Comparative Analysis of X64 (Intel I3 8130U) and Arm-Based (Apple M2) Processors.

# Abstract

|  |  |  |  |
| --- | --- | --- | --- |
| S. No | Parameter | Apple M2 | Intel i3-8130U |
| **1** | Architecture | “Avalanche” and “Blizzard” (Based on ARM big.LITTLE architecture) | “Kaby-Lake U Refresh” |
| **2** | Instruction Set | ARM V8.5-a | X86-amd64 |
| **3** | Max CPU Clock | 3.49GHz | 3.40GHz |
| **4** | Number of Cores | 8 (4 Performance cores, 4 Efficiency cores) | 2 |
| **5** | Multi-threading | No (1 instruction pipeline per core) | Yes (2 instruction pipelines per core) |
| **6** | L1 Cache | 192kB+128kB per core for  performance cores and 192kB+64kB per core for efficiency cores. | 128kB per core |
| **7** | L2 Cache | 16MB (Performance cores), 4MB (Efficiency cores) | 512kB per core |
| **8** | L3 Cache | 8MB Shared | 2MB |
| **9** | Memory | 8GB LPDDR4X Unified Memory | 4GB LPDDR3 Memory |
| **10** | TDP (Max) | 20W | 15W |

There are numerous microprocessors on the market today, each having a unique architecture, instruction set, and memory layout. Every design has a unique set of advantages in terms of functionality, energy usage, stability, etc. The key motivation behind this project was to compare and benchmark two different microprocessors based on different architectures and instruction sets. A slew of parameters regarding the Intel i3-8130U and the Apple M2 processors were compared through a set of standardized tests. Using tools and custom scripts like perf and pthread, among others, several aspects of both CPUs were evaluated, notably single-core performance, multi-core performance, bandwidth, memory latency, and memory security. A brief overview of specs is highlighted in Table-1.

Table-1, Overview of specs.

# Introduction

The project spanned over a period of 8 weeks where several aspects of both the CPUs were extensively studied, tested and compared. A number of specific CPU performance indicators, including single core and multi core performance, memory bandwidth, CPU cache performance, etc., were evaluated employing C/C++ programs. The perf performance data fetcher was used to fetch CPU performance data in real time. Our preliminary literature survey is addressed in Section 3.0, the project's methodology is outlined in Section 4.0, the results are discussed in Section 5.0, and the project's ultimate conclusion is presented in Section 6.0. Brief overview of specs is present in table 1.

# 3.0 Literature Survey

For the purpose of this project, a variety of academic papers, discussion boards, and web articles were referenced. Table 2 highlights all the material referenced for this project.

Table-2, Literature Survey

|  |  |
| --- | --- |
| S. No | Literary Work |
| 1 | V. Selfa, J. Sahuquillo, C. Gómez and M. E. Gómez, "Methodologies and Performance Metrics to Evaluate Multiprogram Workloads," 2015 23rd Euromicro International Conference on Parallel, Distributed, and Network-Based Processing, 2015, pp. 150-154. |
| 2 | Rhonda Kay Gaede, Freddy Golos, Michael D McMahan, Jeffrey H Kulick, Evaluation and performance analysis of the process cache: a partitioned multi-process secondary cache, Computers & Electrical Engineering, Volume 24, Issues 3–4, 1998, Pages 201-221, ISSN 0045-7906, |
| 3 | A. Raha, S. Sutar, H. Jayakumar and V. Raghunathan, "Quality Configurable Approximate DRAM," in IEEE Transactions on Computers, vol. 66, no. 7, pp. 1172-1187, 1 July 2017, doi: 10.1109/TC.2016.2640296. |
| 4 | Kim, K., Yang, H., Son, B., Yoon, H., Yim, K., Lee, M. (2020). A Survey on Attack Cases Exploiting Computer Architectural Vulnerabilities. In: Barolli, L., Xhafa, F., Hussain, O. (eds) Innovative Mobile and Internet Services in Ubiquitous Computing . IMIS 2019. Advances in Intelligent Systems and Computing, vol 994. Springer, Cham. |
| 5 | L. Liu, Y. Li, C. Ding, H. Yang and C. Wu, "Rethinking Memory Management in Modern Operating System: Horizontal, Vertical or Random?," in IEEE Transactions on Computers, vol. 65, no. 6, pp. 1921-1935, 1 June 2016, doi: 10.1109/TC.2015.2462813. |
| 6 | M. Böhnert and C. Scholl, "A dynamic virtual memory management under real-time constraints," 2014 IEEE 20th International Conference on Embedded and Real-Time Computing Systems and Applications, 2014, pp. 1-10, doi: 10.1109/RTCSA.2014.6910522. |

# 3.0 Methodology

A custom script was written to measure each aspect of the performance of a CPU. As the script was compiled and run, the perf performance counter was run on terminal to measure real time performance metrics of the CPU. Sub sections 3.0.1 to 3.0.3 discuss the methodology for testing each characteristic of the CPU.

## 3.0.1 Measurement and Comparison of Single Core Performance

Single core performance was measured by running a program on C that would estimate pi up to the first one million decimal places. The program involved using Leibnitz theorem to estimate pi and the performance metric was evaluated by observing the number of CPU cycles that were taken to compute pi. This is mainly dependent on the clock rate and the instruction set employed in the design of each of the processors, a higher clock rate and a larger instruction set will definitely lead to better performance overall.

## 3.0.2 Measurement and Comparison of Cache Performance

Various aspects of cache performance were measured such as cache read-write time, cache hit/miss rates, hit/miss times etc.

### 3.0.2A Cache read /write time measurement and comparison.

Fig. 3.1 depicts the algorithm deployed to measure cache read and write time. The cache was flushed properly to ensure that it was always cold while its performance was being measured.

Text

Description automatically generated

Fig 3.1 Algorithm used to measure cache read + write time.

The program was run for different stride lengths starting from 4 bytes all the way up to 32 bytes with an interval of 4 bytes between stride lengths and the read-write time was estimated for each of the stride lengths.

### 3.0.2B Measurement and Comparison of Cache Hit/Miss Performance

Using the perf performance counter, the cache hit/miss performance was calculated for both the CPUs. The code under test was the Pi estimator program which was previously used to measure single core performance and clock rate of both the CPUs. The performance is mainly dependent on the cache size and clock performance of the respective CPUs.

### 3.0.2C Measurement and Comparison of Memory Throughput/Bandwidth

By reading and writing data to memory and timing how long each read + write operation required, memory bandwidth was estimated. The algorithm shown in Fig 3.1 was used for this operation, but the minimum stride length was taken to be higher than the total available cache for each of the CPUs to minimize errors due to cache. Bandwidth was then calculated by (No. of bits read + written)/Total time.

### 3.0.2C Measurement and Comparison of Memory Latency

Memory (DRAM) latency was measured by first writing a specific number of bytes to memory and then measuring the total time taken to read the written data. Latency was therefore evaluated by deducting the read time from the overall time required to run the program.

## 3.0.3 Measurement and Comparison of Multi-Core Performance

Multicore performance was measured for both uniform and non-uniform loads across four logical processors. Running a script to look for even numbers within four distinct sequential intervals emulated a uniform load and running a program to look for prime numbers within those same intervals simulated a non-uniform load. Since the number of even numbers in any two sequential intervals of the same size is always the same, this load can be spread uniformly across all processing cores whereas the same cannot be done for searching for prime numbers since the number of prime numbers between two sequential intervals of the same size need not be the same.

# 4.0 Results and Discussion

### 

## 4.0.1 Measurement and Comparison of Single Core Performance

Table -3, highlights the results for the single core performance of both the CPUs. For the purpose of reference, the same code was run on a remote server and then the results for the local machines were compared.

Table-3, Single core performance of CPUs

|  |  |  |
| --- | --- | --- |
| S. No | Machine | Number of Cycles (Average) (Based on wall-clock time) |
| 1 | Server | 19575 |
| 2 | I3 (Windows) | 17044 |
| 2.1 | I3 (Linux) | 28605 |
| 3 | M2 | 10843 |

**Inference:** Apple M2 microprocessor was consistently faster than Intel i3. The execution time required by the Apple processor was significantly lower than its Intel counterpart.

## 4.0.2 Comparison of Memory Architecture

Table-4 gives an overview into the memory architecture of both the CPUs

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| S.no | Property |  | M2 | I3-8130U |
| 1. | Volatile | L1 Cache | Performance cores( 192kb +128kB,per core)  Efficiency Cores(128+64kB, per core) | 64kB (Per Core) |
|  |  | L2 Cache | Performance cores(16 MB)  Efficiency cores( 4 MB) | 256kB (Per Core) |
| L3 Cache | 8MB | 4MB |
| RAM | 8GB LPDDR4X Unified Memory | 4GB LPDDR3 SODIMM Memory |
| 2. | Non-Volatile | Swap Space | 8GB (on NVMe SSD) | 8GB (on NVMe SSD) |

## 4.0.2 Cache read/write time measurement and comparison

|  |  |  |
| --- | --- | --- |
| Apple M2 |  |  |
| S.No |  |  |
| 1 |  |  |
| 2 |  |  |
| 3 |  |  |
| 4 |  |  |
| Intel i3 8130U |  |  |
| 1 |  |  |
| 2 |  |  |
|  |  |  |
|  |  |  |

### 4.0.3 Cache read /write time measurement and comparison.