Parallel Processing Project: Max Flops

Aman Hogan-Bailey

University of Texas at Arlington

CSE-5351: Parallel Processing

Contents

Theoretical Maximum Flops for a Single Core (Part 1)

To determine the maximum GFLOPS of a single core of the Intel Xeon E5-2680 v4, the processor's clock speeds, and floating-point capabilities are needed. The lscpu command displays the base clock of 2.40 GHz and a maximum turbo frequency of 3.30 GHz. The Xeon E5-2680 v4 supports AVX2, allowing for 8 single-precision floating point operations per cycle. So the formula for max FLOPS is as follows:

$$Max\ Flops = ClockSpeed * \frac{Operations}{Cycle}$$

For the base clock speed of 2.40 GHz, the calculation is:

$$19.2\ GFLOPS = 2.40\ GHz * 8 \frac{Operations}{cycle}$$

For the max clock speed using turbo frequency of 3.30 GHz, the calculation is:

$$26.4\ GFLOPS = 3.30\ GHz * 8 \frac{Operations}{cycle}$$

Max Flops Testing

The experiments use level 0 optimization flags to emphasize the effect of cache and loop optimizations. I ran the experiments on both the UTA bell cluster and my PC. For the Bell cluster, on *Part 3*, I used `5120x5120` for the matrix and `3072x3072` for the matrix size on my PC. I did not use profiling tools to measure my GFlops, but rather approximated the GFlops in the program runtime. The calculations start to become slightly off when the block size is nearing the same size as the cache size, but still showed a general trend when increasing the block size.

Max Flops using Loop Orders (Part 2)

In the experiments, the `ikj` loop ordering consistently achieved the fastest execution times (4.54 seconds) and highest GFLOPS compared to other loop permutations like `ijk` (Table 1). These results are expected of how the CPU cache hierarchy is handled. The `ikj` ordering sequentially accesses rows of matrix A and C, which aligns with their row-major storage in memory, enhancing spatial locality and ensuring contiguous memory accesses that are likely to remain in the cache. Additionally, it accesses elements of matrix B in a manner that maximizes temporal locality by reusing cached data across multiple iterations of the *innermost loop*. This effective cache utilization minimizes cache misses and reduces the reliance on slower main memory accesses, leading to improved data reuse and faster computation.

Max Flops with Cache Tiling (Part 3)

The analysis of the `GEBP` algorithm using various block sizes for `kc/mc`, `m_r`, and `n_r` reveals several insights into how these parameters influence performance. The most significant factor affecting performance is the size of `kc/mc`, which determines how much of the matrices can fit into the cache. As `kc/mc` increases from `168x168(220 KB)` to `336x336 (882KB)` and `672x672(3.5MB)`, there is a clear improvement in performance, with GFLOPS reaching

a maximum of 26.4 and near full utilization of the processor, with a margin of error of around 10%. Larger `kc/mc` values improve cache efficiency by reducing memory access latency, allowing more data to be loaded into the L2 cache at once. As hinted by the Goto and Van de Geijn paper, the best performance is achieved at around half the cache size, which proved true in this instance. The best performance occurred at around `672x672 (3.5MB)`, which is approx. half the size of L2 cache (Table 3). Any higher, and unreliable results would appear since the data would be high than the cache size and cache thrashing could occur (Figure 1).

The block size for `mr` also plays a critical role in performance. Smaller values of `mr` (2 or 4) consistently deliver higher GFLOPS and lower execution times. As `mr` increases to 8, performance degrades significantly, likely due to exceeding the CPU's register capacity, causing frequent register spills and reducing computational efficiency. Thus, careful selection of `mr` is essential to balance computational throughput and register usage (Table 3).

Interestingly, variations in `nr`, which affects how many columns of matrices B and C are processed at a time, had minimal impact on overall performance. This suggests that the chosen values for `nr` were well within the L1 cache limits, and any further optimization of `nr` may not lead to significant gains. Overall, the results demonstrate that the performance of matrix multiplication can be greatly enhanced by optimizing cache and register utilization, with larger `kc/mc` and smaller `mr` yielding the best results.

Additionally, I added the results from my PC in the report to confirm the results on the Bell Cluster (Table 4). All GFlops calculations were done inside the programs without profiling tools, so they the calculations are off slightly by around 10%. Regardless, there was still a trend of the GFlops increasing when performing the techniques in the Goto and Van de Geijn paper.

Conclusions

In conclusion, optimizing cache and register utilization is crucial for maximizing the performance of matrix multiplication algorithms. The results demonstrate that larger block sizes for `kc/mc` lead to better cache usage, minimizing memory access delays and improving computational throughput. However, care must be taken with register-level blocking (`mr`) as larger values can lead to performance degradation due to register spills. While `nr` showed minimal impact, it suggests that further optimization here may not yield significant benefits in this context (Figure 1). Ultimately, fine-tuning these parameters based on hardware-specific characteristics is essential to achieving optimal performance in matrix multiplication.

References

Goto, K., & Geijn, R. A. V. D. (2008). Anatomy of high-performance matrix multiplication. ACM Transactions on Mathematical Software (TOMS), 34(3), 1-25.

Tables

Table 1

Ordering of loops and respective time and Gflops on Bell Cluster using O0 optimization flag.

Order	Seconds	GFLOPS
ijk	7.598582	0.282616
ikj	4.549784	0.471997
kij	4.575608	0.469333
kji	9.998569	0.214779
jki	9.903976	0.21683
jik	6.184155	0.347256

Table 2

Ordering of loops and respective time and Gflops on PC using O0 optimization flag.

Order	Seconds	GFLOPS
ijk	10.51146	0.204299
ikj	4.2431	0.506112
kij	4.621336	0.464689
kji	19.57734	0.109692
jki	21.41577	0.100276
jik	9.172952	0.23411

Table 3

Gflops, time, and utilization given kc, nr, and mr sizes on bell cluster using O0 optimization flag.

kc/mc	nr	mr	Gflops	time (seconds)	util
168	2	2	21.68268	12.38018	0.821314
168	4	2	21.69872	12.37103	0.821921
168	8	2	21.73193	12.35212	0.823179
168	2	4	14.79657	18.14173	0.560476
168	4	4	14.7479	18.2016	0.558633
168	8	4	14.2705	18.81052	0.540549
168	2	8	8.89045	30.19369	0.336759
168	4	8	8.973753	29.9134	0.339915
168	8	8	8.996694	29.83712	0.340784
336	2	2	26.4	6.520685	1
336	4	2	26.4	6.530053	1
336	8	2	26.4	6.543957	1
336	2	4	26.4	9.618966	1
336	4	4	26.4	9.595639	1
336	8	4	26.4	9.592237	1
336	2	8	17.04772	15.74612	0.645747
336	4	8	17.35782	15.46481	0.657493
336	8	8	17.5857	15.26442	0.666125
672	2	2	26.4	3.252181	1
672	4	2	26.4	3.24863	1
672	8	2	26.4	3.234754	1
672	2	4	26.4	4.720793	1
672	4	4	26.4	4.72035	1
672	8	4	26.4	4.718992	1
672	2	8	26.4	8.099221	1
672	4	8	26.4	8.088208	1
672	8	8	26.4	8.092853	1

Table 4

Gflops, time, and utilization given kc, nr, and mr sizes on PC cluster using O0 optimization flag.

kc/mc	nr	mr	Gflops	time (seconds)	util
76	2	2	12.61238	4.597235	0.397115
76	4	2	13.02607	4.451234	0.410141
76	8	2	12.5009	4.638232	0.393605
76	2	4	8.360938	6.934875	0.263254
76	4	4	7.961482	7.282822	0.250676
76	8	4	8.409469	6.894854	0.264782
76	2	8	5.272063	10.99798	0.165997
76	4	8	5.270244	11.00178	0.16594
76	8	8	5.275724	10.99035	0.166112
152	2	2	23.75529	2.440806	0.747963
152	4	2	24.07612	2.408281	0.758064
152	8	2	24.25208	2.390807	0.763605
152	2	4	16.54333	3.504861	0.520886
152	4	4	16.74944	3.461731	0.527375
152	8	4	16.36063	3.544	0.515133
152	2	8	10.13933	5.718531	0.319248
152	4	8	10.11935	5.729823	0.318619
152	8	8	10.24303	5.660637	0.322513
304	2	2	31.76	1.341689	1
304	4	2	31.76	1.302642	1
304	8	2	31.76	1.307702	1
304	2	4	30.88827	1.877154	0.972553
304	4	4	30.48555	1.901952	0.959873
304	8	4	30.76397	1.884739	0.968639
304	2	8	18.87146	3.072474	0.594189
304	4	8	19.03236	3.046498	0.599256
304	8	8	19.37014	2.993374	0.609891

Figures

Figure 1.

3D scatter plot of Gflops, nr, mr, and kc/mr sizes on Bell Cluster using O0 optimization flag

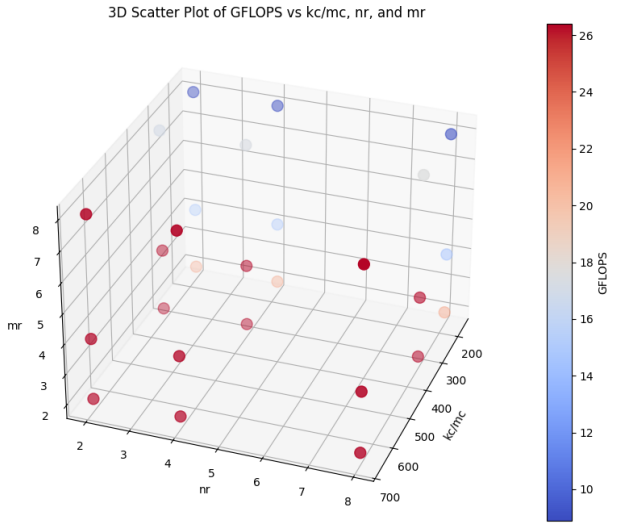

Figure 2.

3D scatter plot of Gflops, nr, mr, and kc/mr sizes on Bell Cluster using O0 optimization flag