I/O Lower Bounds and Algorithms for Matrix-Matrix Multiplication

Tyler M. Smith

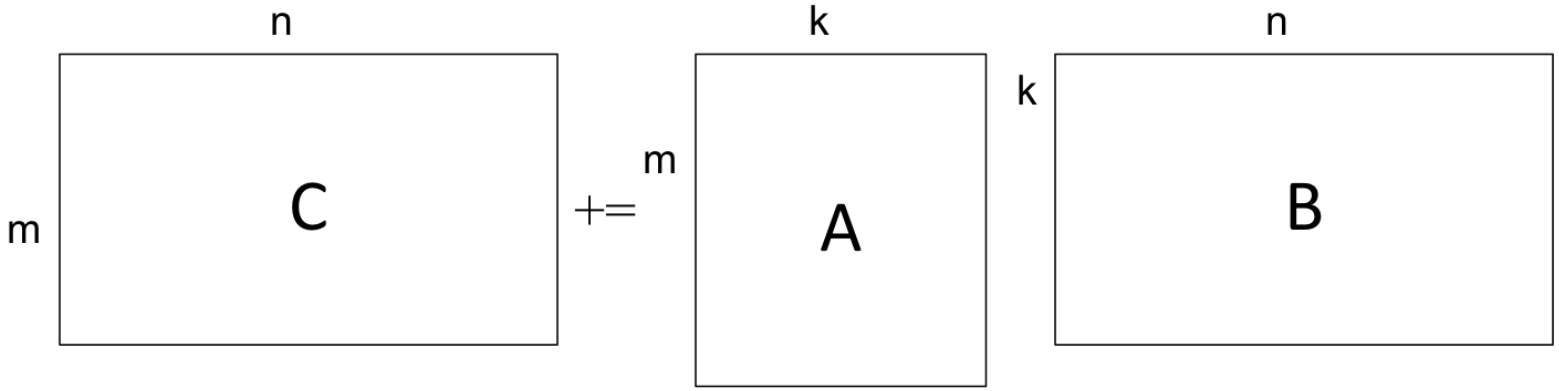
July 5, 2017

Introduction

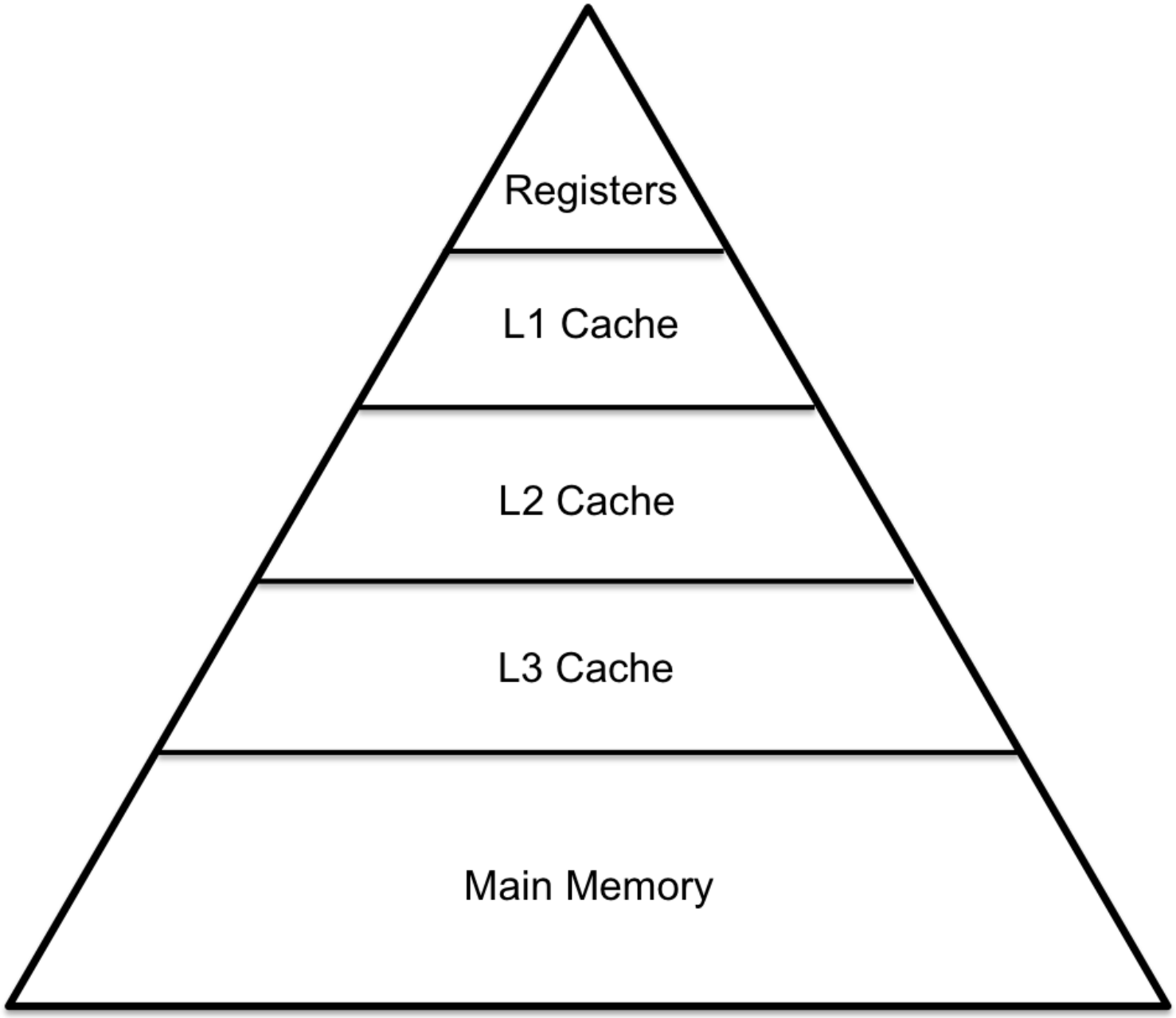
- Dense matrix-matrix multiplication (MMM)
- Goal: Reduce I/O cost for machines with hierarchical memory
- Novel contributions:
  - I/O lower bounds with a tight constant $\frac{2mnk}{\sqrt{S}}$
  - A family of algorithms for machines with any number of levels of memory hierarchy
  - Outperform the state-of-the-art Goto's Algorithm by 38% when there is low bandwidth to main memory

Problem definition

- Classical MMM
  - $C \mathrel{+}= AB$
  - C is $m \times n$, A is $m \times k$, and B is $k \times n$
- Reduce I/O cost for MMM algorithms



Hierarchical memory



Blocked algorithms

- MMM is an operation with a lot of opportunities for reuse
  - Each element of A is used n times
  - Each element of B is used m times
  - Each element of C is used k times
- With $\mathcal{O}(n^2)$ elements, one can perform $\mathcal{O}(n^3)$ flops
  - If all matrices fit into fast memory, amortize $\mathcal{O}(n^2)$ memops with $\mathcal{O}(n^3)$ flops
- Work with blocks of matrices at a time, where the blocks can fit into fast memory

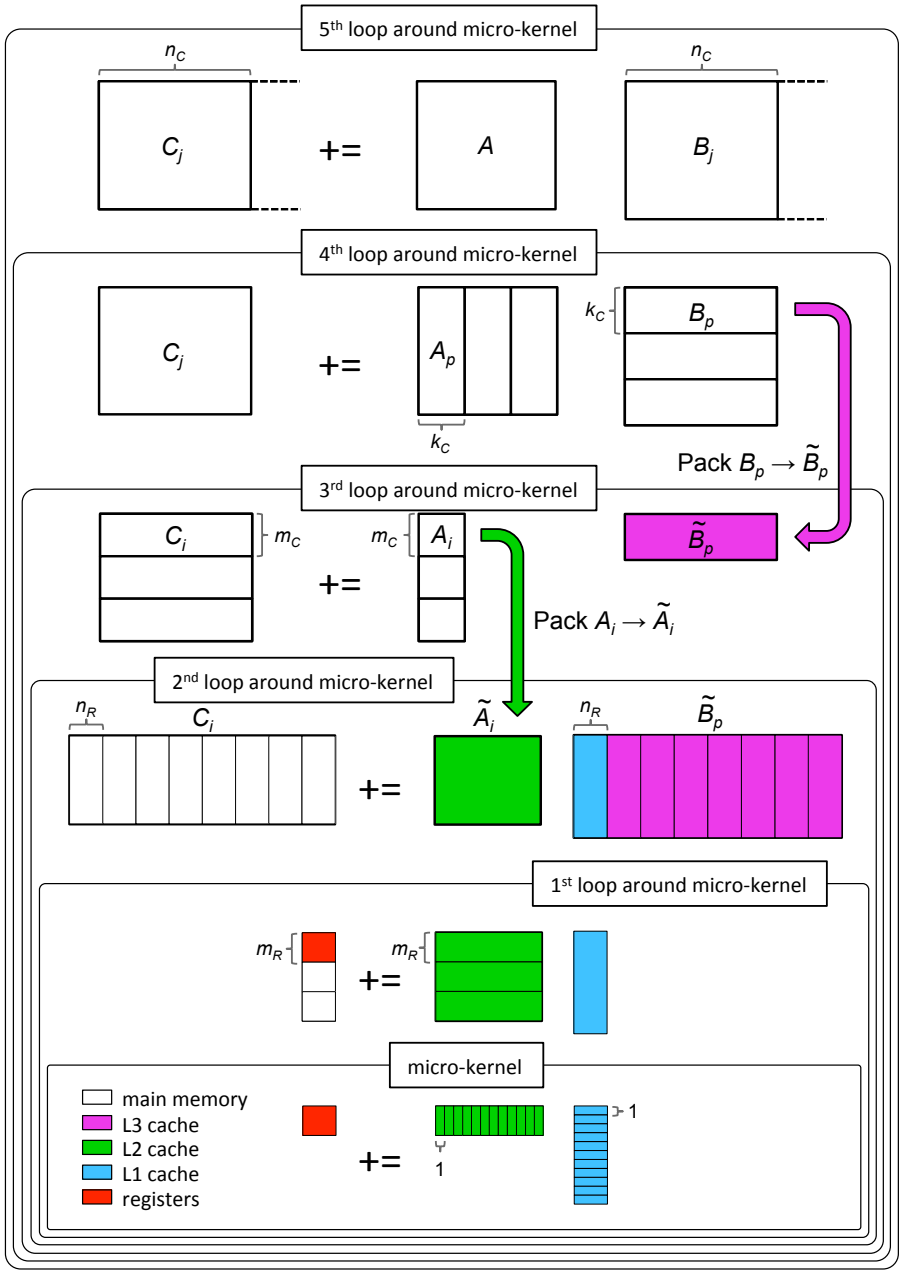
Building blocks of dense linear algebra

- MMM is the bottom of the food chain
- Level-3 BLAS
- LAPACK/FLAME
- ScaLAPACK/Elemental

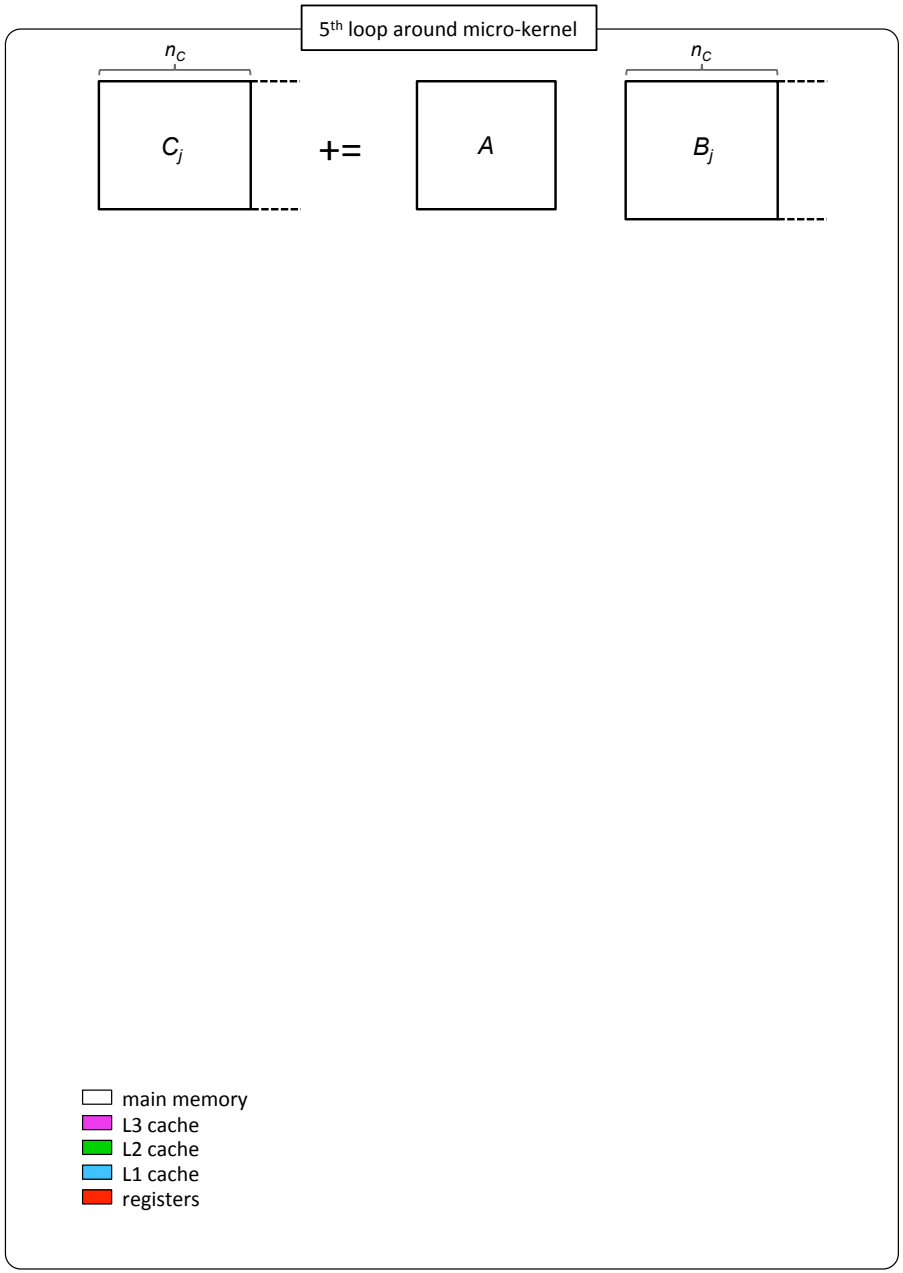
Outline

- Introduction
- State-of-the-art MMM
  - Goto's Algorithm
- Lower bounds
- Algorithms
- Experiments

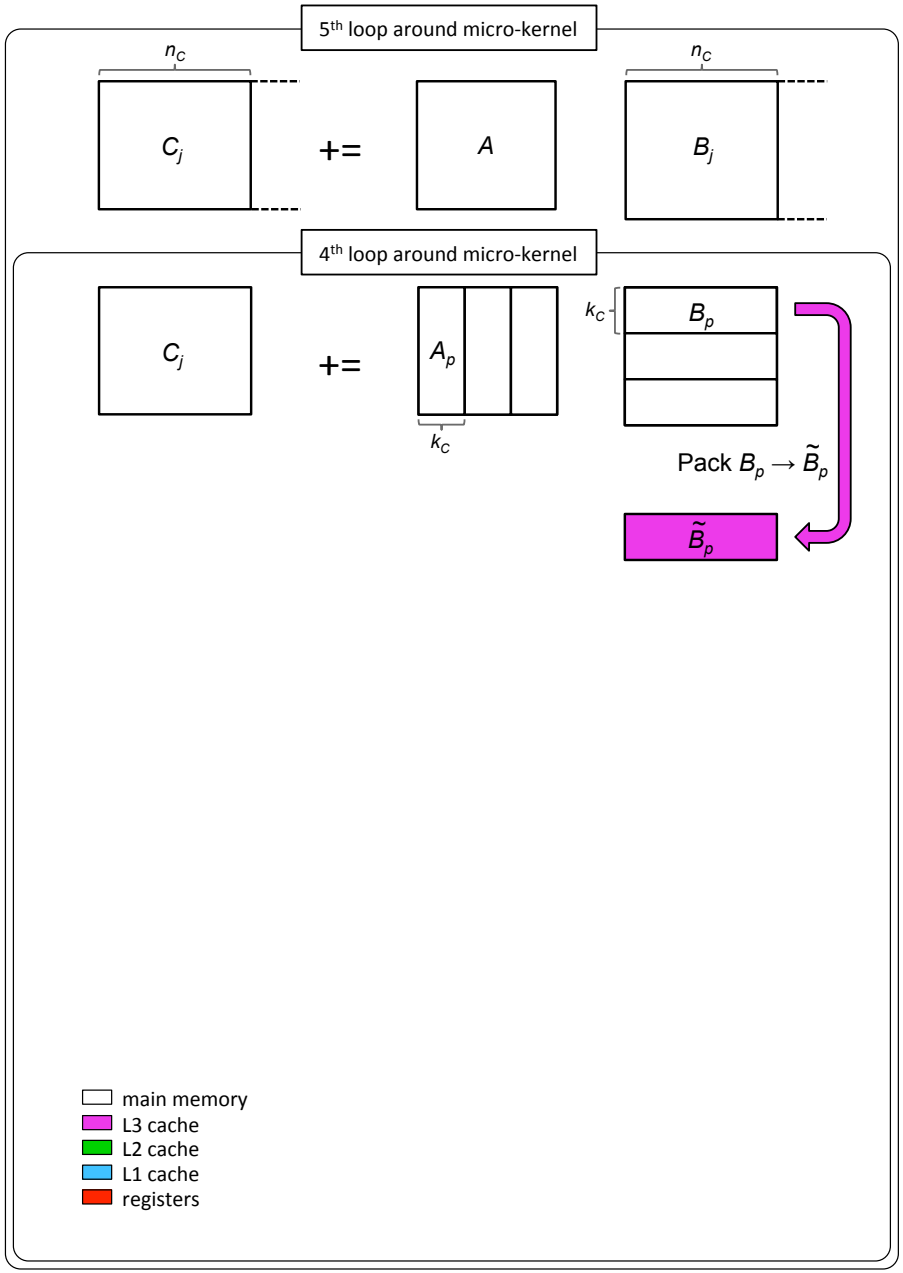
Goto's Algorithm



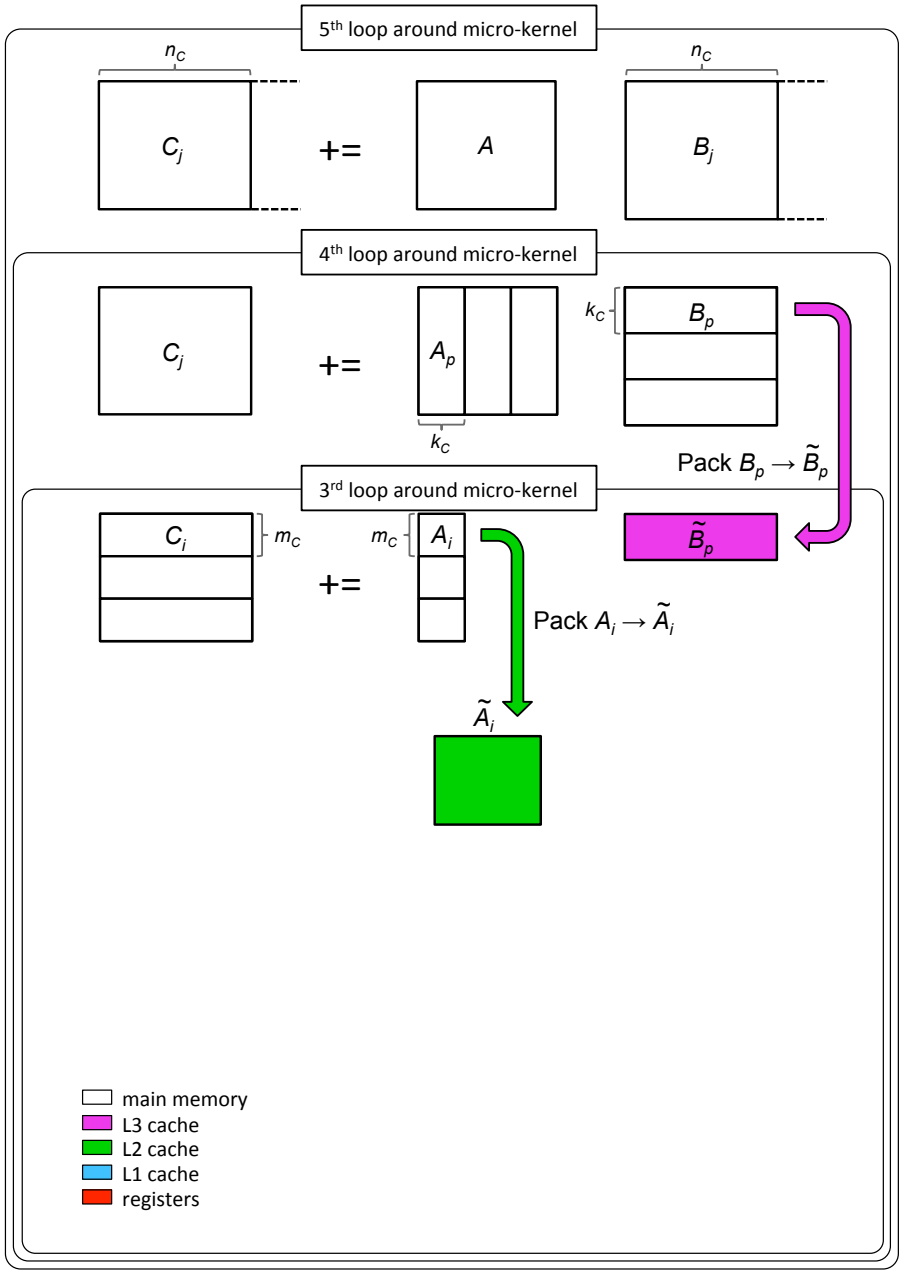
Goto's Algorithm



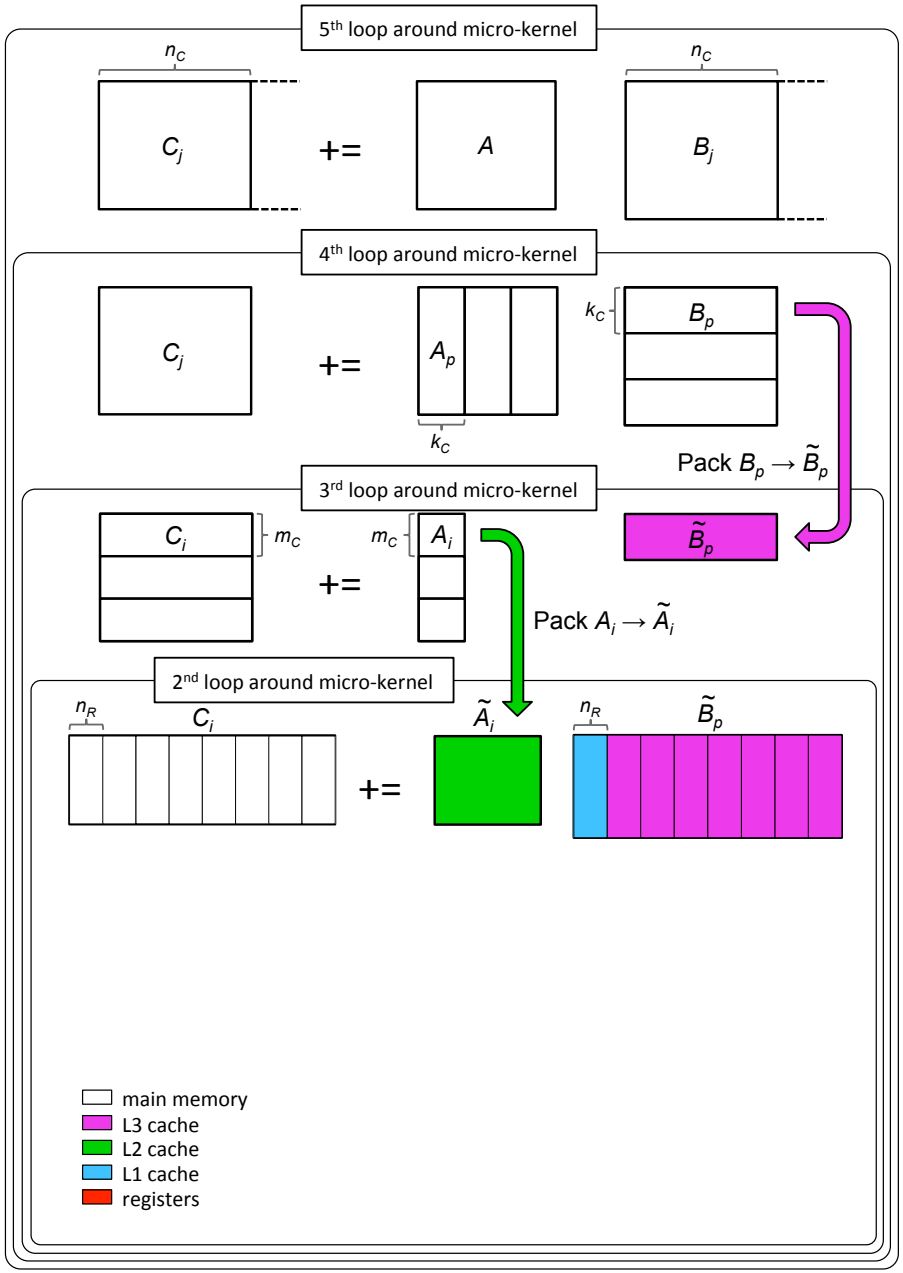
Goto's Algorithm



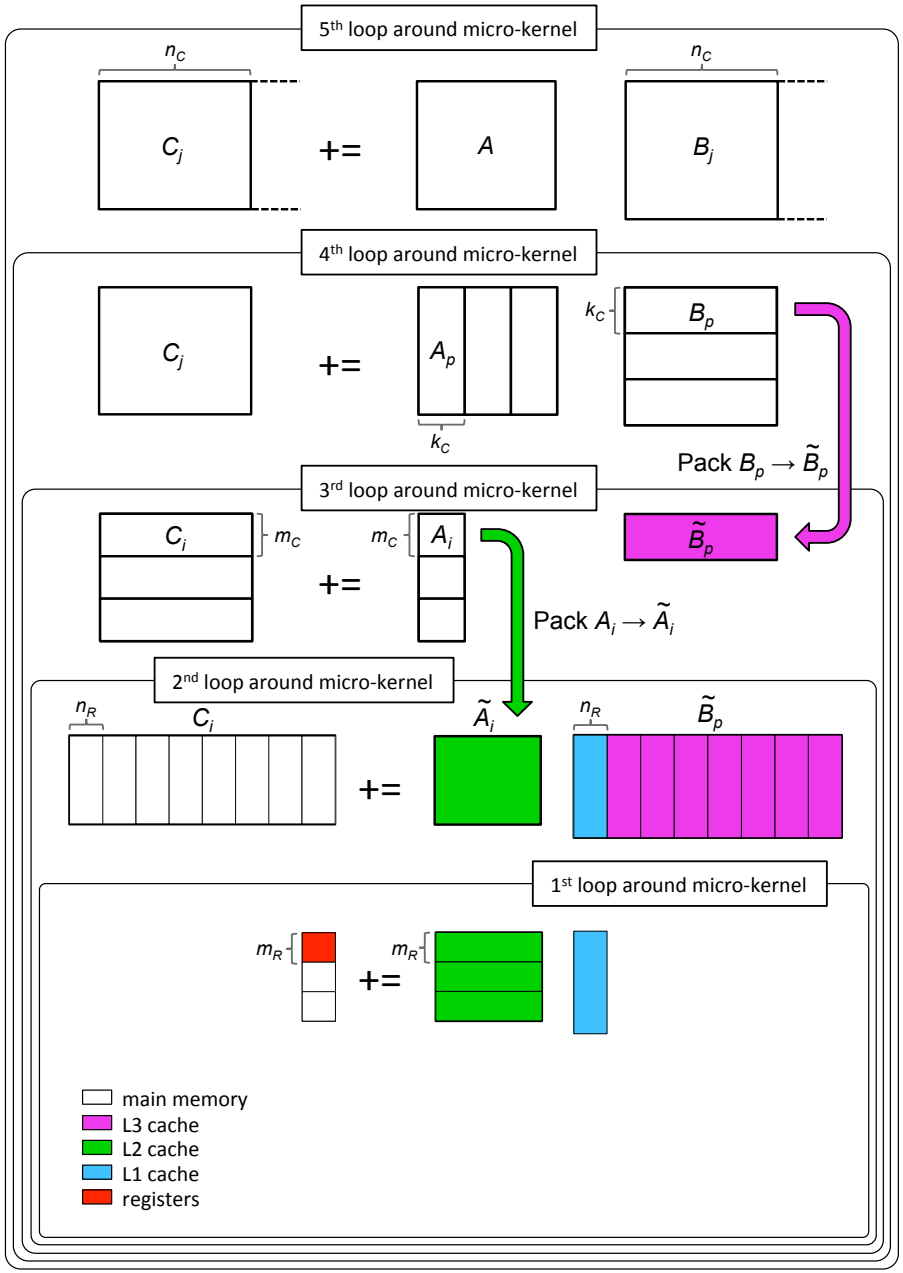
Goto's Algorithm



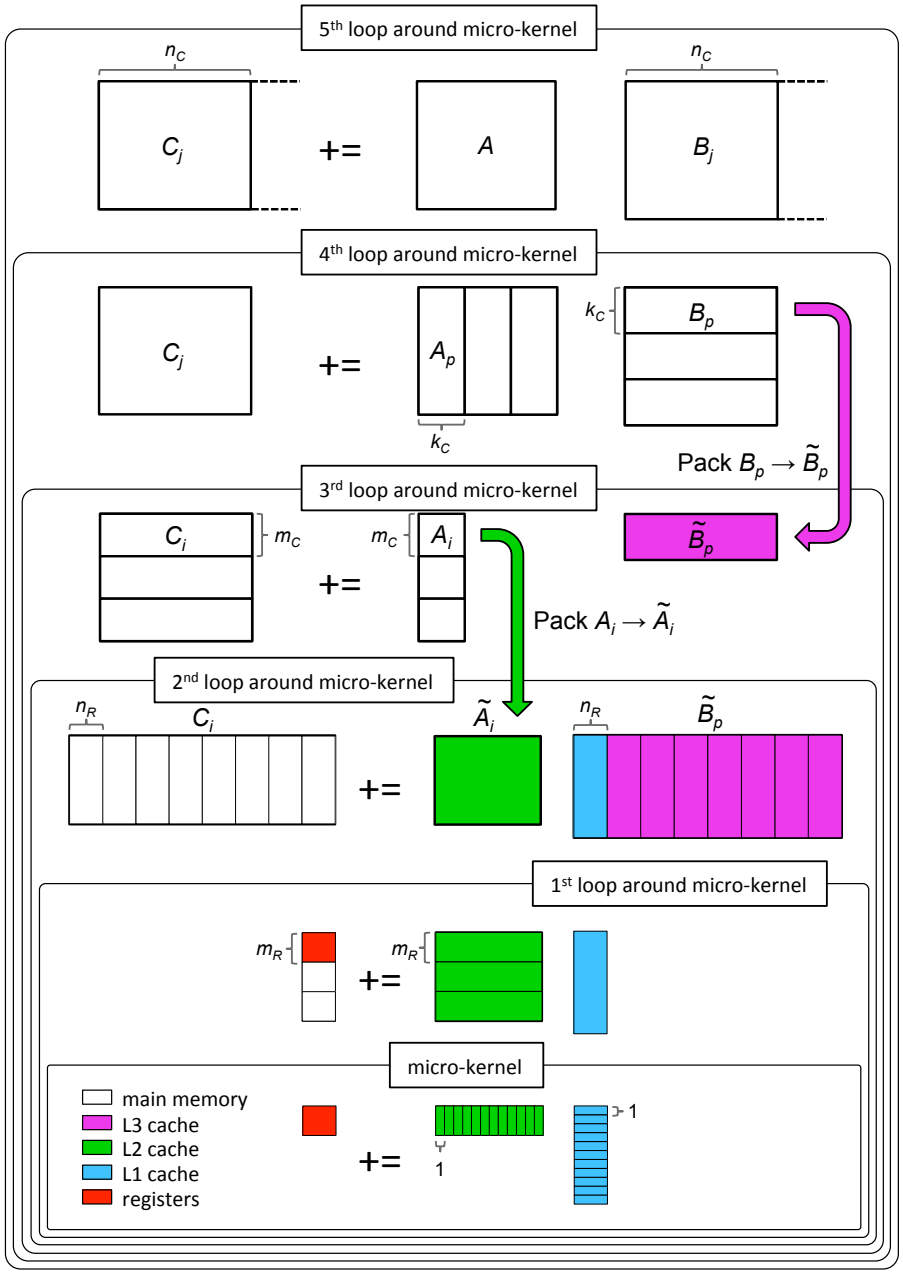
Goto's Algorithm



Goto's Algorithm



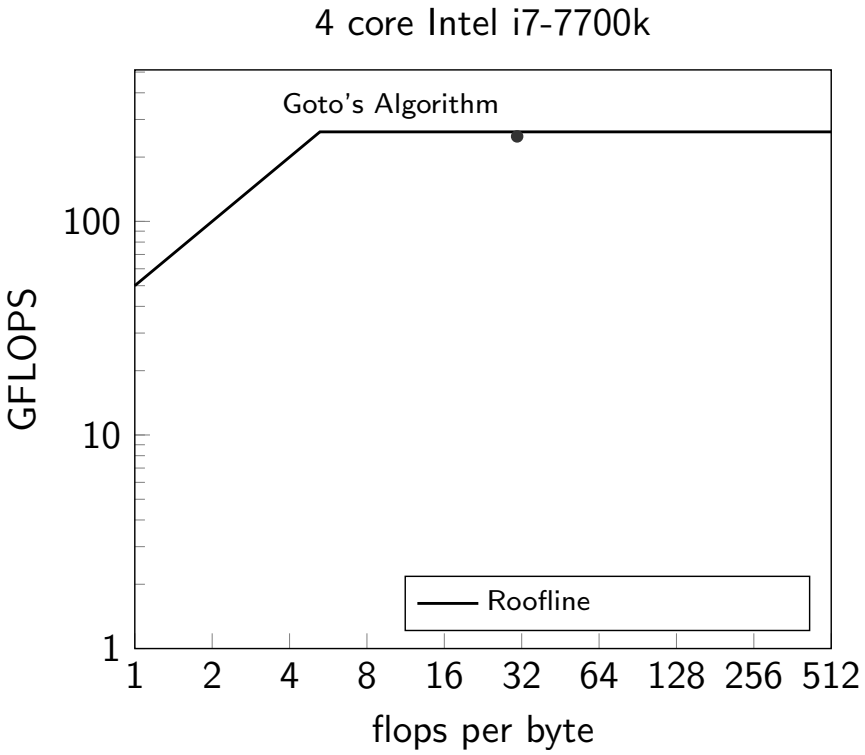
Goto's Algorithm



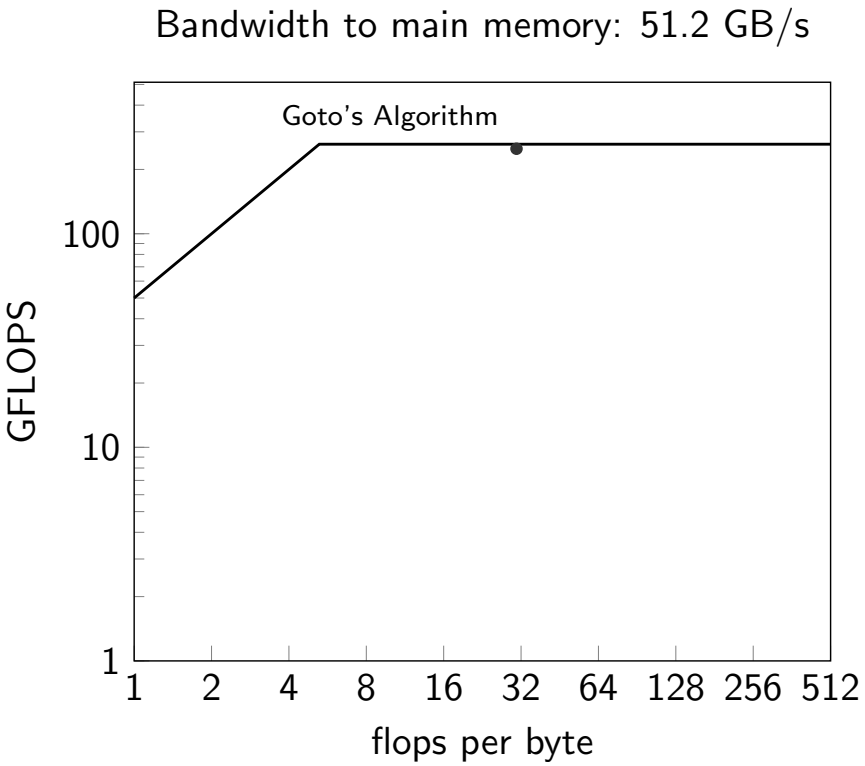
I/O cost of Goto's Algorithm

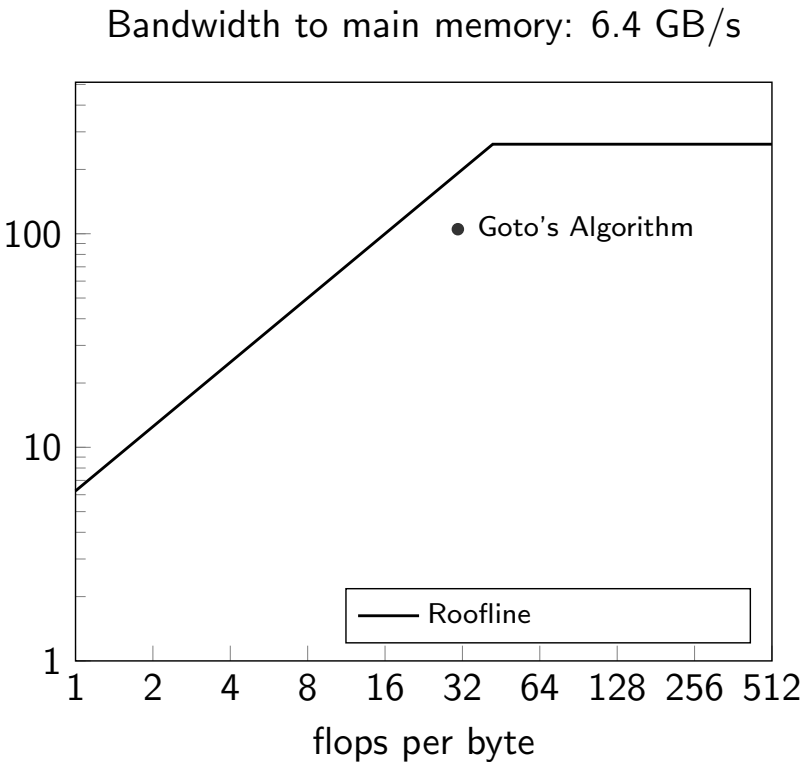
- Reuse dictates the I/O cost for Goto's Algorithm
- Each time an element is read from main memory:
  - An element of A is reused n_c times
  - An element of B is reused m times
  - An element of C is reused k_c times
- Overall I/O costs of:
  - A: $\frac{mnk}{n_c}$
  - B: $\frac{mnk}{m}$
  - C: $\frac{mnk}{k_c}$

Roofline model



Roofline model





Outline

- Introduction
- State-of-the-art MMM
- Lower bounds
- Algorithms
- Experiments

I/O lower bounds

- Theoretical minimum I/O cost for an operation
- We want to find the greatest I/O lower bound
- Model of computation
  - 2 layers of memory: slow and fast
  - Slow memory has unlimited capacity
  - Fast memory has capacity S
  - Data must be in fast memory before computing with it

Related work

- Hong and Kung (1981)
  - I/O lower bound: $\Omega\left(\frac{mnk}{\sqrt{S}}\right)$
- Irony, Toledo, and Tiskin (2004)
  - I/O lower bound: $\frac{mnk}{2\sqrt{2}\sqrt{S}}$
  - With a little calculus this can be improved to $\frac{mnk}{\sqrt{S}}$
- Tyler Smith, Robert van de Geijn, Bradley Lowery, and Julien Langou (2017)
  - I/O lower bound $\frac{2mnk}{\sqrt{S}}$
  - Under submission at ACM TOMS

Lower bound strategy

- Consider any algorithm for MMM
- Break the algorithm into phases
  - Each phase has an I/O cost of exactly S [1]
- If there must be at least h phases, and each phase has an I/O cost of S, the overall I/O cost must be at least Sh.
- Determine minimum number of phases
  - Let F be an upper bound on the multiplications during a phase
  - There are mnk total multiplications during MMM
  - There must be at least $\frac{mnk}{F}$ phases
  - Determine F based on the number of elements available
  - Each phase: $2S$ elements available as inputs and $2S$ elements available as outputs

[1] except the last

Upper bound on elementary multiplications in a phase

Irony, Toledo, and Tiskin (2004)

- Inequality from Loomis and Whitney (1949)
  - Using N_A, N_B, and N_C elements of A, B, and C
  - Can perform at most $\sqrt{N_A N_B N_C}$ multiplications
- At most $2S$ elements available as inputs, and $2S$ elements available as outputs
  - $N_A \leq 2S$, $N_B \leq 2S$, and $N_C \leq 2S$
- At most $\sqrt{8S^3} = \left(2\sqrt{2}\right)\left(S\sqrt{S}\right)$ multiplications in a phase
- Gives an overall lower bound of $\frac{1}{2\sqrt{2}}\frac{mnk}{\sqrt{S}}$

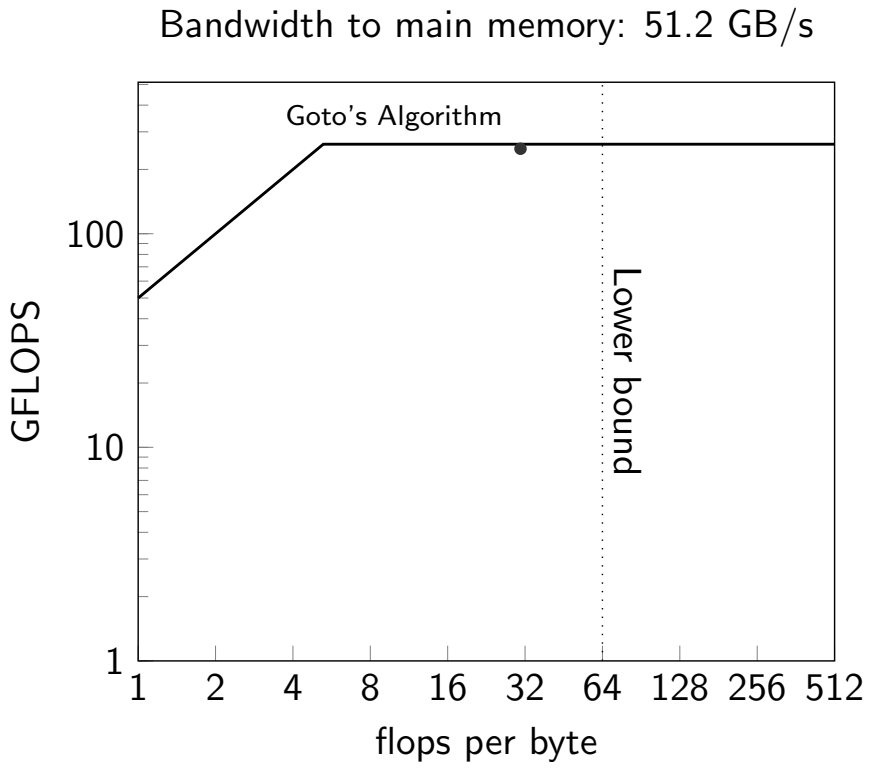
Improving the lower bound

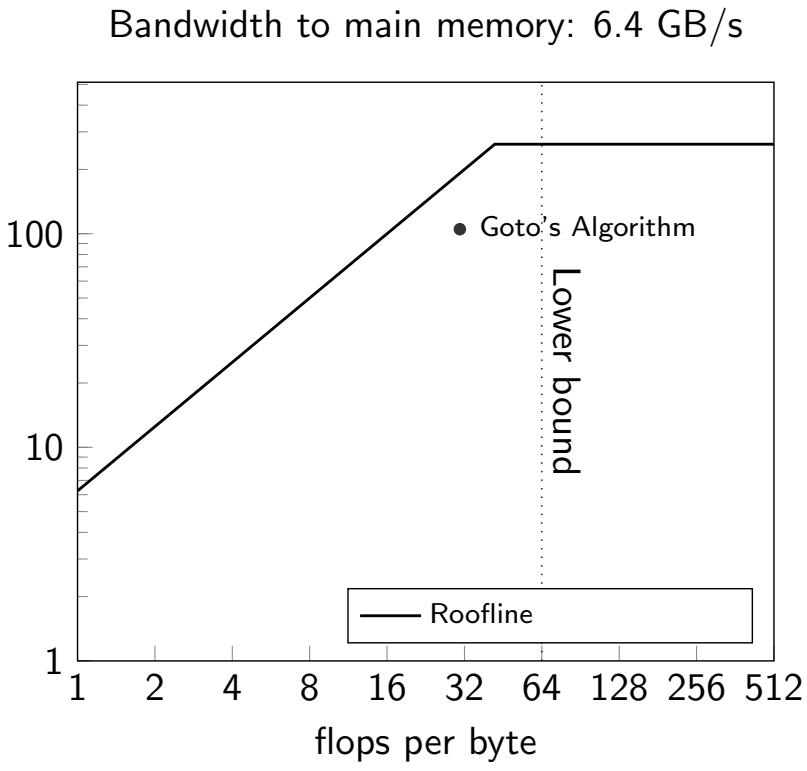
- Assume we perform FMAs instead of elementary multiplications
  - In an FMA, elements of A, B, and C are all inputs
  - We can reason about the input cost of C
- What if we generalize the I/O cost of each phase?
  - Each phase can have $S + M$ inputs and $S + M$ outputs
  - This adds a degree of freedom to our lower bound

Upper bound on FMAs during a phase

- There are at most $S + M$ inputs
  - $N_A + N_B + N_C \leq S + M$
- We again use the Loomis-Whitney inequality
- Maximize $\sqrt{N_A N_B N_C}$ when $N_A + N_B + N_C = S + M$
- Maximized when $N_A = N_B = N_C$
- Then our lower bound is $\frac{3\sqrt{3}Mmnk}{(S+M)\sqrt{S+M}}$
- Finding the greatest lower bound
  - Maximizing over M, this occurs when $M = 2S$
  - The greatest lower bound is $\frac{2mnk}{\sqrt{S}}$

Roofline model

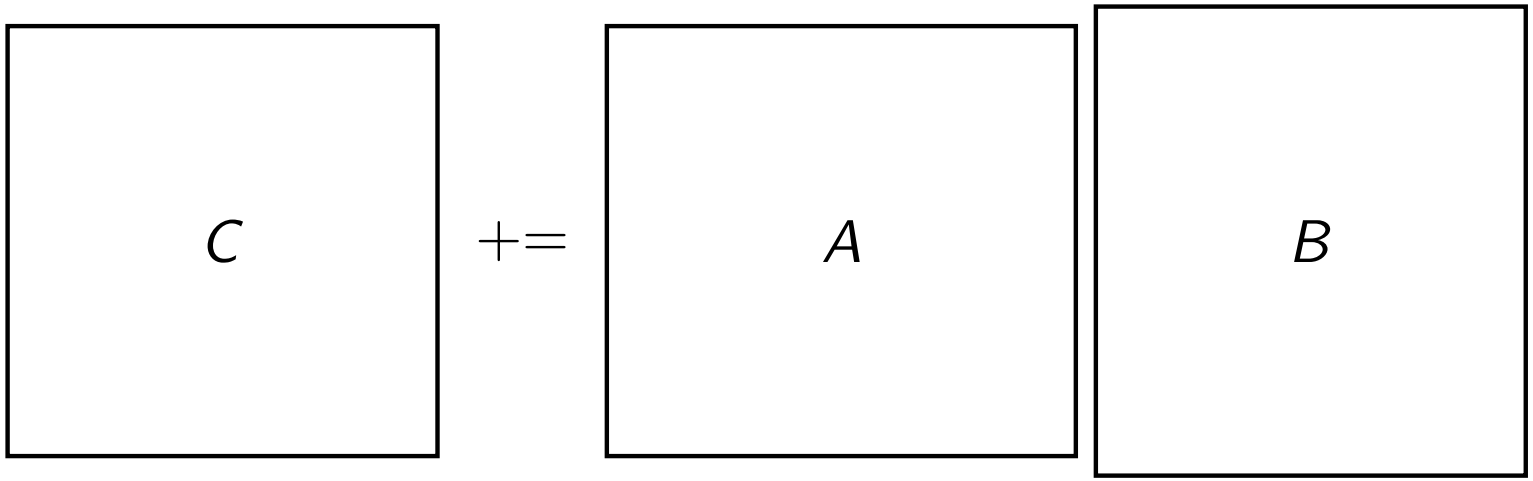




Outline

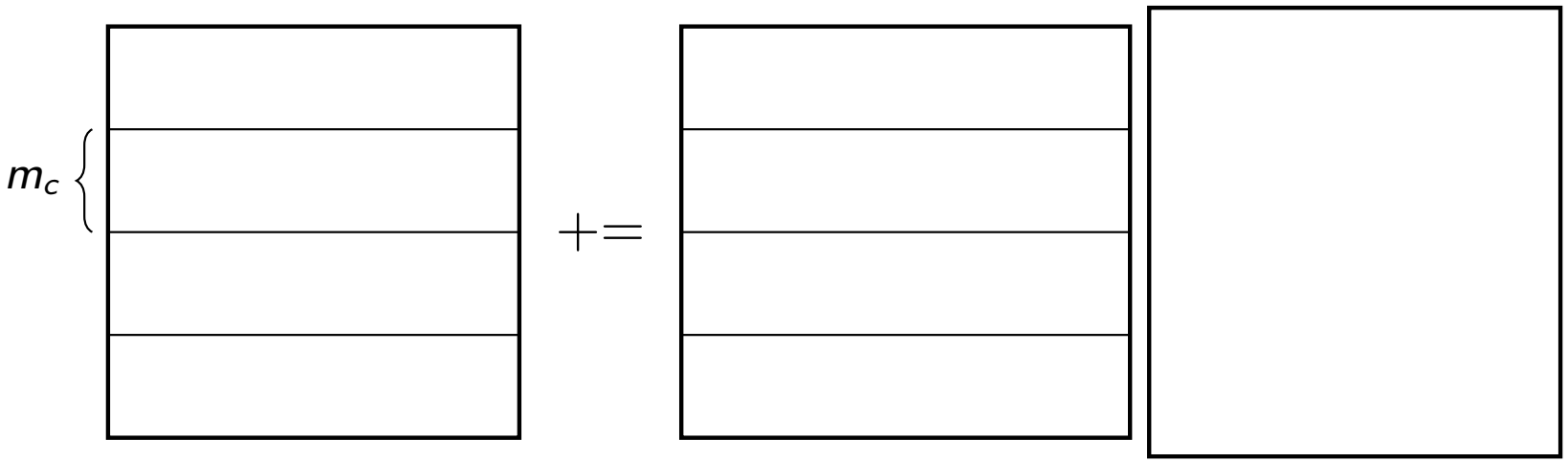
- Introduction
- State-of-the-art MMM
- Lower bounds
- Algorithms
  - Single level of cache
  - Multiple levels of cache
- Experiments

Resident C



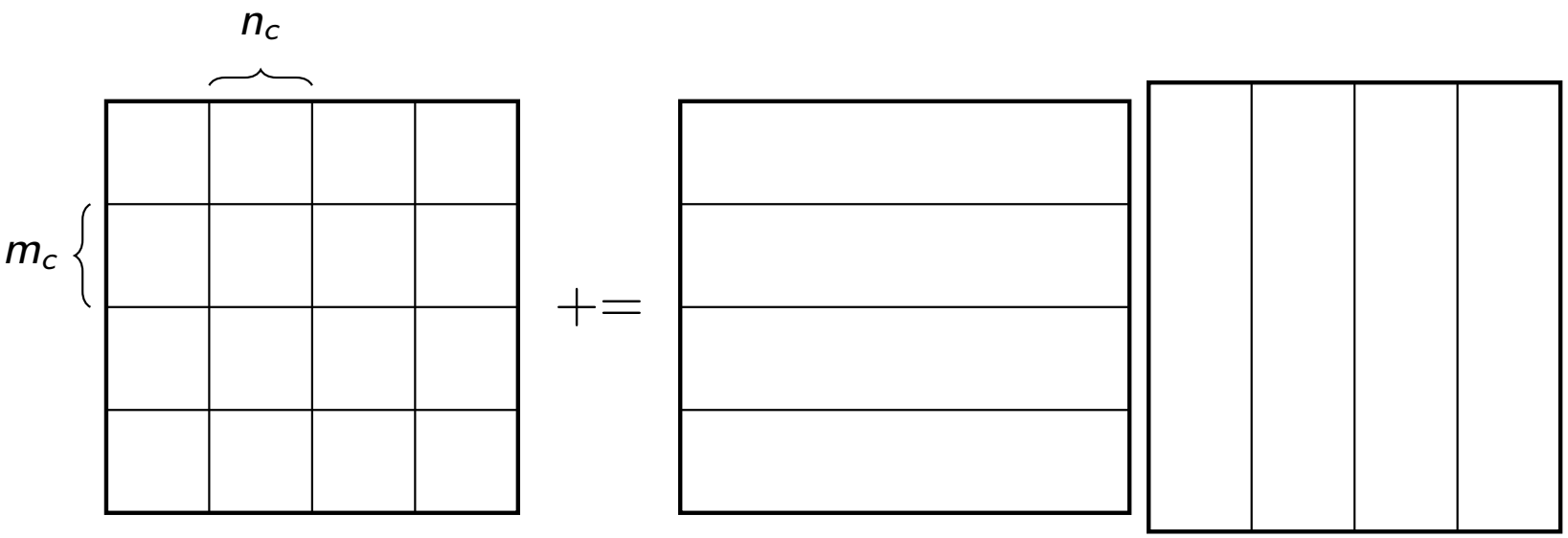
Resident C

Partition m dimension



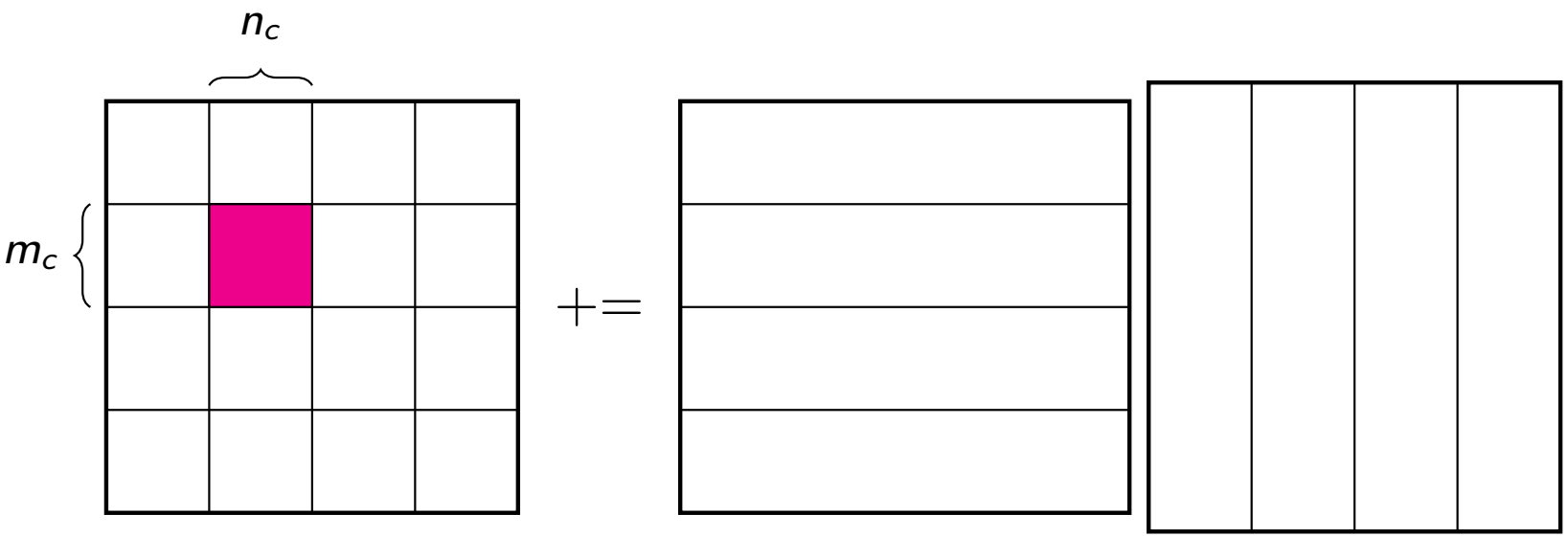
Resident C

Partition n dimension



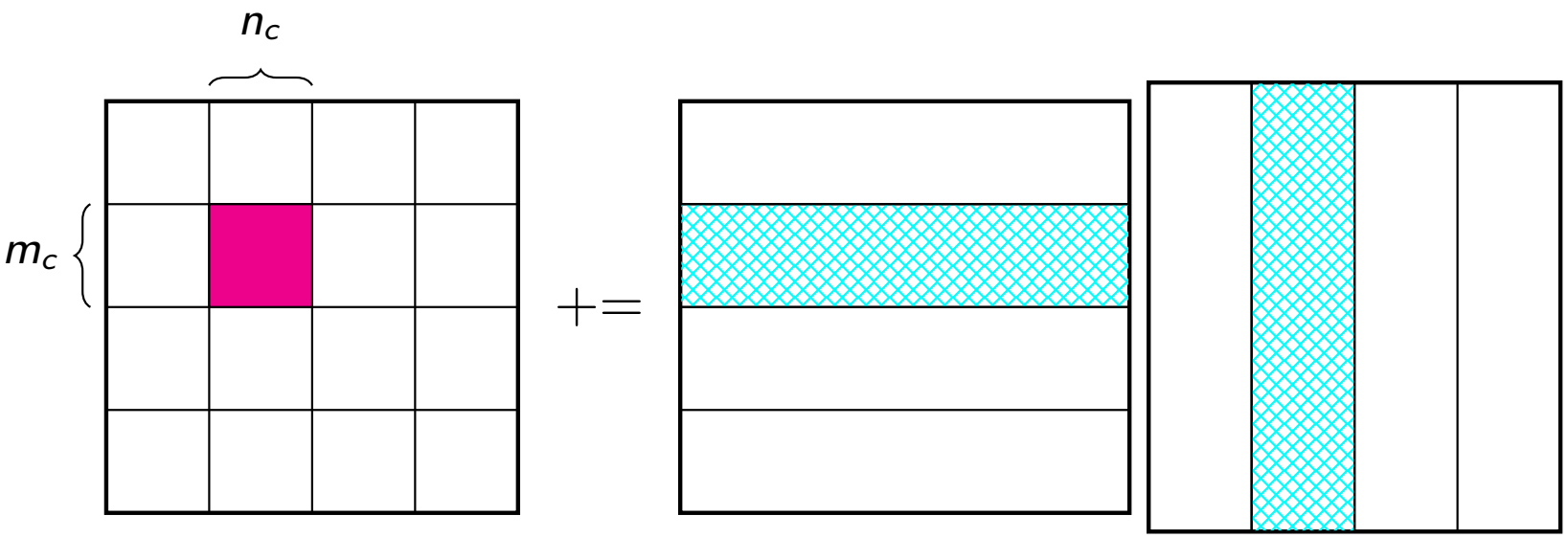
Resident C

Move $m_c \times n_c$ block of C into fast memory



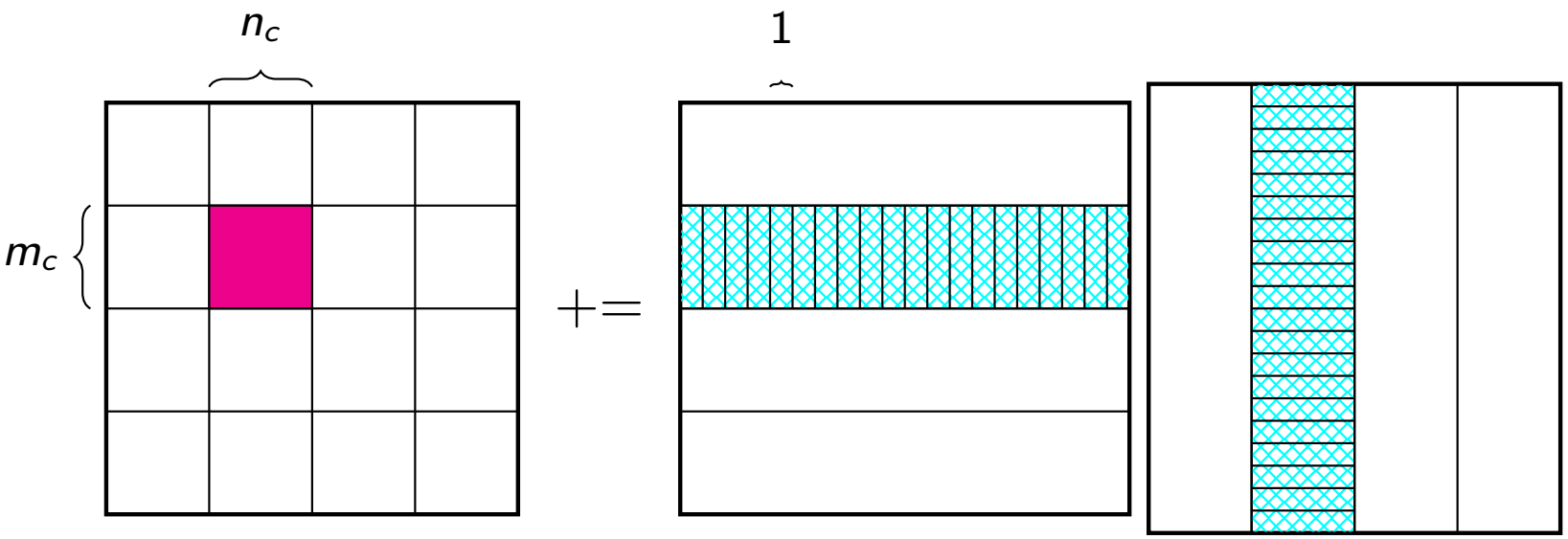
Resident C

Stream panels of A and B from slow memory



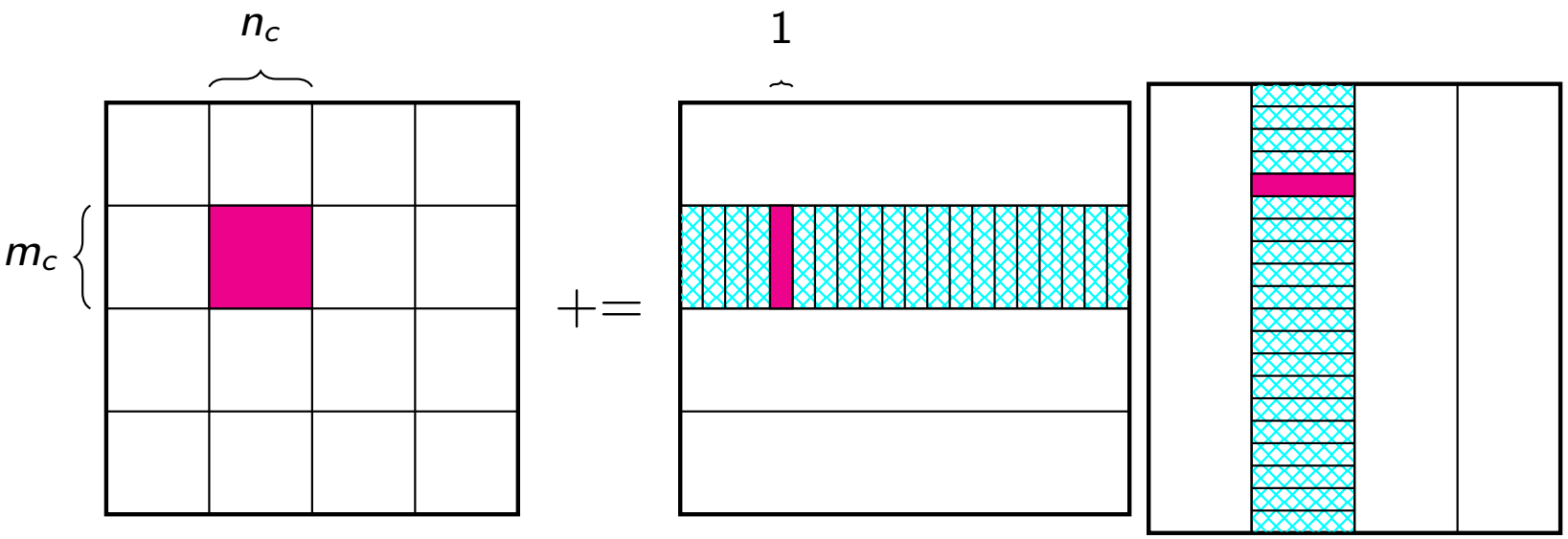
Resident C

Partition k dimension

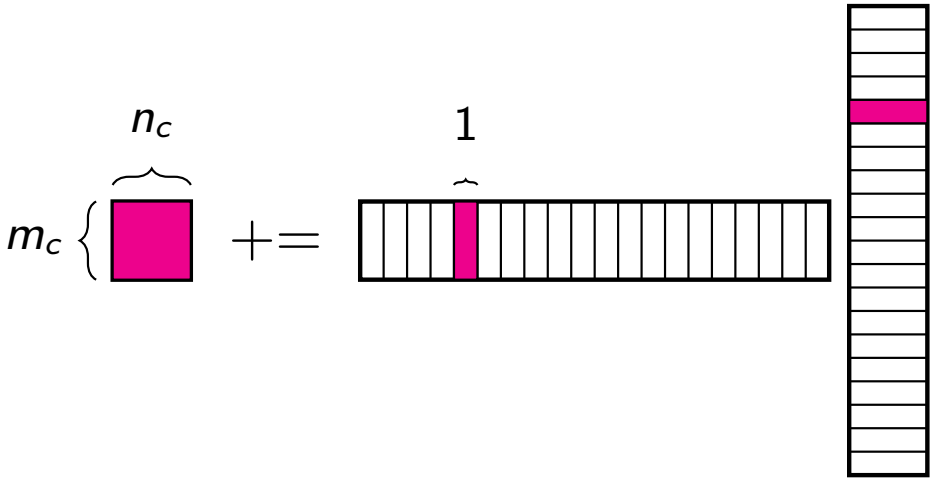


Resident C

Move vectors into fast memory

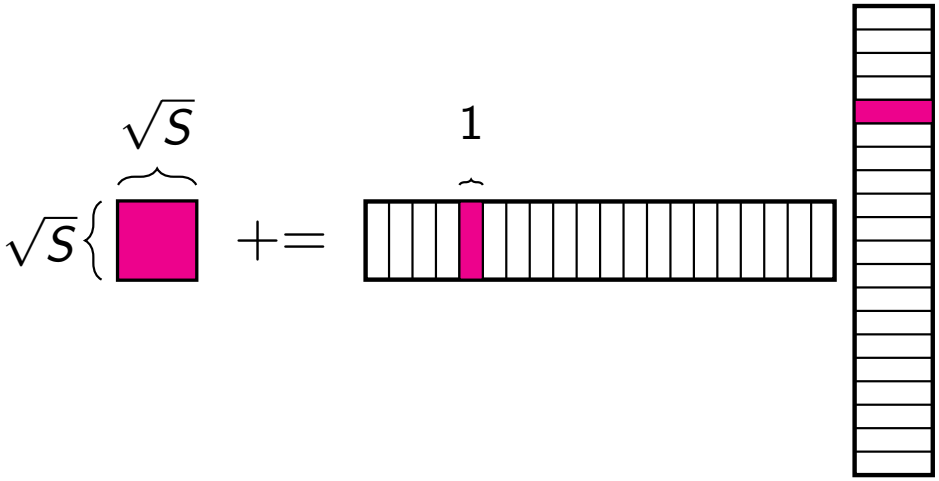


I/O cost for Resident C



- I/O cost per block dot product:
  - $C_{i,j}$: $m_c n_c$ reads and $m_c n_c$ writes.
  - A_i: $m_c k$ reads.
  - B_j: $k n_c$ reads.
- Total I/O cost:
  - C: mn reads and mn writes.
  - A: $\frac{mnk}{n_c}$ reads.
  - B: $\frac{mnk}{m_c}$ reads.

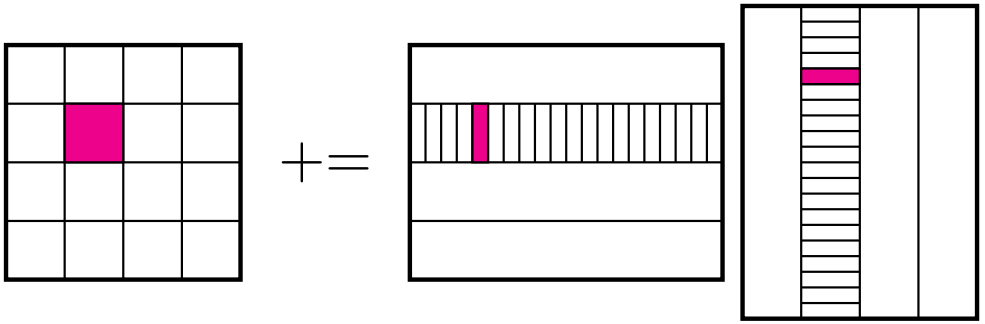
Choosing blocksizes for Resident C



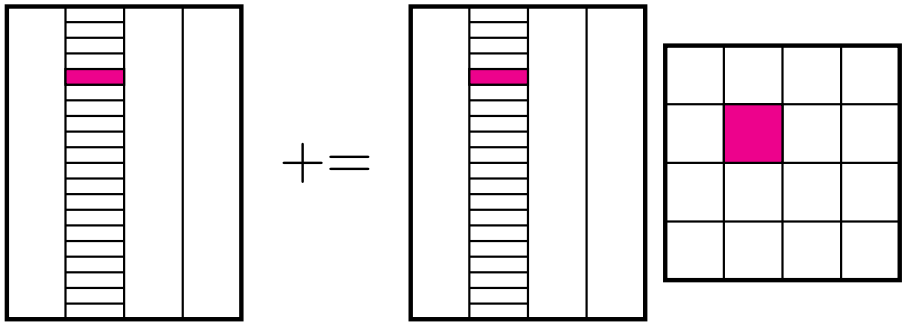
- If $m_c \approx n_c \approx \sqrt{S}$
- Total I/O cost:
  - C: mn reads and mn writes.
  - A: $\frac{mnk}{\sqrt{S}}$ reads.
  - B: $\frac{mnk}{\sqrt{S}}$ reads.
- If m, n, k are large and we can ignore lower ordered terms
  - I/O cost is $\frac{2mnk}{\sqrt{S}}$
  - Same as lower bound

Three algorithms

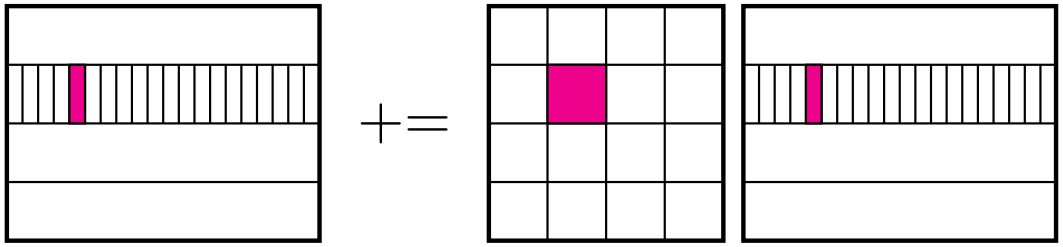
Resident C



Resident B



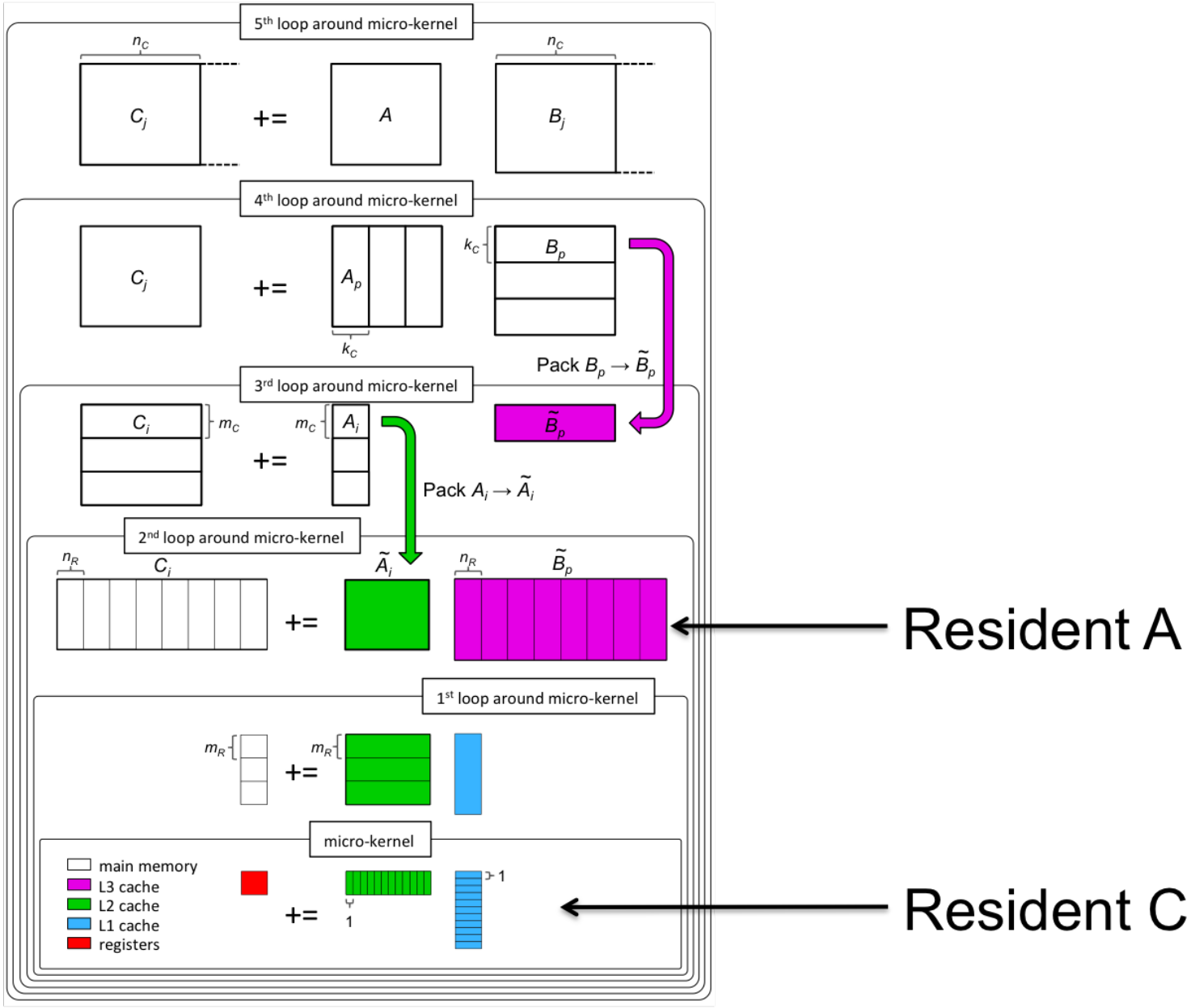
Resident A



Data in cache.

Data in main memory.

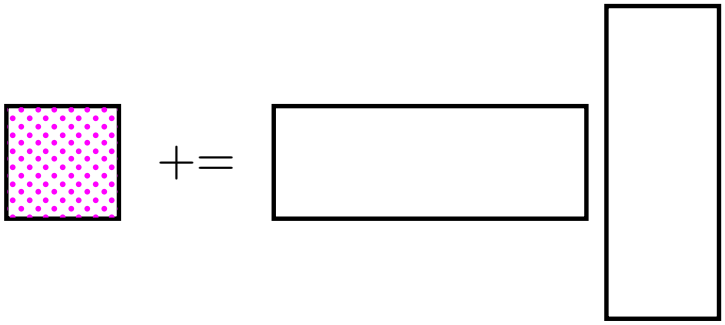
Resident A, B, and C algorithms in Goto's Algorithm



Algorithms for multiple levels of cache

- Suppose we have 2 levels of cache: L_2 and L_1
- We have 3 algorithms
  - Resident A, Resident B, and Resident C
  - Each is associated with a shape of MMM
- Suppose we have one of those shapes at the L_2 level
- Then how do we also encounter one at the L_1 level?
  - We can do it with two loops

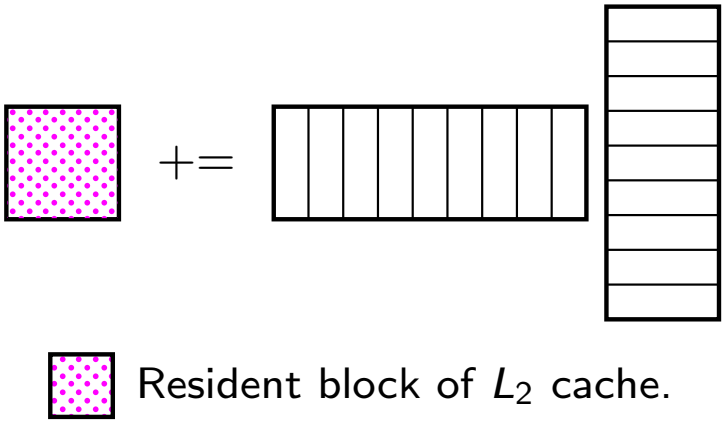
Resident C at the L_2 cache



Resident block of L_2 cache.

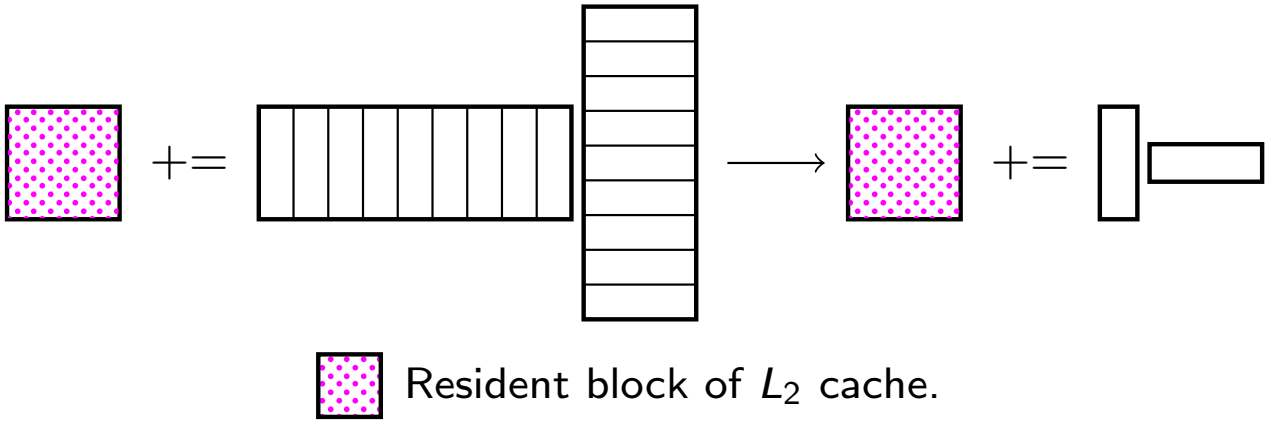
L_1 outer loop

Partition k dimension



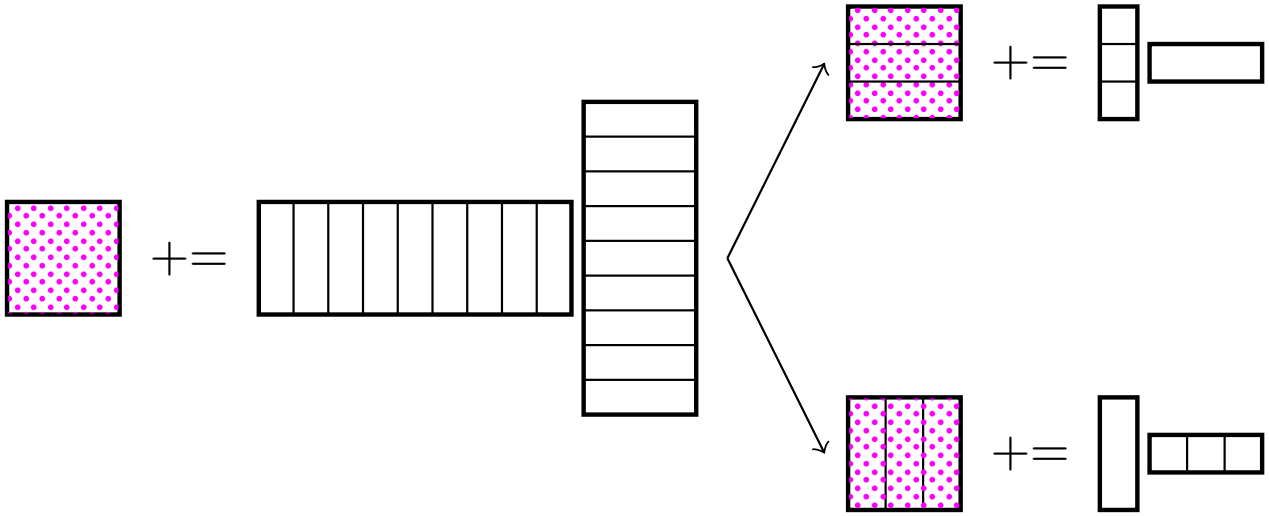
L_1 outer loop

Partition k dimension



L_1 inner loop

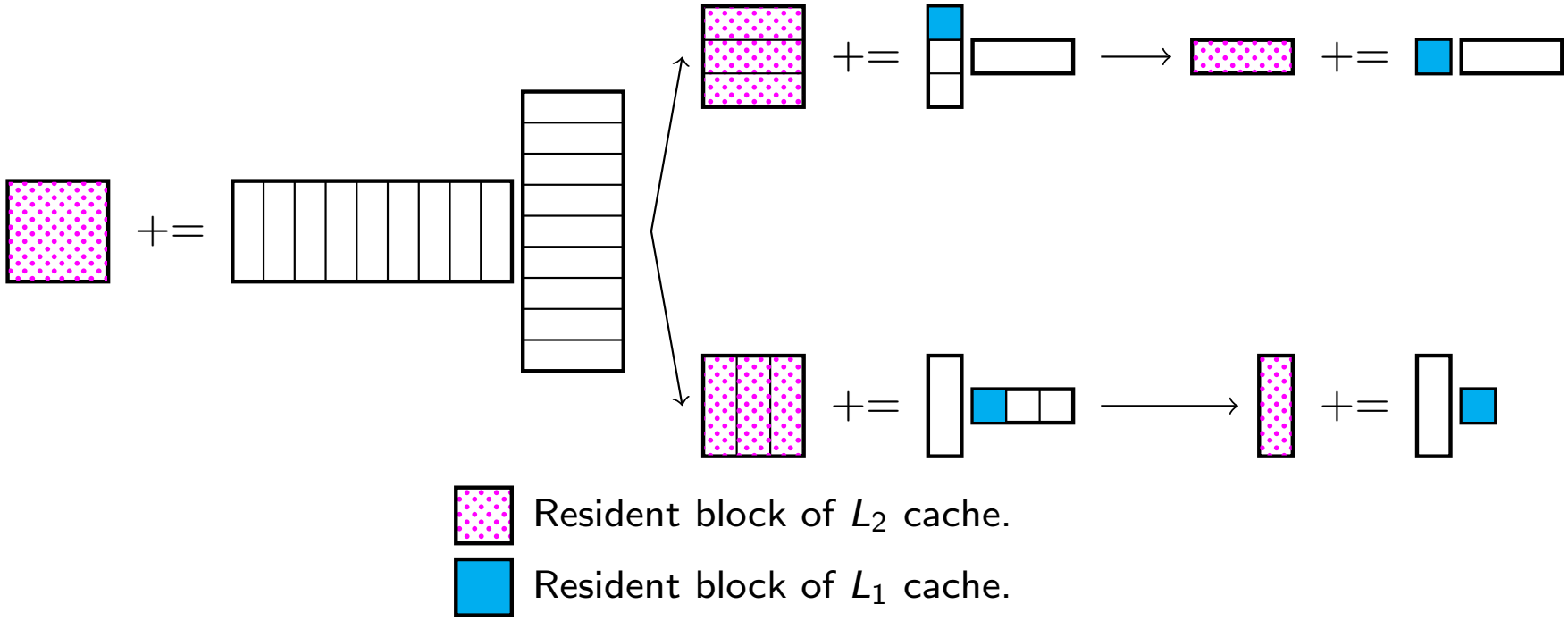
Partition either m or n direction



Resident block of L_2 cache.

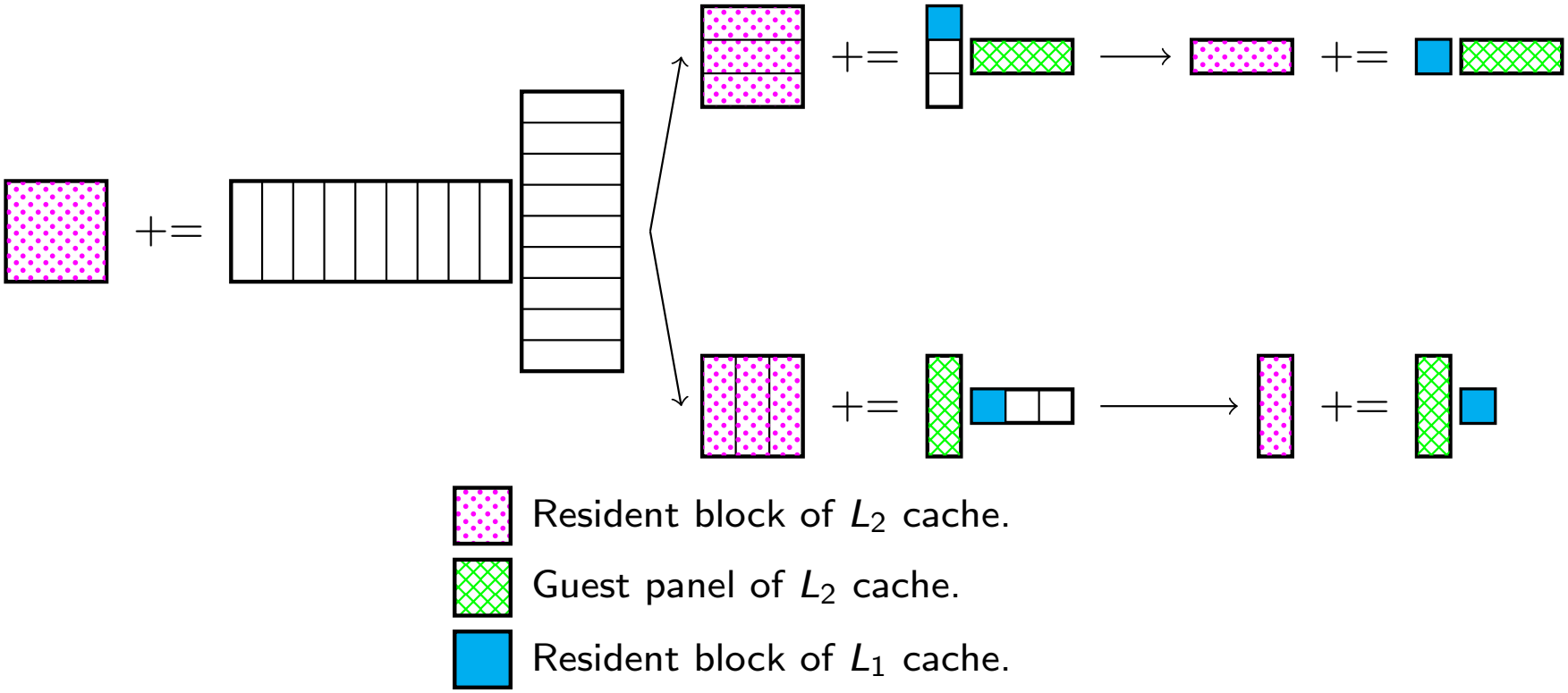
L_1 inner loop

Partition either m or n direction

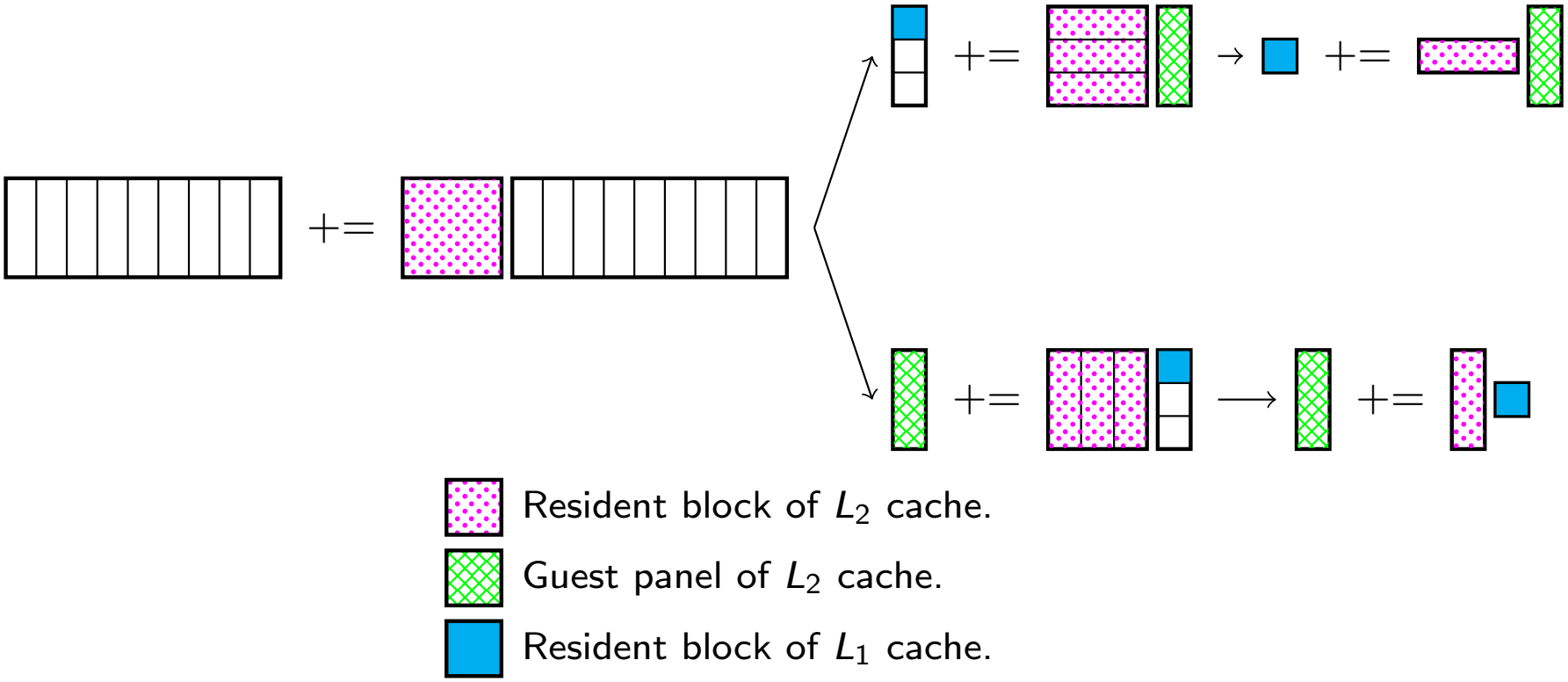


L_1 inner loop

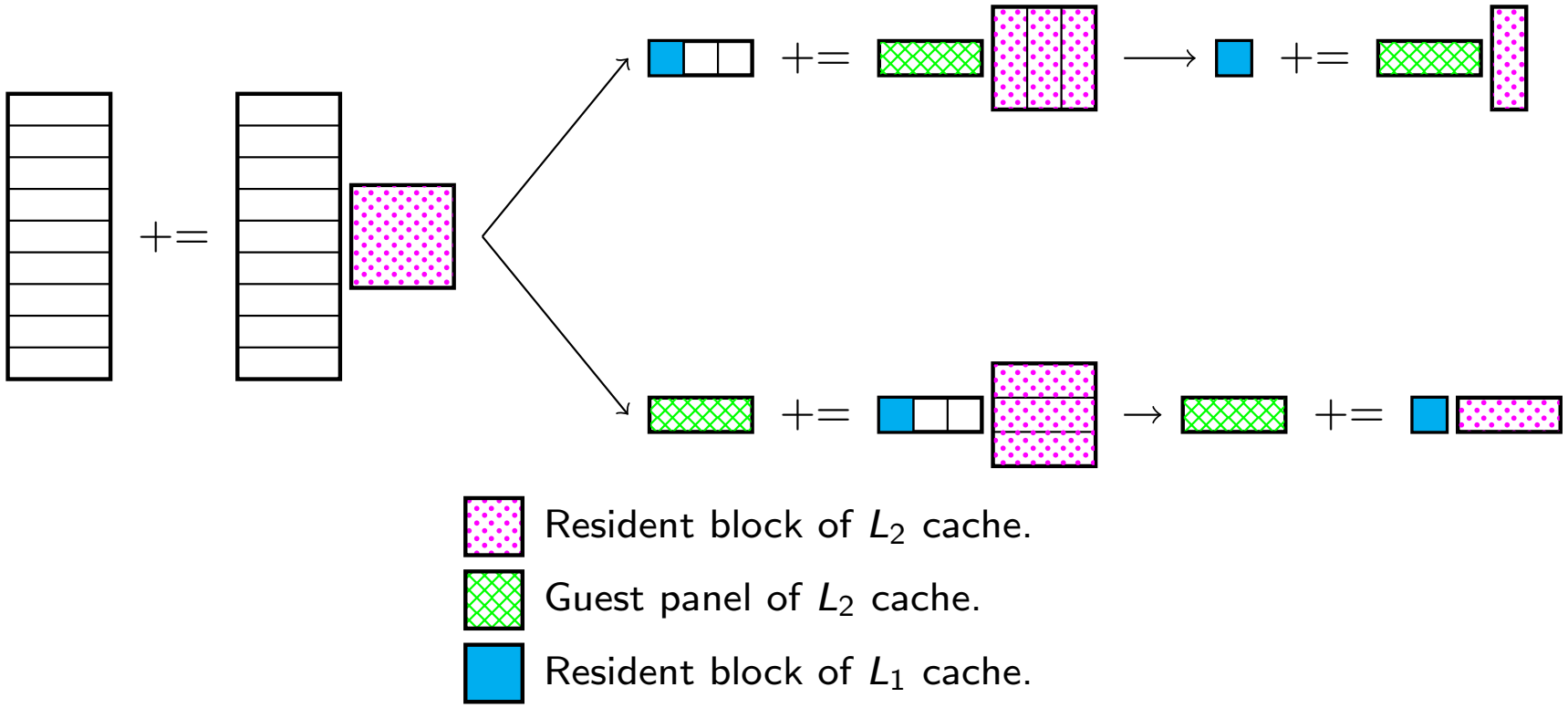
Partition either m or n direction



Resident A at the L_2 cache



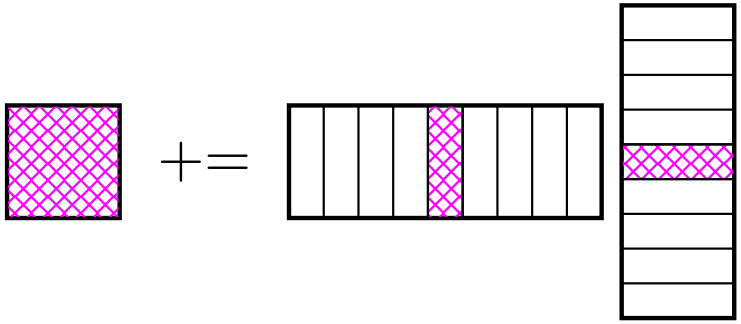
Resident B at the L_2 cache



Families of algorithms

- We start out with one of the three shapes at the L_h cache
- With 2 loops, we have one of the other two shapes at the L_{h-1} cache
- Repeat the process for subsequent levels of cache
- We name algorithms based on the resident matrix at each level of cache
  - e.g. $B_3 A_2 C_1$

Tradeoffs



- Blocking for L_{h-1} cache means more data must fit into L_h
- For LRU caches, all elements used during one iteration of the L_{h-1} outer loop must fit into the L_h cache
- For ideal caches, the L_h resident matrix and L_h guest matrix must fit into the L_h cache
- This increases L_h I/O cost
  - Depends on the ratio between S_h and S_{h-1}

What if it's not worth optimizing for both levels of cache?

- One option is to use smaller blocksizes for the L_{h-1} cache
- Skipping a level of cache
  - Optimize for the L_h and L_{h-2} caches
  - Under the right circumstances, the L_h guest matrix can be placed in the L_{h-1} cache
  - We can think of Goto's Algorithm as "skipping" the L_3 and L_1 caches.
    - We can call Goto's Algorithm "$A_2 C_R$"

Outline

- Introduction
- State-of-the-art MMM
- Lower bounds
- Algorithms
- Experiments

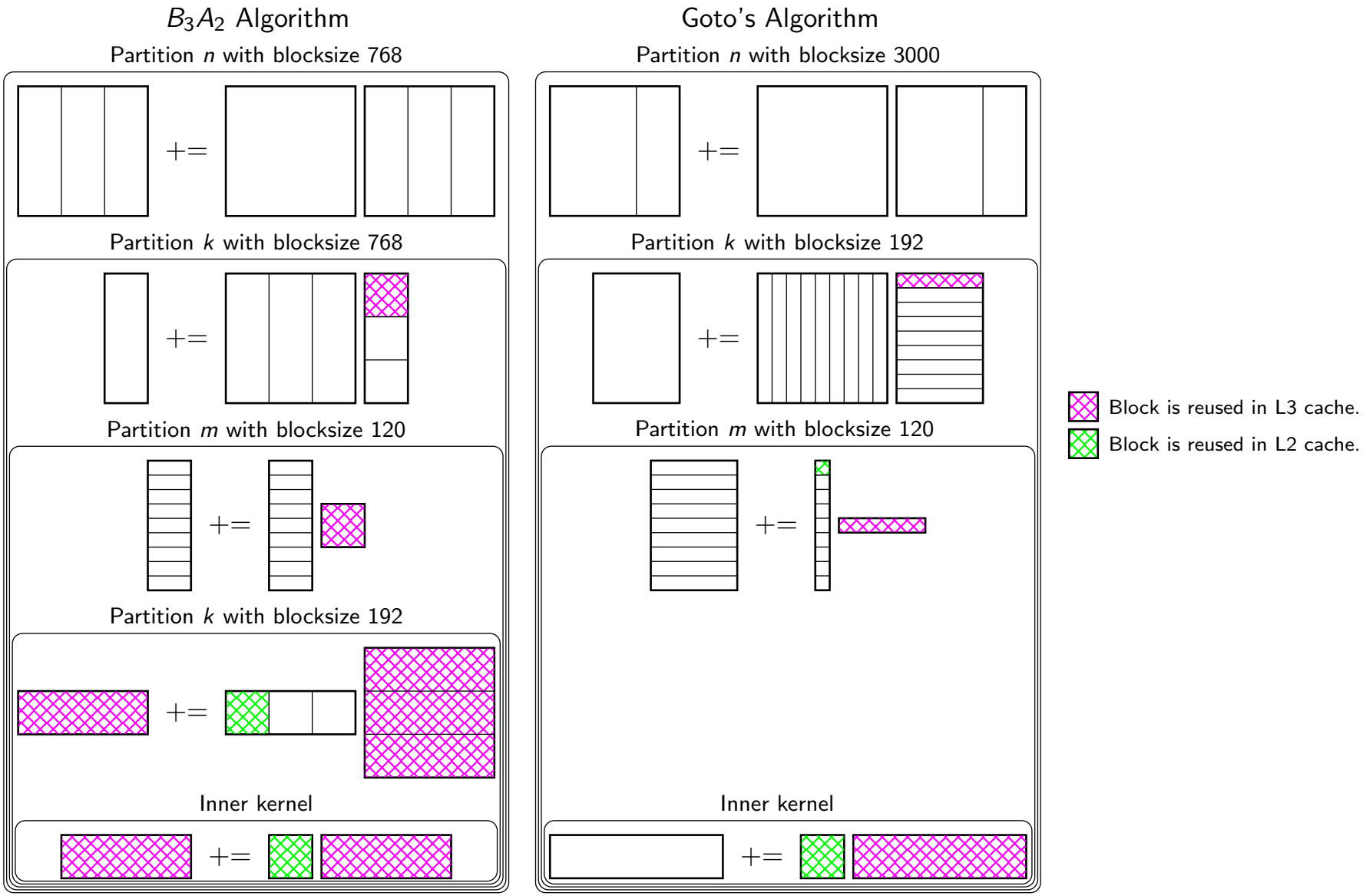
Experimental setup

- Custom-built PC with an unlocked CPU and enthusiast motherboard
- Vary BCLK, CPU multiplier, and the memory multiplier to change system characteristics
- System Details
  - Intel i7-7700K CPU
  - 4 core
  - Hyperthreading disabled
  - Turbomode disabled, CPU set to 4.2 GHz
- Hypothesis: If we reduce bandwidth to main memory, algorithms that better utilize the last level cache become more efficient than those that do not.

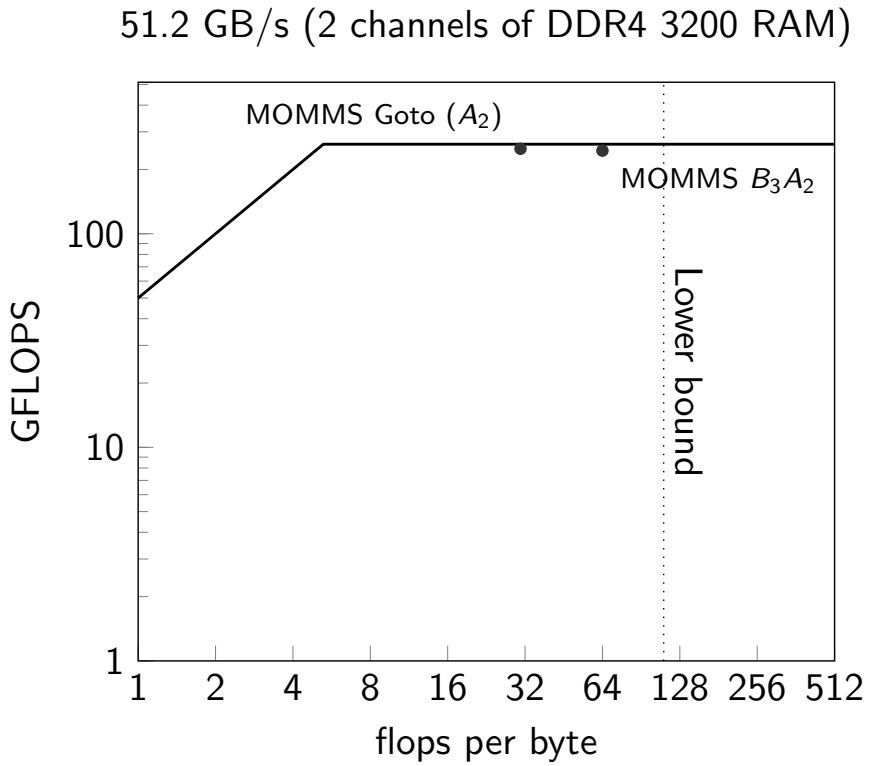
MOMMS

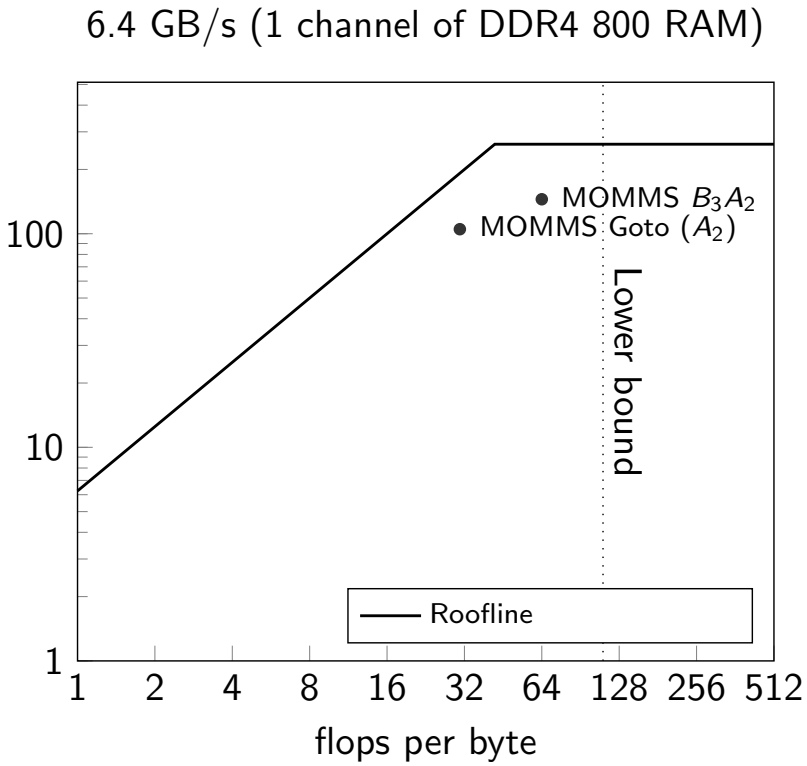
- Multilevel Optimized Matrix-matrix Multiplication Sandbox
- Framework written in Rust
- Use composition to instantiate different algorithms for MMM

Algorithms for an Intel i7-7700K

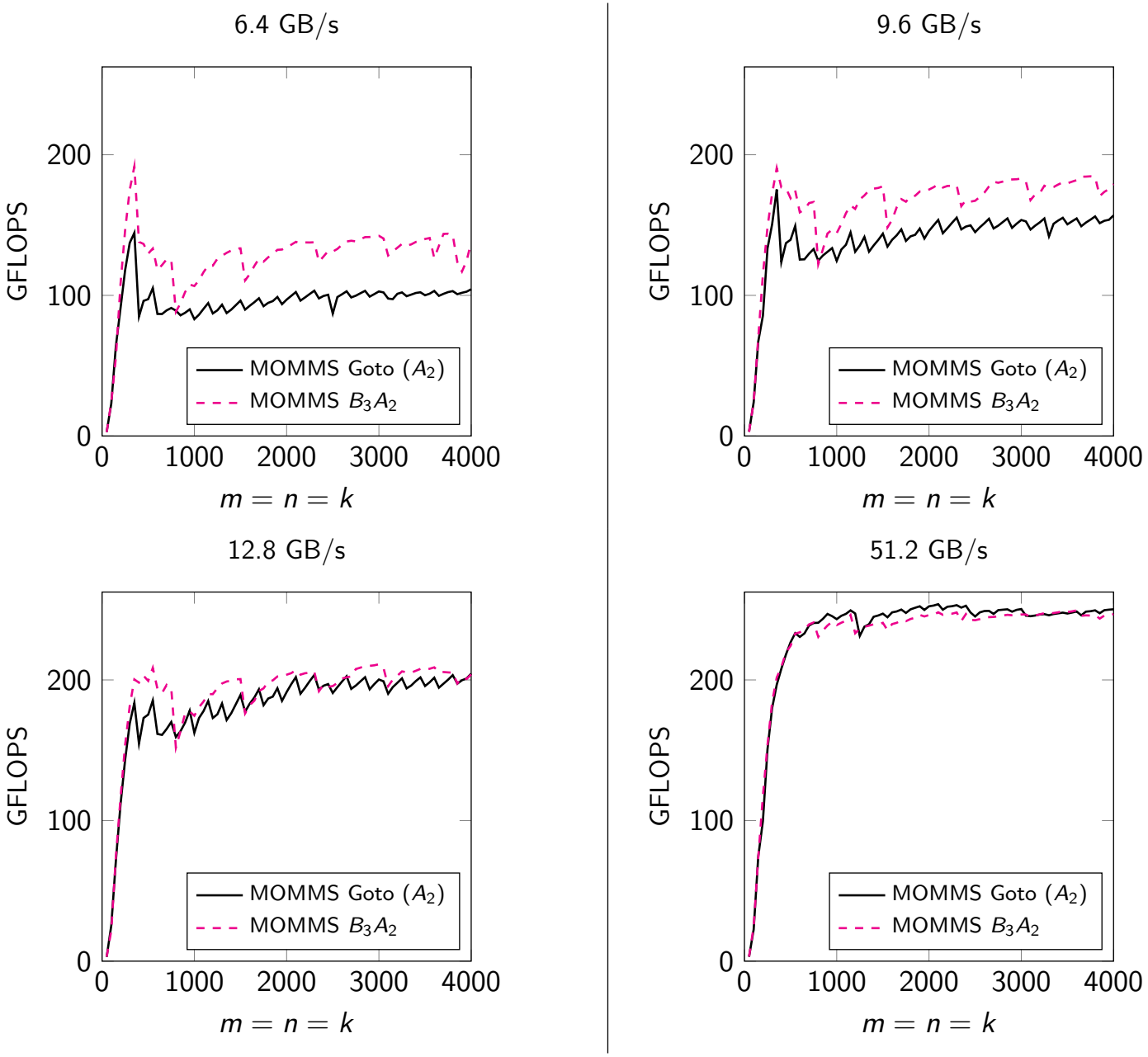


Roofline model

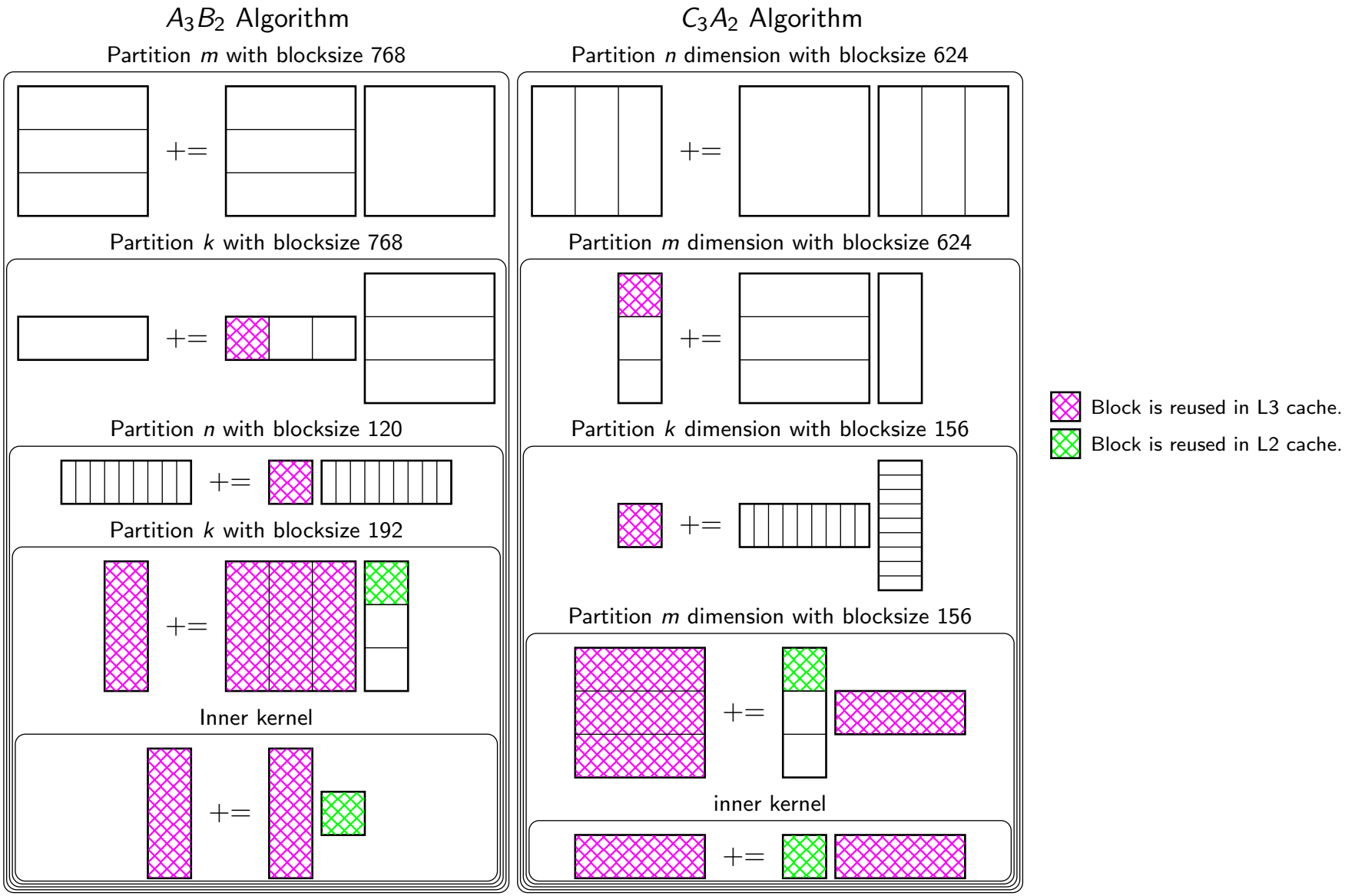




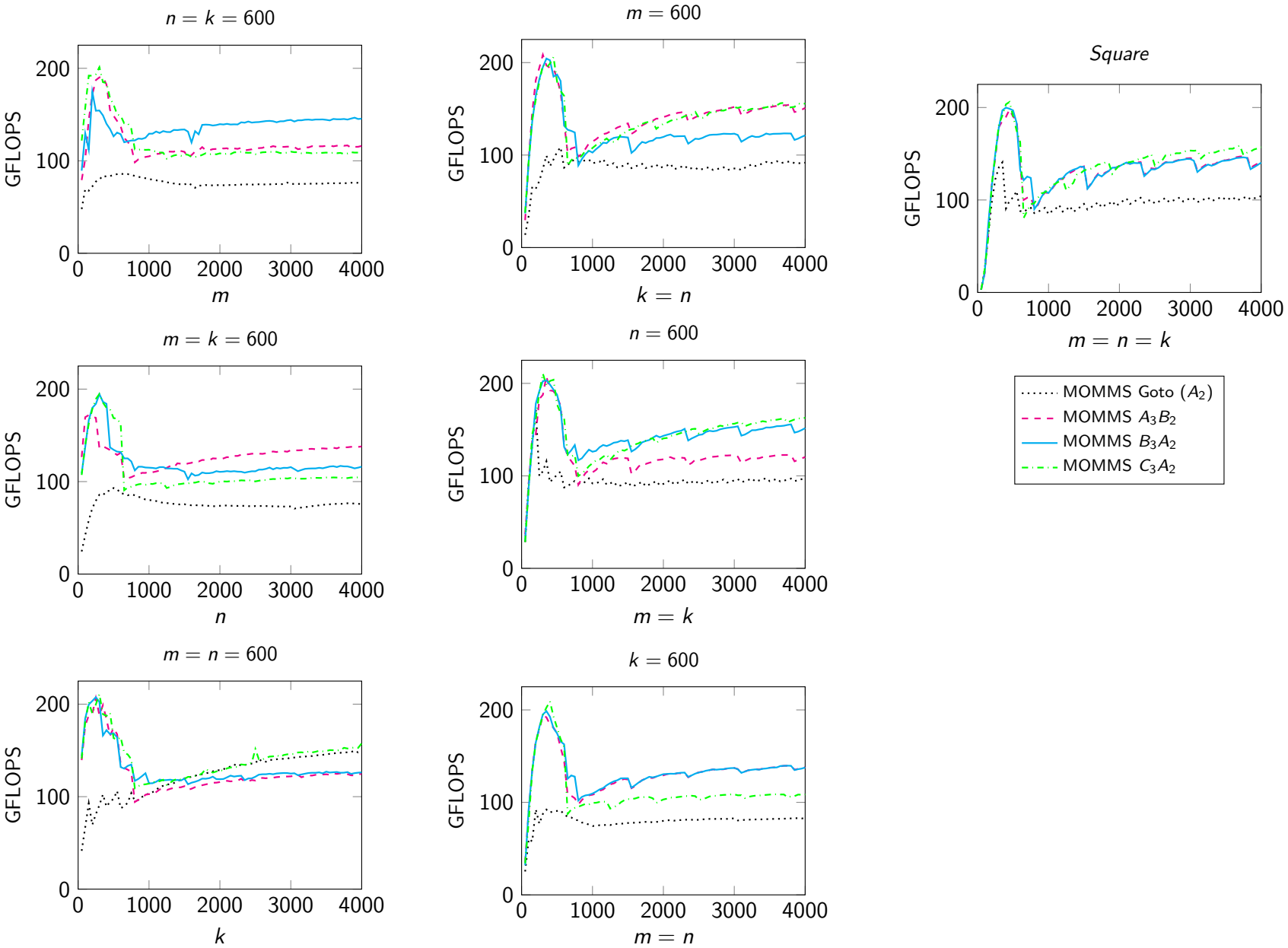
Varying bandwidth for the i7-7700K



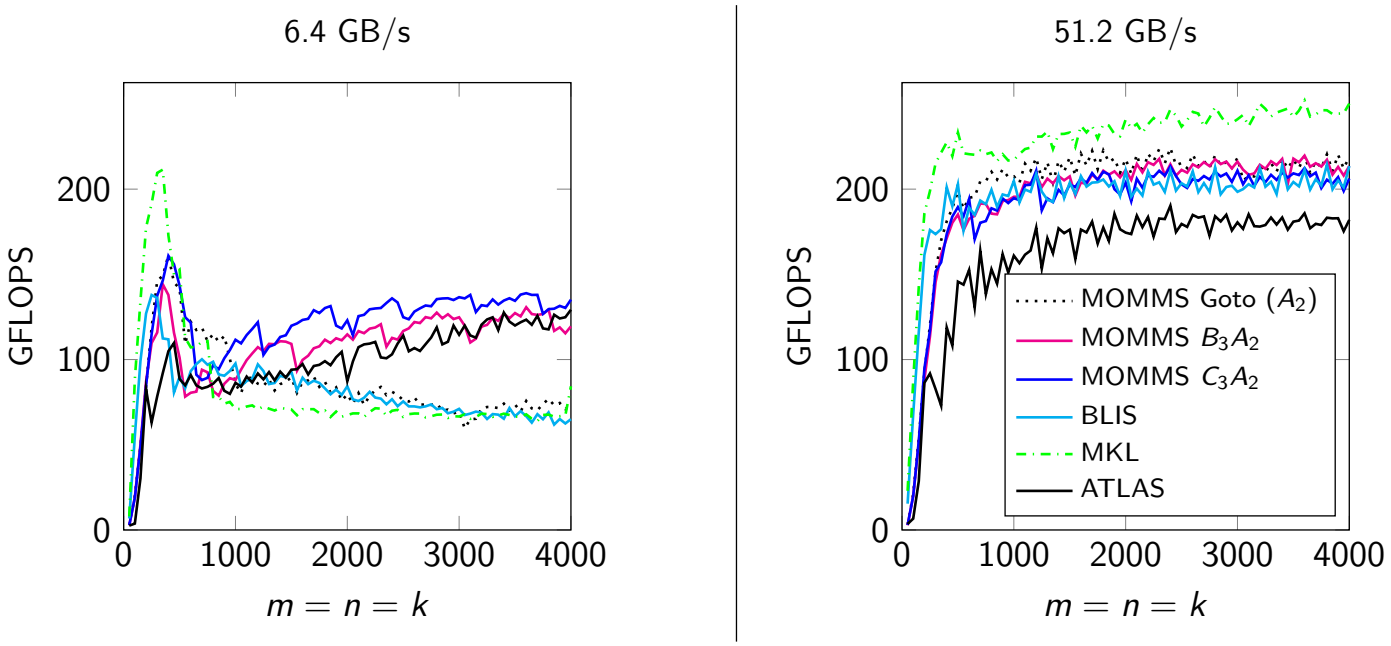
Algorithms for an Intel i7-7700K



Different shapes of MMM



Comparing with other implementations for the i7-7700K



Conclusion

- New lower bounds
  - We can reason about the optimality of algorithms
- A new family of algorithms
  - Better L3 cache utilization
  - We know how to use further levels of the memory hierarchy (L4, out of core, etc)
- Future work
  - Parallelization
  - Algorithms for other operations (rest of the level-3 BLAS, matrix factorizations, etc)

Thank you!

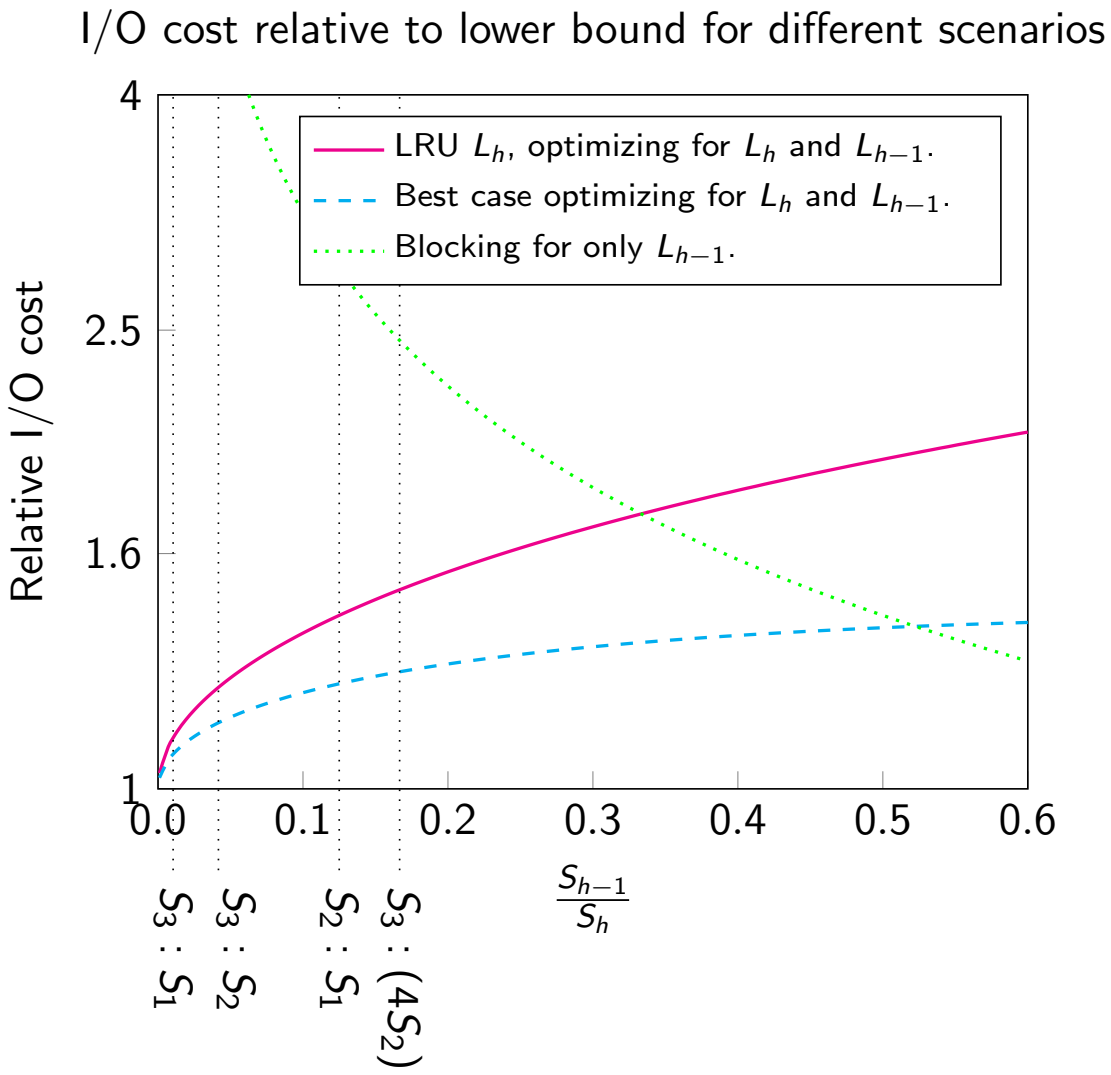
Questions?

- Tyler M. Smith
- tms@cs.utexas.edu

- MOMMS can be found at github.com/tlrmchlsmth/momms

Backup

Tradeoffs

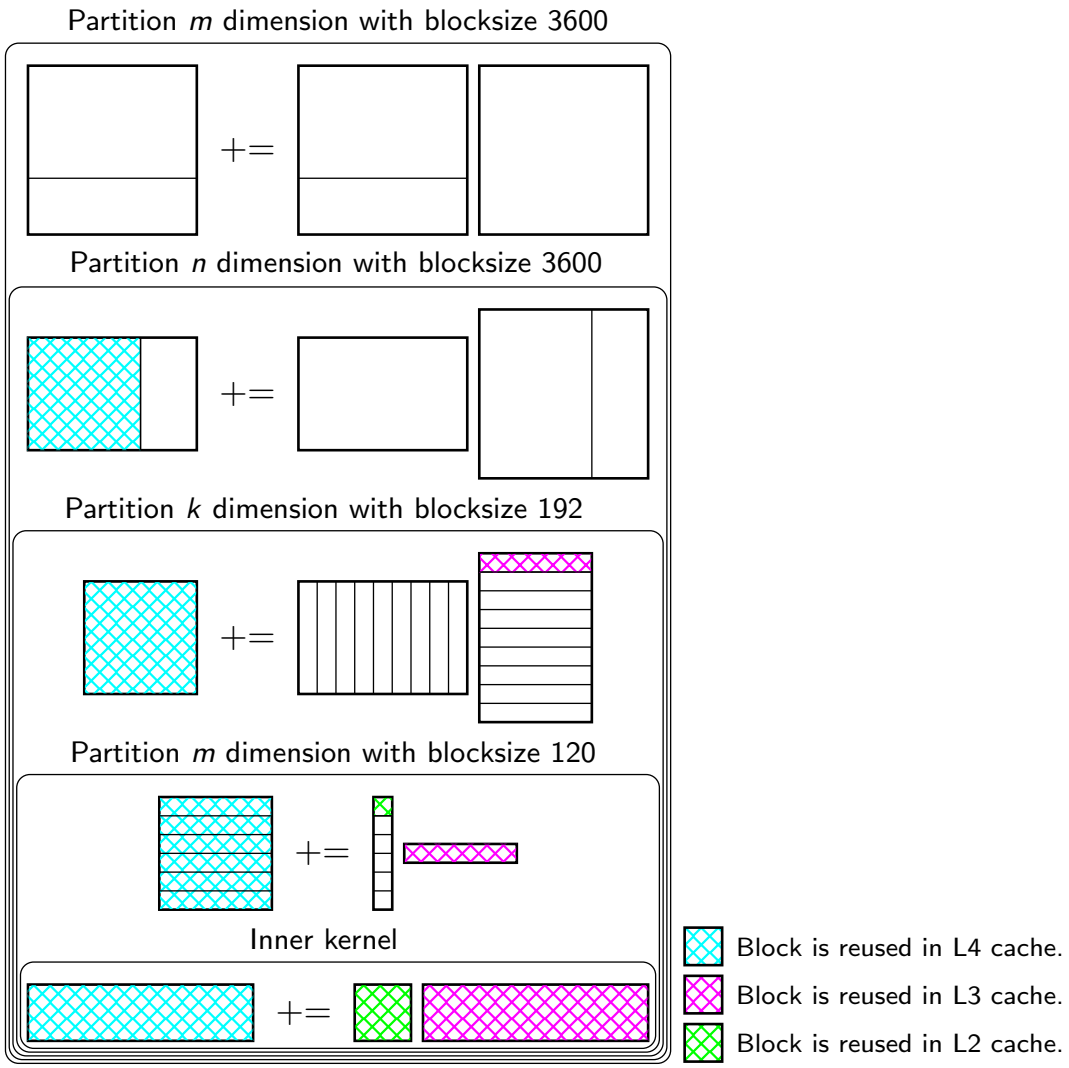

I/O cost relative to lower bound for different scenarios
Relative I/O cost
4
2.5
1.6
1
0.0
0.1
0.2
0.3
0.4
0.5
0.6
$\frac{S_{h-1}}{S_h}$
LRU L_h, optimizing for L_h and L_{h-1}.
Best case optimizing for L_h and L_{h-1}.
Blocking for only L_{h-1}.
$S_3 : S_1$
$S_3 : S_2$
$S_2 : S_1$
$S_3 : (4S_2)$

I/O cost of Goto's Algorithm

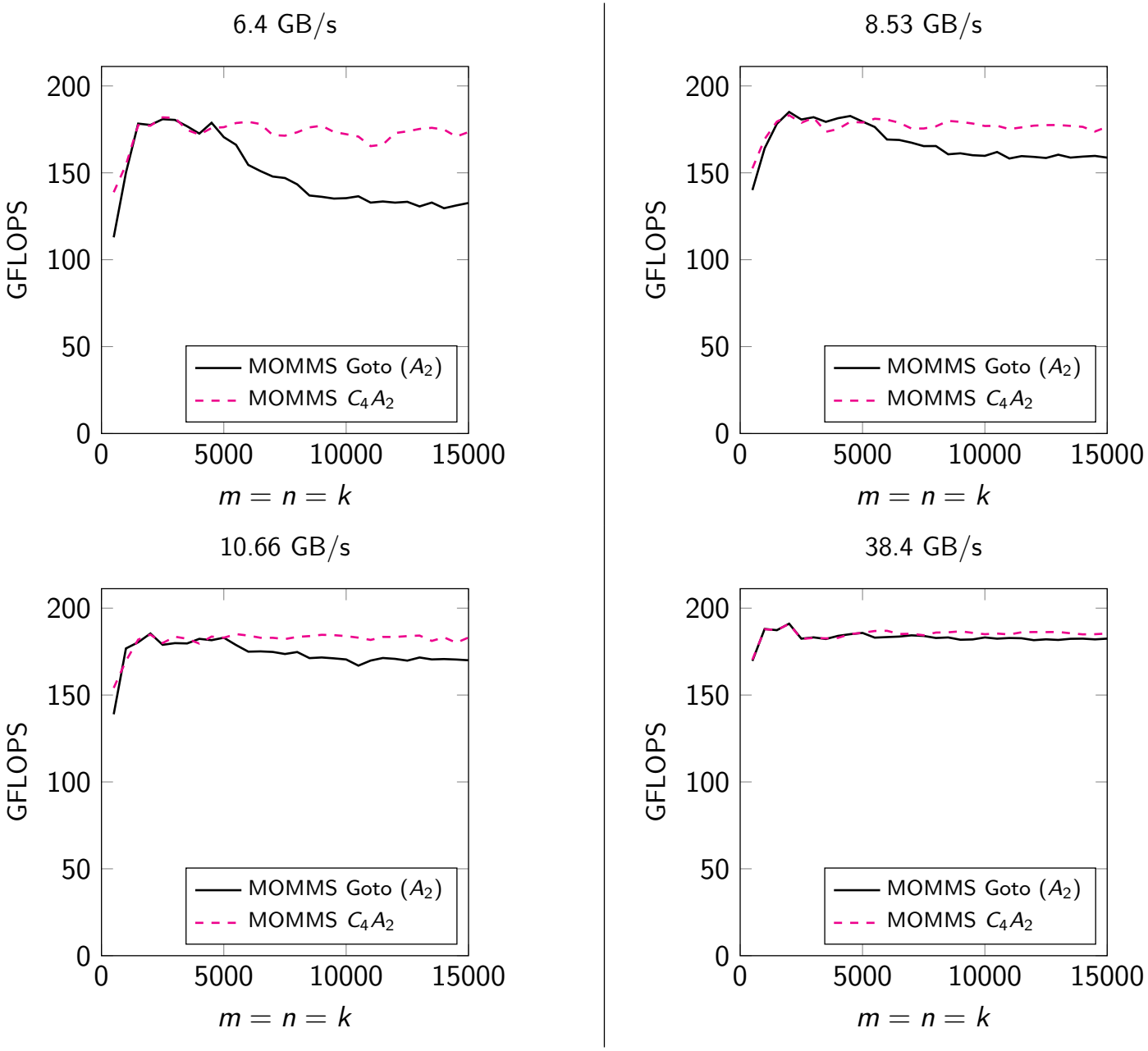
- Let's look at the I/O cost of C
  - Each element of C is involved in k flops
  - k_c flops accumulated into an element of C each time it is read and written from main memory
  - Each element of C is read and written to and from main memory $\frac{k}{k_c}$ times.
  - I/O cost of $\frac{2mnk}{k_c}$
- Can analyze I/O cost of A and B similarly
  - I/O cost of A is $\frac{mnk}{n_c}$
  - I/O cost of B can be amortized completely

An algorithm for an Intel i7-5775C

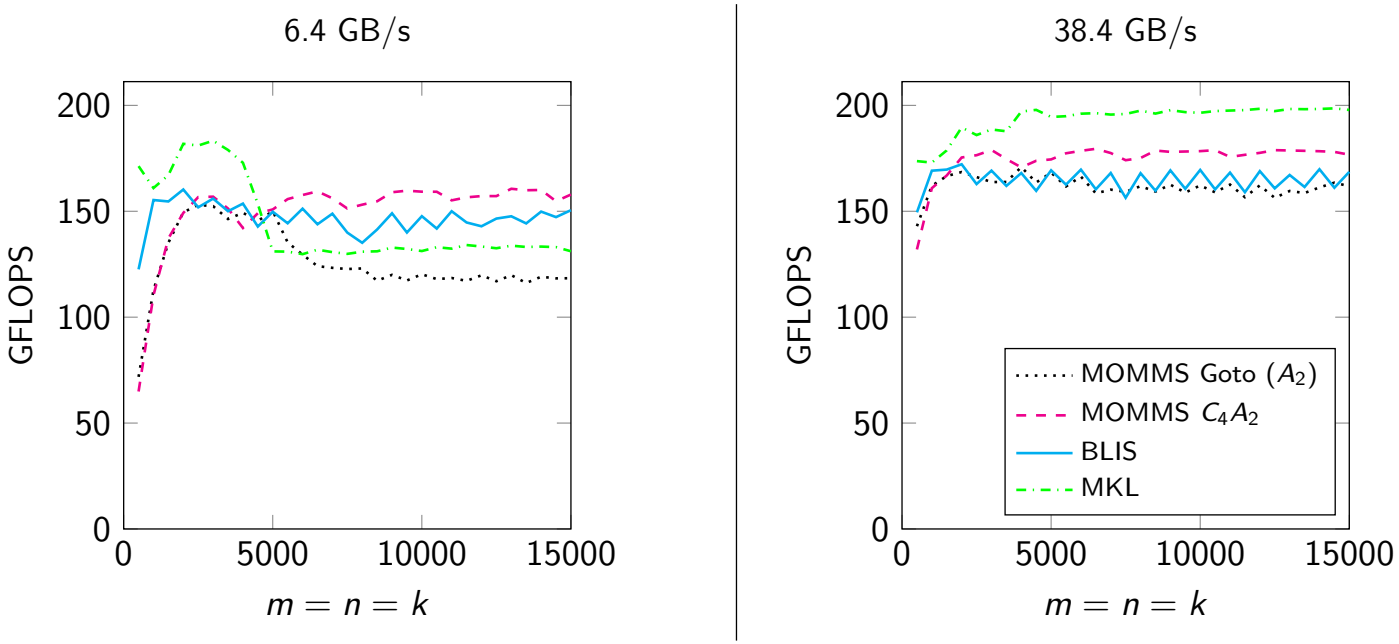
C_4A_2 Algorithm



Varying bandwidth for the i7-5775C



Comparing with other implementations for the i7-5775C



Upper bound on FMAs during a phase with I/O cost S

- Again, we will use the Loomis-Whitney Inequality
- In MMM, A, B, and C are inputs
- There are at most $2S$ inputs
  - $N_A + N_B + N_C \leq 2S$
  - $\sqrt{N_A N_B N_C} \leq \sqrt{xyz}$ for some $x, y, z \in \mathbb{R}$
  - $x + y + z = 2S$
- Maximize $\sqrt{xyz}$ under the constraint $x + y + z = 2S$
  - $x = y = z = \frac{2S}{3}$
- $F = \frac{2\sqrt{2}}{3\sqrt{3}} S\sqrt{S}$
- Then our lower bound is $S\left(\frac{3\sqrt{3}}{2\sqrt{2}}\frac{mnk}{S\sqrt{S}} - 1\right)$
- Or: $\frac{3\sqrt{3}}{2\sqrt{2}}\frac{mnk}{\sqrt{S}} - S = \frac{1.837mnk}{\sqrt{S}} - S$

Analysis of ATLAS

Whaley, Petitet, and Dongarra (2001)

```
for( j=0; j<N-1; j+=NB )
{
    for( i=0; i<M-1; i+=NB )
    {
        for( p=0; p<K-1; p+=NB )
        {
            ON_CHIP_MATMUL( A[i:i+NB][p:p+NB],
                B[p:p+NB][j:j+NB], C[i:i+NB][j:j+NB] );
        }
    }
}
```

Analysis of ATLAS

Whaley, Petitet, and Dongarra (2001)

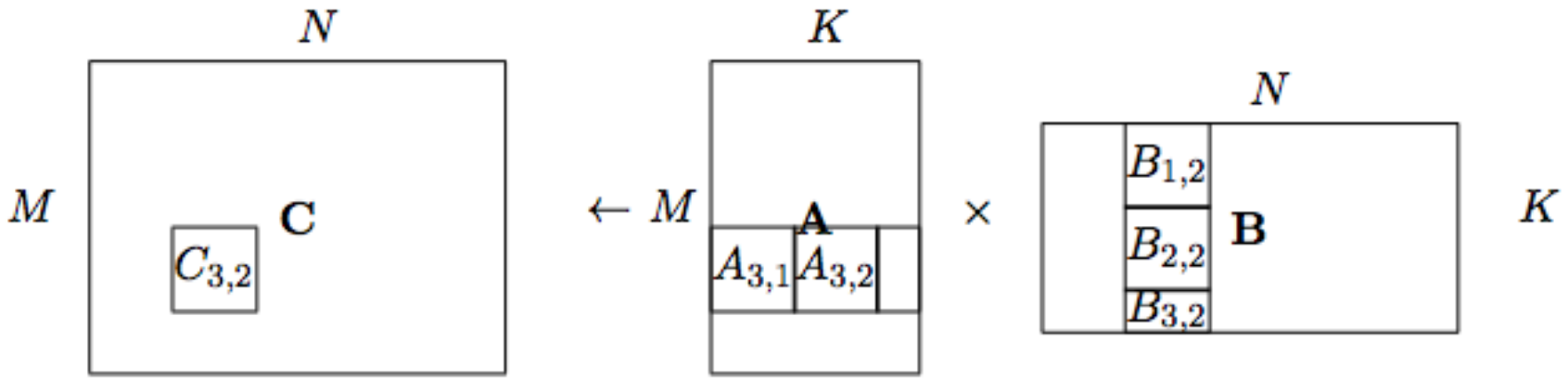


Figure 1: One step of matrix-matrix multiply

- Inner-kernel is an $n_b \times n_b \times n_b$ MMM
  - Fills the L1 cache with a square block of A or B
  - Streams the other two matrices
- The next inner-kernel invocation uses the same block of C, different A and B.
- Each element of A, B, and C reused in cache n_b times
- I/O cost for each of A, B, and C is $\frac{mnk}{\sqrt{S_1}}$
- Overall cost is roughly $\frac{3mnk}{\sqrt{S_1}}$