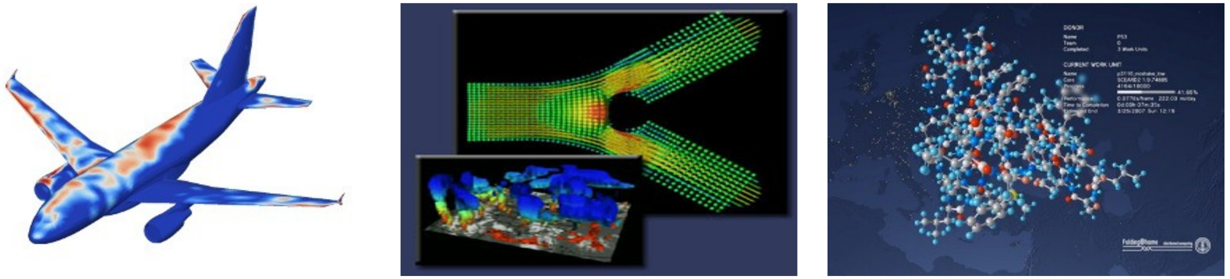
Adding new Flavours to GEMM: ARM big.LITTLE Architectures and Fault Tolerance

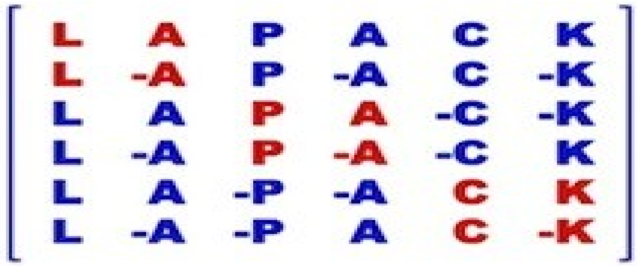
Enrique S. QUINTANA-ORTÍ

UNIVERSITAT JAUME·I

Motivation

- Why GEMM?





BLAS

GEMM

Outline

- High performance GEMM (sequential and multi-threaded)
- GEMM for asymmetric processors: ARM big.LITTLE
- Fault tolerance (and approximate computing) in GEMM

High Performance GEMM

- Commercial libraries for BLAS

MKL

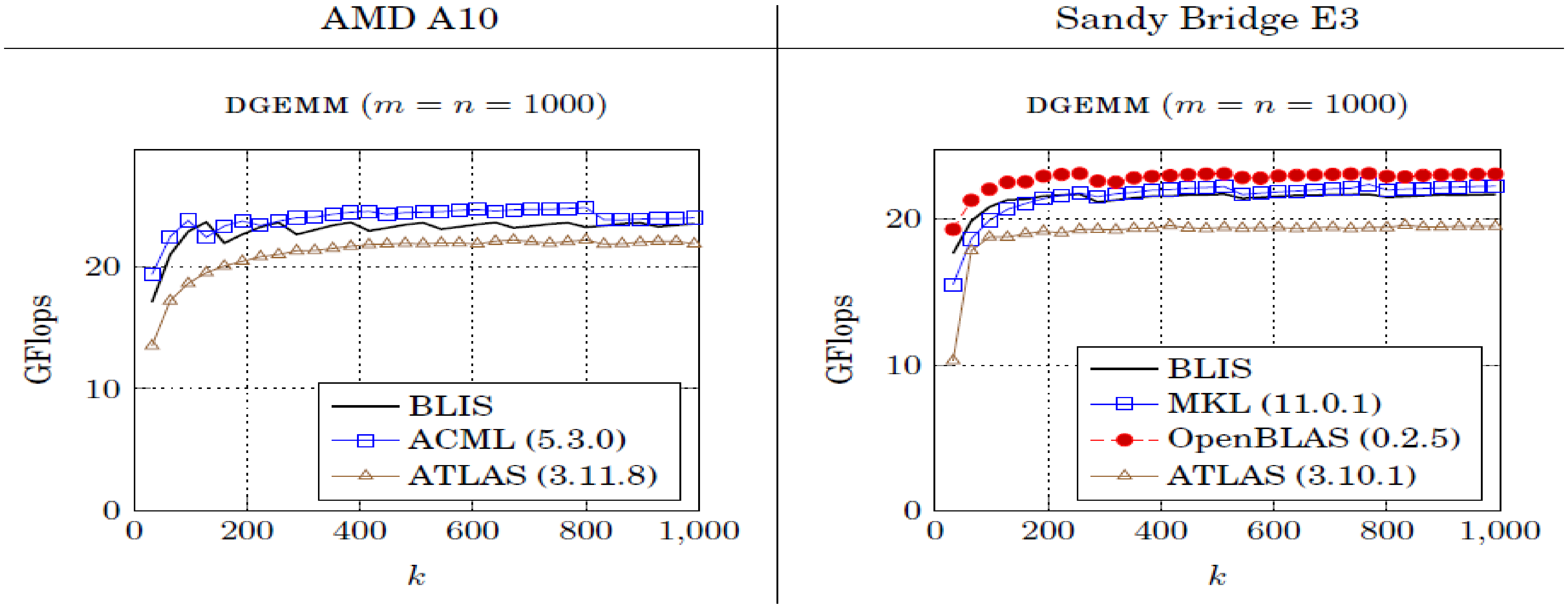
ACML

ESSL

CUBLAS

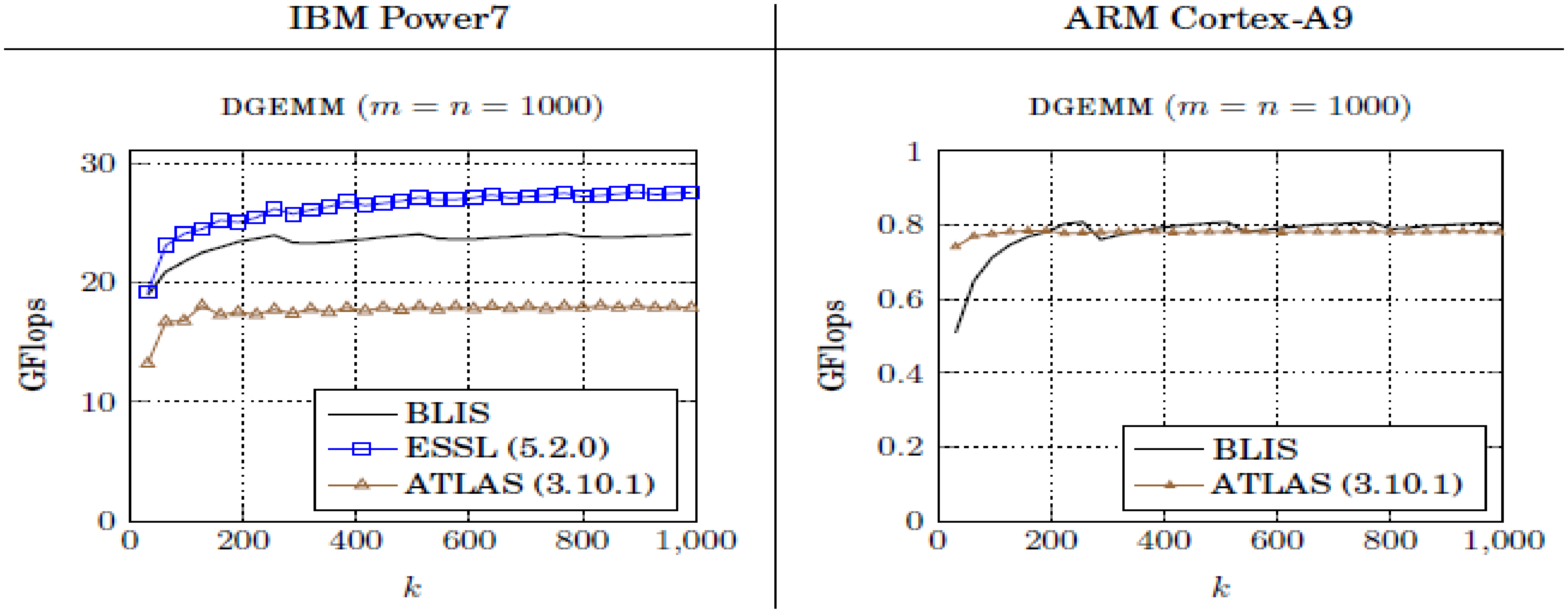
High Performance GEMM

- “Open” sw.: GotoBLAS, ATLAS, OpenBLAS, BLIS



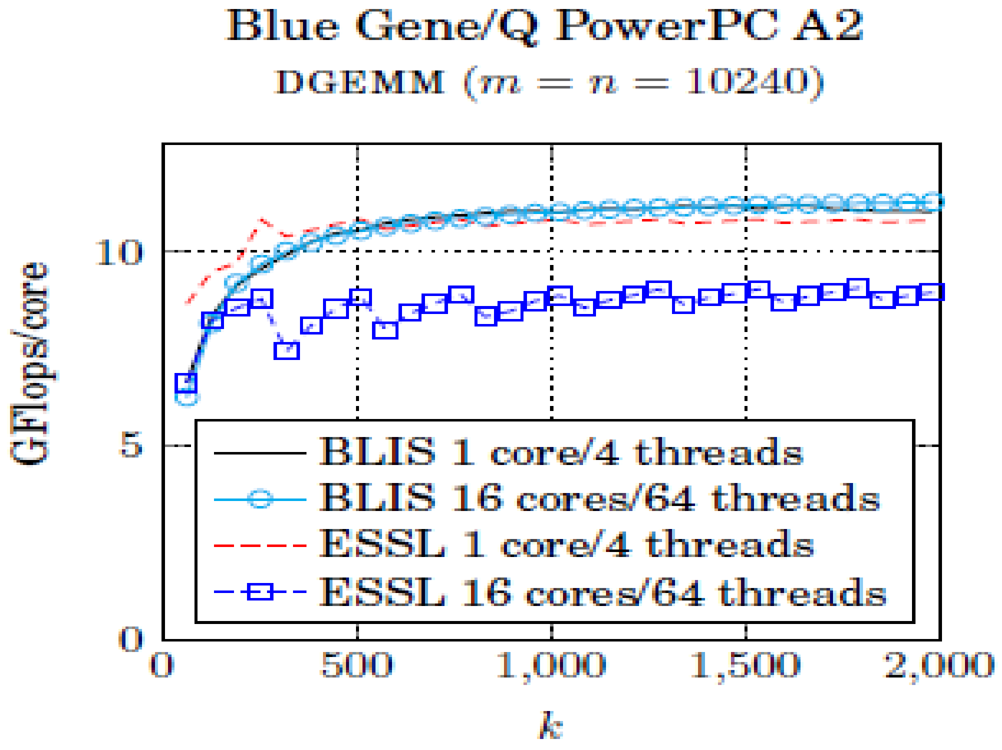
High Performance GEMM

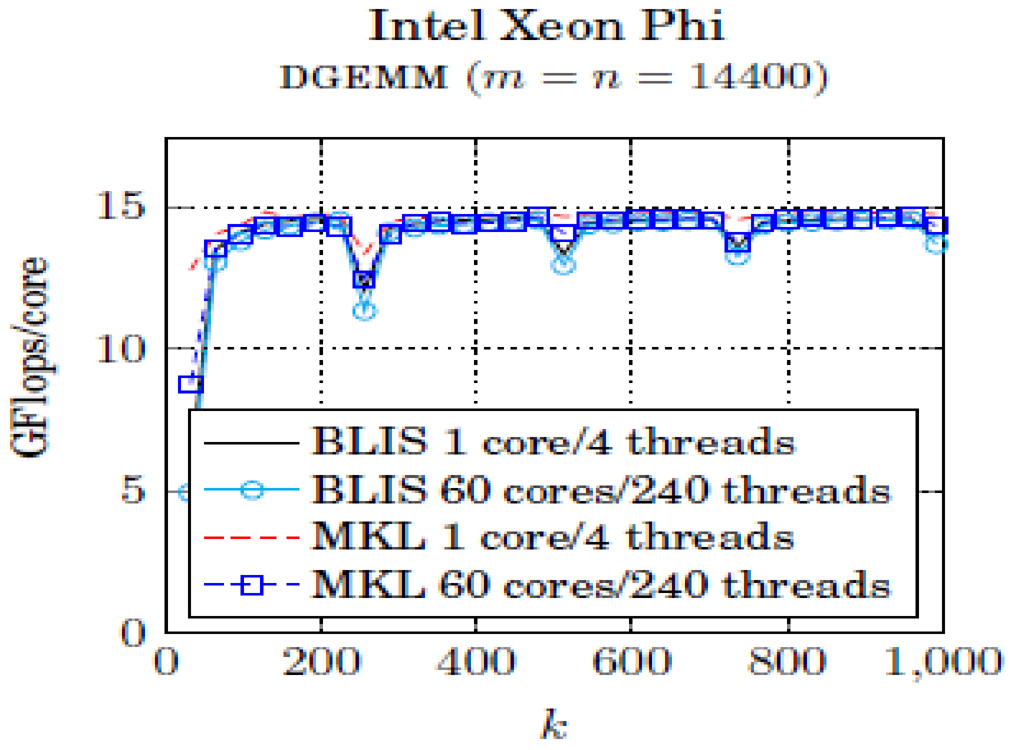
- “Open” sw.: GotoBLAS, ATLAS, OpenBLAS, BLIS



High Performance GEMM

- "Open" sw.: GotoBLAS, ATLAS, OpenBLAS, BLIS



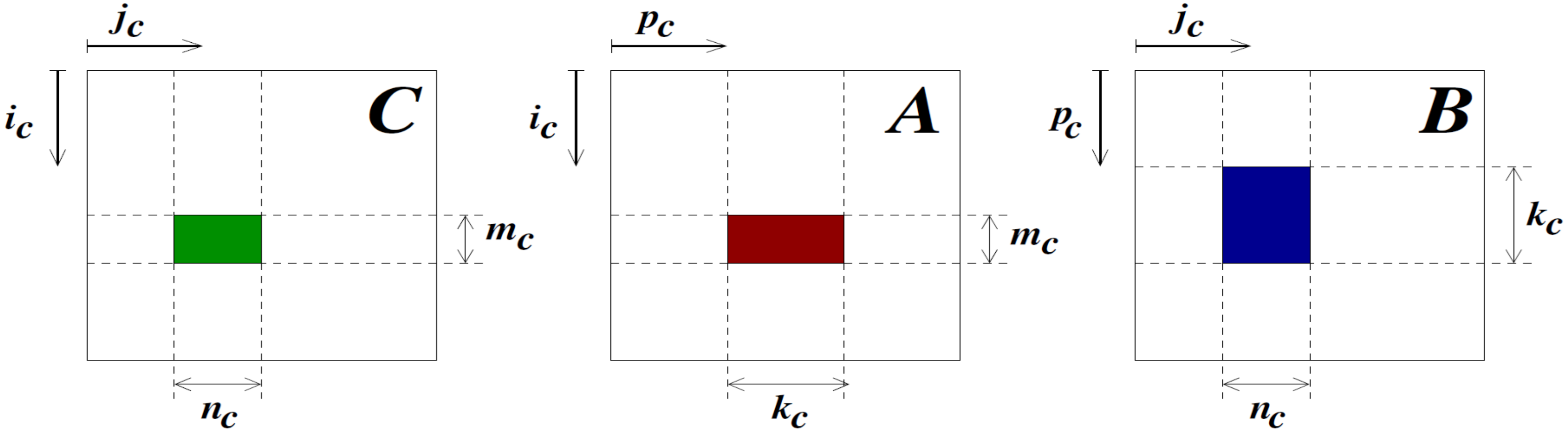


High Performance GEMM

- BLIS
  - Software framework for instantiating high-performance BLAS-like dense linear algebra libraries
  - New/modified/3-clause BSD license
  - https://code.google.com/p/blis/

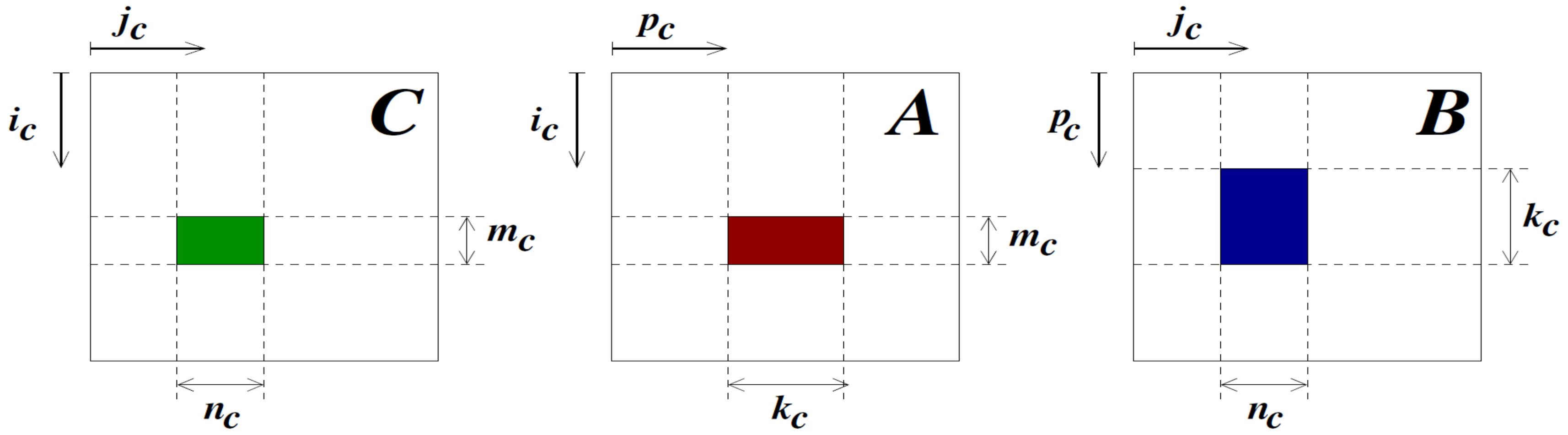
High Performance GEMM
BLIS

```
Loop 1:  for $j_c = 0, \ldots, n-1$ in steps of $n_c$
Loop 2:    for $p_c = 0, \ldots, k-1$ in steps of $k_c$

Loop 3:      for $i_c = 0, \ldots, m-1$ in steps of $m_c$

               $C(i_c : i_c + m_c - 1, j_c : j_c + n_c - 1) \equiv C_c \mathrel{+}= A_c \cdot B_c$   // Macro-kernel
             endfor
           endfor
         endfor
```

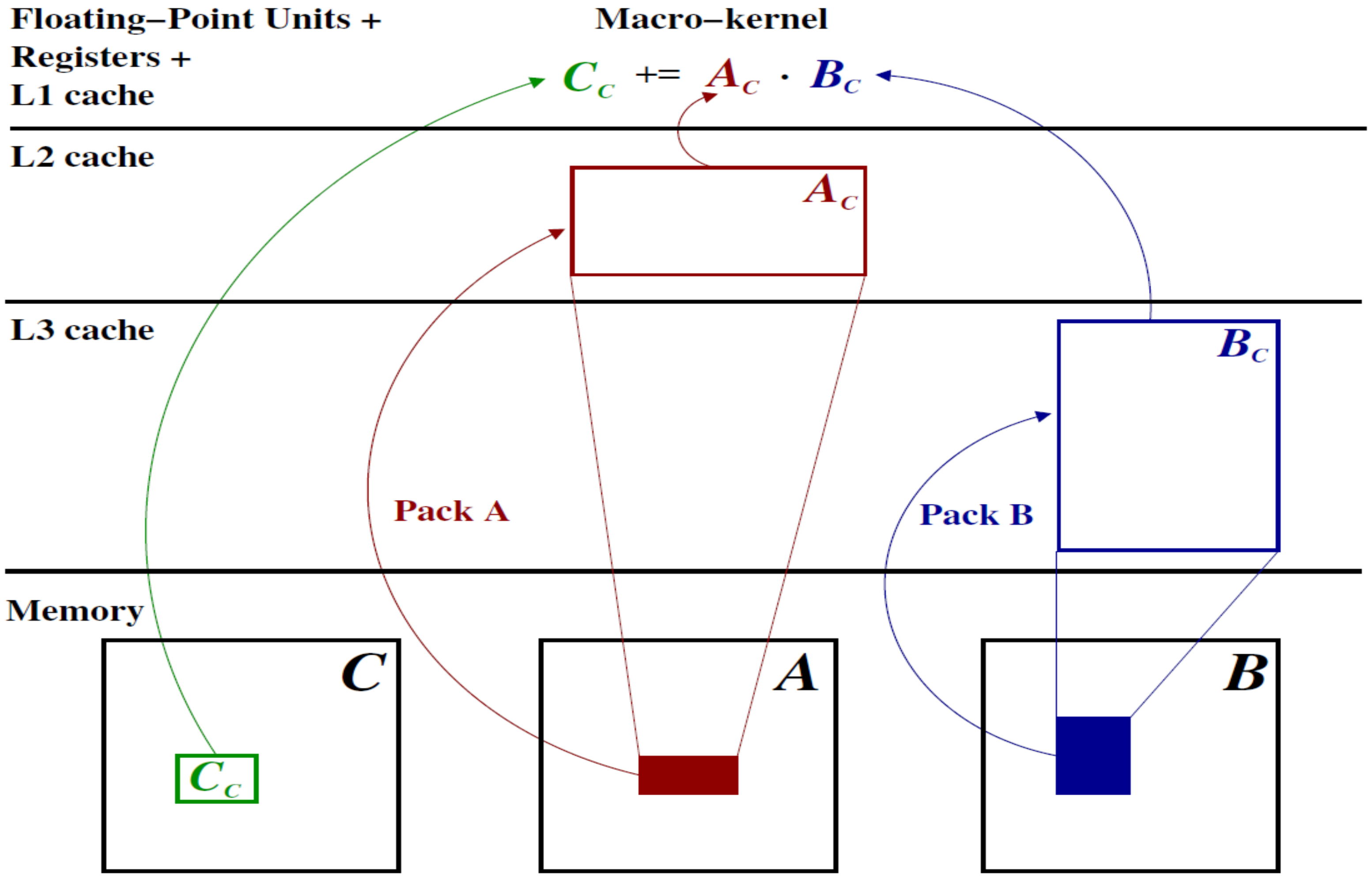


High Performance GEMM
BLIS

```
Loop 1:  for $j_c = 0, \ldots, n-1$ in steps of $n_c$
Loop 2:    for $p_c = 0, \ldots, k-1$ in steps of $k_c$
             $B(p_c : p_c + k_c - 1, j_c : j_c + n_c - 1) \rightarrow B_c$                        // Pack into $B_c$
Loop 3:      for $i_c = 0, \ldots, m-1$ in steps of $m_c$
               $A(i_c : i_c + m_c - 1, p_c : p_c + k_c - 1) \rightarrow A_c$                      // Pack into $A_c$
               $C(i_c : i_c + m_c - 1, j_c : j_c + n_c - 1) \equiv C_c \mathrel{+}= A_c \cdot B_c$    // Macro-kernel
             endfor
           endfor
         endfor
```

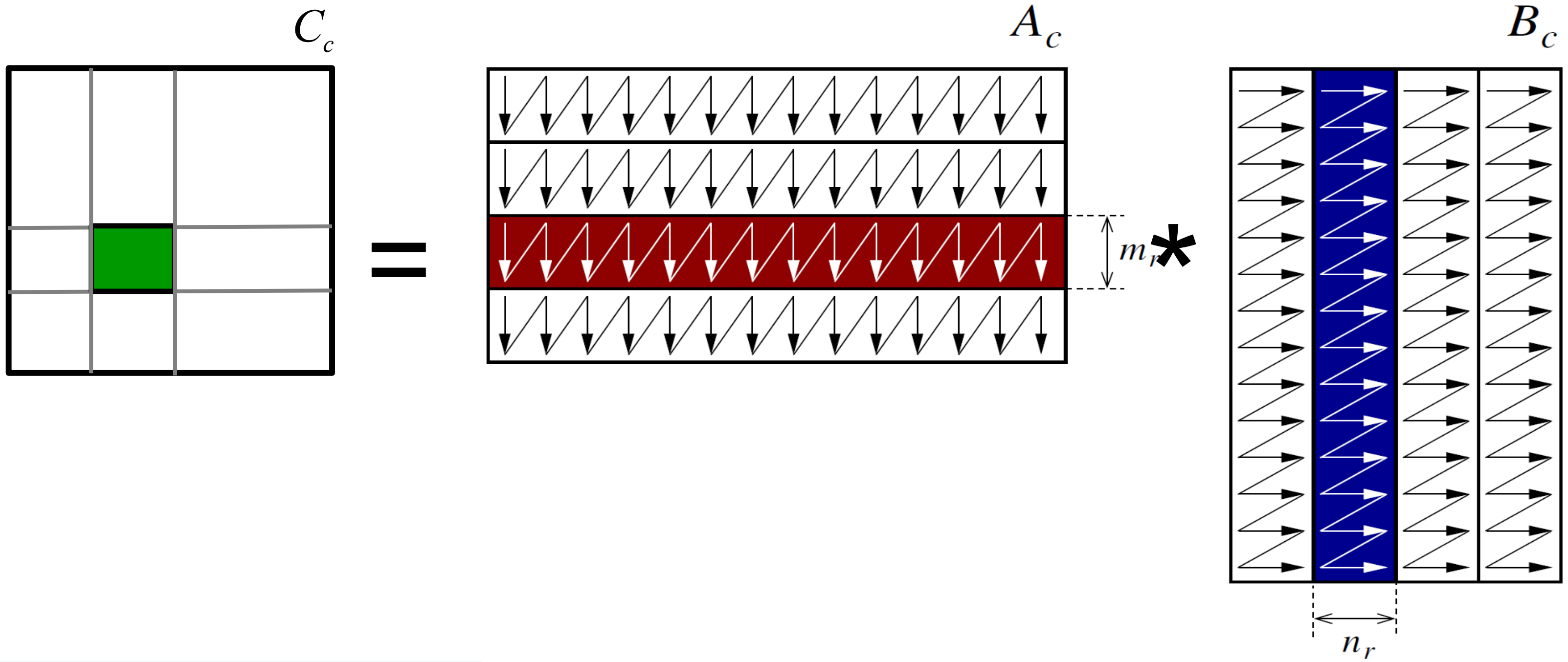


High Performance GEMM
BLIS

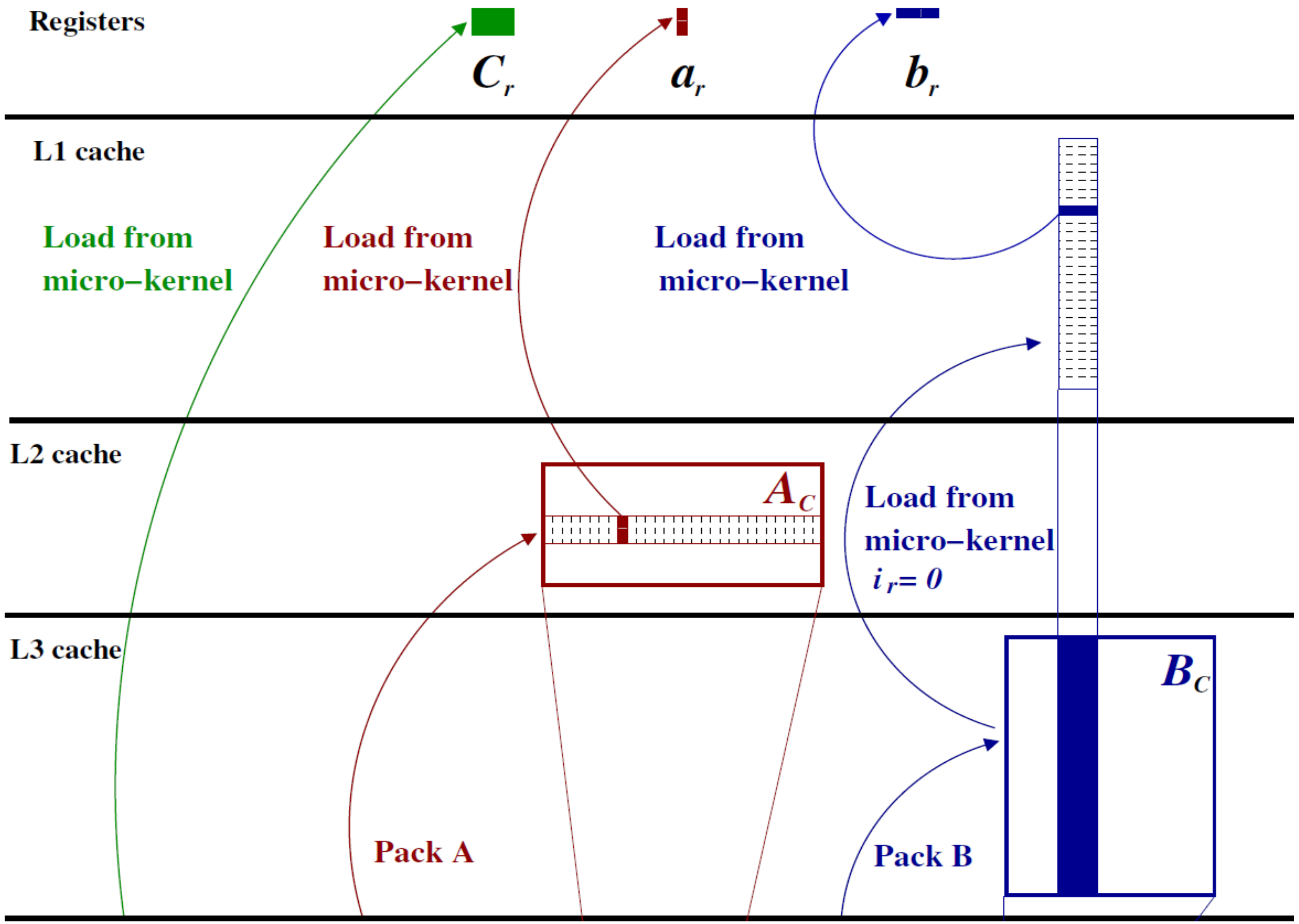


High Performance GEMM
BLIS

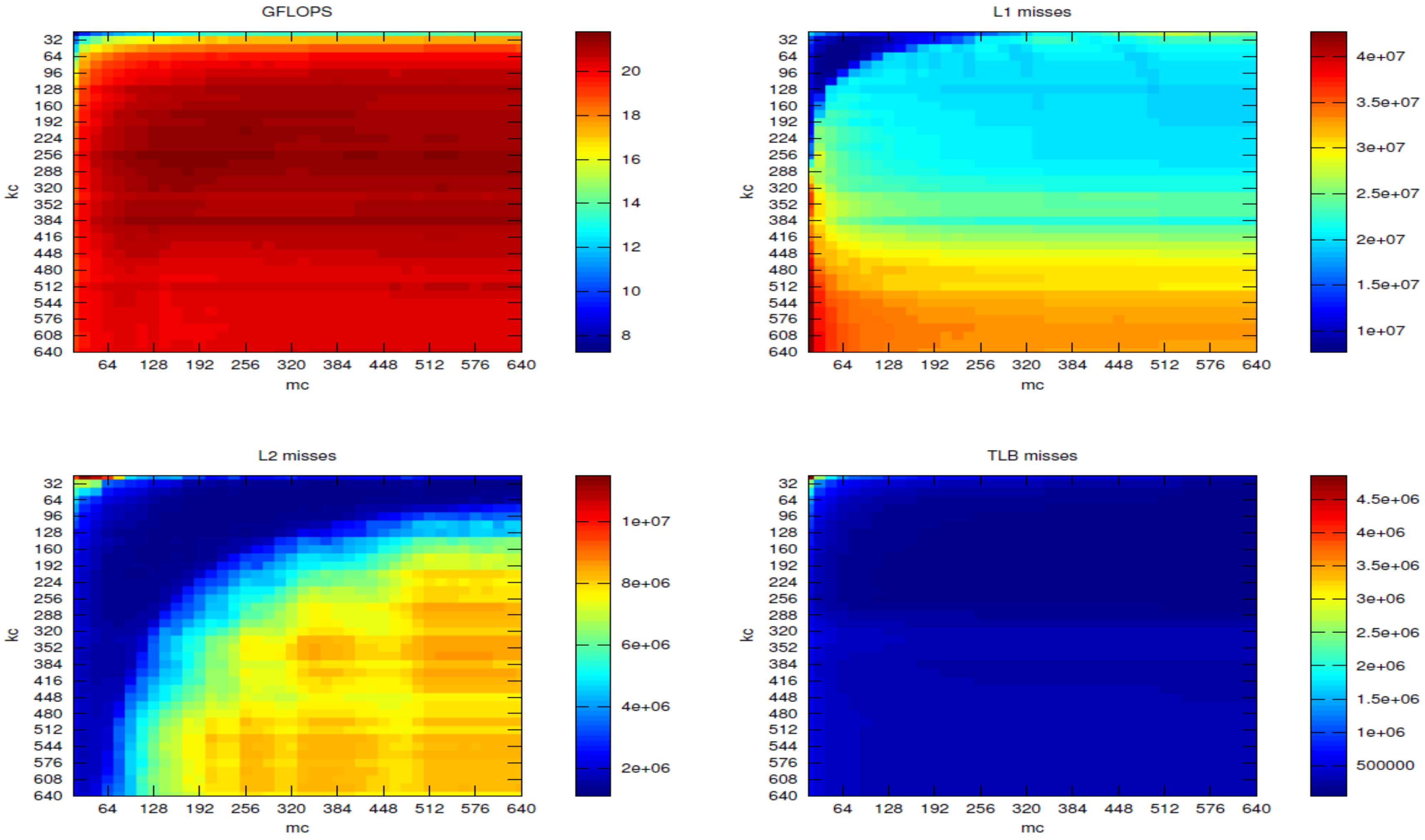
```
Loop 4    for $j_r = 0, \ldots, n_c - 1$ in steps of $n_r$                    // Macro-kernel
Loop 5      for $i_r = 0, \ldots, m_c - 1$ in steps of $m_r$
Loop 6        for $p_r = 0, \ldots, k_c - 1$ in steps of $1$    // Micro-kernel
                $C_c(i_r : i_r + m_r - 1, j_r : j_r + n_r - 1)$
                  $\mathrel{+}= A_c(i_r : i_r + m_r - 1, p_r)$
                  $\cdot\ B_c(p_r, j_r : j_r + n_r - 1)$
              endfor
            endfor
          endfor
```



High Performance GEMM
BLIS



High Performance GEMM
BLIS



High Performance GEMM
BLIS

- How to choose optimal blocking/register tiling parameters?

$$m_c, n_c, k_c, m_r, n_r$$

  - Experimentally

“BLIS: A Framework for Rapid Instantiation of BLAS Functionality”
F. G. Van Zee, R. A. van de Geijn
ACM Transactions on Mathematical Software (TOMS), Vol. 41(3), 2015
http://www.cs.utexas.edu/users/flame

  - Analytically

“Analytical Modeling is Enough for High Performance BLIS”
T. M. Low, F. D. Igual, T. M. Smith, E. S. Quintana-Ortí
Submitted to ACM TOMS. (See also FLAWN 74)
http://www.cs.utexas/edu/users/flame

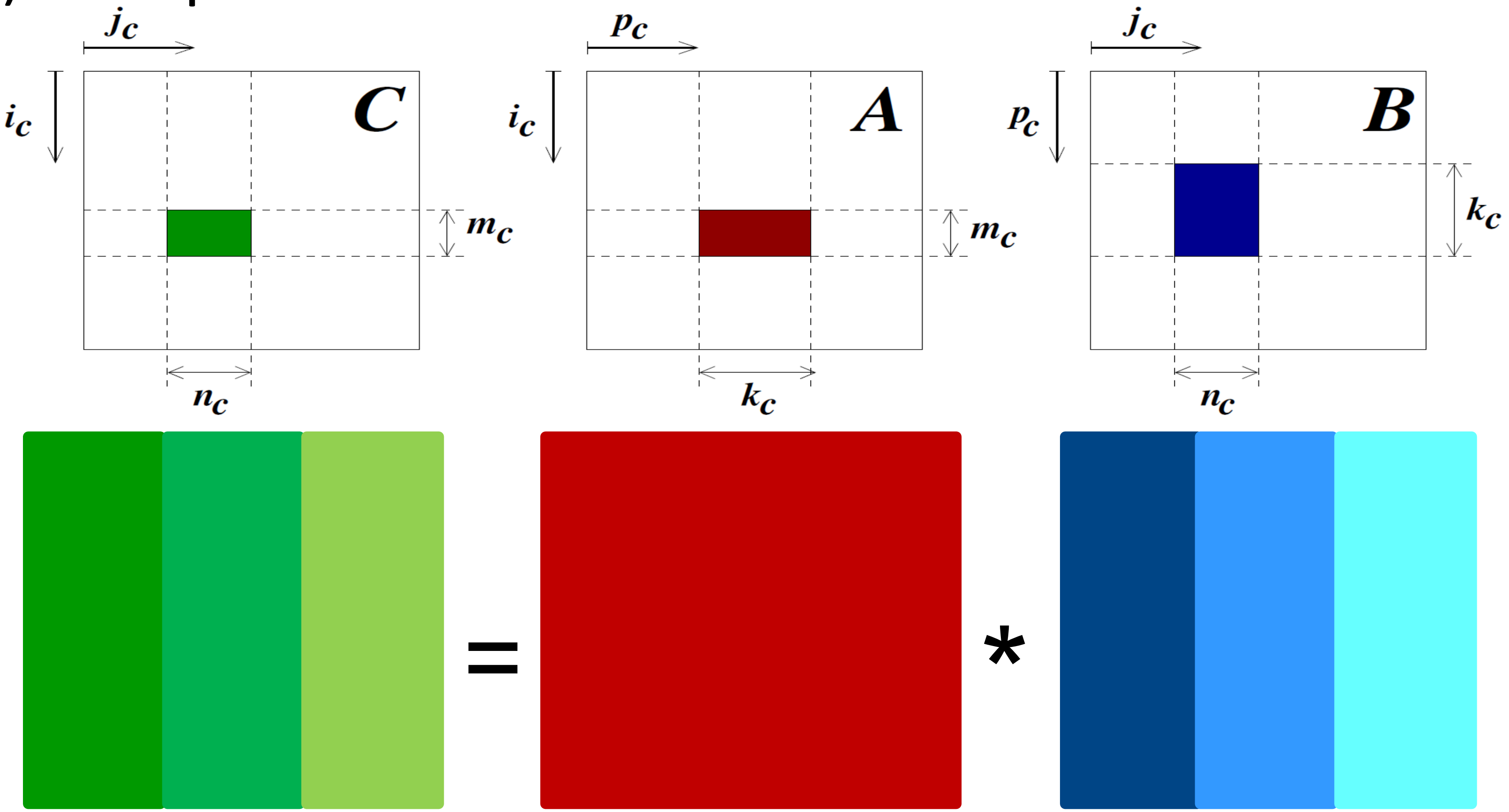
High Performance GEMM
BLIS

- Multi-threaded for multicore. Which loop(s) to target?

```
Loop 1  for $j_c = 0, \dots, n-1$ in steps of $n_c$
Loop 2    for $p_c = 0, \dots, k-1$ in steps of $k_c$
            $B(p_c : p_c + k_c - 1, j_c : j_c + n_c - 1) \rightarrow B_c$          // Pack into $B_c$
Loop 3      for $i_c = 0, \dots, m-1$ in steps of $m_c$
              $A(i_c : i_c + m_c - 1, p_c : p_c + k_c - 1) \rightarrow A_c$       // Pack into $A_c$
Loop 4        for $j_r = 0, \dots, n_c - 1$ in steps of $n_r$                      // Macro-kernel
Loop 5          for $i_r = 0, \dots, m_c - 1$ in steps of $m_r$
Loop 6            for $p_r = 0, \dots, k_c - 1$ in steps of 1    // Micro-kernel
                    $C_c(i_r : i_r + m_r - 1, j_r : j_r + n_r - 1)$
                      $\mathrel{+}= A_c(i_r : i_r + m_r - 1, p_r)$
                      $\cdot \; B_c(p_r, j_r : j_r + n_r - 1)$
                  endfor
                endfor
              endfor
            endfor
          endfor
        endfor
```

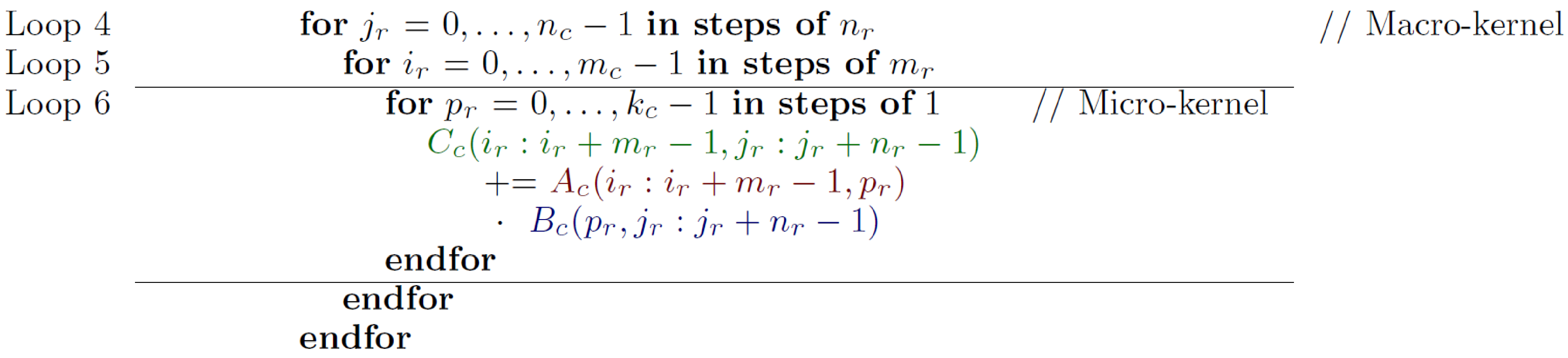
High Performance GEMM
BLIS

- Loop 1 (j_c): Independent GEMMs for multi-socket



High Performance GEMM
BLIS

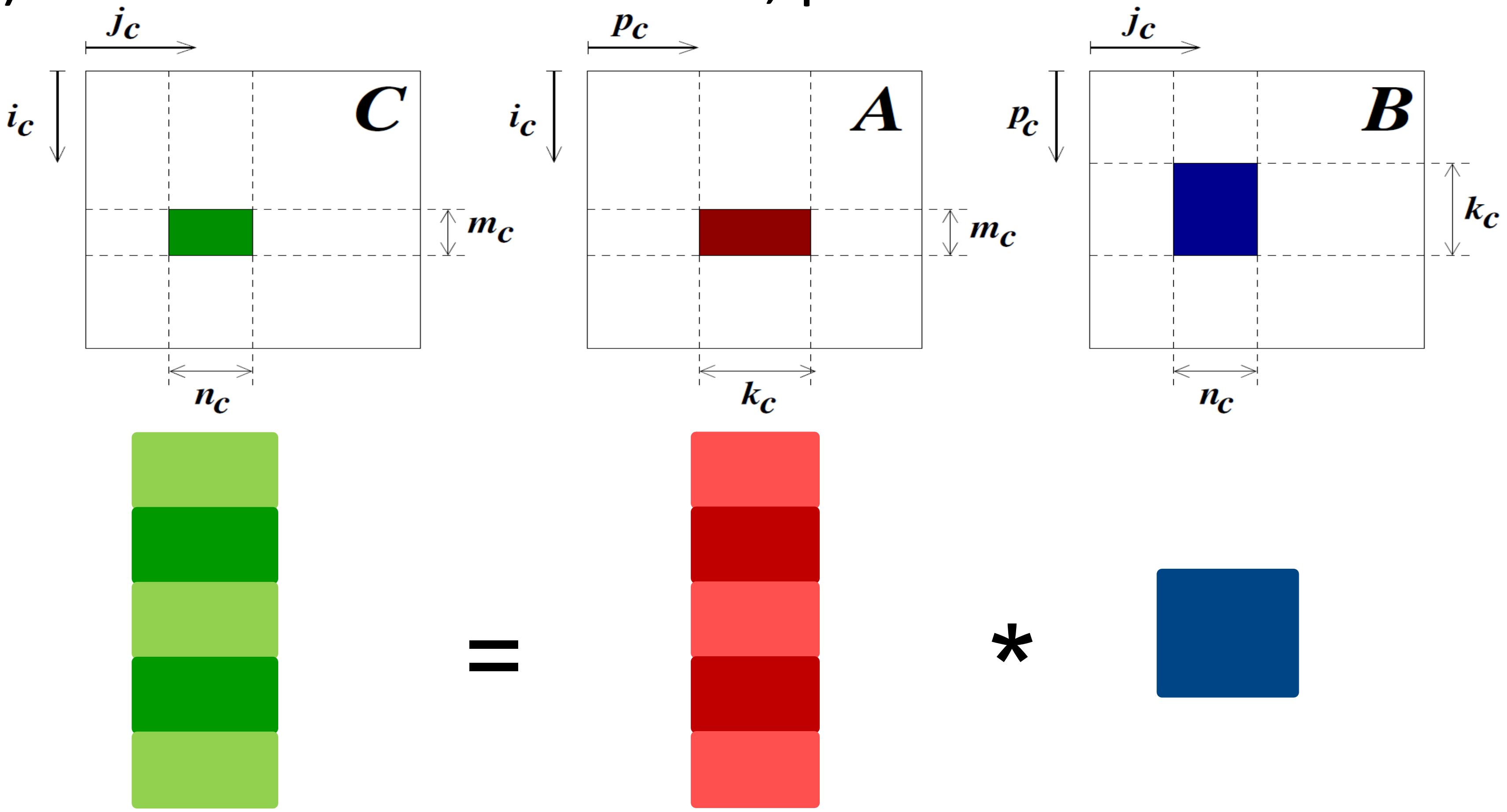
- Loop 2 (also loop 6): race conditions!



```
Loop 4    for $j_r = 0, \ldots, n_c - 1$ in steps of $n_r$                        // Macro-kernel
Loop 5      for $i_r = 0, \ldots, m_c - 1$ in steps of $m_r$
Loop 6        for $p_r = 0, \ldots, k_c - 1$ in steps of $1$          // Micro-kernel
                $C_c(i_r : i_r + m_r - 1, j_r : j_r + n_r - 1)$
                  $\mathrel{+}= A_c(i_r : i_r + m_r - 1, p_r)$
                  $\cdot\ B_c(p_r, j_r : j_r + n_r - 1)$
              endfor
            endfor
          endfor
```

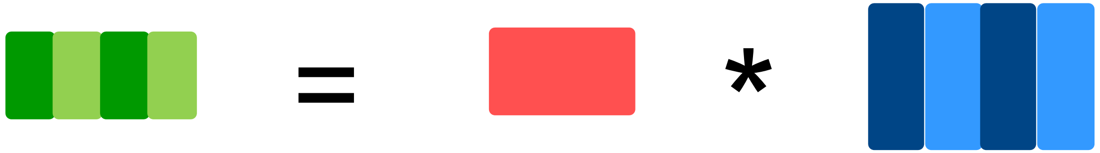
High Performance GEMM
BLIS

- Loop 3 (i_c): multicore with shared L3, private L2



High Performance GEMM
BLIS

- Loops 4 (j_r):
  - Replicated A_c if L2 cache is private. Single copy if shared
  - Single slice of B_c if L1 is private



- Loops 5 (i_r):
  - Fine-grained
  - Replicated slice of B_c in each (private) L1 cache

High Performance GEMM
BLIS

- What is the best combination for multicore/manycore architecture?

“Anatomy of High-Performance Many-Threaded Matrix Multiplication”
T. M. Smith, R. van de Geijn, M. Smelyanskiy, J. R. Hammond, F. G. Van Zee.
International Parallel and Distributed Processing Symposium - IPDPS, 2014
http://www.cs.utexas.edu/users/flame

Outline

- High performance GEMM (sequential and multi-threaded)
- GEMM for asymmetric processors

S. Catalán, R. Mayo,
R. Rodríguez Sánchez, E. S. Quintana-Ortí

UNIVERSITAT JAUME·I

F. D. Igual

UNIVERSIDAD COMPLUTENSE
MADRID

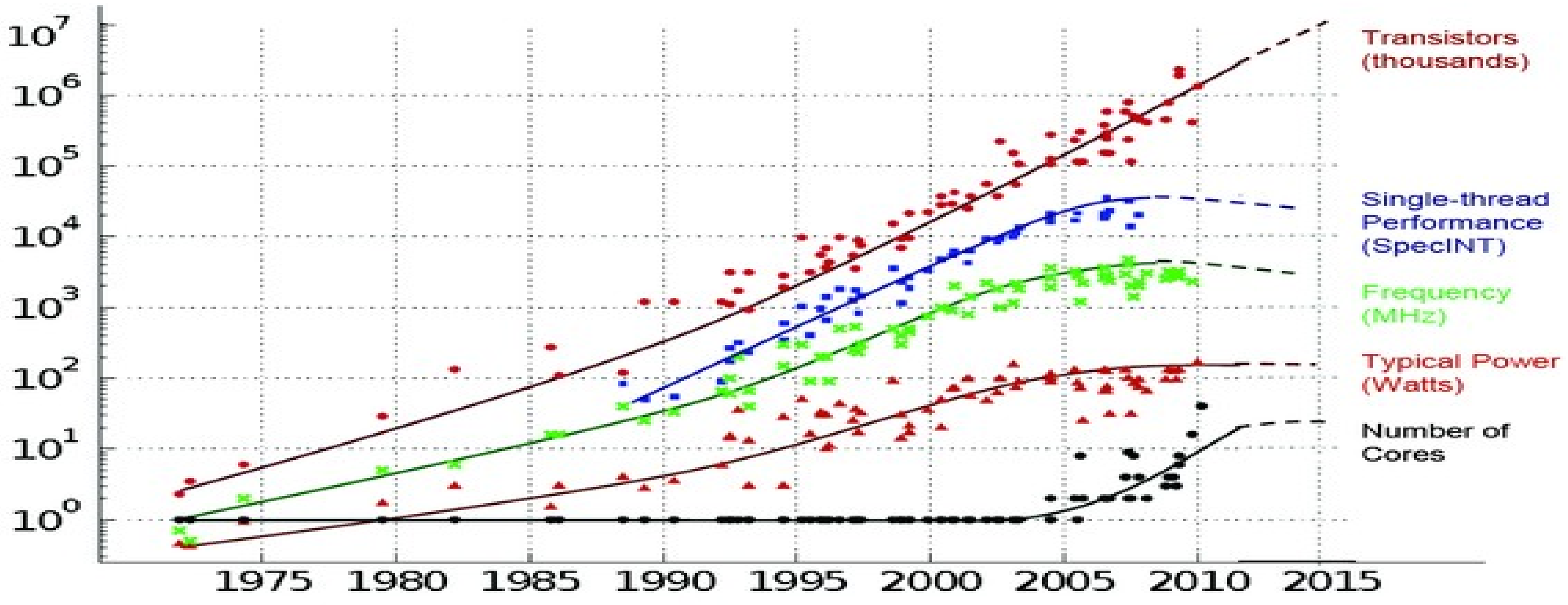
"Architecture-Aware Conguration and Scheduling of Matrix Multiplication on Asymmetric Multicore Processors"
S. Catalán, F. D. Igual, R. Mayo, R. Rodríguez-Sánchez, E. S. Quintana-Ortí
axXiv:1506.08988 [cs.PF], June 2015
Submitted to Parallel Computing

- Fault tolerance (and approximate computing) in GEMM

GEMM for Asymmetric Processors
Motivation

- Moore’s law is alive, but Dennard’s scaling is over

35 YEARS OF MICROPROCESSOR TREND DATA



Original data collected and plotted by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond and C. Batten
Dotted line extrapolations by C. Moore

GEMM for Asymmetric Processors
Motivation

- Welcome “dark silicon”: power/energy/utilization walls!

  …and asymmetric/heterogeneous architectures

Hardkernel Odroid XU3
Samsung Exynos5422
Cortex-A15 quad core + Cortex-A7 quad core
(sorry, also tiny GPU)

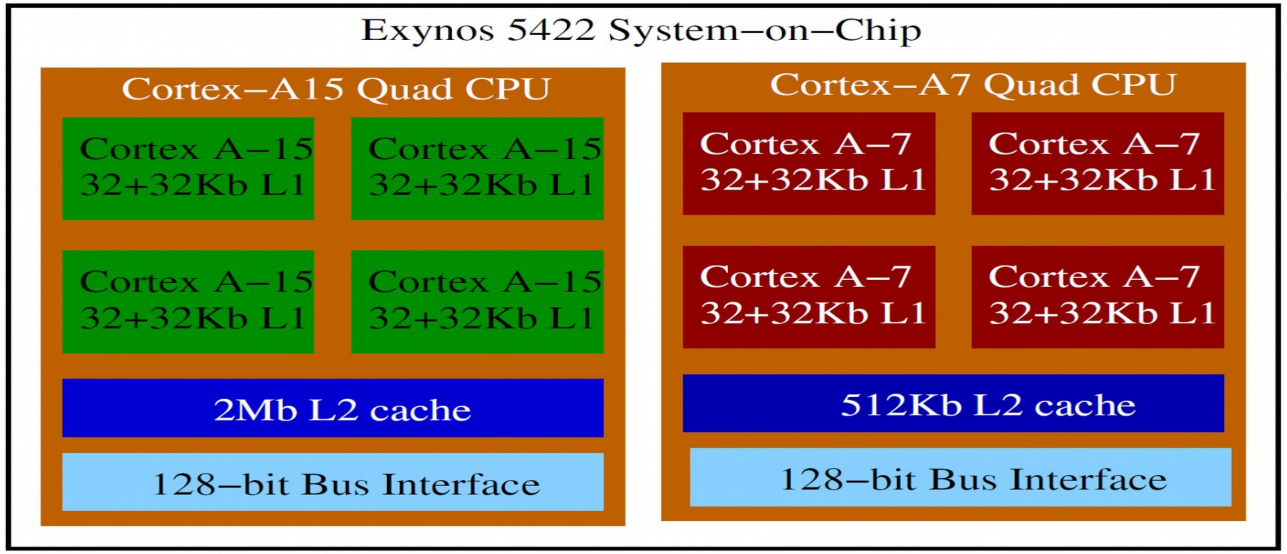
NVIDIA Jetson TK1 Devkit
Kepler GPU with 192 CUDA cores
4-Plus-1 quad-core ARM Cortex A15 CPU

GEMM for Asymmetric Processors
Target architecture

- Samsung Exynos5422
  - Twp clusters, with A7/A15 cores that share the same ISA…
  - but feature very different computational capacity

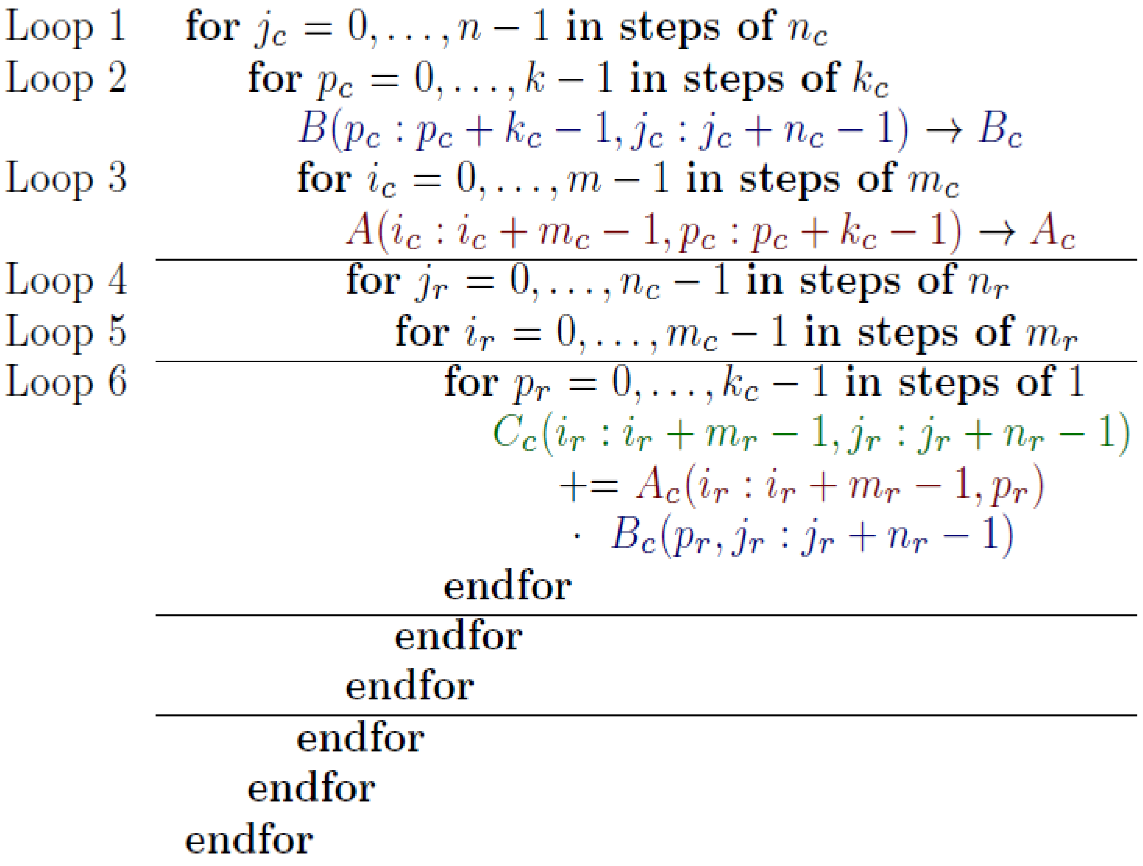
GEMM for Asymmetric Processors
Target architecture

- Samsung Exynos5422
  - Private L1 cache per core
  - Private L2 cache per cluster (shared among cores in the same cluster)
  - No L3 cache



GEMM for Asymmetric Processors

- Scheduling for multi-threaded asymmetric architecture?

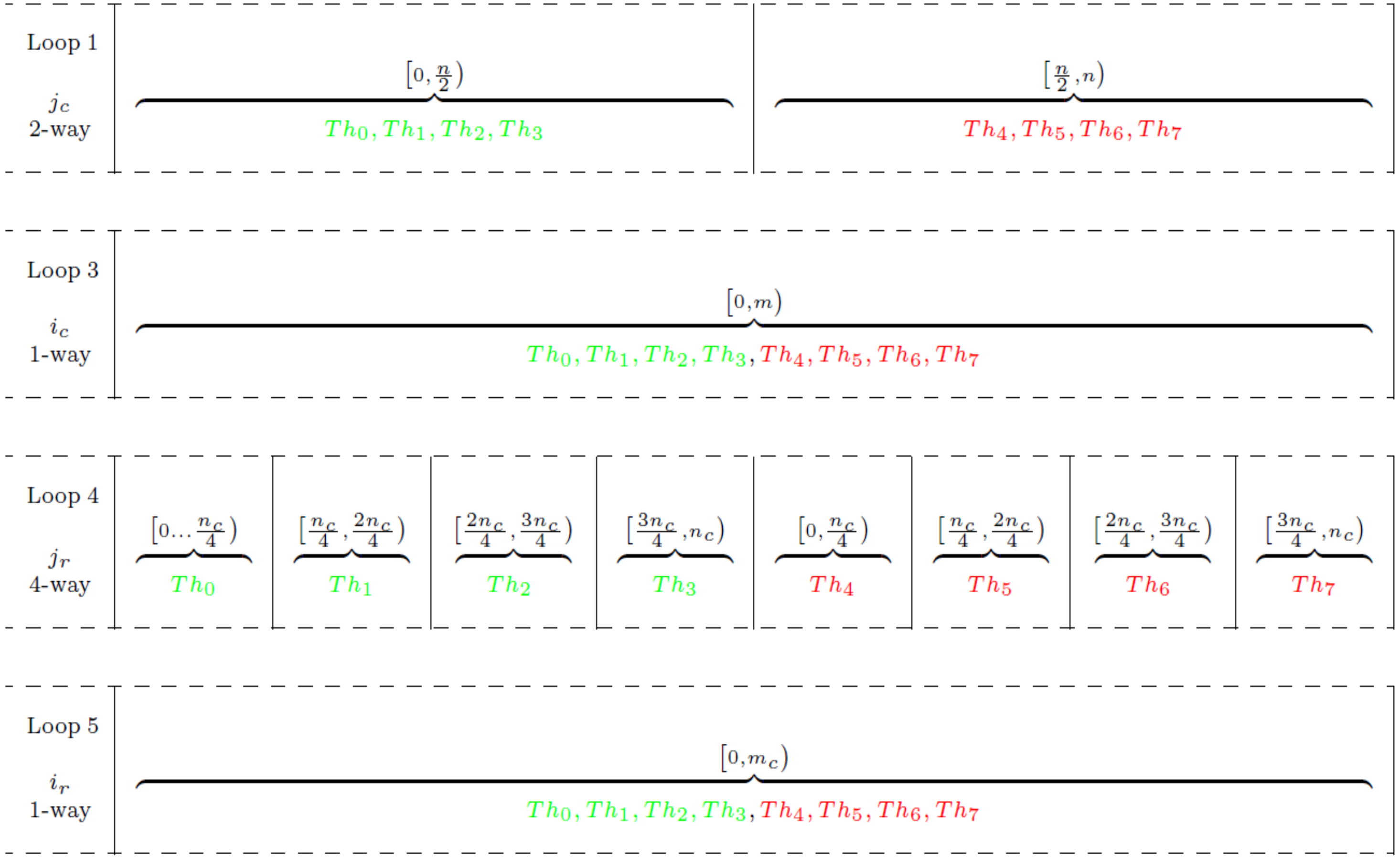


```
Loop 1  for $j_c = 0, \ldots, n-1$ in steps of $n_c$
Loop 2    for $p_c = 0, \ldots, k-1$ in steps of $k_c$
            $B(p_c : p_c + k_c - 1, j_c : j_c + n_c - 1) \rightarrow B_c$
Loop 3      for $i_c = 0, \ldots, m-1$ in steps of $m_c$
              $A(i_c : i_c + m_c - 1, p_c : p_c + k_c - 1) \rightarrow A_c$
Loop 4        for $j_r = 0, \ldots, n_c - 1$ in steps of $n_r$
Loop 5          for $i_r = 0, \ldots, m_c - 1$ in steps of $m_r$
Loop 6            for $p_r = 0, \ldots, k_c - 1$ in steps of $1$
                    $C_c(i_r : i_r + m_r - 1, j_r : j_r + n_r - 1)$
                        $+= A_c(i_r : i_r + m_r - 1, p_r)$
                        $\cdot \ B_c(p_r, j_r : j_r + n_r - 1)$
                  endfor
                endfor
              endfor
            endfor
          endfor
        endfor
```

GEMM for Asymmetric Processors

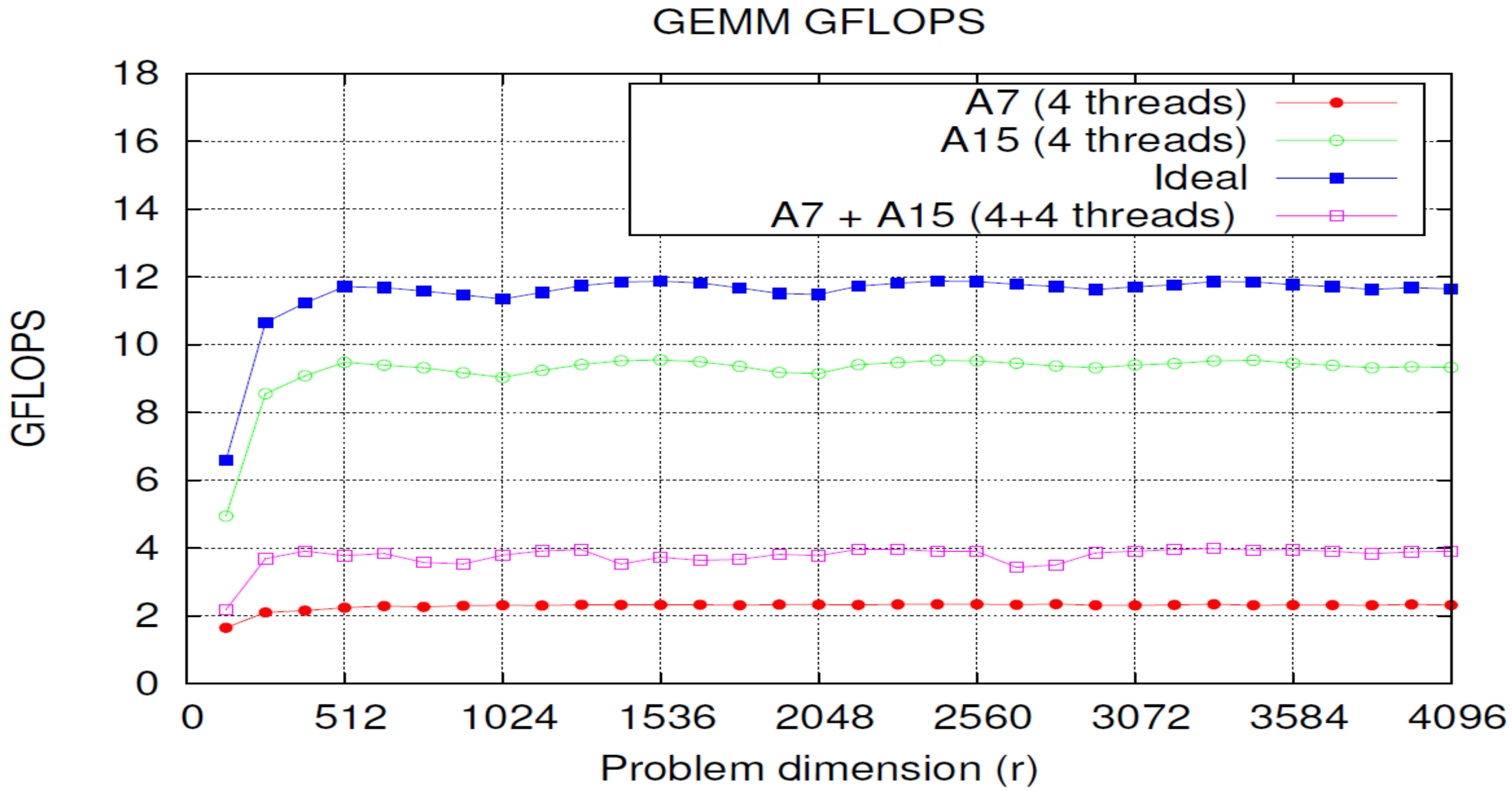
- Static symmetric scheduling between clusters

```
Loop 1   for $j_c = 0, \ldots, n-1$ in steps of $n_c$
Loop 2     for $p_c = 0, \ldots, k-1$ in steps of $k_c$
             $B(p_c : p_c + k_c - 1, j_c : j_c + n_c - 1) \rightarrow B_c$
Loop 3       for $i_c = 0, \ldots, m-1$ in steps of $m_c$
               $A(i_c : i_c + m_c - 1, p_c : p_c + k_c - 1) \rightarrow A_c$
Loop 4         for $j_r = 0, \ldots, n_c - 1$ in steps of $n_r$
Loop 5           for $i_r = 0, \ldots, m_c - 1$ in steps of $m_r$
Loop 6             for $p_r = 0, \ldots, k_c - 1$ in steps of 1
                     $C_c(i_r : i_r + m_r - 1, j_r : j_r + n_r - 1)$
                       $\mathrel{+}= A_c(i_r : i_r + m_r - 1, p_r)$
                       $\cdot\ B_c(p_r, j_r : j_r + n_r - 1)$
                   endfor
                 endfor
               endfor
             endfor
           endfor
         endfor
```



GEMM for Asymmetric Processors

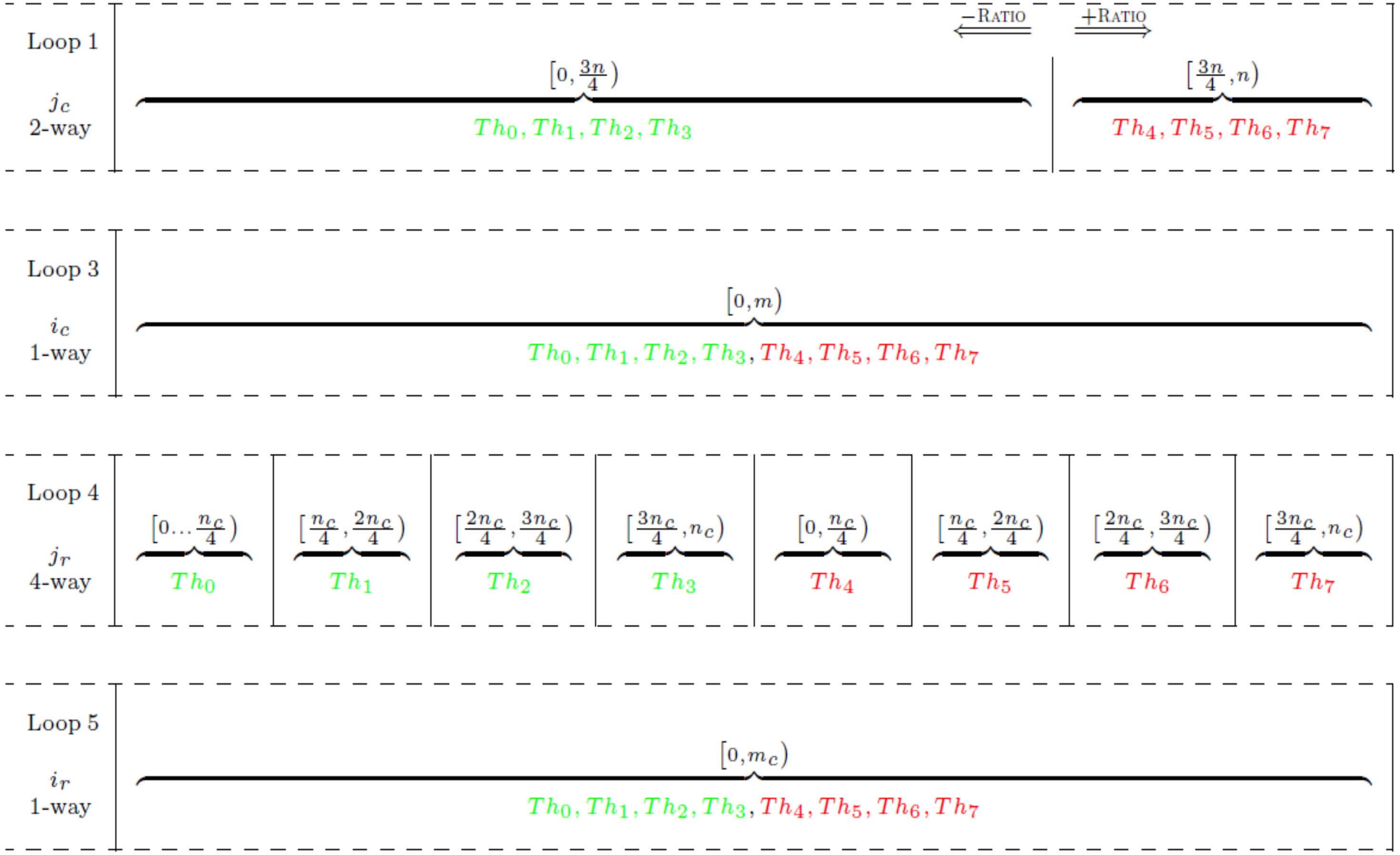
- Static symmetric scheduling between clusters



GEMM for Asymmetric Processors

- Static asymmetric scheduling between clusters

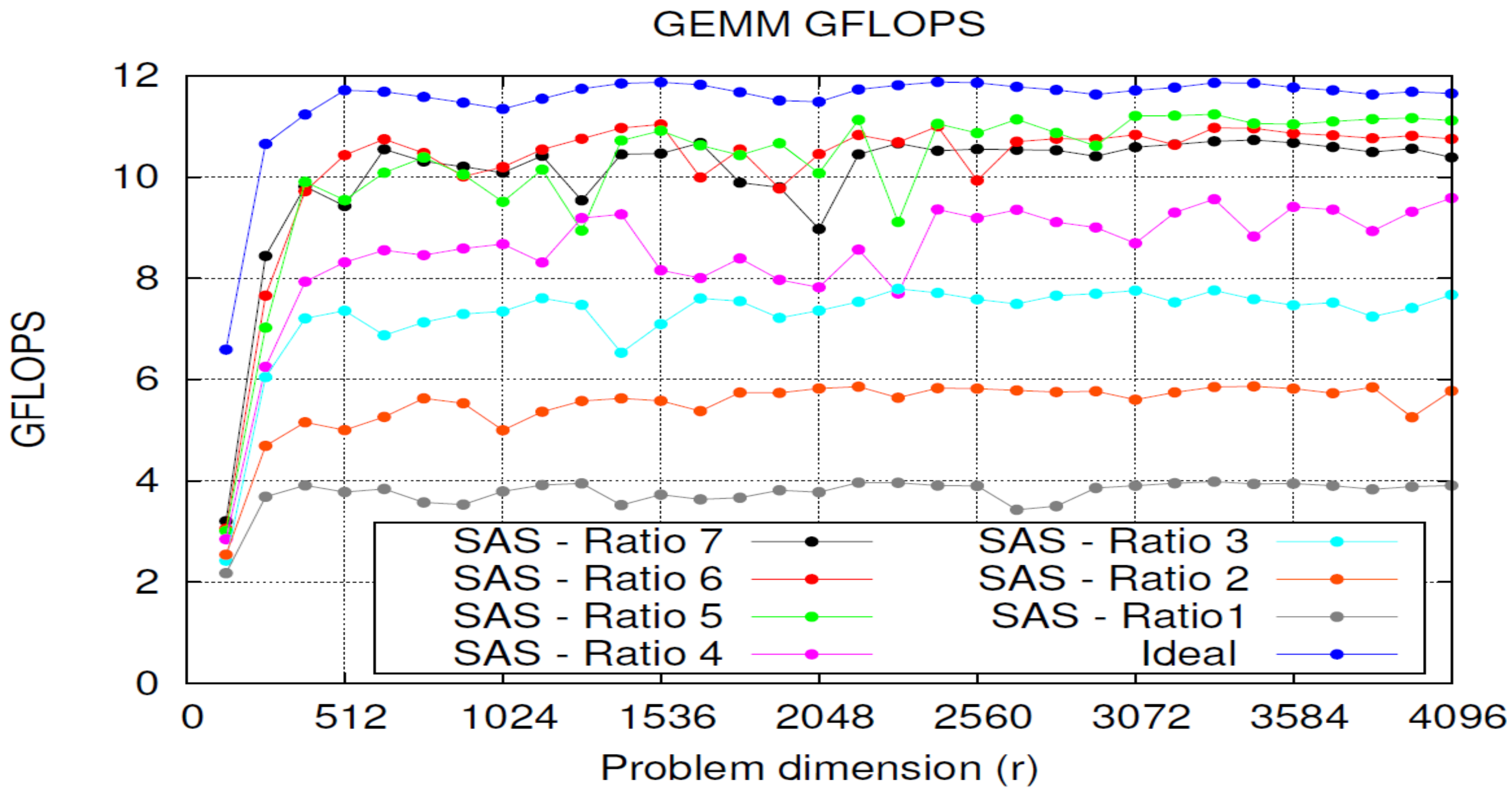
```
Loop 1   for $j_c = 0, \dots, n-1$ in steps of $n_c$
Loop 2     for $p_c = 0, \dots, k-1$ in steps of $k_c$
             $B(p_c : p_c + k_c - 1, j_c : j_c + n_c - 1) \to B_c$
Loop 3       for $i_c = 0, \dots, m-1$ in steps of $m_c$
               $A(i_c : i_c + m_c - 1, p_c : p_c + k_c - 1) \to A_c$
Loop 4         for $j_r = 0, \dots, n_c - 1$ in steps of $n_r$
Loop 5           for $i_r = 0, \dots, m_c - 1$ in steps of $m_r$
Loop 6             for $p_r = 0, \dots, k_c - 1$ in steps of $1$
                     $C_c(i_r : i_r + m_r - 1, j_r : j_r + n_r - 1)$
                       $\mathrel{+}= A_c(i_r : i_r + m_r - 1, p_r)$
                       $\cdot\ B_c(p_r, j_r : j_r + n_r - 1)$
                   endfor
                 endfor
               endfor
             endfor
           endfor
         endfor
```



Loop 1
j_c
2-way
$\xleftarrow{-\text{RATIO}}$
$\xrightarrow{+\text{RATIO}}$
$[0, \frac{3n}{4})$
Th_0, Th_1, Th_2, Th_3
$[\frac{3n}{4}, n)$
Th_4, Th_5, Th_6, Th_7
Loop 3
i_c
1-way
$[0, m)$
$Th_0, Th_1, Th_2, Th_3, Th_4, Th_5, Th_6, Th_7$
Loop 4
j_r
4-way
$[0 \dots \frac{n_c}{4})$
Th_0
$[\frac{n_c}{4}, \frac{2n_c}{4})$
Th_1
$[\frac{2n_c}{4}, \frac{3n_c}{4})$
Th_2
$[\frac{3n_c}{4}, n_c)$
Th_3
$[0, \frac{n_c}{4})$
Th_4
$[\frac{n_c}{4}, \frac{2n_c}{4})$
Th_5
$[\frac{2n_c}{4}, \frac{3n_c}{4})$
Th_6
$[\frac{3n_c}{4}, n_c)$
Th_7
Loop 5
i_r
1-way
$[0, m_c)$
$Th_0, Th_1, Th_2, Th_3, Th_4, Th_5, Th_6, Th_7$

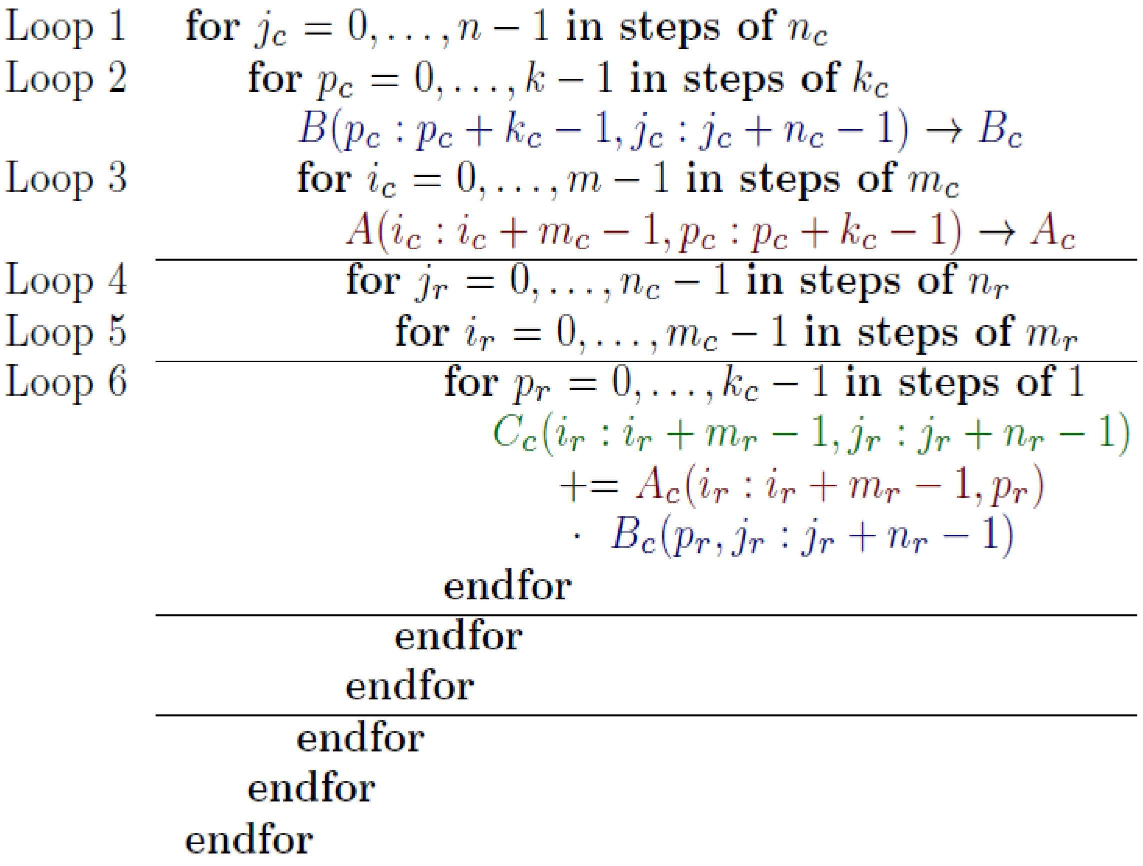
GEMM for Asymmetric Processors

- Static asymmetric scheduling between clusters



GEMM for Asymmetric Processors

- Cache-aware optimization with static asymmetric scheduling



```
Loop 1  for $j_c = 0, \ldots, n-1$ in steps of $n_c$
Loop 2    for $p_c = 0, \ldots, k-1$ in steps of $k_c$
            $B(p_c : p_c + k_c - 1, j_c : j_c + n_c - 1) \rightarrow B_c$
Loop 3      for $i_c = 0, \ldots, m-1$ in steps of $m_c$
              $A(i_c : i_c + m_c - 1, p_c : p_c + k_c - 1) \rightarrow A_c$
Loop 4        for $j_r = 0, \ldots, n_c - 1$ in steps of $n_r$
Loop 5          for $i_r = 0, \ldots, m_c - 1$ in steps of $m_r$
Loop 6            for $p_r = 0, \ldots, k_c - 1$ in steps of 1
                    $C_c(i_r : i_r + m_r - 1, j_r : j_r + n_r - 1)$
                         $\mathrel{+}= A_c(i_r : i_r + m_r - 1, p_r)$
                         $\cdot \; B_c(p_r, j_r : j_r + n_r - 1)$
                  endfor
                endfor
              endfor
            endfor
          endfor
        endfor
```

Use different m_c, k_c depending on the type of core

GEMM for Asymmetric Processors

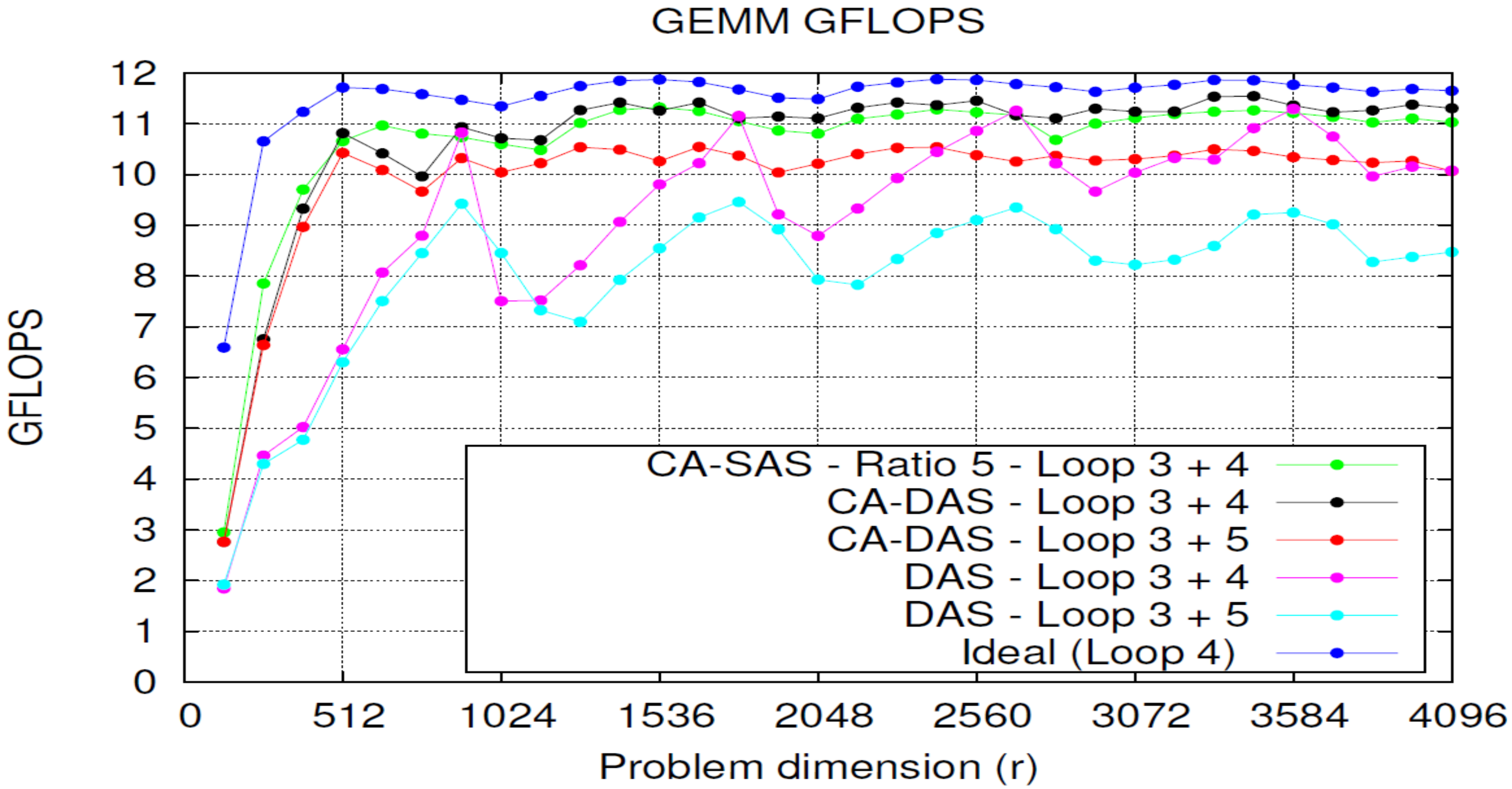
- Cache-aware optimization with dynamic asymmetric scheduling

```
Loop 1  for $j_c = 0, \ldots, n-1$ in steps of $n_c$
Loop 2    for $p_c = 0, \ldots, k-1$ in steps of $k_c$
            $B(p_c : p_c + k_c - 1, j_c : j_c + n_c - 1) \rightarrow B_c$
Loop 3      for $i_c = 0, \ldots, m-1$ in steps of $m_c$
              $A(i_c : i_c + m_c - 1, p_c : p_c + k_c - 1) \rightarrow A_c$
Loop 4        for $j_r = 0, \ldots, n_c - 1$ in steps of $n_r$
Loop 5          for $i_r = 0, \ldots, m_c - 1$ in steps of $m_r$
Loop 6            for $p_r = 0, \ldots, k_c - 1$ in steps of $1$
                    $C_c(i_r : i_r + m_r - 1, j_r : j_r + n_r - 1)$
                       $\mathrel{+}= A_c(i_r : i_r + m_r - 1, p_r)$
                       $\cdot \ B_c(p_r, j_r : j_r + n_r - 1)$
                  endfor
                endfor
              endfor
            endfor
          endfor
        endfor
```

Dynamically distribute the iteration space for Loop 1 between the two clusters

GEMM for Asymmetric Processors

- Cache-aware optimization with dynamic asymmetric scheduling



GEMM for Asymmetric Processors
Concluding remarks

- Easy to integrate support for asymmetric processors into BLIS framework
- Significant increase in GFLOPS with architecture-aware GEMM
- Same techniques applied to rest of BLAS

Outline

- High performance GEMM (sequential and multi-threaded)
- GEMM for asymmetric processors
- Fault tolerance (and approximate computing) GEMM

Tyler M. Smith
Robert A. van de Geijn

Mikhail Smelyanskiy

Enrique S. Quintana-Ortí

"Toward ABFT for BLIS GEMM"
T. M. Smith, R. A. van de Geijn, M. Smelyanskiy, E. S. Quintana-Ortí
FLAME Working note #76. Technical Report TR-15-05. Dept. of Computer Science. The University of Texas at Austin

Fault tolerance in GEMM
Motivation

- Provide a software layer for reliability in numerical libraries for space-borne missions

NASA JPL
Jet Propulsion Laboratory
California Institute of Technology

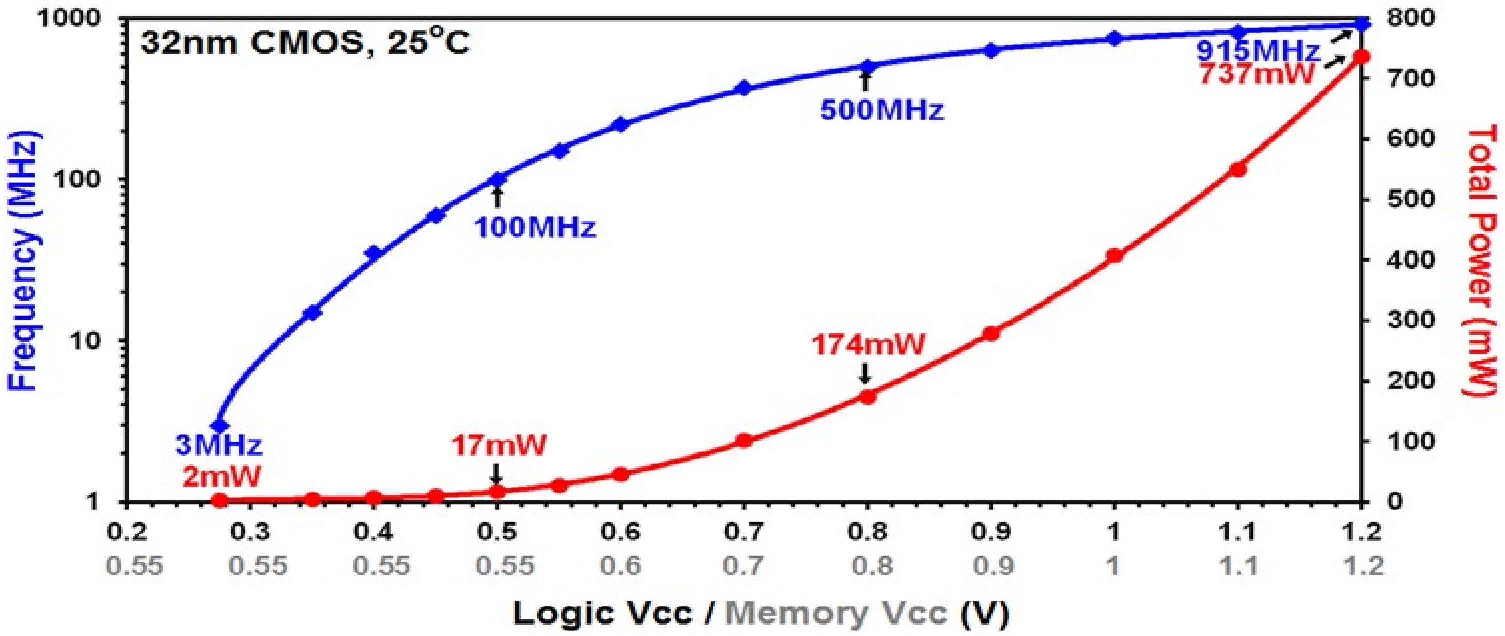
"Fault-tolerant high-performance matrix-matrix multiplication: theory and practice"
John A. Gunnels, Daniel S. Katz, Enrique S. Quintana, Robert van de Geijn
Int. Conference on Dependable Systems and Networks - DSN 2001

Fault tolerance in GEMM
Motivation

- FT GEMM, revisited
  - *Increase in #components with Moore's Law*
    - ASCI Q @ LANL (Terascale): 26.1 CPU failures per week
    - Sequoia @ LLNL (Petascale): MTBF is 1.5 days
    - Exascale requires increasing #components by O(10^3)

Fault tolerance in GEMM
Motivation

- FT GEMM, revisited
  - *Near-threshold voltage computing* (NTVC) reduces power…



at the cost of increasing error rates

Fault tolerance in GEMM

- Consider $C = A \cdot B$, and the augmented matrices

Consider, and the augmented matrices

$$A^* = \left(\begin{array}{c} A \\ \hline v^T A \end{array}\right), \quad B^* = \left(\begin{array}{c|c} B & Bw \end{array}\right), \quad C^* = \left(\begin{array}{c|c} C & Cw \\ \hline v^T C & v^T Cw \end{array}\right),$$

In absence of error, then $C^* = A^* B^*$.

In absence of error, then

Use *left and right checksum* vectors:

Use *left and right checksum* vectors:

$$\begin{array}{rclcl} \|d\|_\infty & = & \|C \cdot w & - & A \cdot (B \cdot w)\|_\infty & > & 0 & \text{or} \\ \|e^T\|_\infty & = & \|v^T \cdot C & - & (v^T \cdot A) \cdot B\|_\infty & > & 0. \end{array}$$

"Algorithm-based fault tolerance for matrix operations"
K.-H. Huang and J. A. Abraham
IEEE Transactions on Computers, Vol. 33(6), 1984

Fault tolerance in GEMM

- In practice, due to finite precision arithmetic, an error is detected if

$$\begin{aligned} \|d\|_\infty &> \tau \cdot \|A\|_\infty \cdot \|B\|_\infty \\ \|e^T\|_\infty &> \tau \cdot \|A\|_\infty \cdot \|B\|_\infty, \end{aligned}$$

where

$$\tau = \max(m, n, k) \cdot u$$

"Fault-tolerant high-performance matrix-matrix multiplication: theory and practice"
John A. Gunnels, Daniel S. Katz, Enrique S. Quintana, Robert van de Geijn
Int. Conference on Dependable Systems and Networks - DSN 2001

or higher for Approximate Computing!

Fault tolerance in GEMM

- Overhead for full GEMM (detection only)

- Overhead for full GEMM (detection only)

$$
\begin{array}{rclcl}
\|d\|_\infty & = & \|C \cdot w & - & A \cdot (B \cdot w)\|_\infty \\
\|e^T\|_\infty & = & \|v^T \cdot C & - & (v^T \cdot A) \cdot B\|_\infty
\end{array}
$$

$$
\mathcal{O}_d(m,n,k) = \frac{4mn + 5mk + 5kn}{\mathcal{O}_c(m,n,k)} = \frac{4mn + 5mk + 5kn}{2mnk},
$$

  - Has to be applied off-line
  - Requires a copy of the full matrix
  - Correction is expensive: recompute the full product

  - Has to be applied off-line
  - Requires a copy of the full matrix C
  - Correction is expensive: recompute the full product

Fault tolerance in GEMM

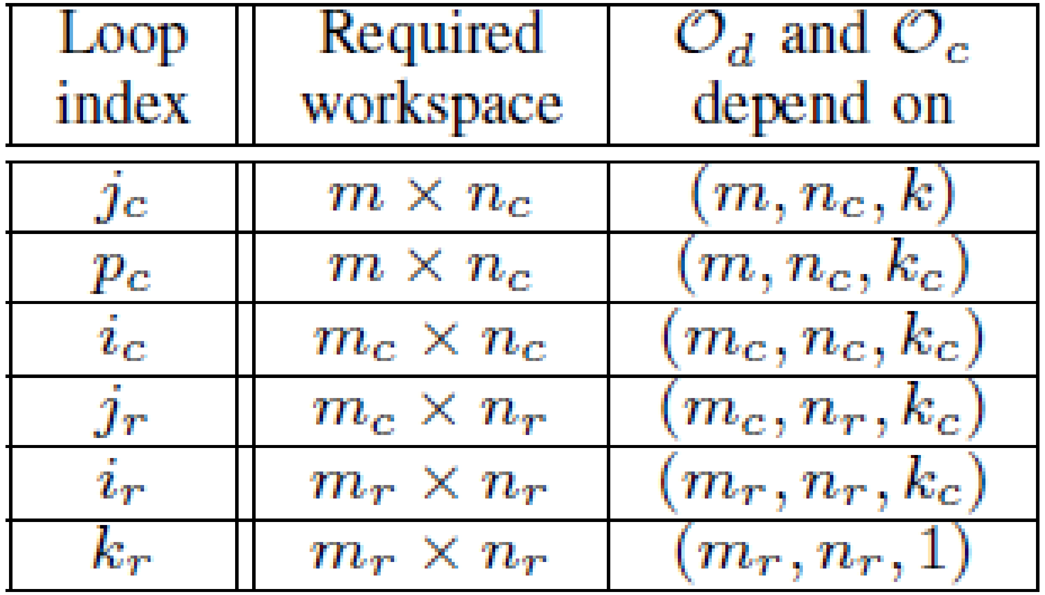
- Apply with finer granularity

```
Loop 1  for $j_c = 0, \ldots, n-1$ in steps of $n_c$
Loop 2    for $p_c = 0, \ldots, k-1$ in steps of $k_c$
            $B(p_c : p_c + k_c - 1, j_c : j_c + n_c - 1) \rightarrow B_c$                    // Pack into $B_c$
Loop 3      for $i_c = 0, \ldots, m-1$ in steps of $m_c$
              $A(i_c : i_c + m_c - 1, p_c : p_c + k_c - 1) \rightarrow A_c$                  // Pack into $A_c$
-----------------------------------------------------------------------------------------------------------
Loop 4        for $j_r = 0, \ldots, n_c - 1$ in steps of $n_r$                                // Macro-kernel
Loop 5          for $i_r = 0, \ldots, m_c - 1$ in steps of $m_r$
-------------------------------------------------------------------------------------
Loop 6            for $p_r = 0, \ldots, k_c - 1$ in steps of 1              // Micro-kernel
                    $C_c(i_r : i_r + m_r - 1, j_r : j_r + n_r - 1)$
                        $+= A_c(i_r : i_r + m_r - 1, p_r)$
                        $\cdot\ B_c(p_r, j_r : j_r + n_r - 1)$
                  endfor
-------------------------------------------------------------------------------------
                endfor
              endfor
-----------------------------------------------------------------------------------------------------------
            endfor
          endfor
        endfor
```

Fault tolerance in GEMM

- Apply with finer granularity

$$\mathcal{O}_d(m,n,k) = \frac{4mn+5mk+5kn}{\mathcal{O}_c(m,n,k)} = \frac{4mn+5mk+5kn}{2mnk},$$



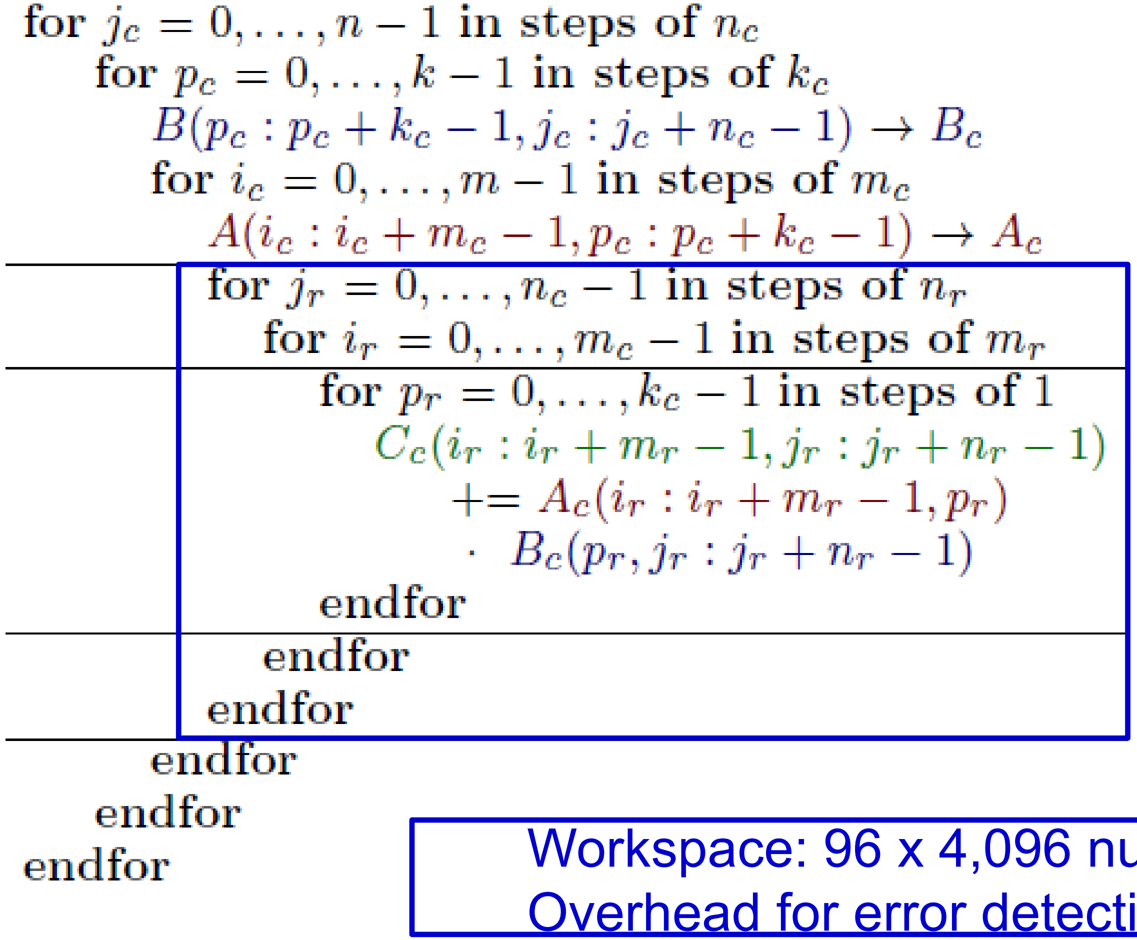
Loop index	Required workspace	$\mathcal{O}_d$ and $\mathcal{O}_c$ depend on
j_c	$m \times n_c$	(m, n_c, k)
p_c	$m \times n_c$	(m, n_c, k_c)
i_c	$m_c \times n_c$	(m_c, n_c, k_c)
j_r	$m_c \times n_r$	(m_c, n_r, k_c)
i_r	$m_r \times n_r$	(m_r, n_r, k_c)
k_r	$m_r \times n_r$	$(m_r, n_r, 1)$

More expensive correction / Larger workspace

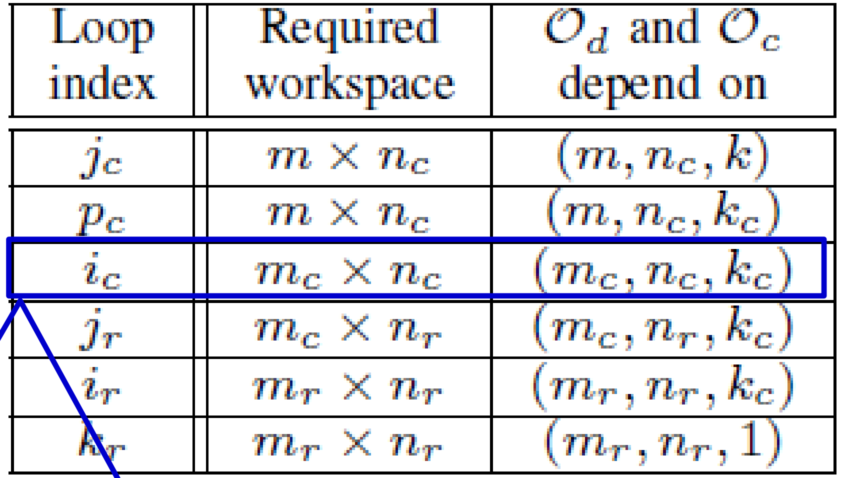
More expensive detection

Fault tolerance in GEMM

- Intel Xeon E5 (Sandy-Bridge): Macro-kernel



```
for $j_c = 0, \ldots, n-1$ in steps of $n_c$
  for $p_c = 0, \ldots, k-1$ in steps of $k_c$
    $B(p_c : p_c + k_c - 1, j_c : j_c + n_c - 1) \rightarrow B_c$
    for $i_c = 0, \ldots, m-1$ in steps of $m_c$
      $A(i_c : i_c + m_c - 1, p_c : p_c + k_c - 1) \rightarrow A_c$
      for $j_r = 0, \ldots, n_c - 1$ in steps of $n_r$
        for $i_r = 0, \ldots, m_c - 1$ in steps of $m_r$
          for $p_r = 0, \ldots, k_c - 1$ in steps of $1$
            $C_c(i_r : i_r + m_r - 1, j_r : j_r + n_r - 1)$
              $\mathrel{+}= A_c(i_r : i_r + m_r - 1, p_r)$
              $\cdot\ B_c(p_r, j_r : j_r + n_r - 1)$
          endfor
        endfor
      endfor
    endfor
  endfor
endfor
```



Loop index	Required workspace	$\mathcal{O}_d$ and $\mathcal{O}_c$ depend on
j_c	$m \times n_c$	(m, n_c, k)
p_c	$m \times n_c$	(m, n_c, k_c)
i_c	$m_c \times n_c$	(m_c, n_c, k_c)
j_r	$m_c \times n_r$	(m_c, n_r, k_c)
i_r	$m_r \times n_r$	(m_r, n_r, k_c)
k_r	$m_r \times n_r$	$(m_r, n_r, 1)$

Workspace: 96 x 4,096 numbers
Overhead for error detection: 2.6%

Fault tolerance in GEMM

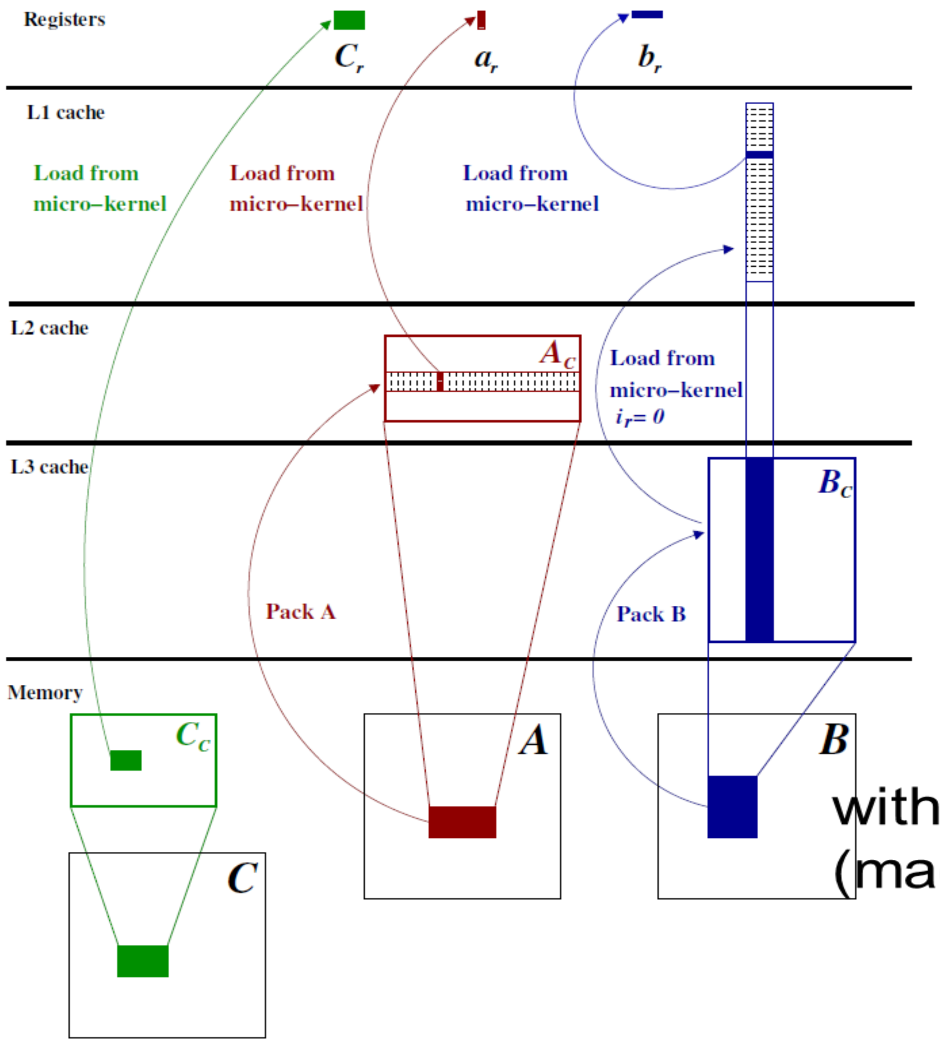
```
Loop 1:   for $j_c = 0, \ldots, n-1$ in steps of $n_c$
Loop 2:       for $p_c = 0, \ldots, k-1$ in steps of $k_c$
                  $B(p_c : p_c + k_c - 1, j_c : j_c + n_c - 1) \rightarrow B_c$          // Pack into $B_c$
Loop 3:           for $i_c = 0, \ldots, m-1$ in steps of $m_c$
                      $A(i_c : i_c + m_c - 1, p_c : p_c + k_c - 1) \rightarrow A_c$      // Pack into $A_c$
                      $C(i_c : i_c + m_c - 1, j_c : j_c + n_c - 1) \equiv C_c \mathrel{+}= A_c \cdot B_c$   // Macro-kernel
                  endfor
              endfor
          endfor
```

$$\begin{aligned} \|d\|_\infty &= \|C \cdot w &- \quad A \cdot (B \cdot w)\|_\infty \\ \|e^T\|_\infty &= \|v^T \cdot C &- \quad (v^T \cdot A) \cdot B\|_\infty \end{aligned}$$

with $C = C_c, A = A_c, B = B_c$
(macro-kernel)

with
(macro-kernel)

Fault tolerance in GEMM



$$\begin{array}{rcll} \|d\|_\infty & = & \|C \cdot w & - \quad A \cdot (B \cdot w)\|_\infty \\ \|e^T\|_\infty & = & \|v^T \cdot C & - \quad (v^T \cdot A) \cdot B\|_\infty \end{array}$$

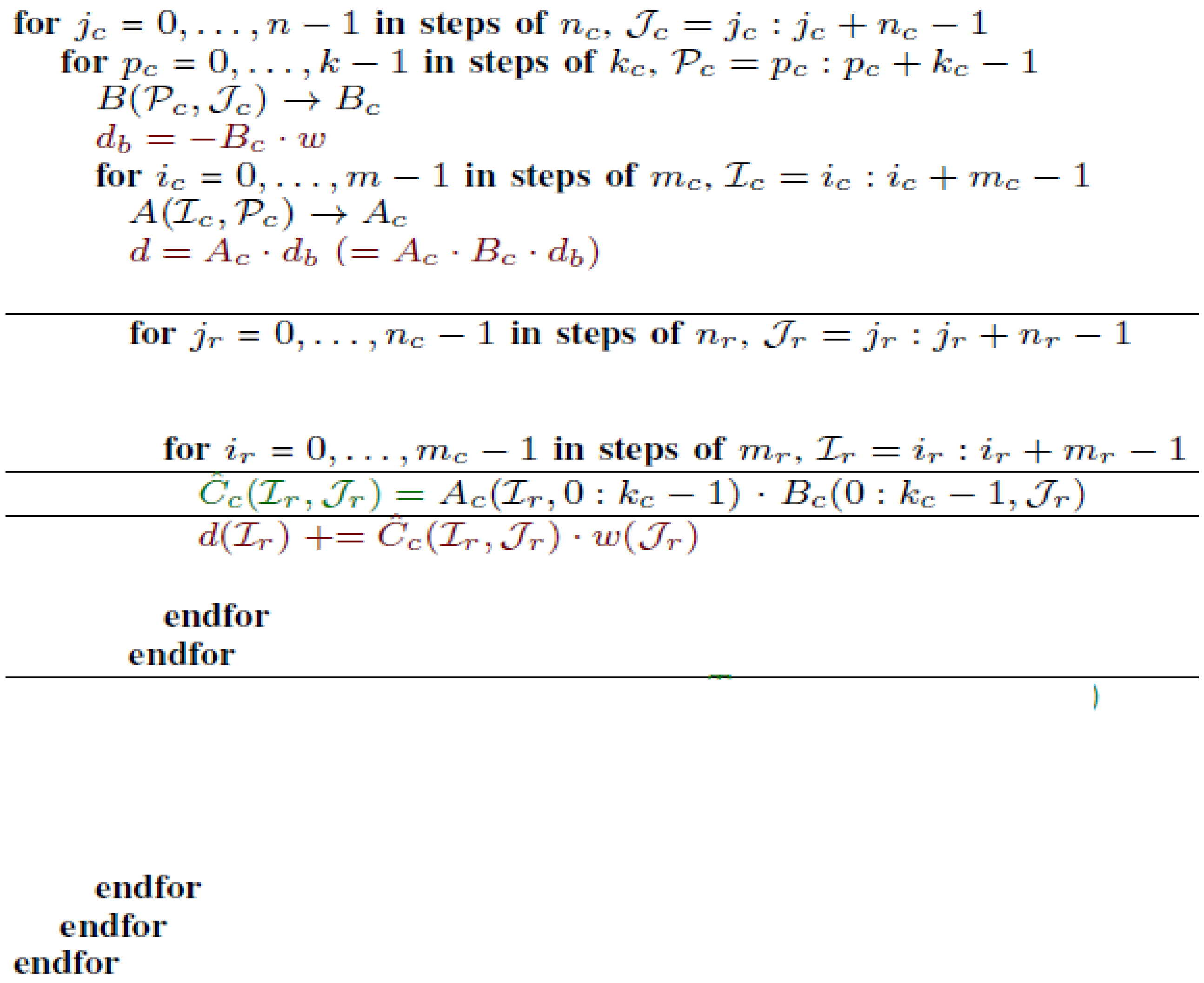
with $C = C_c, A = A_c, B = B_c$
(macro-kernel)

with
(macro-kernel)

Fault tolerance in GEMM

- Right checksum:

$$d = \hat{C}_c \cdot w - A_c \cdot B_c \cdot w$$

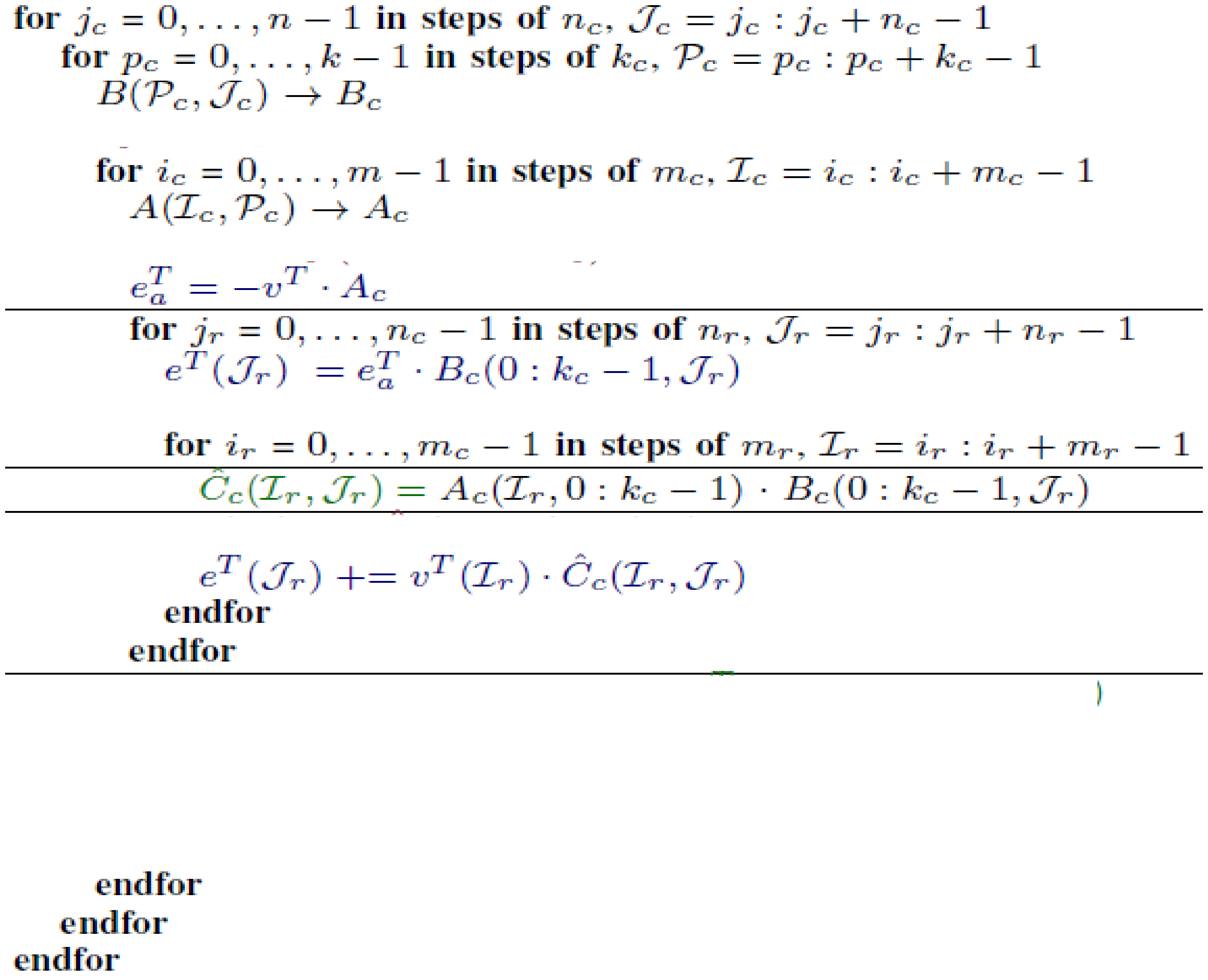


```
for $j_c = 0, \ldots, n-1$ in steps of $n_c$, $\mathcal{J}_c = j_c : j_c + n_c - 1$
  for $p_c = 0, \ldots, k-1$ in steps of $k_c$, $\mathcal{P}_c = p_c : p_c + k_c - 1$
    $B(\mathcal{P}_c, \mathcal{J}_c) \rightarrow B_c$
    $d_b = -B_c \cdot w$
    for $i_c = 0, \ldots, m-1$ in steps of $m_c$, $\mathcal{I}_c = i_c : i_c + m_c - 1$
      $A(\mathcal{I}_c, \mathcal{P}_c) \rightarrow A_c$
      $d = A_c \cdot d_b$ $(= A_c \cdot B_c \cdot d_b)$

      for $j_r = 0, \ldots, n_c - 1$ in steps of $n_r$, $\mathcal{J}_r = j_r : j_r + n_r - 1$


        for $i_r = 0, \ldots, m_c - 1$ in steps of $m_r$, $\mathcal{I}_r = i_r : i_r + m_r - 1$
          $\hat{C}_c(\mathcal{I}_r, \mathcal{J}_r) = A_c(\mathcal{I}_r, 0 : k_c - 1) \cdot B_c(0 : k_c - 1, \mathcal{J}_r)$
          $d(\mathcal{I}_r) \mathrel{+}= \hat{C}_c(\mathcal{I}_r, \mathcal{J}_r) \cdot w(\mathcal{J}_r)$

        endfor
      endfor




    endfor
  endfor
endfor
```

Fault tolerance in GEMM

- Left checksum:

$$e^T = v^T \cdot \hat{C}_c - v^T \cdot A_c \cdot B_c$$



```
for $j_c = 0, \ldots, n-1$ in steps of $n_c$, $\mathcal{J}_c = j_c : j_c + n_c - 1$
  for $p_c = 0, \ldots, k-1$ in steps of $k_c$, $\mathcal{P}_c = p_c : p_c + k_c - 1$
    $B(\mathcal{P}_c, \mathcal{J}_c) \rightarrow B_c$

    for $i_c = 0, \ldots, m-1$ in steps of $m_c$, $\mathcal{I}_c = i_c : i_c + m_c - 1$
      $A(\mathcal{I}_c, \mathcal{P}_c) \rightarrow A_c$

      $e_a^T = -v^T \cdot A_c$
      for $j_r = 0, \ldots, n_c - 1$ in steps of $n_r$, $\mathcal{J}_r = j_r : j_r + n_r - 1$
        $e^T(\mathcal{J}_r) = e_a^T \cdot B_c(0 : k_c - 1, \mathcal{J}_r)$

        for $i_r = 0, \ldots, m_c - 1$ in steps of $m_r$, $\mathcal{I}_r = i_r : i_r + m_r - 1$
          $\hat{C}_c(\mathcal{I}_r, \mathcal{J}_r) = A_c(\mathcal{I}_r, 0 : k_c - 1) \cdot B_c(0 : k_c - 1, \mathcal{J}_r)$

          $e^T(\mathcal{J}_r) \mathrel{+}= v^T(\mathcal{I}_r) \cdot \hat{C}_c(\mathcal{I}_r, \mathcal{J}_r)$
        endfor
      endfor

    endfor
  endfor
endfor
```

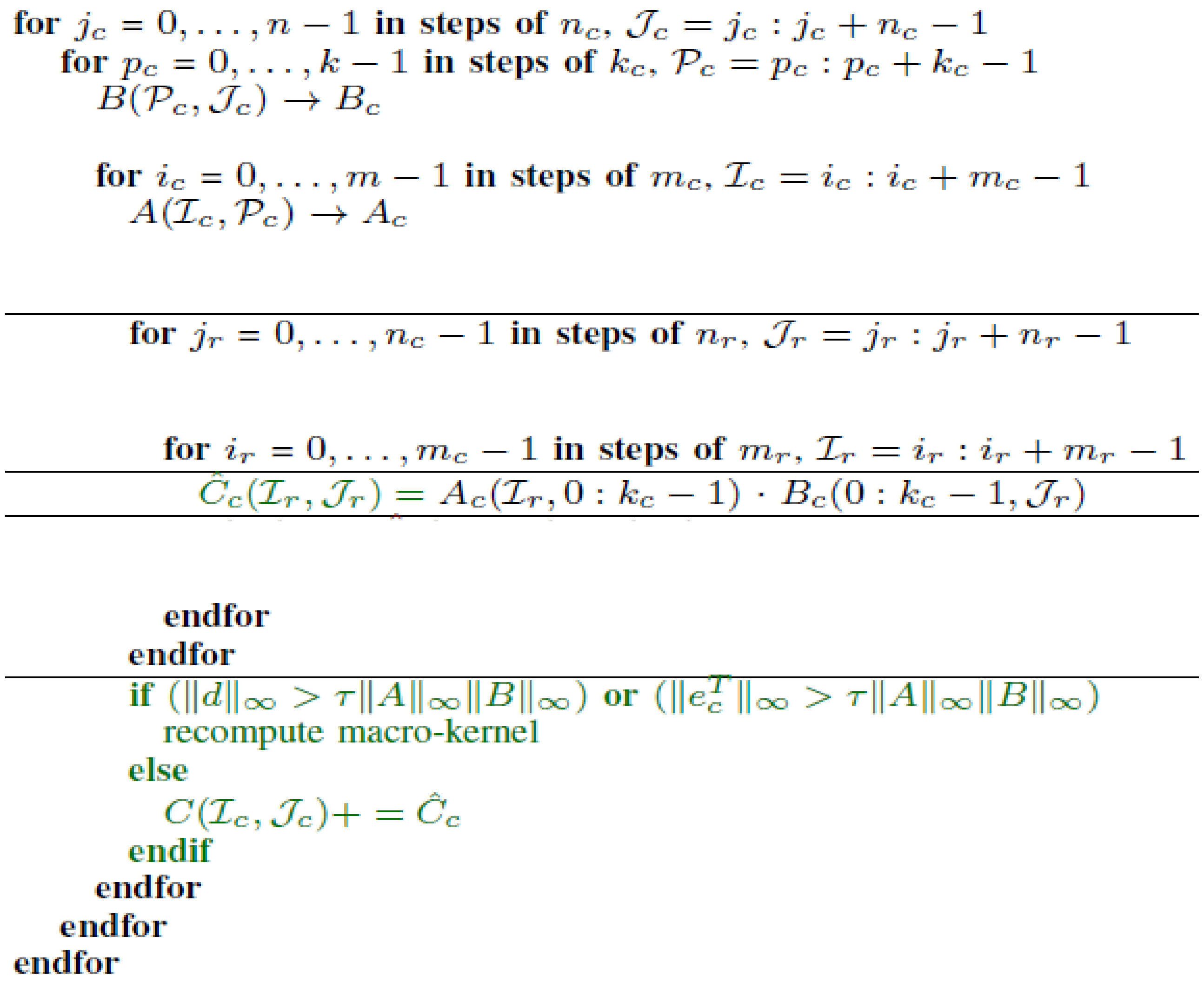
Fault tolerance in GEMM

- Detect and prevent error:

Detect and prevent error:

Check $\|d\|_\infty$ and $\|e^T\|_\infty$

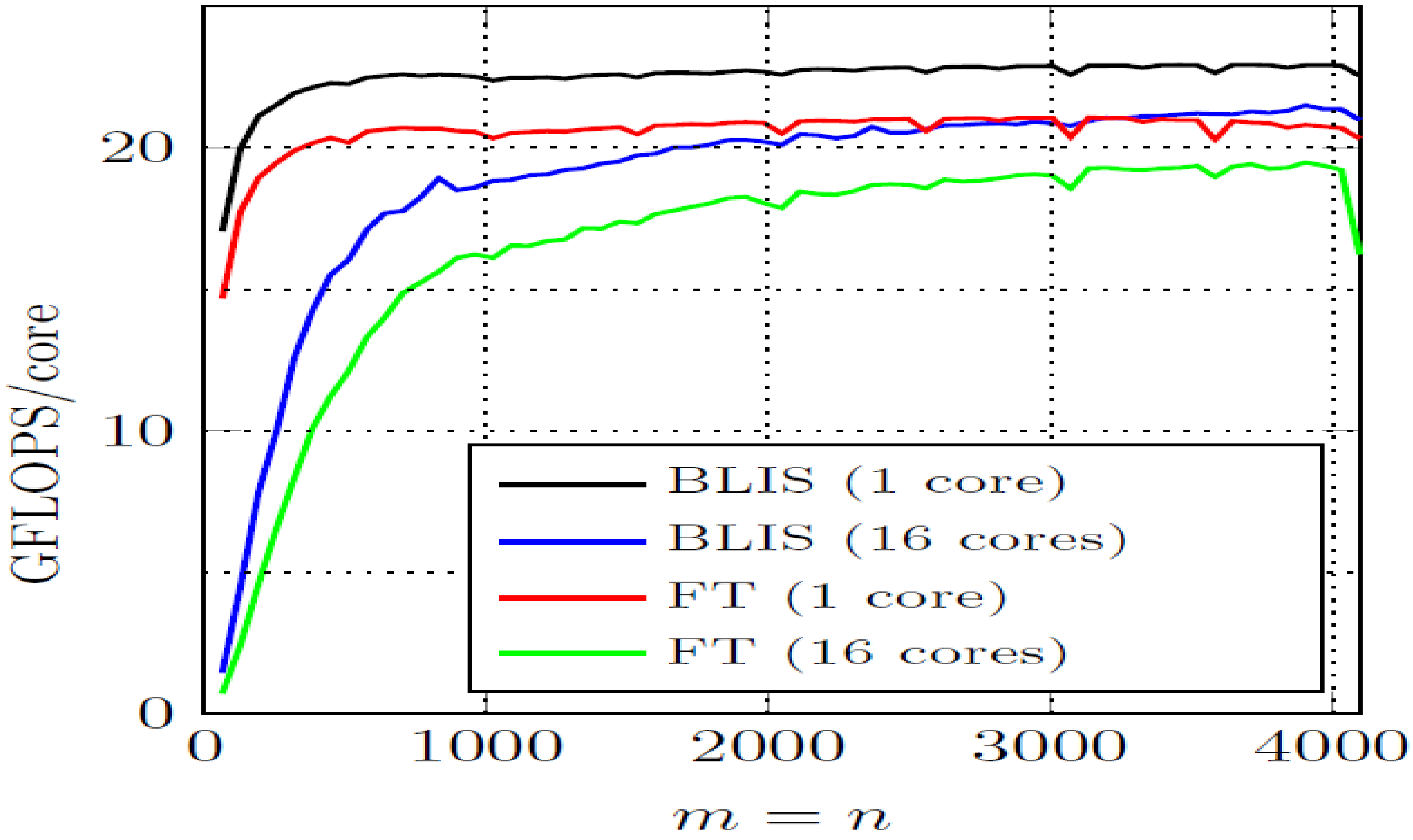
Check and



```
for $j_c = 0, \ldots, n-1$ in steps of $n_c$, $\mathcal{J}_c = j_c : j_c + n_c - 1$
  for $p_c = 0, \ldots, k-1$ in steps of $k_c$, $\mathcal{P}_c = p_c : p_c + k_c - 1$
    $B(\mathcal{P}_c, \mathcal{J}_c) \rightarrow B_c$

    for $i_c = 0, \ldots, m-1$ in steps of $m_c$, $\mathcal{I}_c = i_c : i_c + m_c - 1$
      $A(\mathcal{I}_c, \mathcal{P}_c) \rightarrow A_c$

      for $j_r = 0, \ldots, n_c - 1$ in steps of $n_r$, $\mathcal{J}_r = j_r : j_r + n_r - 1$

        for $i_r = 0, \ldots, m_c - 1$ in steps of $m_r$, $\mathcal{I}_r = i_r : i_r + m_r - 1$
          $\hat{C}_c(\mathcal{I}_r, \mathcal{J}_r) = A_c(\mathcal{I}_r, 0 : k_c - 1) \cdot B_c(0 : k_c - 1, \mathcal{J}_r)$

        endfor
      endfor
      if $(\|d\|_\infty > \tau \|A\|_\infty \|B\|_\infty)$ or $(\|e_c^T\|_\infty > \tau \|A\|_\infty \|B\|_\infty)$
        recompute macro-kernel
      else
        $C(\mathcal{I}_c, \mathcal{J}_c) += \hat{C}_c$
      endif
    endfor
  endfor
endfor
```

Fault tolerance in GEMM

- Intel Xeon E5-2680. BLIS vs FT-BLIS



Fault tolerance in GEMM

- Selective error correction

  Detection at the macro-kernel level:

$$4m_cn_c + 5m_ck_c + 5k_cn_c \qquad \mathcal{O}_d(m_c, n_c, k_c)$$

but correction can proceed at the micro-kernel level:

but correction can proceed at the micro-kernel level:

$$2m_rn_rk_c \qquad \mathcal{O}_c(m_r, n_r, k_c)$$

instead of

instead of

$$2m_cn_ck_c \qquad \mathcal{O}_c(m_c, n_c, k_c)$$

Fault tolerance in GEMM
Concluding remarks

- Easy to integrate FT and AC into the same framework for BLIS
- Left and right checksums yield acceptable overhead for high performance GEMM