An In-depth Analysis of Compiler and SIMD Enhancements in Linear Algebra

Introduction

The goal of this assignment was to delve into the intricacies of optimizing linear algebra operations, specifically matrix-matrix and matrix-vector multiplication, in C++. This exploration entailed leveraging both compiler optimizations and SIMD (Single Instruction Multiple Data) instructions to enhance the performance of the implemented algorithms. The emphasis was on understanding how different optimization strategies influence execution speed and efficiency, particularly across varying matrix sizes, optimization settings in g++, and the utilization of float vs. double data types. This report provides a comprehensive analysis of these factors, underscoring their impact on computational throughput as measured by Giga Floating-Point Operations Per Second (GFLOPS).

Methodology

Implementation Details

The development process began with the implementation of the fundamental algorithms for matrix-matrix and matrix-vector multiplication. These operations were initially coded in a straightforward manner without employing any external libraries for the core computations. The implemented algorithms supported operations on both single and double-precision floating-point numbers, accommodating the evaluation of performance across different data types.

Classic vs. SIMD Implementations

- The "classic" implementation served as the baseline, utilizing conventional loop structures for iterating through matrix elements. Memory allocation for matrices and vectors was handled dynamically, with inputs initialized to random values within a specified range.

- In contrast, the "SIMD" version was designed to exploit the parallel processing capabilities of ARM NEON, a SIMD architecture that allows for simultaneous operations on multiple data points. This approach necessitated modifications to the original algorithms, including the transposition of matrices to align data for efficient SIMD processing and the introduction of cleanup code to handle cases where matrix sizes were not multiples of the SIMD vector length.

Notably, the SIMD implementation incorporated specialized cleanup loops to ensure compatibility with all matrix and vector sizes. This was crucial because ARM NEON supports operations on four floats or two doubles at a time, potentially leading to unprocessed elements in matrices or vectors whose sizes are not exact multiples of these numbers. The cleanup code sequentially processes any remaining elements, ensuring the algorithm's correctness across varying sizes.

However, it's important to acknowledge that the input sizes for benchmarking were chosen as multiples of 4, which might introduce a bias favoring SIMD benchmarks due to the minimized impact of the cleanup loops.

Compiler Optimization Flags

The exploration of compiler optimizations was an integral part of this study, encompassing several optimization levels:

- **O0**: This level avoids code optimization to facilitate debugging and reduce compilation time. It results in slower execution for matrix operations due to the lack of optimization for loop structures and memory access patterns.
- **O1**: Introduces basic optimizations like loop unrolling and limited function inlining, which can improve memory access patterns and cache utilization.
- **O2**: Employs advanced loop optimizations and mathematical computation enhancements, leading to significant changes in memory access patterns and increased execution efficiency.

- **O3**: Utilizes aggressive loop transformations, prefetching hints, and vectorization, optimizing memory access and enabling simultaneous processing of multiple data points.
- **Ofast**: Allows the compiler to disregard IEEE or ISO rules for floating-point arithmetic, potentially simplifying operations and improving performance at the cost of accuracy and standards compliance.

Performance Analysis

The performance of each kernel was evaluated using benchmarks across different matrix sizes and optimization settings. The GFLOPS metric, calculated based on the number of floating-point operations and execution time, served as the primary indicator of performance efficiency.

Results and Discussion

GFLOPS Comparisons and Analysis

Based on the analysis, matrix-vector multiplication generally reports higher GFLOPS values across various optimization levels and particularly shines in SIMD-optimized scenarios with float data types. This can be attributed to the operation's simpler computational model, which allows for more efficient parallelization and benefits more directly from SIMD's capabilities.

Matrix-matrix multiplication, while also seeing significant performance improvements, especially at higher optimization levels and with SIMD instructions, tends to be more computationally intensive. The additional complexity and overhead involved in these operations mean that, while GFLOPS values increase significantly with optimizations, they do not quite match the peak efficiency observed in matrix-vector multiplication tasks.

Performance Impact of Compiler Optimizations

The application of different compiler optimization flags resulted in varied performance outcomes, as expected. The transition from **O0** to **O1** showcased modest improvements in execution speed, attributed to basic optimizations like loop unrolling and function inlining, which enhance memory access efficiency.

Moving to **O2** and **O3** brought significant performance leaps, thanks to more sophisticated optimizations that effectively rearrange loop structures and utilize vectorization to process data more efficiently. The **Ofast** level, with its aggressive assumptions about floating-point arithmetic, further amplified performance, particularly in operations where such assumptions did not compromise the correctness of results.

SIMD Optimizations and Data Size Considerations

The SIMD-optimized implementations demonstrated notable performance gains over their classic counterparts, especially when handling float data types. This improvement is directly linked to the ability of SIMD instructions to execute multiple operations in parallel, significantly reducing the time required for matrix computations.

However, the advantage of SIMD was more pronounced for float operations than for double, consistent with the ARM NEON's capacity to process four floats or only two doubles simultaneously. This hardware limitation underscored the importance of choosing the appropriate data type for SIMD-optimized algorithms to maximize computational throughput.

Conclusion

This assignment provided a deep dive into the optimization of linear algebra operations in C++, revealing the profound impact of compiler optimizations and SIMD instructions on performance. The exploration highlighted not only the potential for significant speedups but also the importance of tailoring optimization strategies to the specific characteristics of the computational task and the underlying hardware. Through detailed analysis and benchmarking, this report has underscored the effectiveness of these optimization techniques, contributing to a richer understanding of high-performance computing in the context of linear algebra.

References Used

- *ARM Architecture Reference Manual.* ARM Limited. This manual provides detailed information on ARM's architecture, including the SIMD instructions set (NEON) which was crucial for the SIMD optimizations in this project.
- *GCC Online Documentation.* Free Software Foundation. The GCC manual was referenced for understanding and applying different compiler optimization flags (-O0, -O1, -O2, -O3, -Ofast) and their expected impact on the performance of C++ code.

İlhan Yavuz İskurt

31112

Appendix: Benchmark Results

Optimization Level: O0				
Benchmark	Time (ns)	CPU Time (ns)	Iterations	GFLOPS
BM_MatrixMultiply<float>/16	10965	10944	63264	0.7485380117
BM_MatrixMultiply<float>/32	97110	96748	7247	0.6773886799
BM_MatrixMultiply<float>/64	892576	888906	820	0.5898126461
BM_MatrixMultiply<float>/128	7336195	7304074	95	0.5742417177
BM_MatrixMultiply<float>/256	59419104	59271750	12	0.5661117143
BM_MatrixVectorMultiply<float>/16	622	621	1127977	0.8244766506
BM_MatrixVectorMultiply<float>/32	2770	2761	253678	0.7417602318
BM_MatrixVectorMultiply<float>/64	12925	12872	53982	0.6364201367
BM_MatrixVectorMultiply<float>/128	54012	53872	12987	0.6082566083
BM_MatrixVectorMultiply<float>/256	223053	222423	3140	0.5892915751
BM_MatrixMultiply<double>/16	10930	10904	63880	0.7512839325
BM_MatrixMultiply<double>/32	99904	99239	7061	0.6603855339
BM_MatrixMultiply<double>/64	884491	864604	816	0.6063909027
BM_MatrixMultiply<double>/128	7413049	7295546	97	0.5749129674
BM_MatrixMultiply<double>/256	61111174	59754750	12	0.561535811
BM_MatrixVectorMultiply<double>/16	617	616	1134081	0.8311688312
BM_MatrixVectorMultiply<double>/32	2896	2877	245640	0.7118526243
BM_MatrixVectorMultiply<double>/64	13191	13136	53016	0.6236297199
BM_MatrixVectorMultiply<double>/128	55153	54877	12987	0.5971171894
BM_MatrixVectorMultiply<double>/256	223052	222231	3151	0.5898007029

Optimization Level: O1				
Benchmark	Time (ns)	CPU Time (ns)	Iterations	GFLOPS
BM_MatrixMultiply<float>/16	5118	5104	137217	1.605015674
BM_MatrixMultiply<float>/32	57451	57272	12222	1.144293896
BM_MatrixMultiply<float>/64	569496	567954	1213	0.923117013
BM_MatrixMultiply<float>/128	5895114	5881319	119	0.7131570316
BM_MatrixMultiply<float>/256	53193978	53052385	13	0.6324773523
BM_MatrixVectorMultiply<float>/16	107	107	6769630	4.785046729
BM_MatrixVectorMultiply<float>/32	433	432	1601032	4.740740741
BM_MatrixVectorMultiply<float>/64	2172	2166	323208	3.782086796
BM_MatrixVectorMultiply<float>/128	11333	11301	61987	2.89956641
BM_MatrixVectorMultiply<float>/256	55954	55837	12540	2.347404051
BM_MatrixMultiply<double>/16	5100	5089	137600	1.609746512
BM_MatrixMultiply<double>/32	58180	58034	12046	1.129269049
BM_MatrixMultiply<double>/64	588002	577184	1229	0.908355048
BM_MatrixMultiply<double>/128	5951778	5940145	117	0.7060945482
BM_MatrixMultiply<double>/256	52798942	52612846	13	0.6377612038
BM_MatrixVectorMultiply<double>/16	106	106	6621514	4.830188679
BM_MatrixVectorMultiply<double>/32	436	436	1606979	4.697247706
BM_MatrixVectorMultiply<double>/64	2207	2204	326127	3.716878403
BM_MatrixVectorMultiply<double>/128	11287	11270	62304	2.907542147
BM_MatrixVectorMultiply<double>/256	55186	55135	12674	2.377292101

Optimization Level: O2				
Benchmark	Time (ns)	CPU Time (ns)	Iterations	GFLOPS
BM_MatrixMultiply<float>/16	2009	2004	355234	4.087824351
BM_MatrixMultiply<float>/32	16417	16381	42728	4.000732556
BM_MatrixMultiply<float>/64	159254	158770	4331	3.302185551
BM_MatrixMultiply<float>/128	1687467	1678225	418	2.499250101
BM_MatrixMultiply<float>/256	17626582	17358439	41	1.933032803
BM_MatrixVectorMultiply<float>/16	58.7	58.5	11985070	8.752136752
BM_MatrixVectorMultiply<float>/32	226	225	3074693	9.102222222
BM_MatrixVectorMultiply<float>/64	1185	1178	594349	6.954159593
BM_MatrixVectorMultiply<float>/128	6799	6761	104221	4.846620322
BM_MatrixVectorMultiply<float>/256	32483	31922	21952	4.106008395
BM_MatrixMultiply<double>/16	2208	2162	321852	3.789084181
BM_MatrixMultiply<double>/32	18267	18173	38661	3.606229021
BM_MatrixMultiply<double>/64	182690	181750	3854	2.88466575
BM_MatrixMultiply<double>/128	1856082	1844584	380	2.273848196
BM_MatrixMultiply<double>/256	17557286	17439375	40	1.924061613
BM_MatrixVectorMultiply<double>/16	105	105	6682641	4.876190476
BM_MatrixVectorMultiply<double>/32	449	441	1595598	4.64399093
BM_MatrixVectorMultiply<double>/64	2300	2263	307905	3.619973487
BM_MatrixVectorMultiply<double>/128	11408	11348	61644	2.887557279
BM_MatrixVectorMultiply<double>/256	55811	55663	12312	2.354741929

Optimization Level: O3				
Benchmark	Time (ns)	CPU Time (ns)	Iterations	GFLOPS
BM_MatrixMultiply<float>/16	2005	1998	351718	4.1001001
BM_MatrixMultiply<float>/32	16405	16329	42944	4.013472962
BM_MatrixMultiply<float>/64	162628	161742	4327	3.241508081
BM_MatrixMultiply<float>/128	1725089	1720111	407	2.438391476
BM_MatrixMultiply<float>/256	17262414	17187268	41	1.952284214
BM_MatrixVectorMultiply<float>/16	56.0	55.9	12547501	9.15921288
BM_MatrixVectorMultiply<float>/32	228	227	3050760	9.022026432
BM_MatrixVectorMultiply<float>/64	1183	1175	592879	6.971914894
BM_MatrixVectorMultiply<float>/128	6790	6753	103342	4.852361913
BM_MatrixVectorMultiply<float>/256	32493	32200	22033	4.070559006
BM_MatrixMultiply<double>/16	2160	2144	324439	3.820895522
BM_MatrixMultiply<double>/32	18456	18274	38605	3.586297472
BM_MatrixMultiply<double>/64	186919	183951	3802	2.850150312
BM_MatrixMultiply<double>/128	1851311	1839710	379	2.279872371
BM_MatrixMultiply<double>/256	18051584	17963400	40	1.867933242
BM_MatrixVectorMultiply<double>/16	106	105	6688644	4.876190476
BM_MatrixVectorMultiply<double>/32	446	440	1598681	4.654545455
BM_MatrixVectorMultiply<double>/64	2327	2290	308185	3.577292576
BM_MatrixVectorMultiply<double>/128	11608	11543	60097	2.838776748
BM_MatrixVectorMultiply<double>/256	57223	56420	12288	2.32314782

Optimization Level: Ofast				
Benchmark	Time (ns)	CPU Time (ns)	Iterations	GFLOPS
BM_MatrixMultiply<float>/16	2008	2000	351307	4.096
BM_MatrixMultiply<float>/32	17722	16641	42733	3.938224866
BM_MatrixMultiply<float>/64	168021	163455	4339	3.207537243
BM_MatrixMultiply<float>/128	1748878	1720624	418	2.437664475
BM_MatrixMultiply<float>/256	17332116	17225512	41	1.947949762
BM_MatrixVectorMultiply<float>/16	42.6	41.8	17175891	12.24880383
BM_MatrixVectorMultiply<float>/32	71.1	70.8	9803098	28.92655367
BM_MatrixVectorMultiply<float>/64	238	237	2906361	34.56540084
BM_MatrixVectorMultiply<float>/128	927	922	764676	35.54013015
BM_MatrixVectorMultiply<float>/256	4824	4773	146641	27.46113555
BM_MatrixMultiply<double>/16	2154	2135	335227	3.837002342
BM_MatrixMultiply<double>/32	18189	18103	38877	3.620173452
BM_MatrixMultiply<double>/64	185196	182724	3760	2.869289201
BM_MatrixMultiply<double>/128	1852241	1842659	378	2.276223653
BM_MatrixMultiply<double>/256	17608516	17519600	40	1.915251033
BM_MatrixVectorMultiply<double>/16	41.8	41.6	16736202	12.30769231
BM_MatrixVectorMultiply<double>/32	128	128	5473411	16
BM_MatrixVectorMultiply<double>/64	477	475	1473148	17.24631579
BM_MatrixVectorMultiply<double>/128	1991	1984	359611	16.51612903
BM_MatrixVectorMultiply<double>/256	9429	9399	74326	13.94531333

Optimization Level: O0 (SIMD)				
Benchmark	Time (ns)	CPU Time (ns)	Iterations	GFLOPS
BM_MatrixMultiply<float>/16	7834	7817	88935	1.047972368
BM_MatrixMultiply<float>/32	53924	53380	13177	1.22772574
BM_MatrixMultiply<float>/64	439958	436988	1612	1.199776653
BM_MatrixMultiply<float>/128	3573152	3554320	194	1.18005807
BM_MatrixMultiply<float>/256	29479052	29248708	24	1.147210742
BM_MatrixVectorMultiply<float>/16	559	555	1399440	0.9225225225
BM_MatrixVectorMultiply<float>/32	1816	1803	384787	1.135884637
BM_MatrixVectorMultiply<float>/64	7077	7033	100144	1.16479454
BM_MatrixVectorMultiply<float>/128	29011	28601	24155	1.145694206
BM_MatrixVectorMultiply<float>/256	115856	115643	6047	1.13341923
BM_MatrixMultiply<double>/16	13064	13043	53705	0.6280763628
BM_MatrixMultiply<double>/32	104111	103935	6724	0.6305479386
BM_MatrixMultiply<double>/64	875716	874267	800	0.5996886535
BM_MatrixMultiply<double>/128	7231428	7217823	96	0.5811037483
BM_MatrixMultiply<double>/256	59380948	59285417	12	0.5659812092
BM_MatrixVectorMultiply<double>/16	896	895	787579	0.5720670391
BM_MatrixVectorMultiply<double>/32	3520	3516	200151	0.582480091
BM_MatrixVectorMultiply<double>/64	14257	14231	49084	0.5756447193
BM_MatrixVectorMultiply<double>/128	59355	59260	12122	0.5529530881
BM_MatrixVectorMultiply<double>/256	234697	234294	2989	0.5594338737

Optimization Level: O1 (SIMD)				
Benchmark	Time (ns)	CPU Time (ns)	Iterations	GFLOPS
BM_MatrixMultiply<float>/16	1182	1180	588220	6.942372881
BM_MatrixMultiply<float>/32	4872	4863	144061	13.47645486
BM_MatrixMultiply<float>/64	30569	30279	23775	17.31523498
BM_MatrixMultiply<float>/128	246519	245044	2831	17.11653417
BM_MatrixMultiply<float>/256	2413845	2390118	296	14.03881817
BM_MatrixVectorMultiply<float>/16	55.7	55.3	12678174	9.258589512
BM_MatrixVectorMultiply<float>/32	134	134	5244153	15.28358209
BM_MatrixVectorMultiply<float>/64	400	400	1773822	20.48
BM_MatrixVectorMultiply<float>/128	1543	1541	454424	21.26411421
BM_MatrixVectorMultiply<float>/256	7791	7779	90051	16.84946651
BM_MatrixMultiply<double>/16	1608	1606	456279	5.100871731
BM_MatrixMultiply<double>/32	8681	8665	83051	7.563300635
BM_MatrixMultiply<double>/64	72147	72017	9722	7.280058875
BM_MatrixMultiply<double>/128	694233	693316	988	6.049628164
BM_MatrixMultiply<double>/256	6099329	6090348	115	5.509444124
BM_MatrixVectorMultiply<double>/16	68.1	68	10366991	7.529411765
BM_MatrixVectorMultiply<double>/32	188	188	3732040	10.89361702
BM_MatrixVectorMultiply<double>/64	771	769	910474	10.65279584
BM_MatrixVectorMultiply<double>/128	3961	3955	175079	8.285208597
BM_MatrixVectorMultiply<double>/256	18687	18659	37657	7.024599389

Optimization Level: O2 (SIMD)				
Benchmark	Time (ns)	CPU Time (ns)	Iterations	GFLOPS
BM_MatrixMultiply<float>/16	1275	1273	549451	6.435192459
BM_MatrixMultiply<float>/32	4990	4981	139592	13.15719735
BM_MatrixMultiply<float>/64	30678	30617	23189	17.12408139
BM_MatrixMultiply<float>/128	245137	244718	2861	17.13933589
BM_MatrixMultiply<float>/256	2424821	2415170	289	13.89319675
BM_MatrixVectorMultiply<float>/16	62	61.6	12714558	8.311688312
BM_MatrixVectorMultiply<float>/32	129	129	5746324	15.87596899
BM_MatrixVectorMultiply<float>/64	377	376	1850124	21.78723404
BM_MatrixVectorMultiply<float>/128	1550	1548	452272	21.16795866
BM_MatrixVectorMultiply<float>/256	7799	7783	89984	16.84080689
BM_MatrixMultiply<double>/16	1535	1532	453671	5.347258486
BM_MatrixMultiply<double>/32	8927	8768	81001	7.474452555
BM_MatrixMultiply<double>/64	72444	72028	9716	7.278947076
BM_MatrixMultiply<double>/128	713391	712205	982	5.889180784
BM_MatrixMultiply<double>/256	6260310	6251580	112	5.367352253
BM_MatrixVectorMultiply<double>/16	70.9	70.5	9766579	7.262411348
BM_MatrixVectorMultiply<double>/32	193	193	3616767	10.61139896
BM_MatrixVectorMultiply<double>/64	781	779	898219	10.51604621
BM_MatrixVectorMultiply<double>/128	3853	3847	180424	8.517806083
BM_MatrixVectorMultiply<double>/256	18559	18532	37739	7.072739046

Optimization Level: O3 (SIMD)				
Benchmark	Time (ns)	CPU Time (ns)	Iterations	GFLOPS
BM_MatrixMultiply<float>/16	1167	1165	604778	7.031759657
BM_MatrixMultiply<float>/32	4649	4603	154623	14.23767108
BM_MatrixMultiply<float>/64	28586	28471	23460	18.41480805
BM_MatrixMultiply<float>/128	239273	237801	2937	17.63787368
BM_MatrixMultiply<float>/256	2415254	2397870	292	13.9934325
BM_MatrixVectorMultiply<float>/16	47.9	47.6	13472939	10.75630252
BM_MatrixVectorMultiply<float>/32	125	124	6231472	16.51612903
BM_MatrixVectorMultiply<float>/64	364	362	1952384	22.62983425
BM_MatrixVectorMultiply<float>/128	1548	1538	460669	21.30559168
BM_MatrixVectorMultiply<float>/256	7841	7806	89358	16.79118627
BM_MatrixMultiply<double>/16	1630	1596	481248	5.13283208
BM_MatrixMultiply<double>/32	8904	8772	80750	7.471044232
BM_MatrixMultiply<double>/64	70558	70215	9961	7.466894538
BM_MatrixMultiply<double>/128	724972	712916	1012	5.88330743
BM_MatrixMultiply<double>/256	6315685	6252304	112	5.366730728
BM_MatrixVectorMultiply<double>/16	66.5	66.4	9903652	7.710843373
BM_MatrixVectorMultiply<double>/32	192	192	3628729	10.66666667
BM_MatrixVectorMultiply<double>/64	767	766	913397	10.69451697
BM_MatrixVectorMultiply<double>/128	3978	3972	178456	8.249748238
BM_MatrixVectorMultiply<double>/256	18615	18582	37859	7.053707889

Optimization Level: Ofast (SIMD)				
Benchmark	Time (ns)	CPU Time (ns)	Iterations	GFLOPS
BM_MatrixMultiply<float>/16	1160	1157	601778	7.080380294
BM_MatrixMultiply<float>/32	4539	4529	154689	14.4703025
BM_MatrixMultiply<float>/64	28670	28618	24406	18.32021804
BM_MatrixMultiply<float>/128	238321	237089	2936	17.69084184
BM_MatrixMultiply<float>/256	2423542	2415232	289	13.89284011
BM_MatrixVectorMultiply<float>/16	46.5	46.4	14902336	11.03448276
BM_MatrixVectorMultiply<float>/32	120	120	6296380	17.06666667
BM_MatrixVectorMultiply<float>/64	373	372	1924938	22.02150538
BM_MatrixVectorMultiply<float>/128	1774	1771	395152	18.50254094
BM_MatrixVectorMultiply<float>/256	9170	9156	76873	14.31542158
BM_MatrixMultiply<double>/16	1454	1451	478685	5.645761544
BM_MatrixMultiply<double>/32	8184	8167	84293	8.024488796
BM_MatrixMultiply<double>/64	70132	69984	10002	7.491540924
BM_MatrixMultiply<double>/128	711713	708667	978	5.918582353
BM_MatrixMultiply<double>/256	6237363	6212429	112	5.401177543
BM_MatrixVectorMultiply<double>/16	69.9	69.7	9818498	7.345767575
BM_MatrixVectorMultiply<double>/32	197	197	3561272	10.39593909
BM_MatrixVectorMultiply<double>/64	897	896	781294	9.142857143
BM_MatrixVectorMultiply<double>/128	4600	4596	152268	7.129677981
BM_MatrixVectorMultiply<double>/256	23286	23249	30281	5.63774786