PA1 Report: BlisLab GEMM Optimization

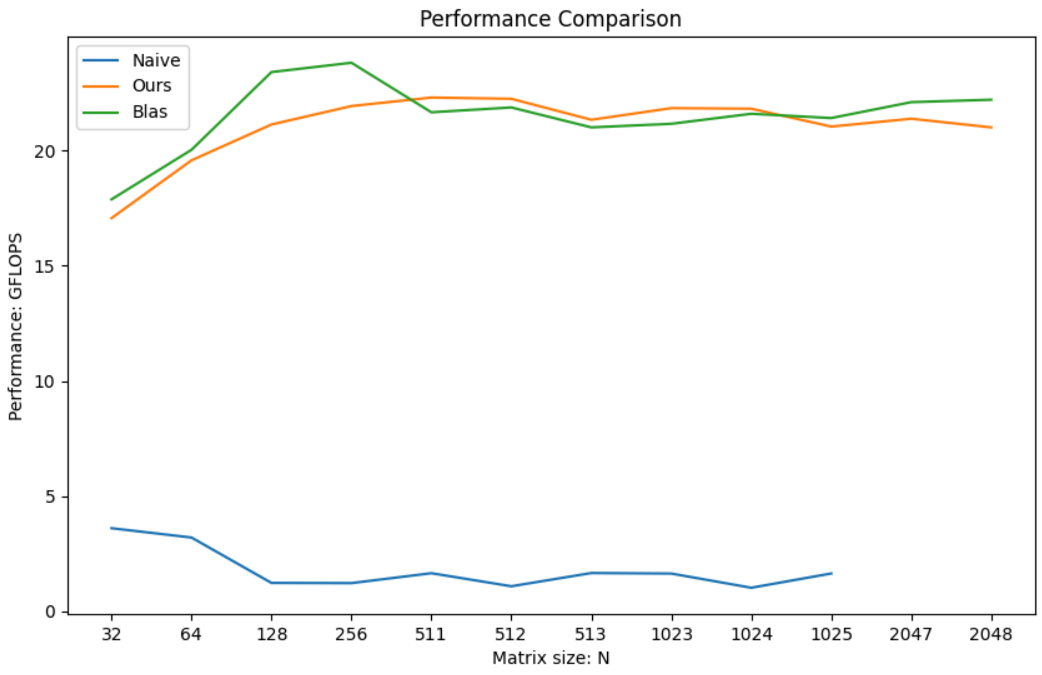
Jingyu Wu, A16157847
Haoxuan Wang, A16143053

Q1. Results

Q1.a. Optimized code performance:

N	Peak GF
32	17.07
64	19.58
128	21.135
256	21.935
511	22.305
512	22.25
513	21.34
1023	21.845
1024	21.82
1025	21.045
2047	21.385
2048	21.01

Q1.b. Performance Comparison (OpenBLAS, our code, naive code)



Note that the X-axis is equally spaced regardless of the N value.

Q2. Analysis

Q2.a. How does the program work

Key files:

benchmark.cpp: driver code to run matrix-matrix multiplications with different options and report results.

genDATA.sh: driver script (of the driver code) that takes average results over 20 runs on each matrix size.

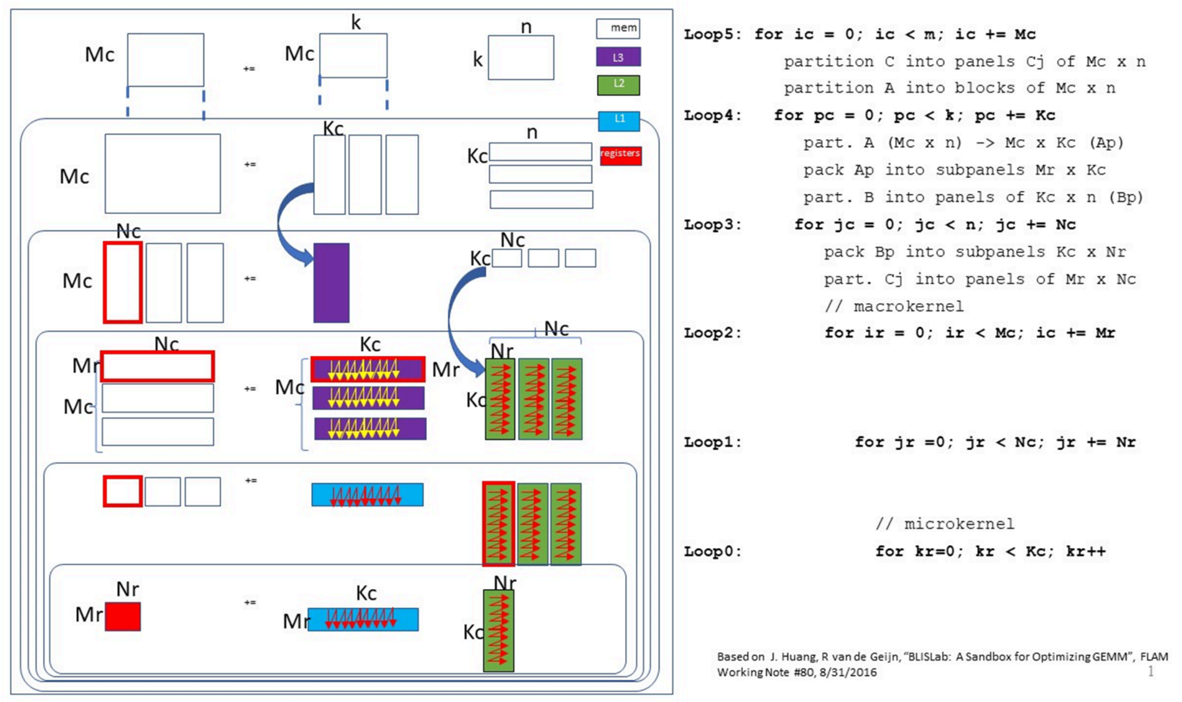
my_dgemm.c: main GEMM (general matrix multiplication) code, includes:

- Function bl_dgemm() for Loop 5 to 3, see below figure.
- Macro Kernel function bl_macro_kernel() for Loop 2 and 1.
- Packing functions packA_mcxkc_d() and packB_kcxnc_d()

bl_dgemm_ukr.c: highly optimized microkernel functions.

blislab/bl_config.h: definitions of hyperparameters, which include Mc, Kc, Nc, Mr, Nr in the below figure, and which microkernel to use.

Program Workflow:



```
Loop5:  for ic = 0; ic < m; ic += Mc
            partition C into panels Cj of Mc x n
            partition A into blocks of Mc x n
Loop4:    for pc = 0; pc < k; pc += Kc
              part. A (Mc x n) -> Mc x Kc (Ap)
              pack Ap into subpanels Mr x Kc
              part. B into panels of Kc x n (Bp)
Loop3:      for jc = 0; jc < n; jc += Nc
                pack Bp into subpanels Kc x Nr
                part. Cj into panels of Mr x Nc
                // macrokernel
Loop2:          for ir = 0; ir < Mc; ic += Mr

Loop1:              for jr =0; jr < Nc; jr += Nr

                      // microkernel
Loop0:                for kr=0; kr < Kc; kr++
```

Based on J. Huang, R van de Geijn, "BLISLab: A Sandbox for Optimizing GEMM", FLAM Working Note #80, 8/31/2016 1

The graph above shows how our program divides up the work of multiplying two big matrices into smaller blocks to enhance performance. Here is a further breakdown with regards to the code that better explains the blocking process:

- 5th Loop (ic loop): Iterates over matrix A by blocks of rows (DGEMM_MC rows at a time). DGEMM_MC is a predefined size that specifies how many rows of A are considered in one block.
- 4th Loop (pc loop): Iterates over the shared dimension k (in A[m*k] * B[k*n]) taking blocks of size DGEMM_KC. This dimension dictates the depth of the multiplication, akin to the inner dimension in matrix multiplication.
    - Within the 4th loop, blocks of matrices A and B are prepared for efficient multiplication:
        - Packing A: Blocks of matrix A are packed using packA_mcxkc_d, ensuring they are stored in column-major order within the packed block. This

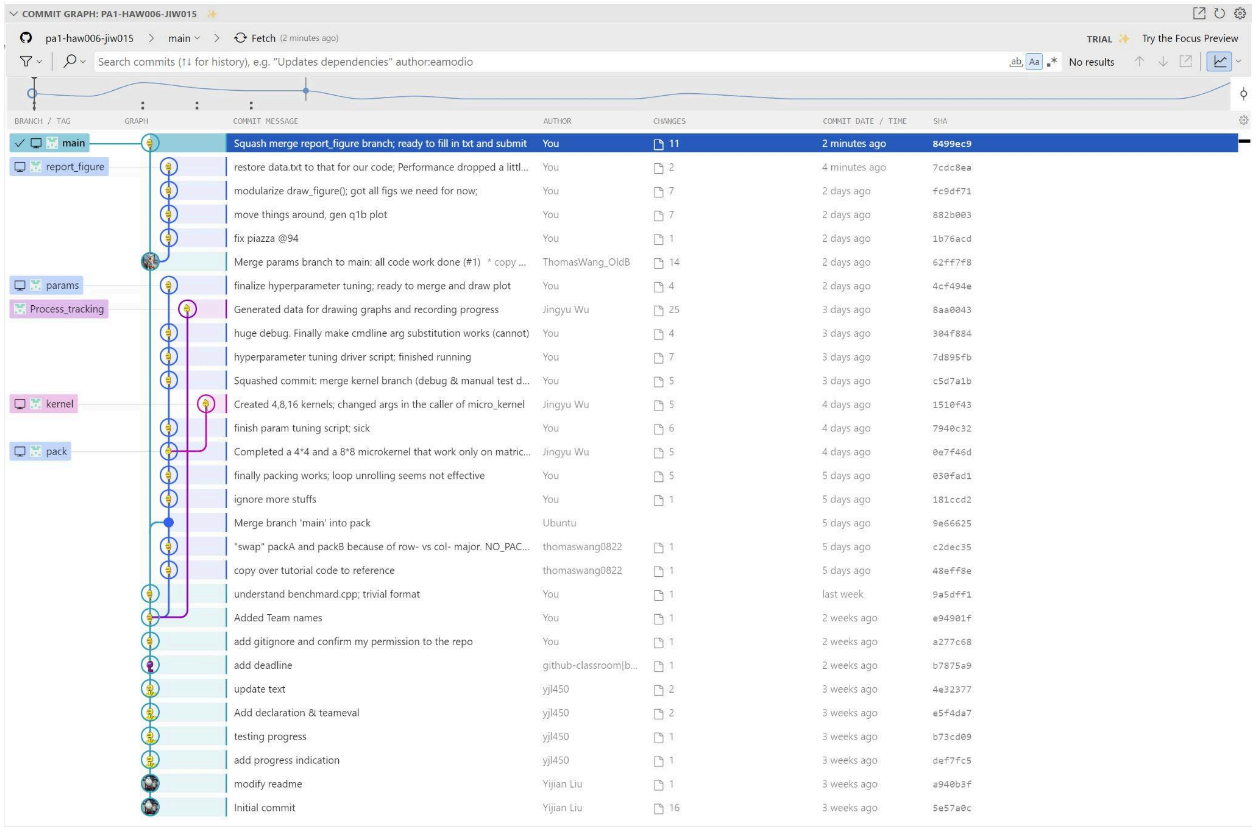
function also handles padding with zeros if the block does not fully match DGEMM_MR.

    - Packing B: Similarly, blocks of matrix B are packed using packB_kcxnc_d, but in row-major order. Padding with zeros is similarly managed for unmatched DGEMM_NR.

- 3rd Loop (jc loop): Iterates over matrix B by blocks of columns (DGEMM_NC columns at a time).
- Nested Inner Loops: These loops set the stage for the actual multiplication:
    - They determine the exact size of each block considering edge cases where the dimensions of the matrices do not align perfectly with the block sizes (DGEMM_MC, DGEMM_NC, DGEMM_KC).
    - The micro-kernel, referenced as bl_micro_kernel, is invoked for each block pair from matrices A and B. The results are accumulated directly into the appropriate block of matrix C.

Q2.b. Development process:

We ran the benchmark program after each step, but we didn't record the data or generate the plot until we finished everything. Nevertheless, we maintained good code structure and commit histories such that we could easily restore the code to any previous state to draw plots and compare performance. This GitLens commit graph helps with tracking progress:



COMMIT GRAPH: PA1-HAW006-JIW015
pa1-haw006-jiw015
main
Fetch (2 minutes ago)
TRIAL
Try the Focus Preview
Search commits (↑↓ for history), e.g. "Updates dependencies" author:eamodio
No results

BRANCH / TAG	COMMIT MESSAGE	AUTHOR	CHANGES	COMMIT DATE / TIME	SHA
✓ main	Squash merge report_figure branch; ready to fill in txt and submit	You	11	2 minutes ago	8499ec9
report_figure	restore data.txt to that for our code; Performance dropped a littl...	You	2	4 minutes ago	7cdc8ea
	modularize draw_figure(); got all figs we need for now;	You	7	2 days ago	fc9df71
	move things around, gen q1b plot	You	7	2 days ago	882b003
	fix piazza @94	You	1	2 days ago	1b76acd
	Merge params branch to main: all code work done (#1) * copy ...	ThomasWang_OldB	14	2 days ago	62ff7f8
params	finalize hyperparameter tuning; ready to merge and draw plot	You	4	2 days ago	4cf494e
Process_tracking	Generated data for drawing graphs and recording progress	Jingyu Wu	25	3 days ago	8aa0043
	huge debug. Finally make cmdline arg substitution works (cannot)	You	4	3 days ago	304f884
	hyperparameter tuning driver script; finished running	You	7	3 days ago	7d895fb
	Squashed commit: merge kernel branch (debug & manual test d...	You	5	3 days ago	c5d7a1b
kernel	Created 4,8,16 kernels; changed args in the caller of micro_kernel	Jingyu Wu	5	4 days ago	1510f43
	finish param tuning script; sick	You	6	4 days ago	7940c32
pack	Completed a 4*4 and a 8*8 microkernel that work only on matric...	Jingyu Wu	5	4 days ago	0e7f46d
	finally packing works; loop unrolling seems not effective	You	5	5 days ago	030fad1
	ignore more stuffs	You	1	5 days ago	181ccd2
	Merge branch 'main' into pack	Ubuntu		5 days ago	9e66625
	"swap" packA and packB because of row- vs col- major. NO_PAC...	thomaswang0822	1	5 days ago	c2dec35
	copy over tutorial code to reference	thomaswang0822	1	5 days ago	48eff8e
	understand benchmard.cpp; trivial format	You	1	last week	9a5dff1
	Added Team names	You	1	2 weeks ago	e94901f
	add gitignore and confirm my permission to the repo	You	1	2 weeks ago	a277c68
	add deadline	github-classroom[b...	1	2 weeks ago	b7875a9
	update text	yjl450	2	3 weeks ago	4e32377
	Add declaration & teameval	yjl450	2	3 weeks ago	e5f4da7
	testing progress	yjl450	1	3 weeks ago	b73cd09
	add progress indication	yjl450	1	3 weeks ago	def7fc5
	modify readme	Yijian Liu	1	3 weeks ago	a940b3f
	Initial commit	Yijian Liu	16	3 weeks ago	5e57a0c

1. Starter Code:

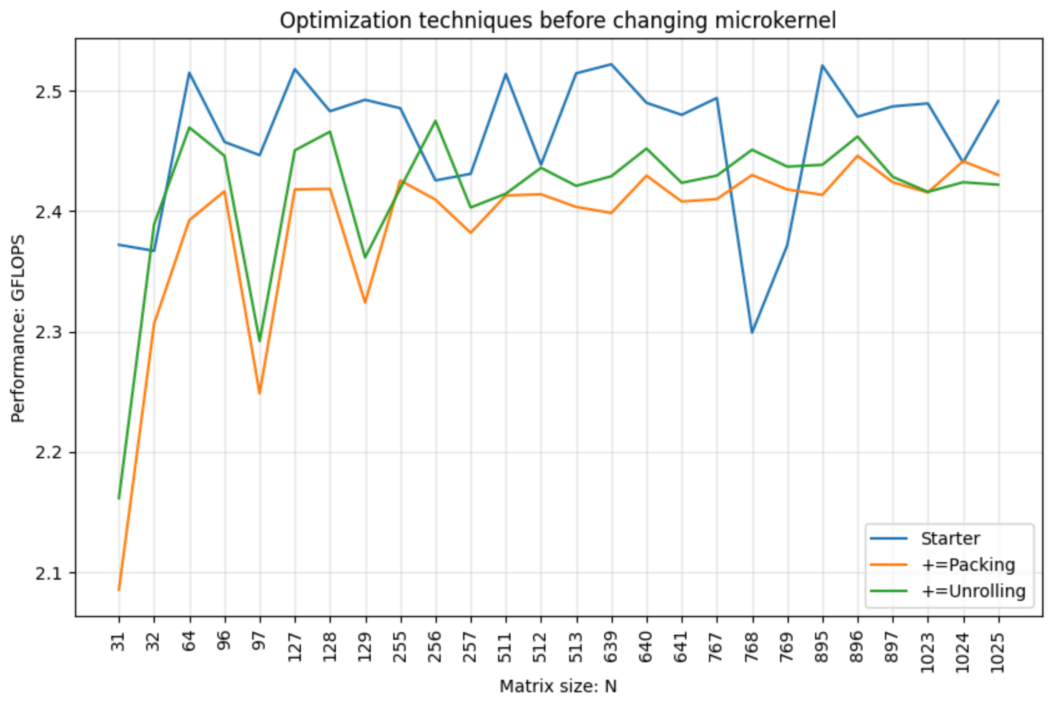
   We first tried running the starter code, which used the most basic ijk micro-kernel.

2. Packing:

   The first job is the packing routines. After that, the performance reached just above 2.0 GFLOPS, which is below our expectations. This suggests that packing overhead counters most of the acceleration it brings.

3. Loop Unrolling:

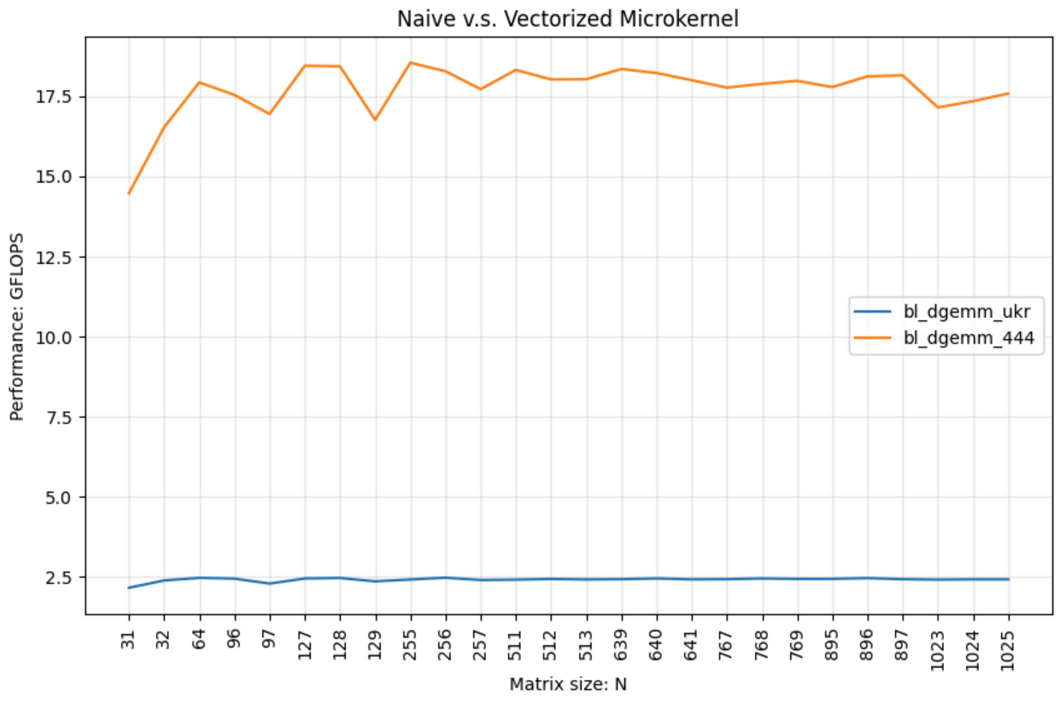
   We then added the loop unrolling in the packing code with some easy changes. There was an improvement, but not significant at all.



In the plot legend, “+=” means the optimization technique is built upon previous ones in the order above.

4. First microkernel

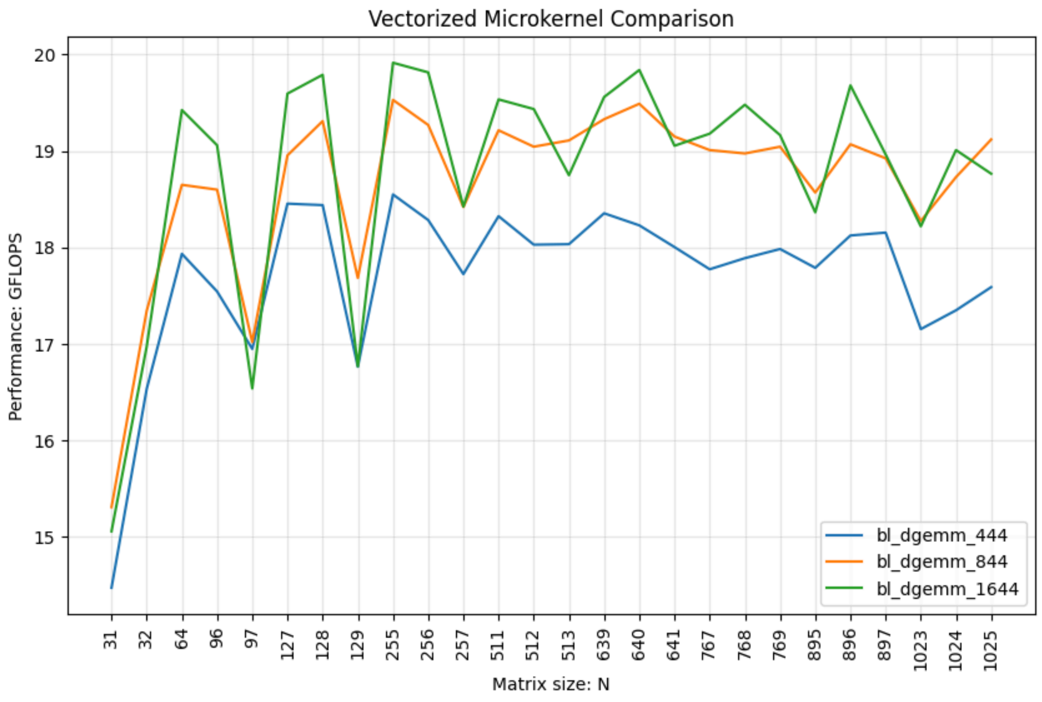
   We started to build microkernels. The first one operates 4 numbers at a time in a vector. The new kernel immediately boosted the performance from under 3 GFLOPS to around 18 GFLOPS.



In this graph, the data of the blue line (‘bl_dgemm_ukr’) are the same as those of ‘+=Unrolling’ in the previous plot.

5. Write more microkernels and do manual testing

We then implemented more microkernels with the same style but operated on more numbers (8 and 16) at a time. The improvement isn’t as significant, but still, they push our performance above 20 GFLOPS. Here is their comparison:



6. Automated parameter tuning

To begin with, we figured out what values of the hyperparameters (namely, Nr, Mr, Kc, Nc, Mc) should be taken by examining cache sizes with `$ lscpu | grep -i cache`. Some brief derivation is noted down in “results/cache_hyperparm.txt”.

Writing the Python script that automates the process of “compile with different macros (hyperparameters), run, record result” isn’t easy. In particular, the biggest difficulty comes from passing key-value pairs to substitute the preprocessor macros defined in the code. Hours were spent before we realized that to use ‘-D’ to pass in key-value pairs, those macros in the code must be commented out because they take high priority.

After all, we decided that (Nr, Mr, Kc, Nc, Mc) = (4, 16, 256, 128, 1024) gives the best result in all combinations. And we use them to generate data and draw the plot in Q1.b.

Q2.c. Irregularities in the data

N	Performance (GFlops)
32	17.07
64	19.58

128	21.135
256	21.935
511	22.305
512	22.25
513	21.34
1023	21.845
1024	21.82
1025	21.045

Based on the table above and the graph in section Q1.b, we found out that the performance of our matrix multiplication subroutine reaches its peak at around n=512, where it no longer improves when n gets larger.

We also noticed that the performance metrics are higher at values like n=511 or n=512 compared with n=513. We conjecture that it may be due to the additional overhead created by the padding in the packing functions.

Q2.d. Supporting data

Cache Sizes and hyperparameter choices:

Using the command "lscpu | grep -i cache", we found out the sizes for the three levels of caches in the machine:

L1d cache:	64 KiB (1 instance)
L1i cache:	64 KiB (1 instance)
L2 cache:	1 MiB (1 instance)
L3 cache:	32 MiB (1 instance)

Hence, we can decide the upper bounds of our hyperparameters (Nr, Mr, Kc, Nc, Mc) based on the following calculations:

Nr: has to be 4 for a row of C to fit into one vector register.

Mr: Based on our implementation of micro-kernels, Mr = 4,8, or 16

Kc: Kc*Mr block of A has to fit into the L1 cache (64KB), so Kc * Mr < 64*1024/8 = 8192

Nc: Nc*Kc block of B has to fit into the L2 cache (1 MB), so Nc*Kc < 1024^2/8 = 131072

Mc: Mc*Kc block of A has to fit into the L3 cache (32 MB), so Nc*Kc < 32*1024^2/8 = 4194304

Difference from the BLISlab tutorial

The first modification involves changing to row-major matrices for the input matrices A and B and for the output matrix C, differing from the BLISLab tutorial which uses column-major order. This adjustment is necessary as our machine uses row-major order, and we also need to adapt the skeleton code to accommodate this.

The second modification involves managing specific scenarios by padding the matrices with 0s, particularly in cases where N is not evenly divisible by the various block sizes. However, paddings introduce additional overhead, which affects the performance when N is small.

Q2.e. Future work

Our PA, or BLAS (Basic Linear Algebra Subprograms) in general, is concerned with optimizations in low-level CPU routines. After this, we could look at other sources of acceleration, such as multi-core, distributed (multi-device) computations. Also, in this AI era, multiplications of huge matrices arise from the training of gigantic neural networks. Therefore, storage is also an important issue.

Our performance could likely be improved if we pick different hyperparameters and microkernels for different matrix sizes. The code shouldn’t be difficult to write, but determining the condition robustly needs a lot more benchmarking.

Q3. References

Class resources:
PA 1 Instructions, Lecture slides, Discussion slides

Papers:
[1]Jianyu Huang and Robert A Van de Geijn. 2016. BLISlab: A sandbox for optimizing GEMM. arXiv preprint arXiv:1609.00076 (2016).
[2]Kazushige Goto and Robert A van de Geijn. 2008. Anatomy of high-performance matrix multiplication. ACM Transactions on Mathematical Software (TOMS) 34, 3 (2008), 1–25.
[3]Field G Van Zee and Robert A Van De Geijn. 2015. BLIS: A framework for rapidly instantiating BLAS functionality. ACM Transactions on Mathematical Software (TOMS) 41, 3 (2015), 1–33.

Links (to look for some easy commands on StackOverflow, and some other tools):
https://stackoverflow.com/a/30211477/14697376
https://stackoverflow.com/a/9890599/14697376
https://stackoverflow.com/a/37708190/14697376
BibTex Converter: https://asouqi.github.io/bibtex-converter/
Matplotlib API: https://matplotlib.org/stable/index.html

Q4. Extra Credit

Unfortunately we didn't have time to complete any extra credit option.