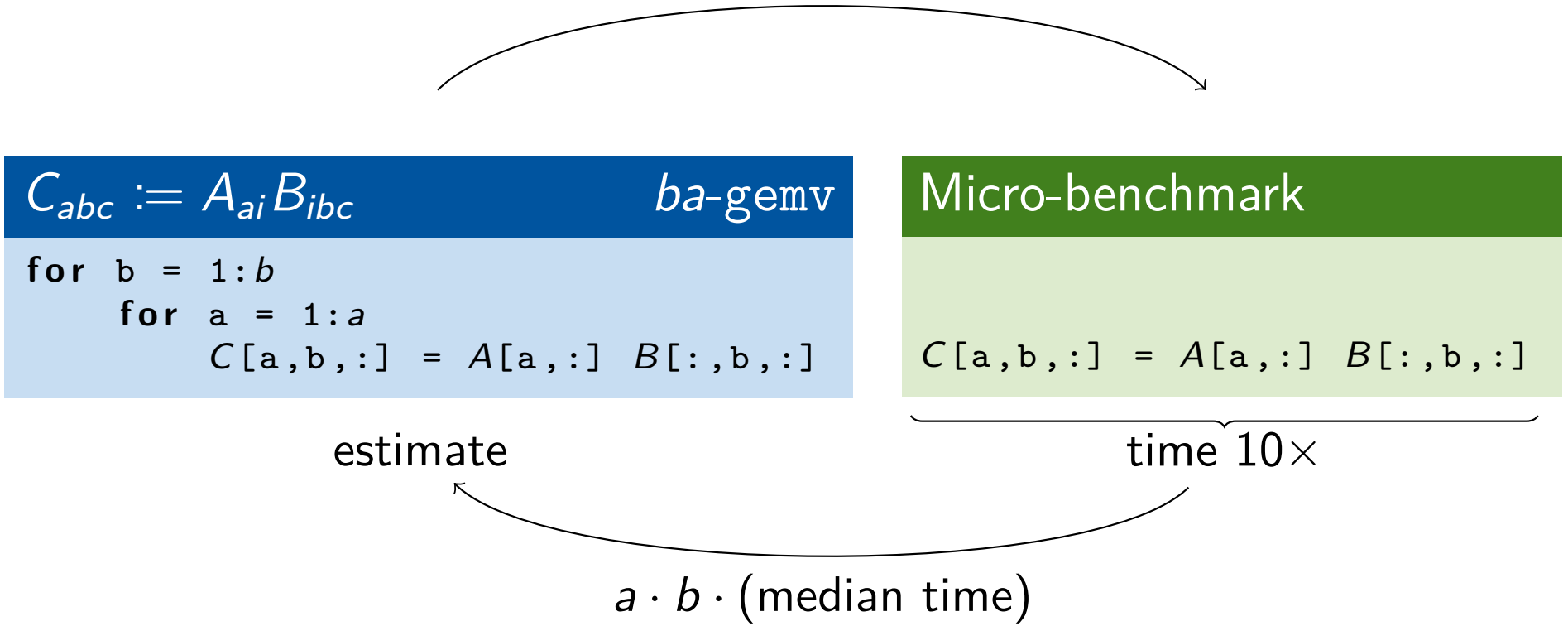
Algorithms for $C_{abc} := A_{ai} B_{ibc}$



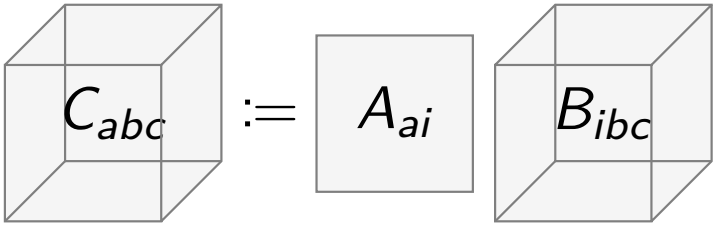
36 algorithms:

- 6 dot-based:
  abc-dot *acb*-dot *bac*-dot *bca*-dot *cab*-dot *cba*-dot
- 18 axpy-based:
  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
- 6 gemv-based:
  bc-gemv *cb*-gemv *ac*-gemv *ca*-gemv *ab*-gemv *ba*-gemv
- 4 ger-based:
  ic-ger *ci*-ger *ib*-ger *bi*-ger
- 2 gemm-based:
  c-gemm *b*-gemm

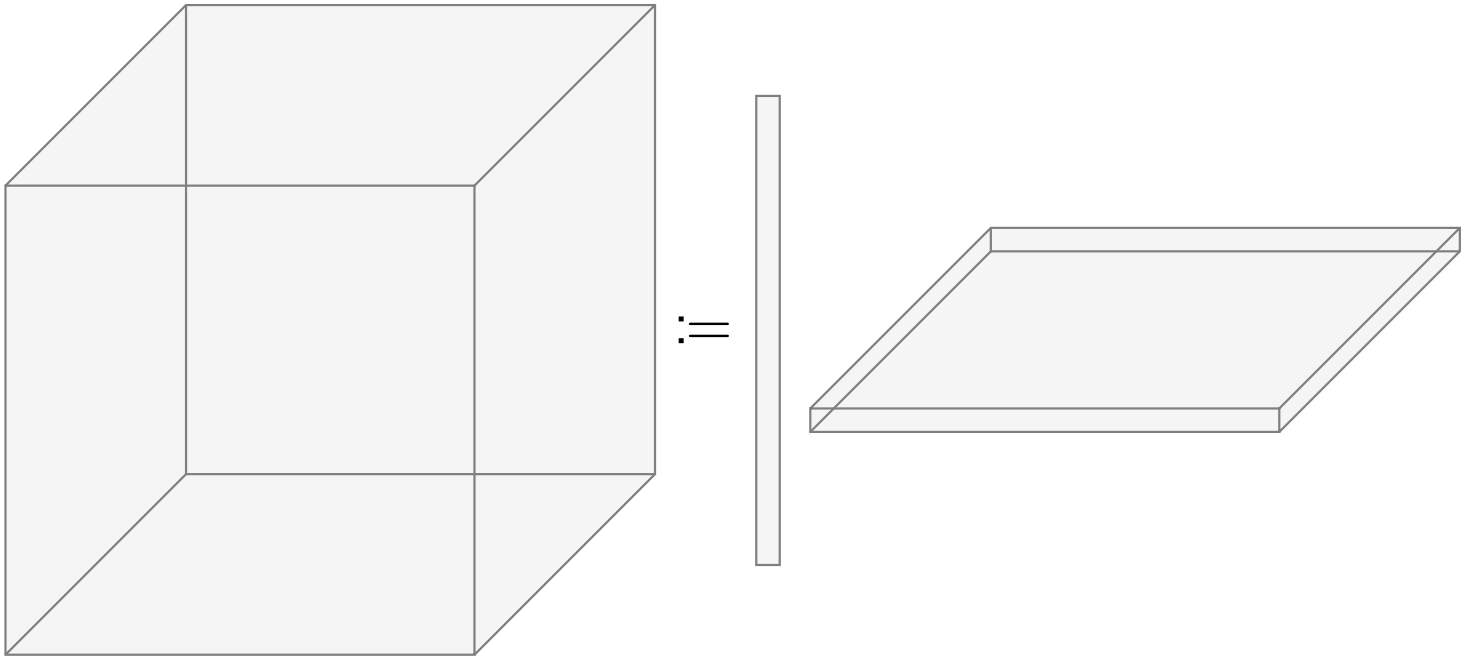
Micro-Benchmarks



Test Setup

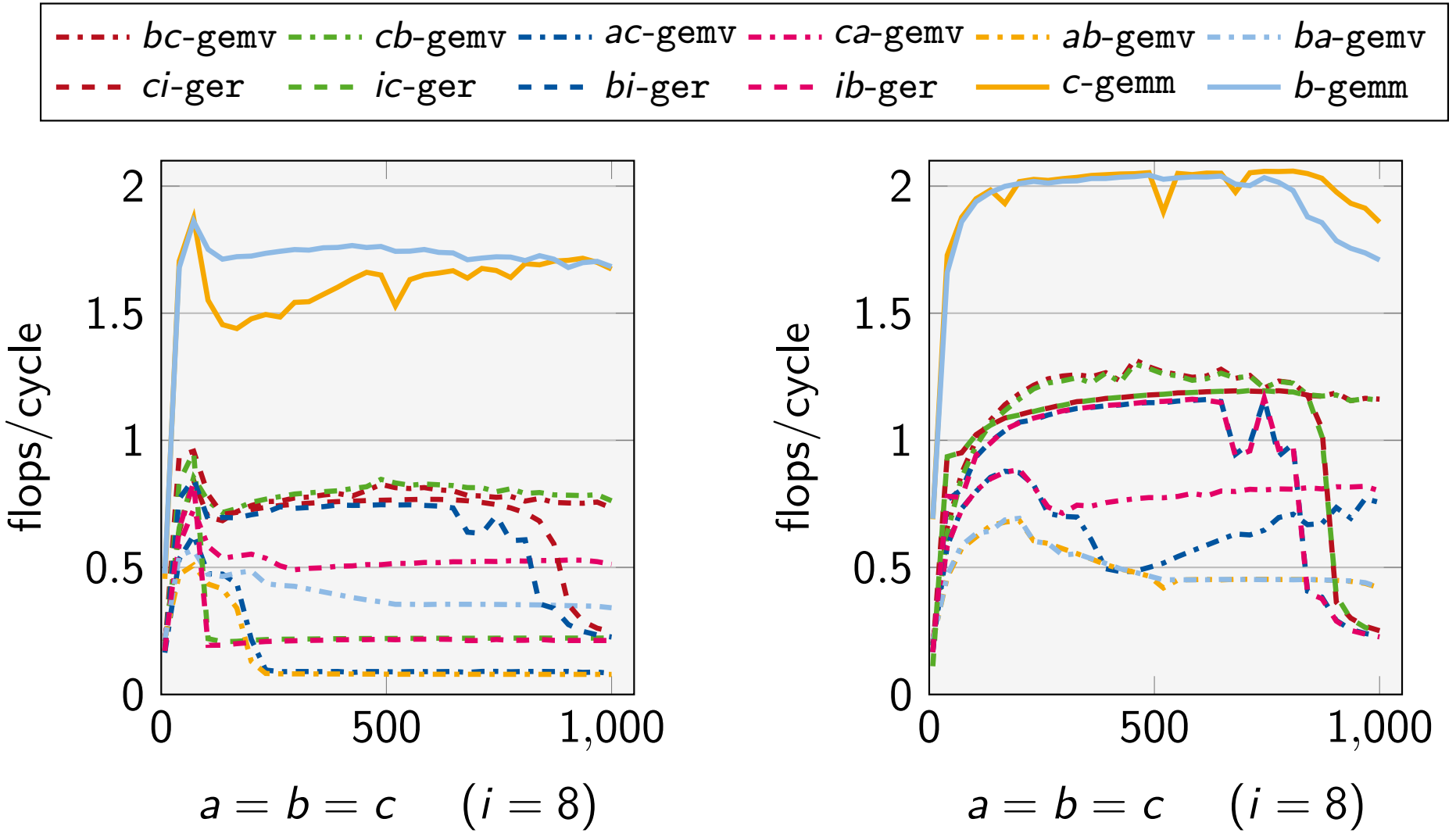
- 

- $i = 8$, $a = b = c = 8 \ldots 1{,}000$



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

Repeated Execution



Problem: general overestimation

Caching!

Problem: general overestimation

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

Solution Approach

Recreate cache precondition within micro-benchmark

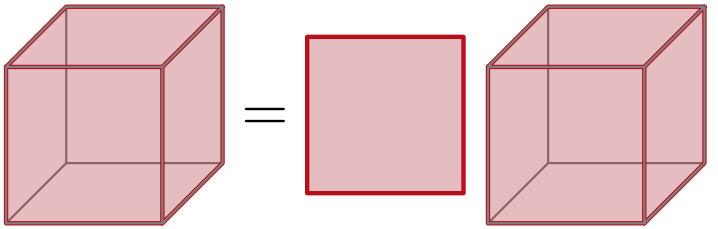
Assumption: Fully associative Least Recently Used (LRU) cache replacement policy

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

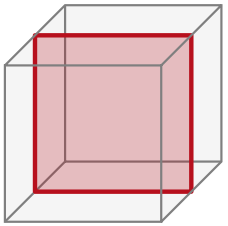
Access Distance

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

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



Access Distance $d(M)$: How much data was loaded since M's last access.

- 

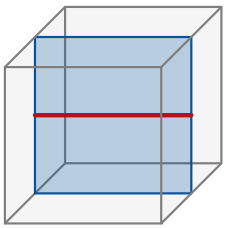
  $B[:,:,c]$ doesn't depend on **for** a

  $d(B[:,:,c]) = 0$

- 

  $A[a,:]$ depends on **for** a; doesn't depend on **for** c

  $$\begin{aligned} d(A[a,:]) &= \text{size}(\text{join}_a(C[a,:,c], A[a,:], B[:,:,c]) \\ &= \text{size}(C[:,:,c], A[:,:], B[:,:,c]) \\ &= a \cdot b + a \cdot i + i \cdot b \end{aligned}$$

- 

  $C[a,:,c]$ depends on **for** a; depends on **for** c

  $$\begin{aligned} d(C[a,:,c]) &= \text{size}(\text{join}_{ac}(C[a,:,c], A[a,:], B[:,:,c]) \\ &= \text{size}(C[:,:,:], A[:,:], B[:,:,:]) \\ &= a \cdot b \cdot c + a \cdot i + i \cdot b \cdot c \end{aligned}$$

Setup

Access distances

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

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

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

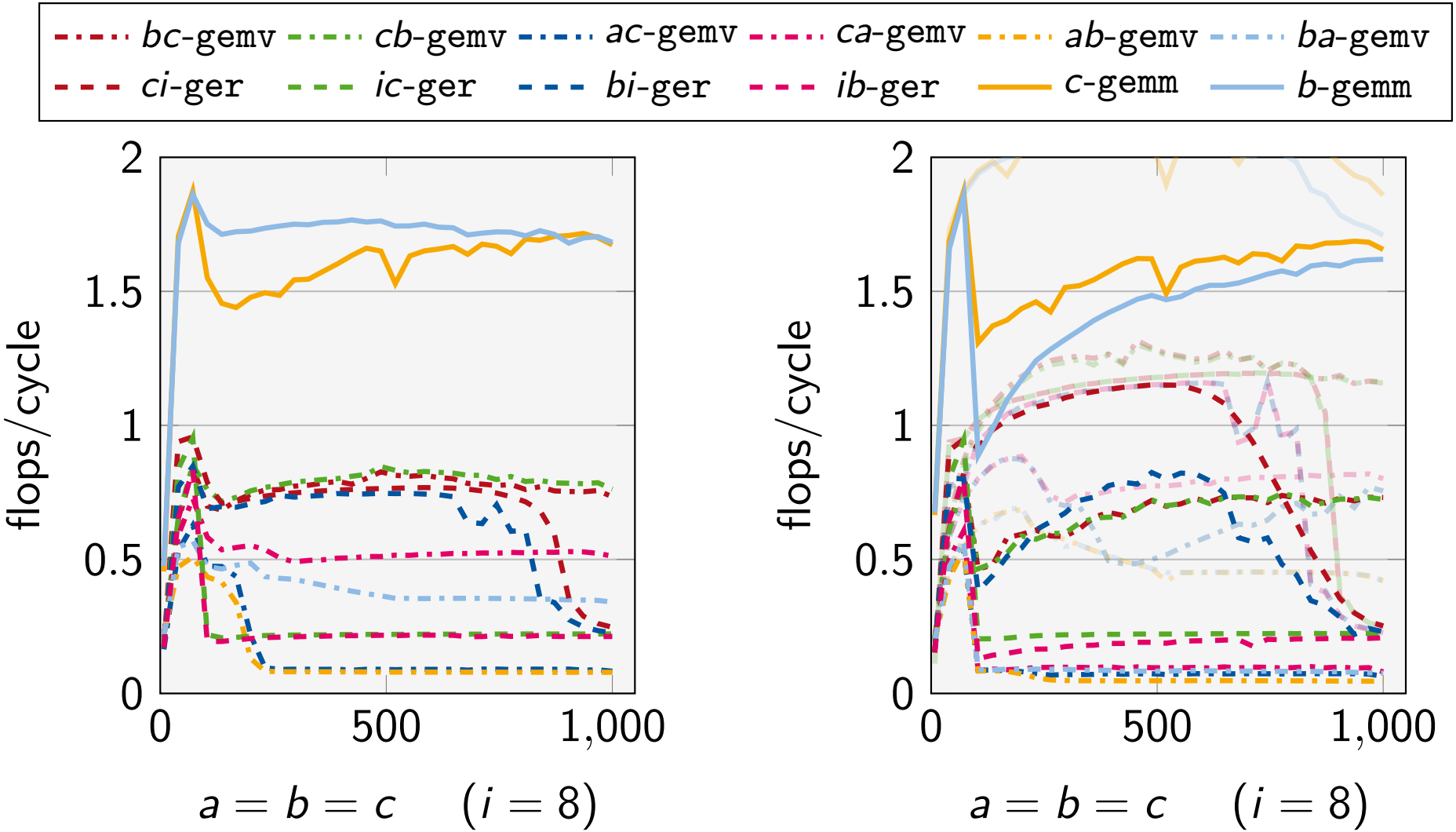
```
for c = 1:c
    for a = 1:a
        C[a,:,c] = A[a,:] B[:,:,c]
```

Micro-benchmark

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

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

Estimates with Cache Setup



Problem: underestimation (including *ca*-gemv ( ))

Prefetching!

Problem: selective underestimation

Cause: Prefetching not accounted for

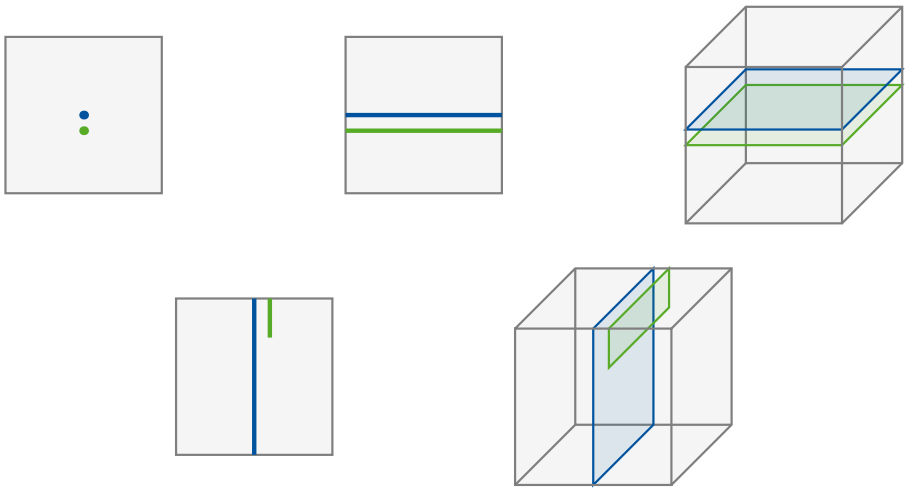
Solution Approach

Imitate prefetching

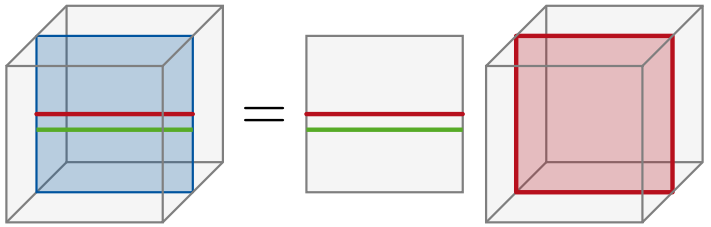
Prefetching:

access

prefetch



Prefetching Imitation



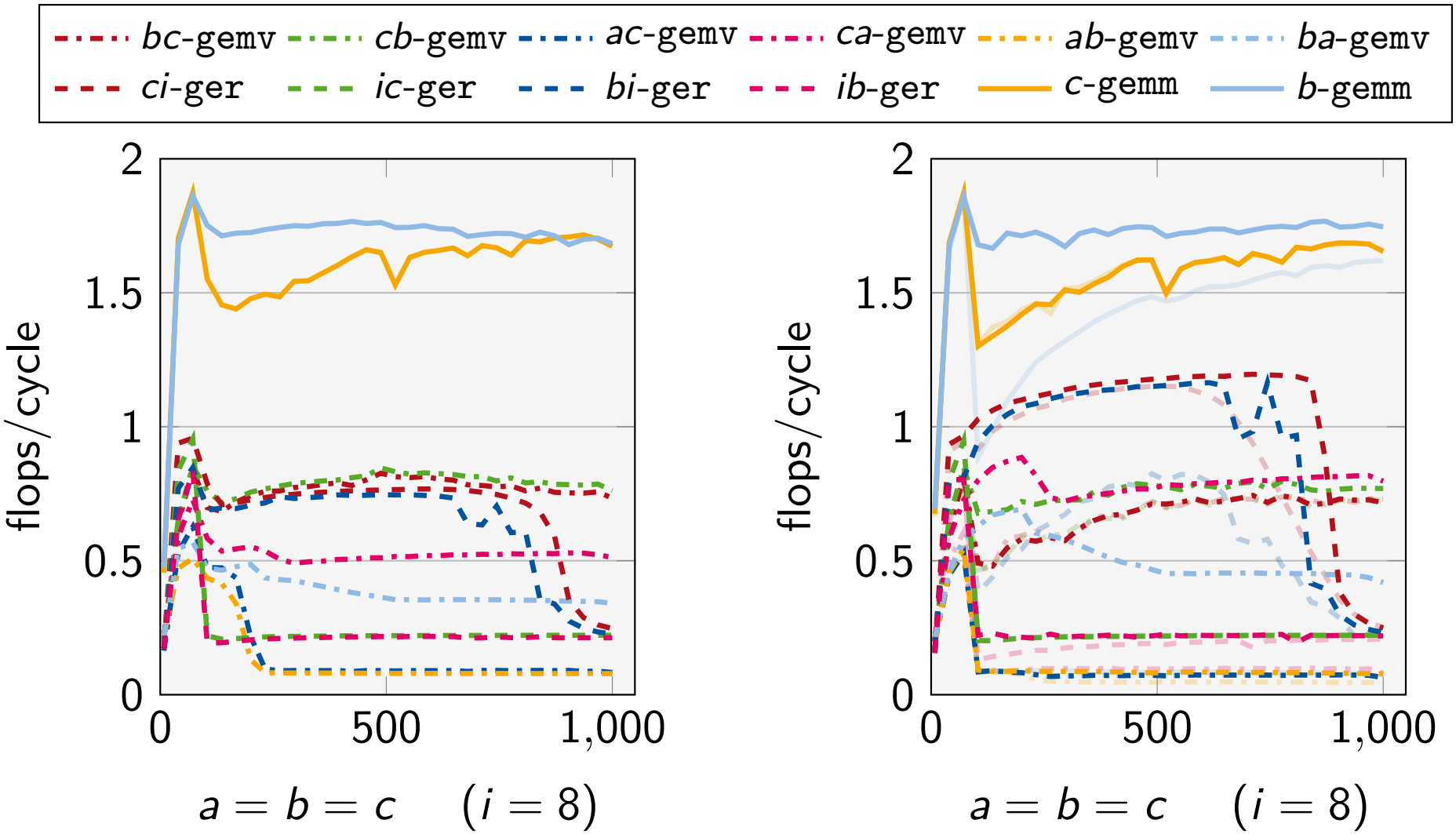
$C_{abc} := A_{ai} B_{ibc}$ *ca*-gemv

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

Micro-benchmark

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

Estimates with Prefetching



Problem: some incorrect prefetching (including *ca*-gemv (- - - -))

Prefetching Failures

Problem: selective prefetchinf faliure

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)

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

time 10×

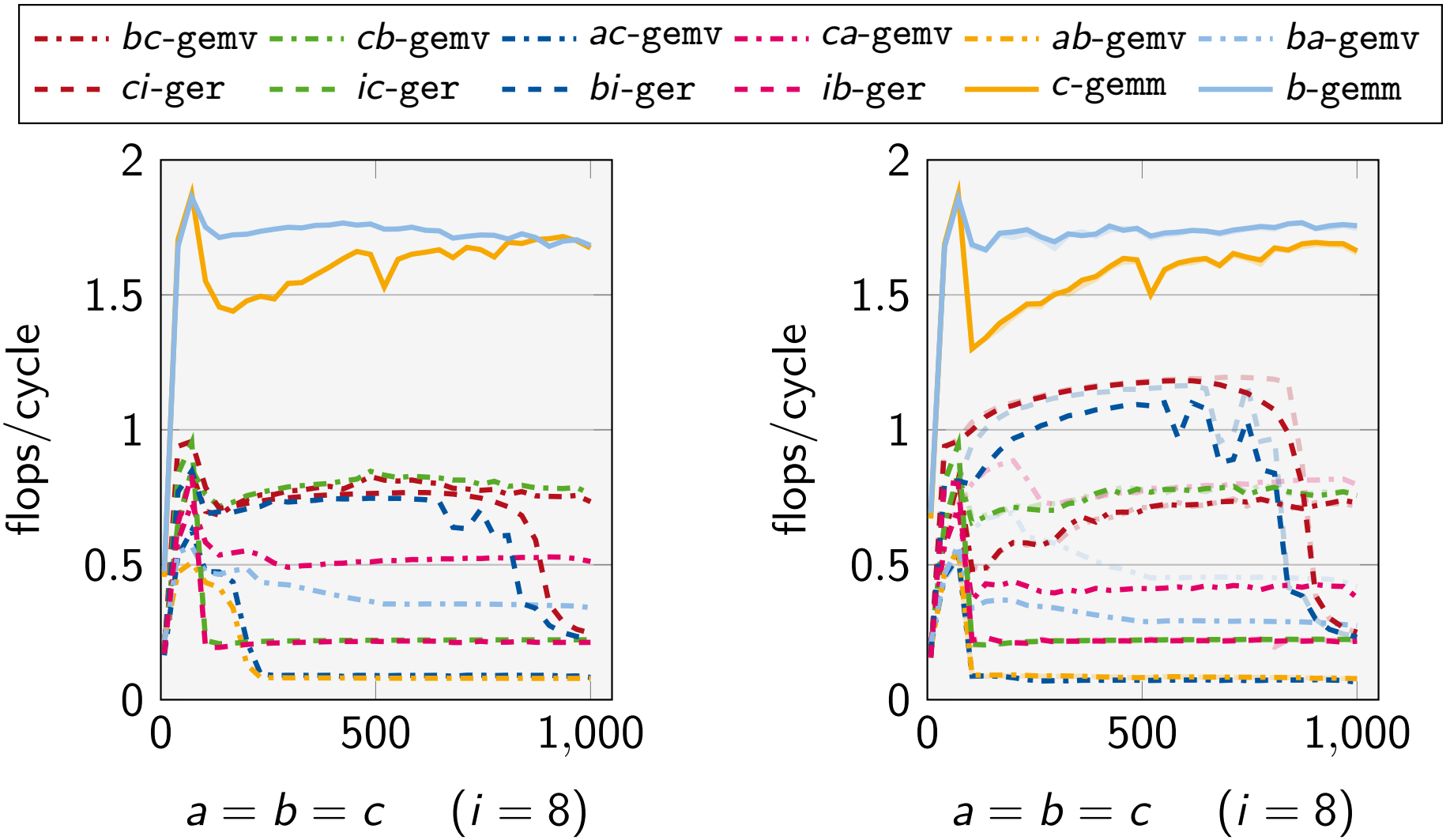
Micro-benchmark (no pre)

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

time 10×

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

Estimates with Prefetching Failures



Problem: selective overestimation (including *bi*-ger (- - -))

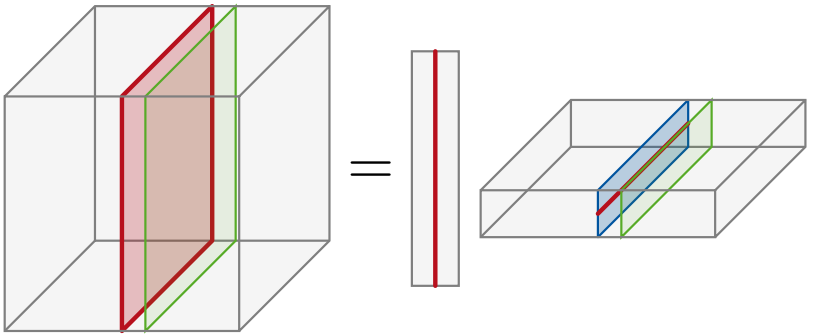
Small Loops' First Iterations

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

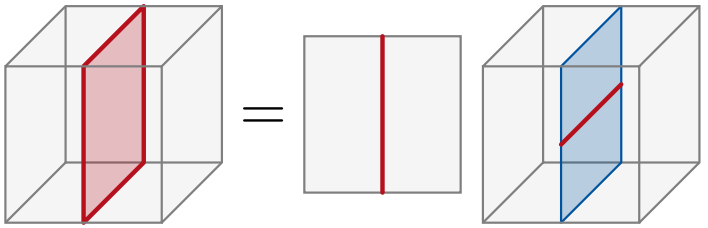
Separate micro-benchmarks for first iteration of small loops

First Iteration Benchmark

- **Access Distance**:

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

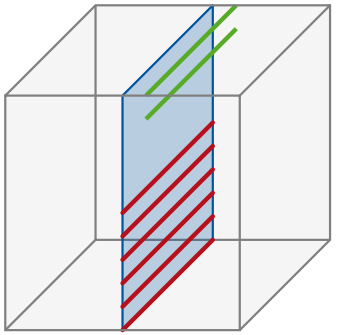
```
for b = 1:b
    for i = 1:i
        C[:,b,:] = A[:,i] B[i,b,:]
```



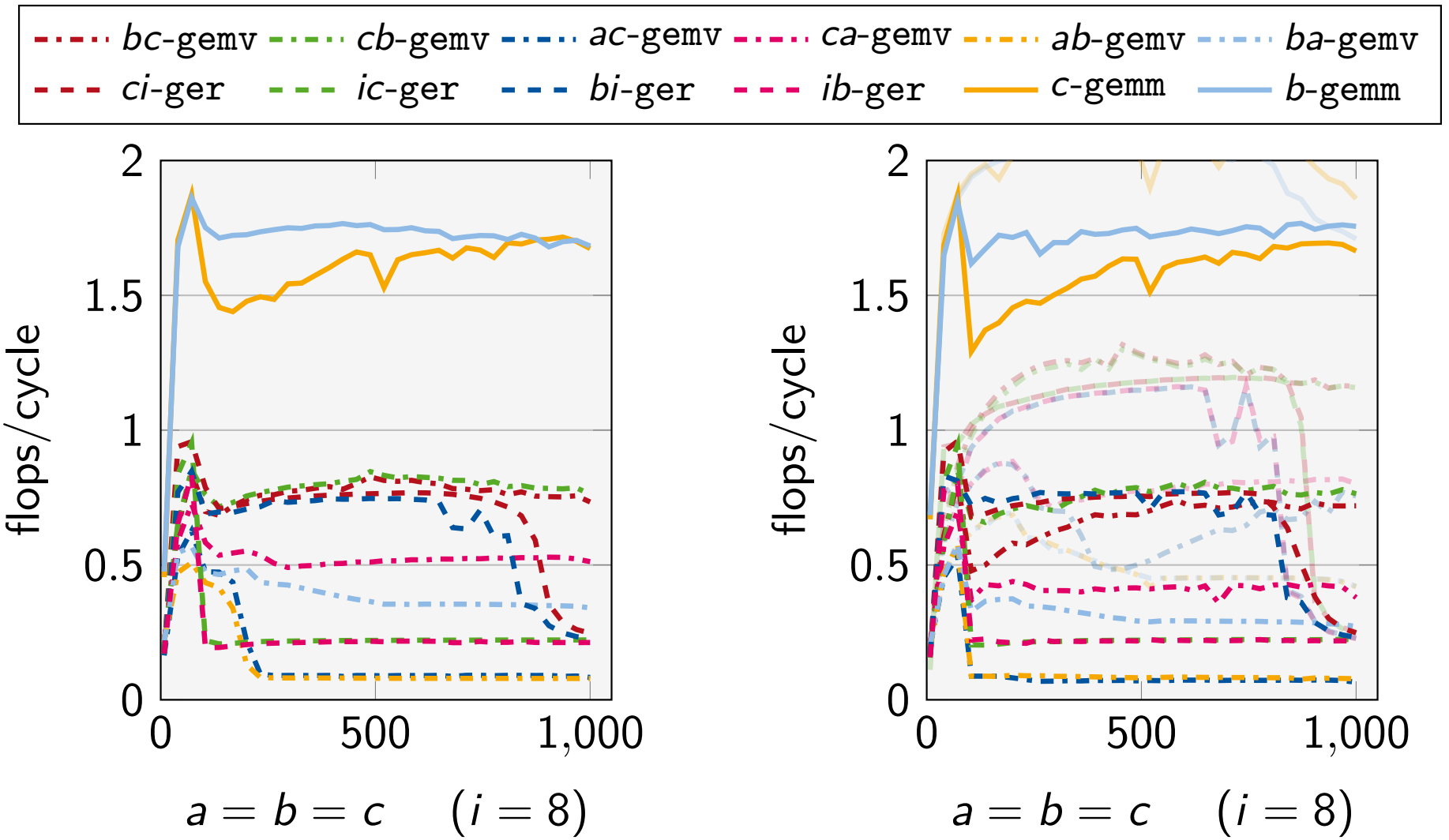
~~Find last access to A[:,i], B[i,:,c], C[:,:,c] within **for** i~~

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

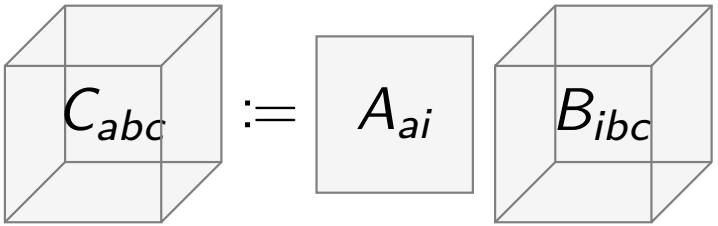
- **Prefetching**:



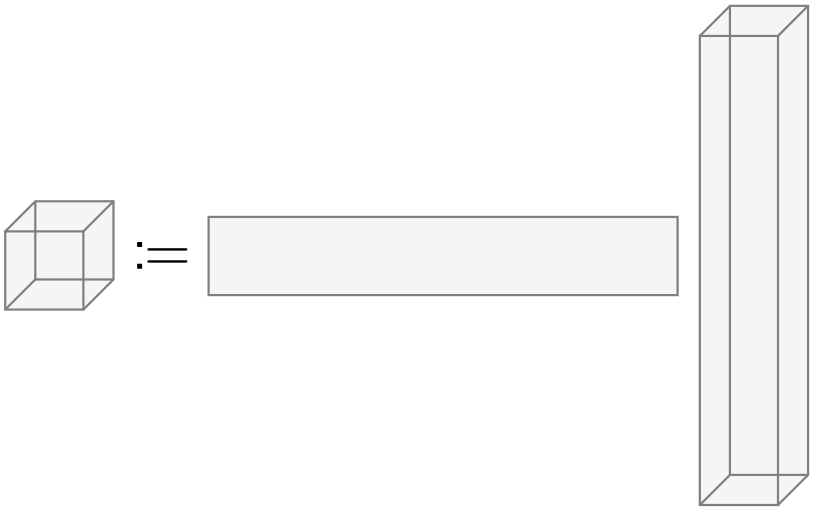
Final Estimates



$C_{abc} := A_{ai} B_{ibc}$ — Setup 2

- 

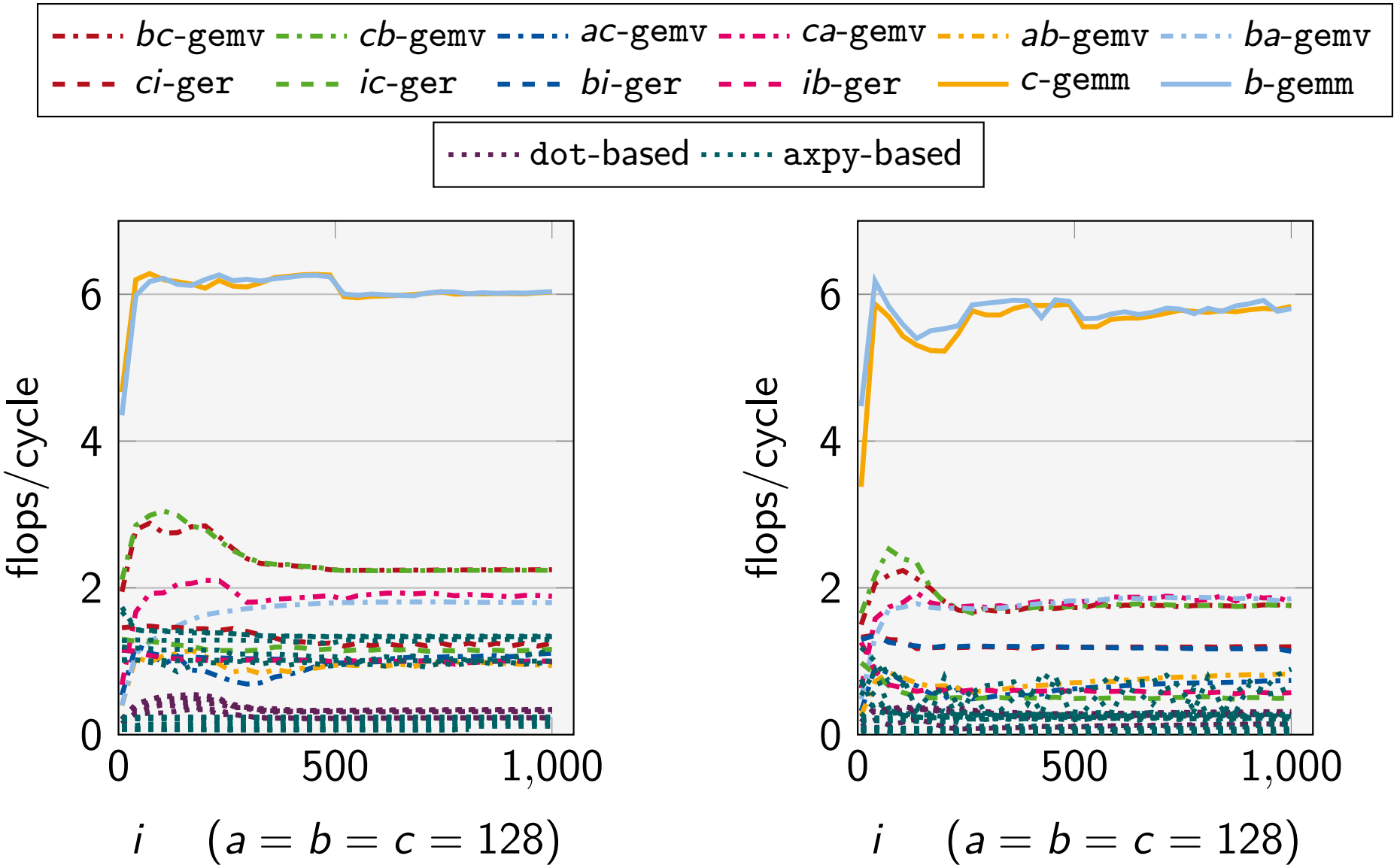
- $a = b = c = 128,\ i = 8 \ldots 1{,}000$



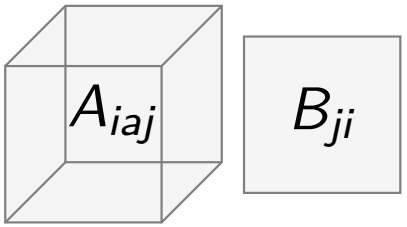
- Intel Ivy Bridge E5-2680 v2
- Single-threaded Intel MKL

$C_{abc} := A_{ai} B_{ibc}$ — Setup 2

Results



$C_a := A_{iaj} B_{ji}$ — 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

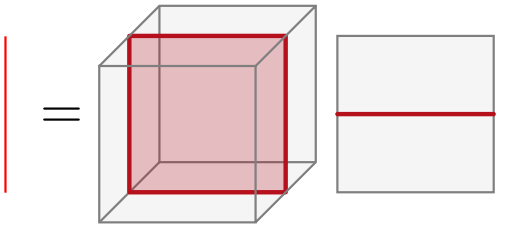
```
for j = 1:j
    C[:] += A[:,:,j] B[j,:]
```

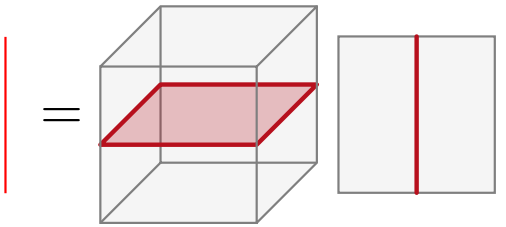
$C_a := A_{iaj} B_{ji}$ — i'-gemv

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

- 2 `axpy`-based:

  ij-`axpy` *ji*-`axpy`

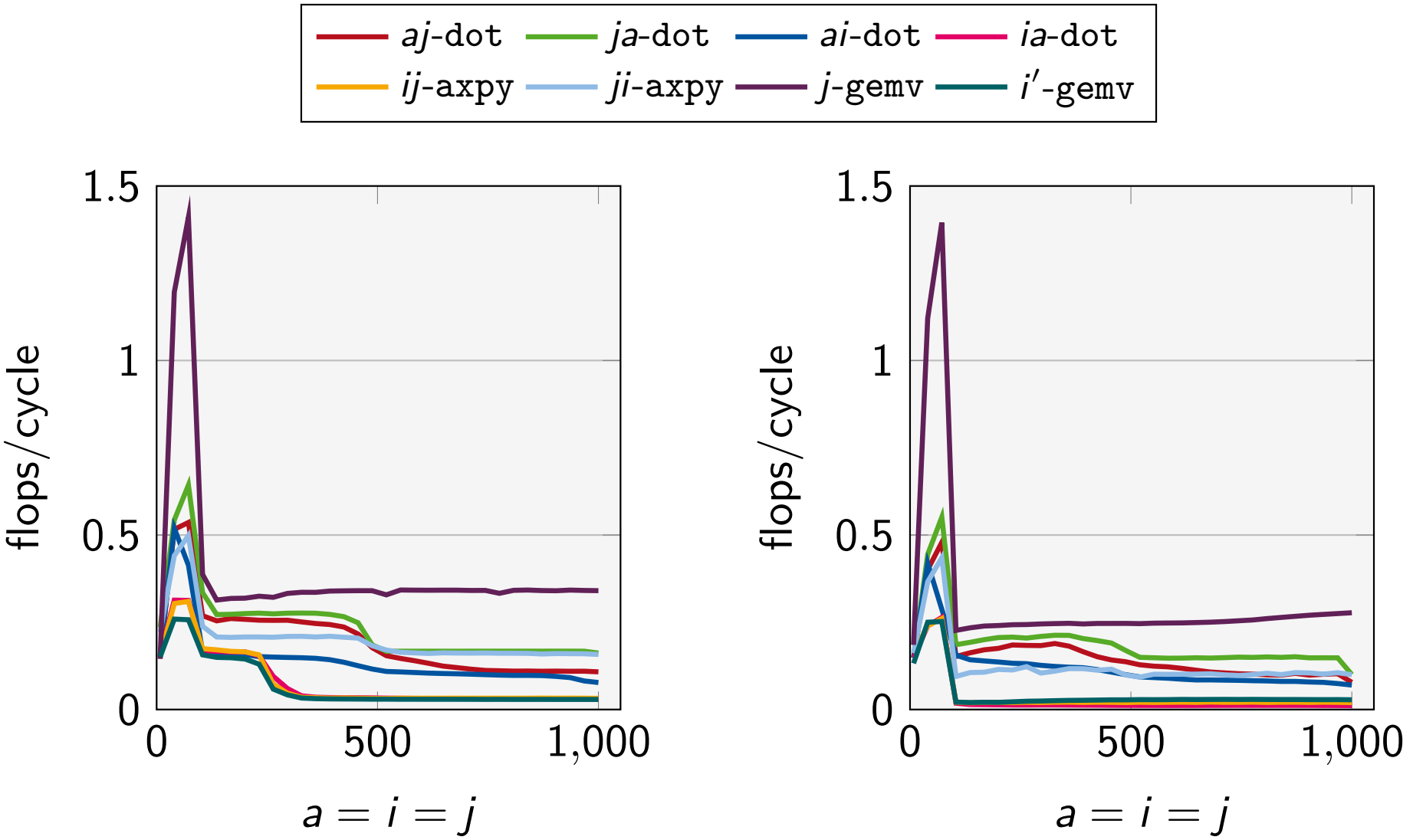




- $a = i = j = 8 \ldots 1{,}000$

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

Results

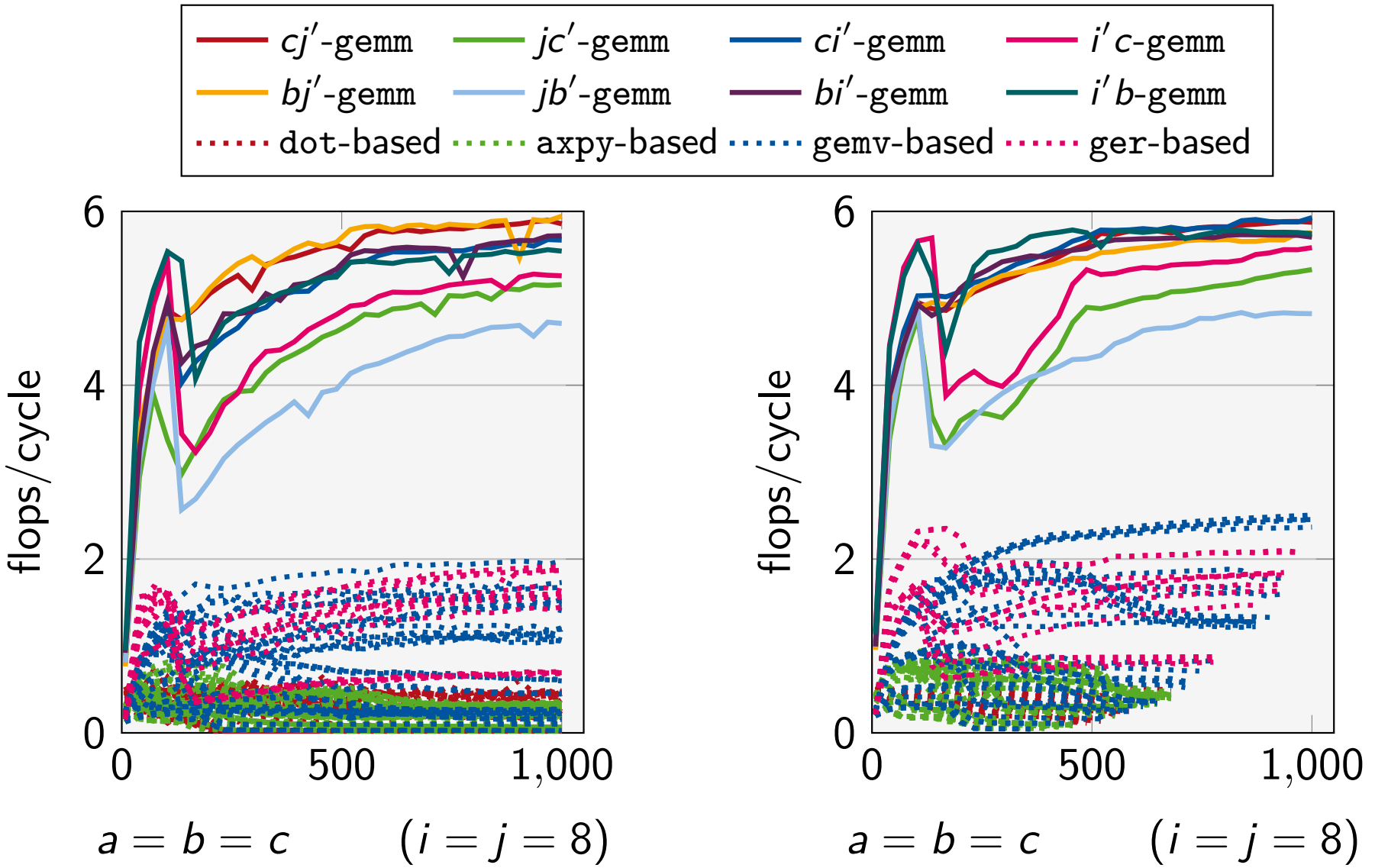


$C_{abc} := A_{aij} B_{jbic}$ — Challenging Contraction

- $C_{abc} := A_{aij} B_{jbic}$
  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
- $i = j = 8$, $a = b = c = 8 \ldots 1{,}000$
- Intel Ivy Bridge E5-2680 v2
- Single-threaded OpenBLAS

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

Results

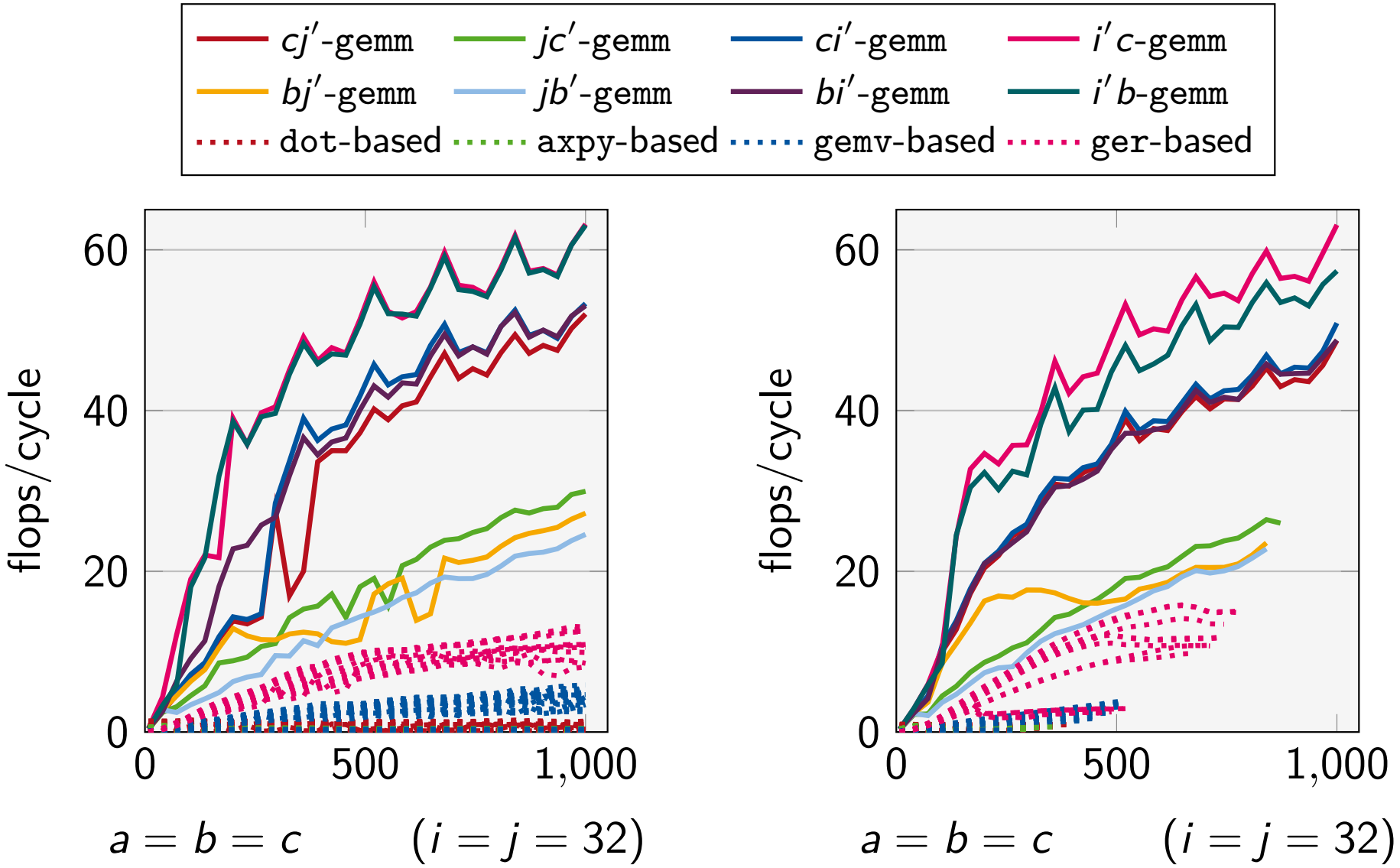


$C_{abc} := A_{aij} B_{jbic}$ — 10 Threads

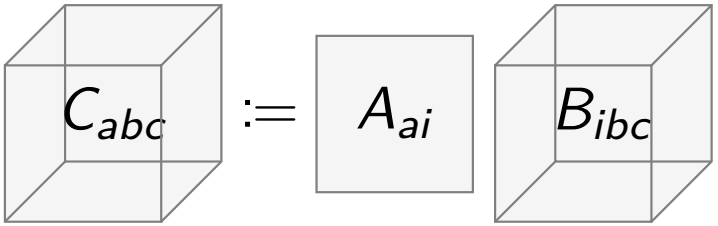
- $C_{abc} := A_{aij} B_{jbic}$
- $i = j = \mathbf{32}$, $a = b = c = 8 \ldots 1{,}000$
- Intel Ivy Bridge E5-2680 v2
- OpenBLAS, 10 threads

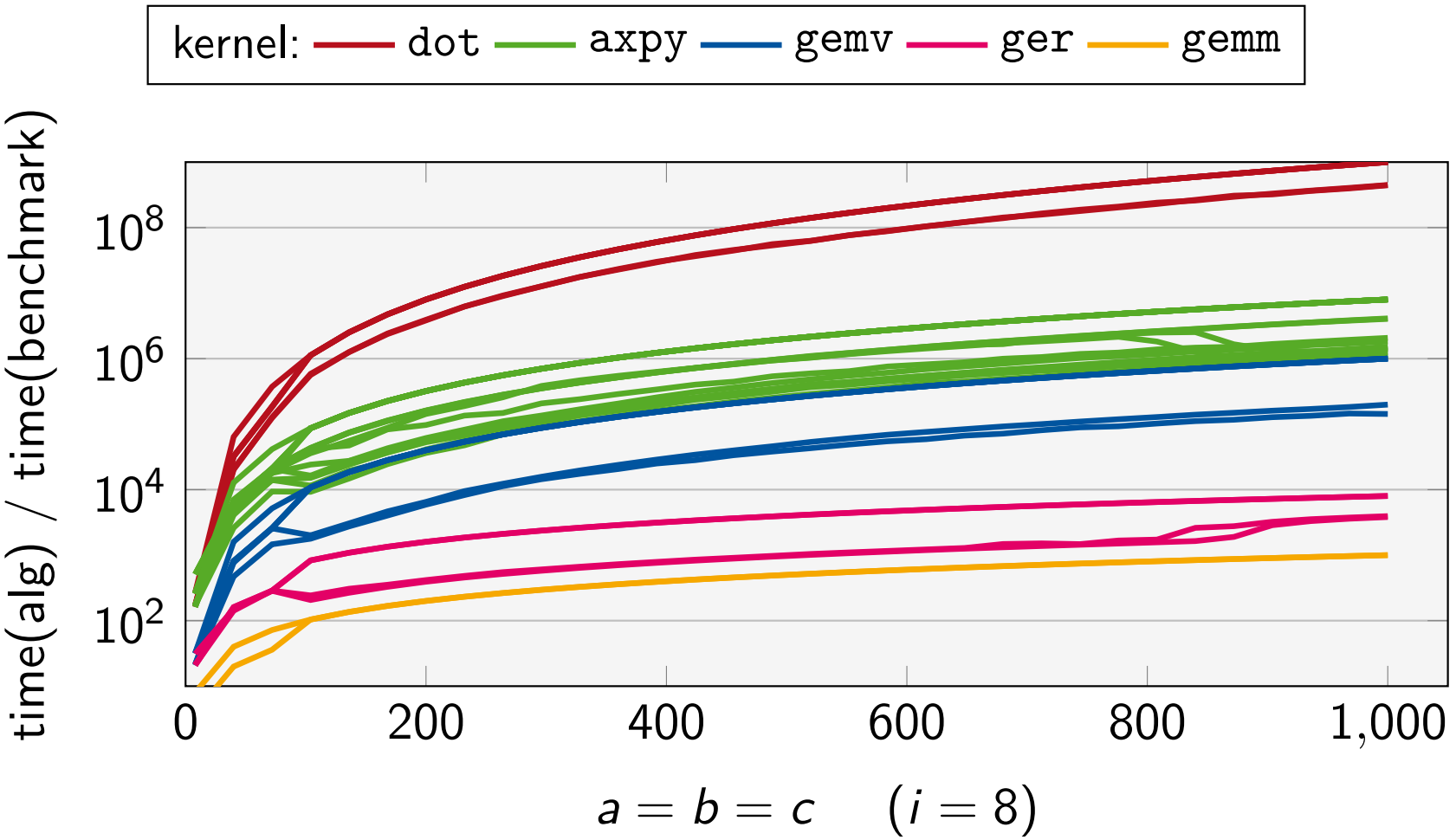
$C_{abc} := A_{aij} B_{jbic}$ — 10 Threads

Results



Prediction Efficiency





Estimating the Efficiency 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.