An Analytical Model for BLIS

Tze Meng Low [1] Francisco D. Igual [2]
Tyler M. Smith [3] Enrique Quitana-Ortiz [4]

[1]Carnegie Mellon University

[2]Universidad Complutense de Madrid, Spain

[3]The University of Texas at Austin

[4]Universidad Jaume I, Spain

2^{nd} BLIS Retreat
25-26 September 2015

Background

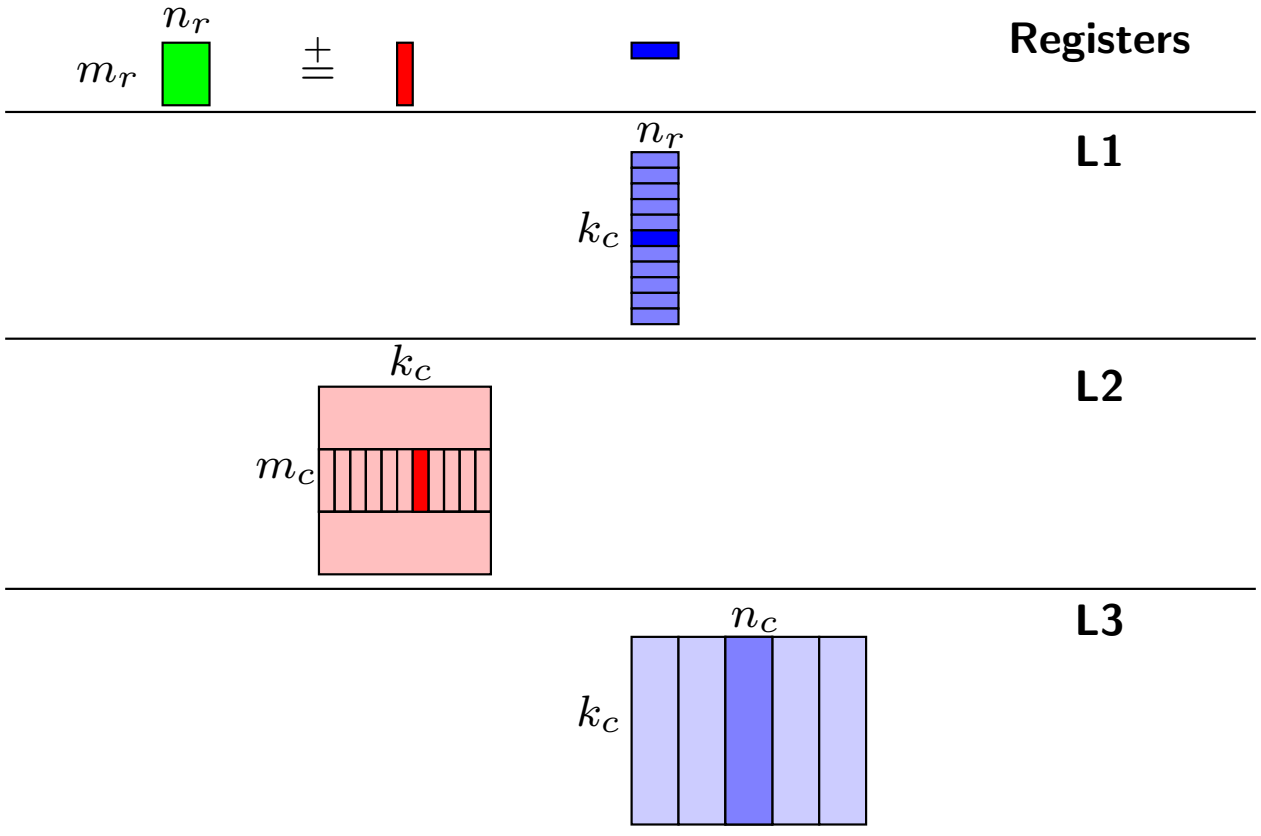
- BLAS-like Library Instantiation Software (BLIS)
  - Framework for rapidly instantiating the BLAS (or BLAS-like) functionalities using the GotoBLAS approach
  - Productivity multiplier for the developer
- With BLIS, an expert has to
  - Identifying parameter values (e.g. block sizes); and
  - Implementing an efficient micro-kernel in assembly (in essence, a series of outer-products)

Background

- "Is Search Really Necessary to Generate High-Performance BLAS?" [Yotov et al, 2005]
- Showed that empirical search in ATLAS can be replaced with simple analytical models
- Key differences
  - ATLAS
    - Scalar instructions
    - Single level, Fully Associative Cache
    - Compared against ATLAS generated code (no user kernels)
  - BLIS
    - SIMD instructions
    - Hierarchy of Set Associative Caches
    - Compared against hand-coded implementations

GotoBLAS at a glance

- 5 parameters ($m_r, n_r, k_c, m_c,$ and n_c)

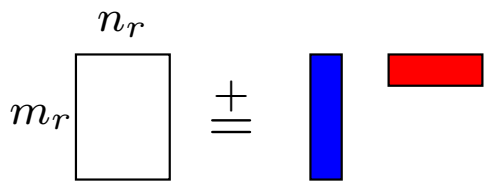


Model Architecture

- Vector registers
  - Each vector register holds N_{vec} elements.
- FMA instructions
  - Throughput of N_{fma} per clock cycle.
  - Instruction latency is given by L_{fma}.
- Caches
  - All caches are set-associative.
  - Cache replacement policy is LRU.
  - Cache lines are the same for all caches.

Parameters: m_r, n_r

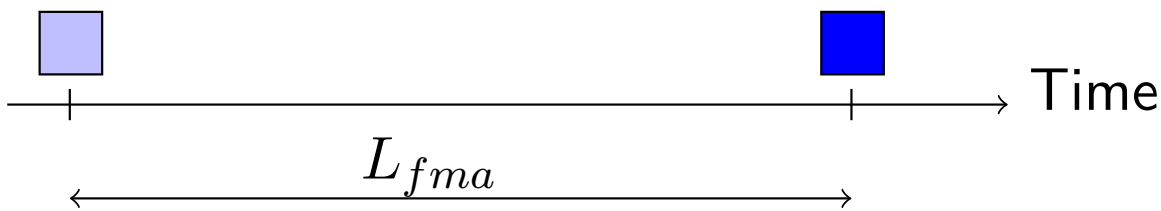
- Recall:
  - m_r and n_r determine the size of the micro-block of C
  - Each element is computed exactly once in each iteration of the micro-kernel



- Strategy
  - Pick the smallest micro-block of C ($m_r \times n_r$) such that no stalls arising from dependencies and instruction latency occur when computing one iteration of the micro-kernel.

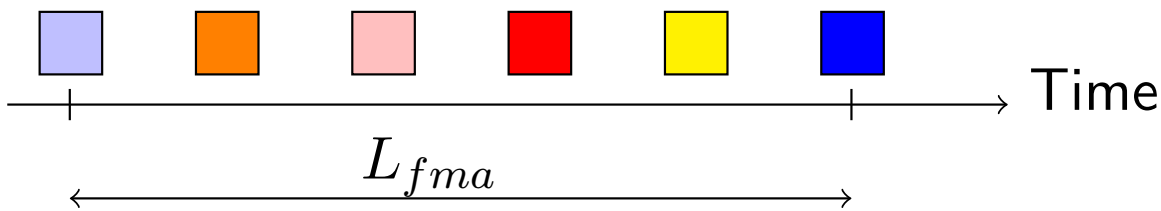
Parameters: m_r, n_r

- Recall:
  - Each FMA instruction has a latency of L_{fma}
  -
  -



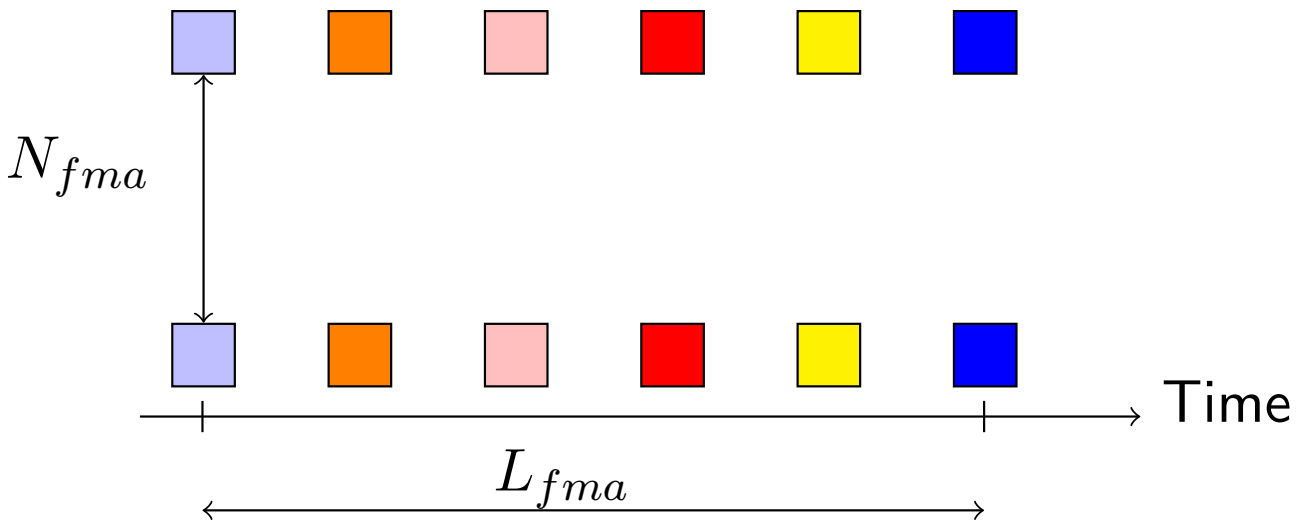
Parameters: m_r, n_r

- Recall:
  - Each FMA instruction has a latency of L_{fma}
  -
  -



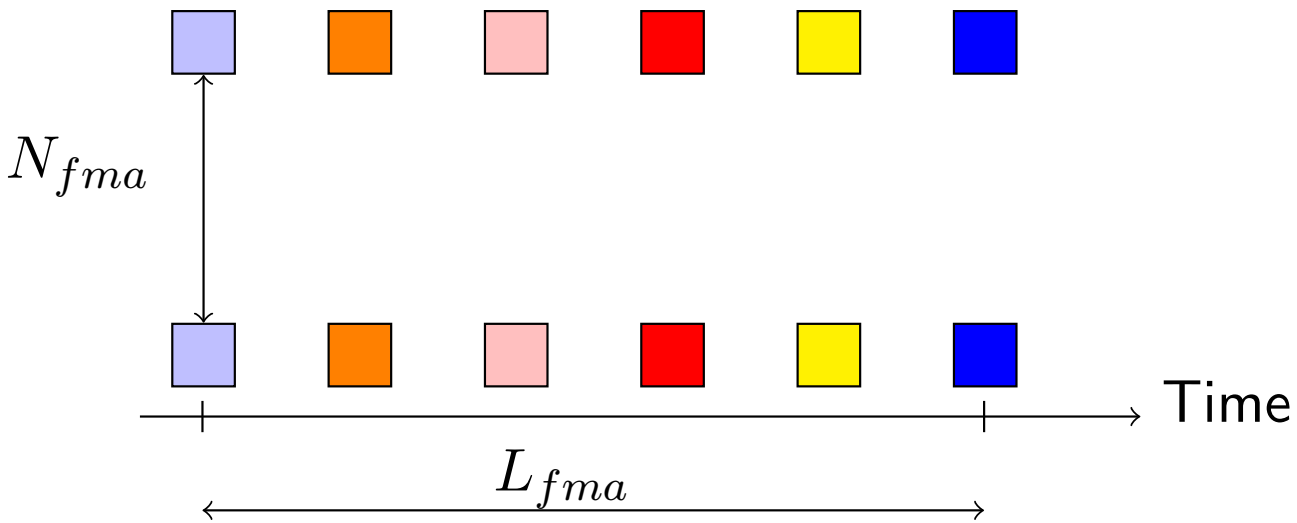
Parameters: m_r, n_r

- Recall:
  - Each FMA instruction has a latency of L_{fma}
  - N_{fma} FMA instructions can be issued per clock cycle
  -



Parameters: m_r, n_r

- Recall:
  - Each FMA instruction has a latency of L_{fma}
  - N_{fma} FMA instructions can be issued per clock cycle
  - Each FMA instruction computes N_{vec} elements



Parameters: m_r, n_r

- Minimum size of the micro-block of C

$$m_r n_r \geq N_{vec} L_{fma} N_{fma}$$

- Ideally,

$$m_r, n_r \approx \sqrt{N_{vec} L_{fma} N_{fma}}$$

- In practice,

$$m_r (\text{or } n_r) = \left\lceil \frac{\sqrt{N_{vec} L_{fma} N_{fma}}}{N_{vec}} \right\rceil N_{vec}$$

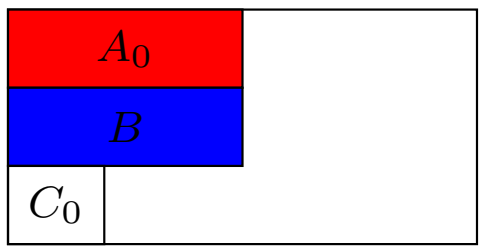
Parameters: k_c, m_c, n_c

- Recall that k_c, m_c, and n_c are dimensions of the matrices that are kept in different caches
  - L1 : Micro-panel of B - $k_c \times n_r$
  - L2 : Packed block of A - $m_c \times k_c$
  - L3 (if available) : Packed block of B - $k_c \times n_c$
- Pick largest k_c, m_c and n_c such that the matrices will still be kept in their caches

Parameters: k_c, m_c, n_c

- Consider the L1 cache:
  - Same micro-panel of B is used between different invocations of the micro-kernel
  - Micro-panels of A are used only once
  - For simplicity, micro-panels of A and B are the same size

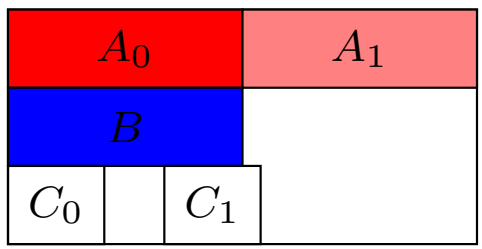
Option 1



Parameters: k_c, m_c, n_c

- Consider the L1 cache:
  - Same micro-panel of B is used between different invocations of the micro-kernel
  - Micro-panels of A are used only once
  - For simplicity, micro-panels of A and B are the same size

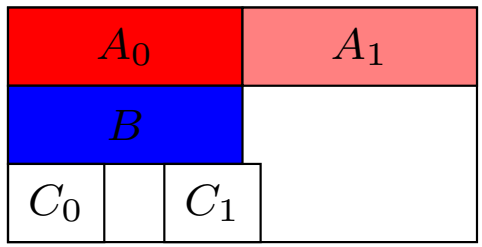
Option 1



Parameters: k_c, m_c, n_c

- Consider the L1 cache:
  - Same micro-panel of B is used between different invocations of the micro-kernel
  - Micro-panels of A are used only once
  - For simplicity, micro-panels of A and B are the same size

Option 1

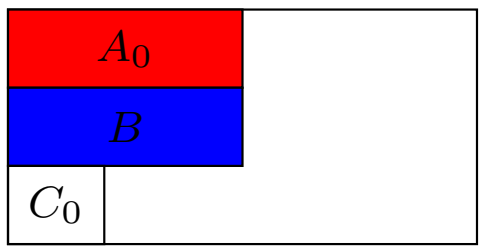


Not efficient use of cache!

Parameters: k_c, m_c, n_c

- Consider the L1 cache:
  - Same micro-panel of B is used between different invocations of the micro-kernel
  - Micro-panels of A are used only once
  - For simplicity, micro-panels of A and B are the same size

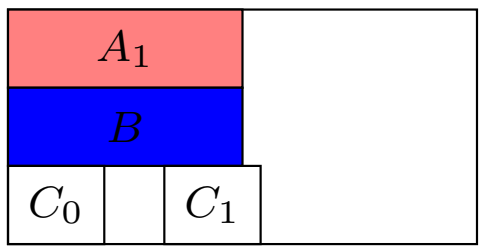
Option 2



Parameters: k_c, m_c, n_c

- Consider the L1 cache:
  - Same micro-panel of B is used between different invocations of the micro-kernel
  - Micro-panels of A are used only once
  - For simplicity, micro-panels of A and B are the same size

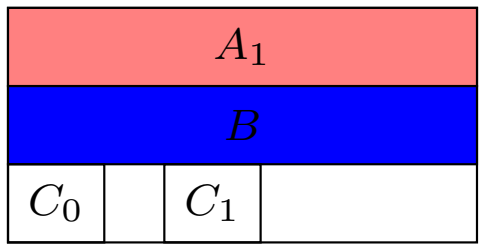
Option 2



Parameters: k_c, m_c, n_c

- Consider the L1 cache:
  - Same micro-panel of B is used between different invocations of the micro-kernel
  - Micro-panels of A are used only once
  - For simplicity, micro-panels of A and B are the same size

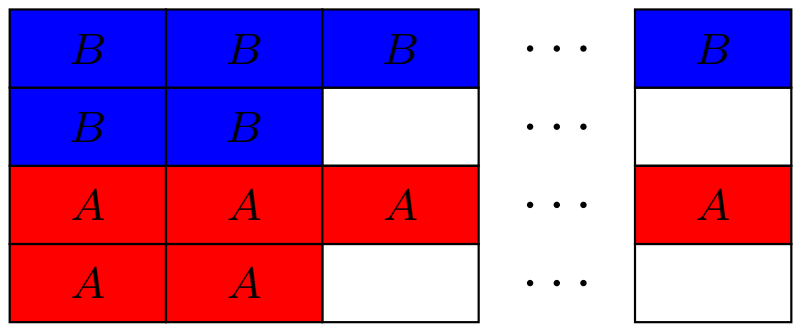
Option 2



Larger micro-panel of B can be kept in the L1 cache

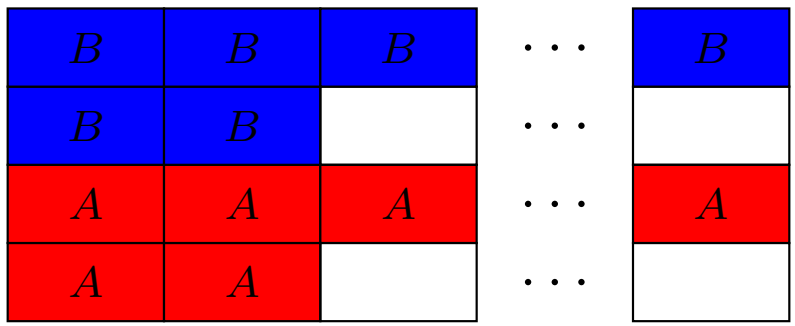
Parameters: k_c, m_c, n_c

- Observation 1: A and B are packed
  - Elements of A and B are in contiguous memory locations



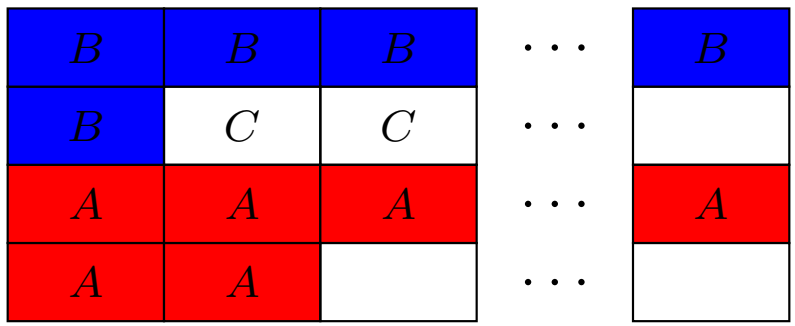
Parameters: k_c, m_c, n_c

- Observation 1: A and B are packed
  - Elements of A and B are in contiguous memory locations
- Observation 2: Caches are set associative
  - Cache lines are evicted when all W cache lines in a set is filled
  - At least one cache line is filled with elements from C.
  - Micro-panels of A and B can, at most, fill $W - 1$ cache lines in each set



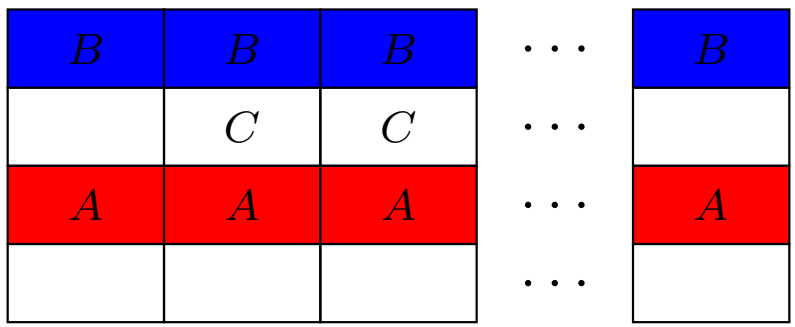
Parameters: k_c, m_c, n_c

- Observation 1: A and B are packed
  - Elements of A and B are in contiguous memory locations
- Observation 2: Caches are set associative
  - Cache lines are evicted when all W cache lines in a set is filled
  - At least one cache line is filled with elements from C.
  - Micro-panels of A and B can, at most, fill $W - 1$ cache lines in each set



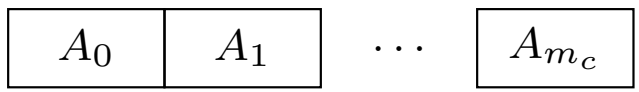
Parameters: k_c, m_c, n_c

- Observation 1: A and B are packed
  - Elements of A and B are in contiguous memory locations
- Observation 2: Caches are set associative
  - Cache lines are evicted when all W cache lines in a set is filled
  - At least one cache line is filled with elements from C.
  - Micro-panels of A and B can, at most, fill $W - 1$ cache lines in each set



Parameters: k_c, m_c, n_c

- Recall: Want new micro-panel of A to evict old micro-panels of A
  - Starting location of each micro-panel of A must be mapped to the same set
  - Size of a micro-panel of A must be a multiple (C_{A_r}) of the number of sets in the cache



  - C_{A_r} is the number of cache lines in each set allocated to a micro-panel of A.
  - k_c can then be computed as follow

$$k_c = \frac{C_{A_r} N_{\mathrm{L1}} C_{\mathrm{L1}}}{m_r S_{\mathrm{DATA}}}$$

Validation

- Compare parameter values from model against OpenBLAS and manually optimized BLIS implementations
- Model should yield similar (if not identical) parameter values as those in existing implmentations since all three apporaches use the GotoBLAS approach

Validation

- Size of micro-block of C, m_r and n_r

Architecture	OpenBLAS		BLIS		Model	
	m_r	n_r	m_r	n_r	m_r	n_r
Intel Dunnington	4	4	4	4	4	4
Intel SandyBridge	8	4	8	4	8	4
TI C6678	-	-	4	4	4	4
AMD Piledriver	8 (6)	2 (4)	4	6	4	6

Validation

- Values of k_c, and m_c.
- n_c not shown because either architecture had no L3 cache, or varying n_c resulted in minimal performance variation

Architecture	BLIS		Model	
	k_c	m_c	k_c	m_c
Intel Dunnington	256	384	256	384
Intel SandyBridge	256	96	256	96
TI C6678	256	128	256	128
AMD Piledriver	120	1088	128	1792

Conclusion

- An analytical model for determining the parameter values required by BLIS
- Parameter values that are similar if not identical to those in expert-tuned implementations
- Consistent result with Yotov et. al:
  Analytical modeling is sufficient for high performance BLIS

Future Work

- Relax Assumptions
  - Include bandwidth considerations
  - Different cache replacement policies
  - Complex arithmetics
- More complicated linear algebra algorithms (e.g. LAPACK)
  - Extend model to LAPACK-type algorithms
  - Can BLIS parameters be used to determine optimal block for LAPACK algorithms?
- Hardware Co-design
  - Analytical model for LAP [Pedram et. al. 2012] is similar to the analytical model presented here
  - Possible for model to be used in cache design/cache replacement policies?