Project 2 Report

Hisen Zhang <zhangz29>
Michael Chen<chenj45>

Test Environment

processor : 1
vendor_id : GenuineIntel
cpu family : 6
model : 158
model name : Intel(R) Core(TM) i5-9300H CPU @ 2.40GHz
stepping : 10
cpu MHz : 2400.000
cache size : 8192 KB
line size :64 bytes
physical id : 0
siblings : 4
core id : 1
cpu cores : 4
apicid : 1
initial apicid : 1
fpu : yes
fpu_exception : yes
cpuid level : 22
wp : yes

Extensions & Technologies

MMX instructions
SSE / Streaming SIMD Extensions
SSE2 / Streaming SIMD Extensions 2
SSE3 / Streaming SIMD Extensions 3
SSSE3 / Supplemental Streaming SIMD Extensions 3
SSE4 / SSE4.1 + SSE4.2 / Streaming SIMD Extensions 4
AES / Advanced Encryption Standard instructions
AVX / Advanced Vector Extensions
AVX2 / Advanced Vector Extensions 2.0
BMI / BMI1 + BMI2 / Bit Manipulation instructions
F16C / 16-bit Floating-Point conversion instructions
FMA3 / 3-operand Fused Multiply-Add instructions
EM64T / Extended Memory 64 technology / Intel 64
HT / Hyper-Threading technology
VT-x / Virtualization technology
VT-d / Virtualization for directed I/O
TBT 2.0 / Turbo Boost technology 2.0

Procedure

We compared the time for Naive, Tiled, SIMD and Mixed for both floating-point and fixed-point matrix multiplications, using the cache line size of 64 shown in Table 1, Table 2, Figure 1, and Figure 2. The mixed method outperforms the other three methods in every condition. The value of percentage boost = (old-new) / new * 100%

Table 1a. Floating-point Performance at Cachline Size = 64

N#	Naive	Mixed	SIMD	Tiled
1000	6.043505	1.438622	1.429837	5.278191
2500	201.395331	22.313691	25.45587	90.708012
5000	1617.061194	179.748622	274.376188	690.847525
10000	13886.40311	1683.934482	3421.119802	5381.189239

Table 1b. Floating-point Performance by Boost at Cachline Size = 64

N#	Naive+%	Mixed+%	SIMD+%	Tiled+%
1000	Ref	320.09%	322.67%	14.50%
2500	Ref	802.56%	691.15%	122.03%
5000	Ref	799.62%	489.36%	134.07%
10000	Ref	724.64%	305.90%	158.05%

Table 2a. FIxed-point Performance at Cachline Size = 64

N#	Naive	Mixed	SIMD	Tiled
1000	4.684445	0.338868	0.32083	3.934597
2000	77.529334	3.423011	2.971956	33.485324
5000	1439.02155	43.611462	47.356169	546.781761
10000	9959.939435	304.035409	418.992672	3776.290993

Table 2b. FIxed-point Performance by Boostat Cachline Size = 64

N#	Naive+%	Mixed+%	SIMD+%	Tiled+%
1000	Ref	1282.38%	1360.10%	19.06%
2000	Ref	2164.95%	2508.70%	131.53%
5000	Ref	3199.64%	2938.72%	163.18%
10000	Ref	3175.91%	2277.12%	163.75%

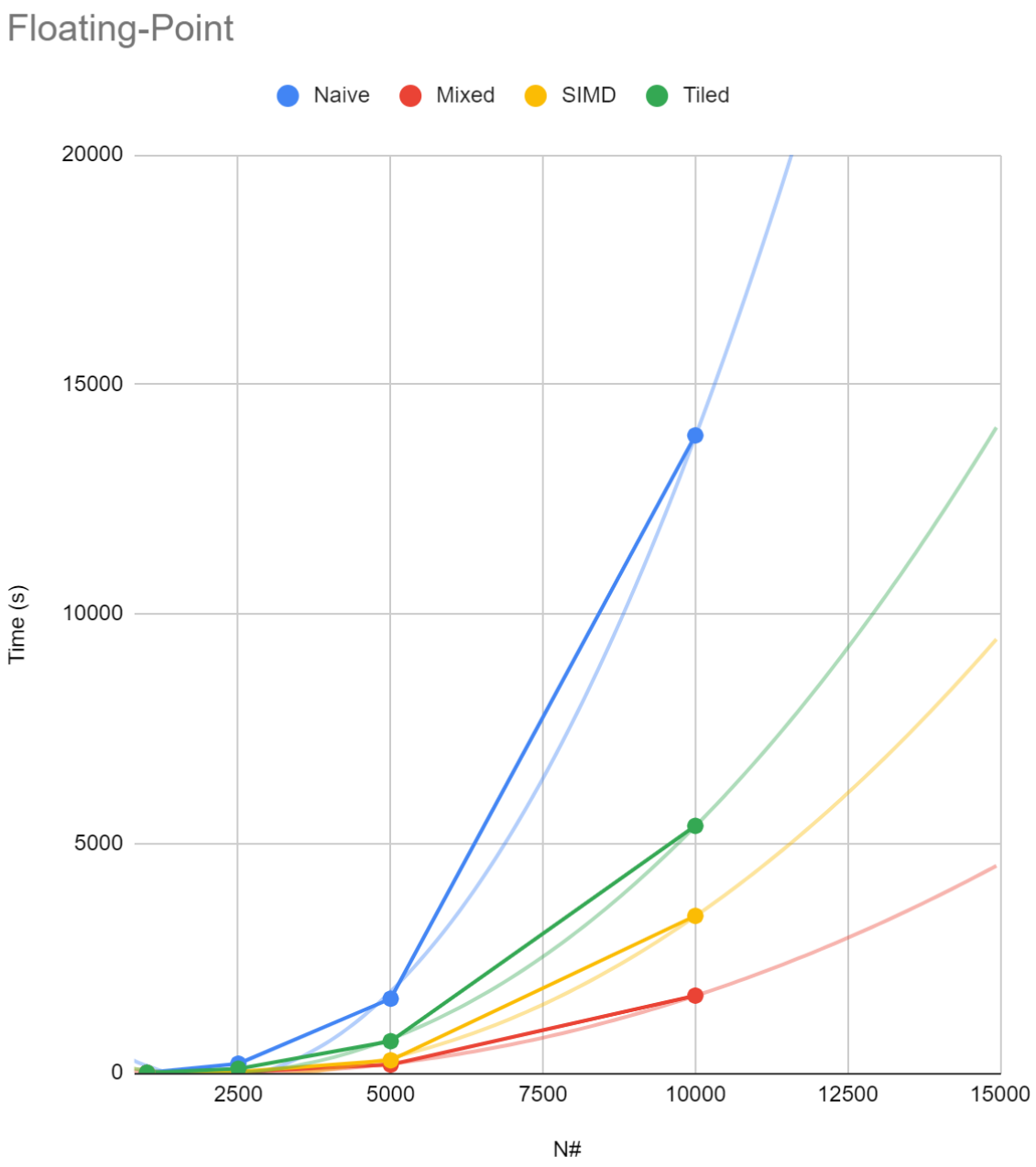

Figure 1. Floating-point Performance and Projection by Methods

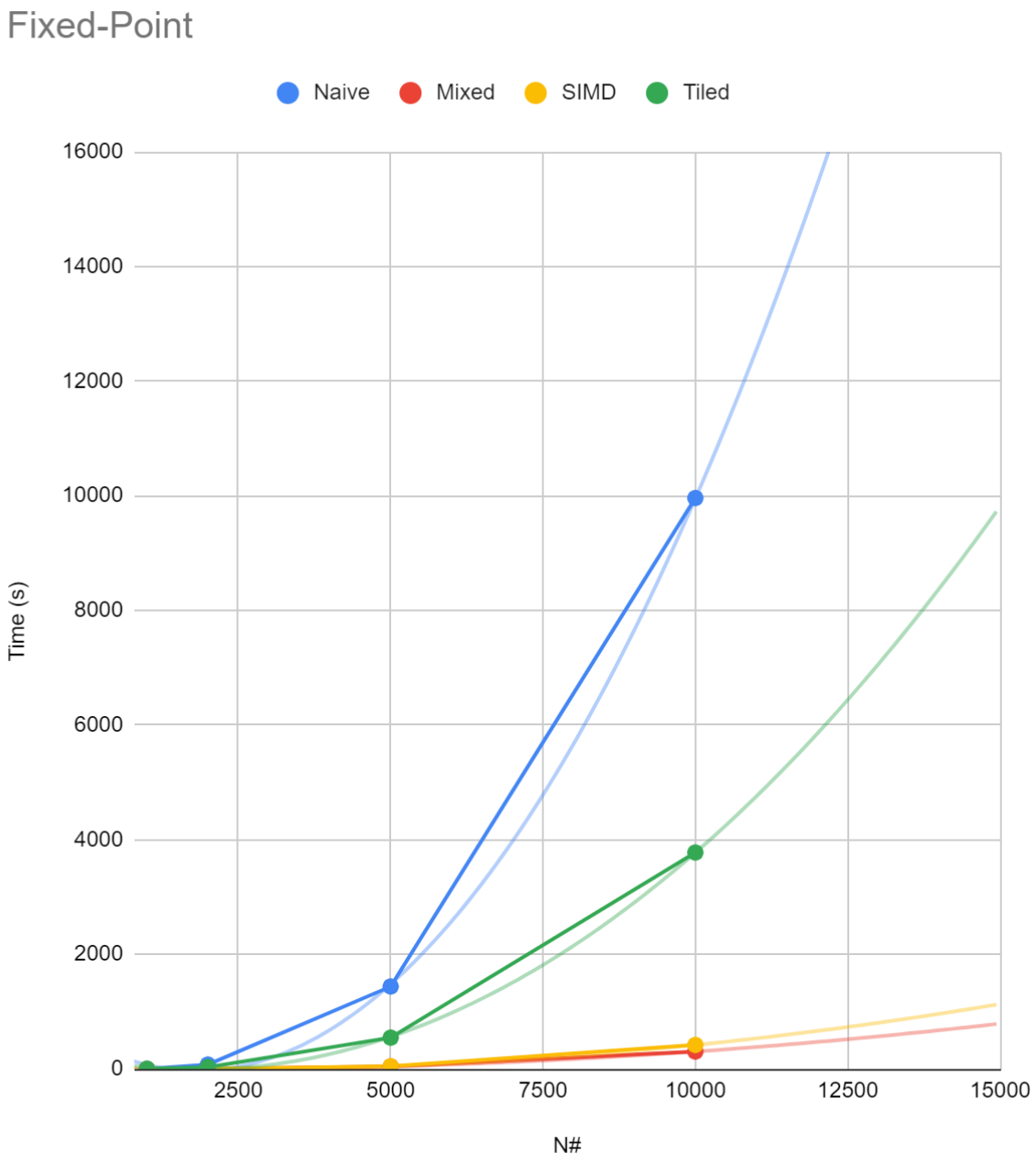

Figure 2. Fixed-point Performance and Projection by Methods

We also tested the performance boost with varying block and matrix sizes, as shown in Table 3 and Figure 3. We compared the naive and tiled matrix multiplication methods, using the blocksize of 32, 64, 128, 256, and 512, and calculated the time cost for both methods when multiplying 128x128, 256x256, 512x512, 1024x1024, 2048x2048 matrix sizes.

Table 3. Tiling And Naive Method Performance with Varying Block Size and Matrix Size

#(NxN)	BlockSize	Tiled Time	Naive Time	Boost
128	32	0.00779	0.009709	24.63%

128	64	0.008727	0.00986	12.98%
128	128	0.009212	0.010716	16.33%
128	256	0.009365	0.010124	8.10%
128	512	0.008582	0.010543	22.85%
256	32	0.07741	0.096613	24.81%
256	64	0.108528	0.096359	-11.21%
256	128	0.073801	0.112274	52.13%
256	256	0.135035	0.128911	-4.54%
256	512	0.106326	0.104119	-2.08%
512	32	0.673	0.993	47.55%
512	64	0.705	0.96	36.17%
512	128	0.814	0.882	8.35%
512	256	0.96	0.894	-6.87%
512	512	1.017	0.877	-13.77%
1024	32	5.983	10.544	76.23%
1024	64	6.271	10.103	61.11%
1024	128	7.095	9.975	40.59%
1024	256	8.299	10.555	27.18%
1024	512	8.737	10.513	20.33%
2048	32	49.378	236.41	378.78%
2048	64	53.476	233.44	336.53%
2048	128	61.474	238.2	287.48%
2048	256	65.429	236.17	260.96%
2048	512	66.092	236.9	258.44%

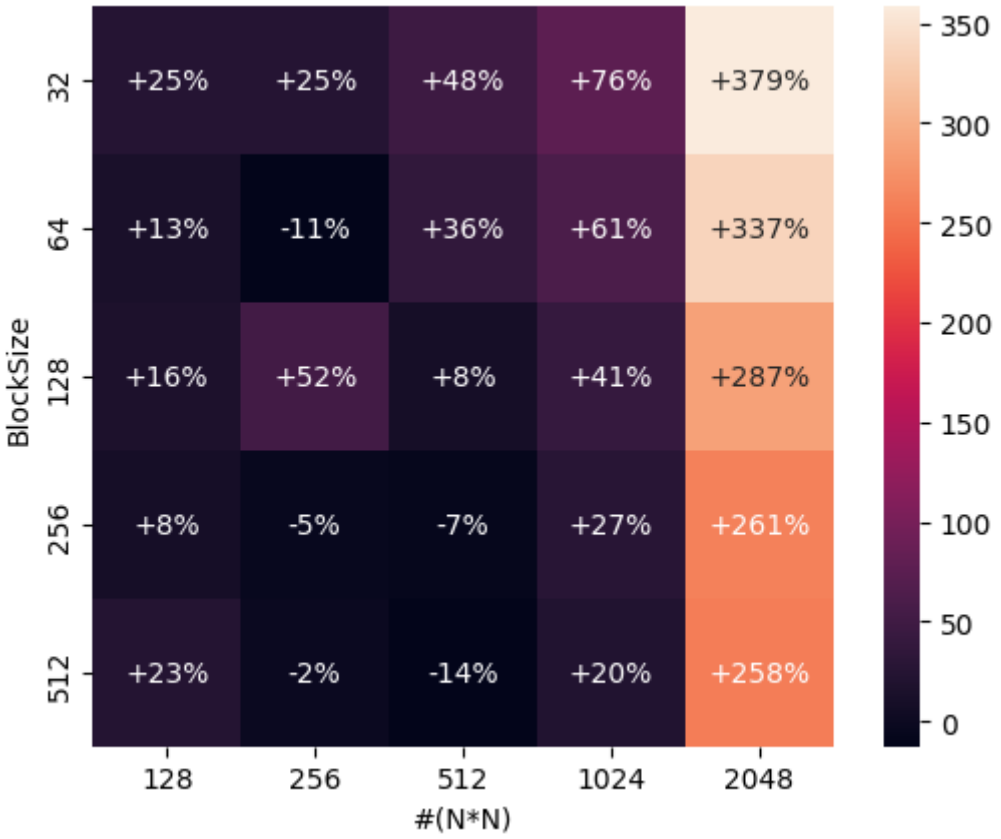

Figure 3. Percentage Boost by Block Size and Matrix Size

Analysis & Discussion

Performance by Methods

According to Table 1 and Table 2, mixed methods bring the best performance compared to the naive reference implementation. The SIMD contributes to most of the boost, while the tiled memory access pattern also contributes to the boost significantly. Thus, both methods are effective and may apply jointly.

We expect the execution time grows in the second-order polynomial time as a matrix has two dimensions. We can verify the assumption with more samples, but it takes too long to perform such a task. Still, we confidently conclude that the mixed method will accelerate the process progressively as N grows larger.

Block Size vs. Matrix Size

Using the results from Table 3 (N >=512), we could conclude that:

- Running time for both methods increase as matrix size increases
- Running time for Tiled increase as the blocksize increases (evident when the blocksize >=512)

- Running time for Naive is not affected by the blocksize
- Tiled method has more performance boost than Naive method as the matrix size increases
- Tiled method has more performance boost than Naive method as the blocksize decreases

Figure 3 also confirms the trend identified above:

1. The boost tends to be more significant for larger matrix size N. For example, the gain for N=2048 is at least 2.5X at a block size of 512. This is because, for N less than 1024, the program only takes ~10 seconds to run, so the other overheads are more significant than the memory access cost. Sometimes, tiling may take longer execution time (the negative gain) for small N due to fluctuation.

2. Generally speaking, smaller block size brings higher gain. However, because the cache size is limited, the larger block size may still pull data from memory. Although this case requires less access, we can eliminate the need by setting the block size properly to minimize the number of memory access.