

Mining DETI coins

Rafael Gil (118377)

David Raposo (93395)

Abstract –This project focused on mining DETI coins—52-byte files whose MD5 hashes end with a specified number of zero bits. Using various computational methods, including SIMD instructions, multi-threading and GPU acceleration, the goal was to optimize the search process and achieve high-performance results.

This report summarizes the techniques implemented, the performance of different approaches, and the insights gained into leveraging advanced computational architectures for this task.

Resumo –Este projeto centrou-se na extração de ficheiros DETI de moedas-52-byte cujos hashes MD5 terminam com um número especificado de bits zero. Utilizando vários métodos computacionais, incluindo instruções SIMD, multi-threading e aceleração de GPU, o objetivo era otimizar o processo de pesquisa e obter resultados de elevado desempenho.

Este relatório resume as técnicas implementadas, o desempenho das diferentes abordagens e os conhecimentos adquiridos sobre o aproveitamento de arquiteturas computacionais avançadas para esta tarefa.

Keywords –High-Performance Computing, MD5 Hashing, SIMD Instructions, GPU Acceleration

Palavras chave –Computação de alto desempenho, hashing MD5, instruções SIMD, aceleração de GPU

I. Introduction

This report presents the development and implementation of high-performance computing techniques for mining DETI coins, 52-byte files whose MD5 hashes meet specific criteria. Various computational methods, including SIMD instructions, multi-threading, GPU acceleration using CUDA, and distributed computing, were employed to optimize the mining process. Performance metrics, such as the number of MD5 computations per hour, were gathered across different platforms, and the results were analyzed to compare efficiency. The findings highlight the impact of algorithmic optimizations and hardware capabilities on computational throughput, offering insights into advanced techniques for solving cryptographic challenges.

II. Analysis of the Algorithms Implemented

A. AVX

The *AVX* search function, in file *deti_coins_avx_search.h*, is designed to mine DETI coins efficiently using Advanced Vector Extensions (*AVX*) to process multiple candidates simultaneously. It operates with four lanes in parallel,

allowing the program to evaluate four potential DETI coins in a single operation, significantly speeding up the search process compared to traditional methods.

The function initializes templates for potential coins and dynamically updates their content during each search iteration, using the function *inc()*, which increments a 32-bit hexadecimal value, wrapping individual bytes within the *ASCII* printable range. If a byte reaches *0x7F*, it is reset to *0x20*, and the next higher byte is incremented. This cascading mechanism continues until either no overflow occurs or all bytes are reset to *0x20*, at which point the function returns 1 to signal completion. By managing the coin values in this way, the *inc* function ensures that the search process covers all valid *ASCII* combinations systematically and avoids invalid characters.

Using the *AVX*-based *MD5* computation, the program calculates hashes for the coin candidates in parallel. Each hash is then validated to determine its "power," and coins meeting the required threshold are saved.

B. AVX2

The *AVX2* search function, in file *deti_coins_avx2_search.h*, builds on the principles of the *AVX* implementation but leverages the enhanced capabilities of *AVX2* to process eight candidates concurrently. This increase in parallelism significantly boosts the efficiency of DETI coin mining.

The function initializes the coin templates and hash storage arrays, aligning them in memory for optimal *SIMD* operations, now being 32 memory spaces instead of the *AVX* 16 memory spaces. Each coin template is pre-filled with static and dynamic portions to ensure unique values for all eight lanes.

During the mining loop, the function updates the coin templates dynamically and computes the *MD5* hashes using the *AVX2*-specific *MD5* computation function. The results are processed lane by lane to validate their "power." Valid DETI coins are saved to the output buffer, and relevant counters for coins found and total attempts are updated.

Once again the *inc()* function is used to ensure that the values used to generate coins are different maintaining the validity of the generated candidates while allowing the function to cover all possible combinations efficiently.

The *AVX2* implementation demonstrates superior performance due to its higher degree of parallelism. It processes twice as many candidates as the *AVX* version in each iteration, making it an essential up-

grade for scenarios requiring high-throughput DETI coin mining.

C. POSIX Threads

The POSIX Threads - *Pthreads* - search function, in file *deti_coins_threaded_search.h*, extends the DETI coin mining process by utilizing multi threading to distribute the workload across multiple CPU cores. This approach significantly improves performance by enabling concurrent execution, with each thread working on a unique subset of the problem space to avoid redundant calculations.

Each thread executes a worker function, which processes eight lanes of coin candidates simultaneously using *AVX2* instructions. To ensure that threads do not perform duplicated work, each thread incorporates its unique *ID* into the initialization of the coin templates. This guarantees that the coin values processed by one thread differ from those handled by other threads, effectively dividing the workload and eliminating overlap.

Within the mining loop, each thread dynamically updates its assigned coin templates and computes their *MD5* hashes in parallel. Valid coins are saved and the thread's individual counters for attempts and coins found are updated.

The number of threads corresponds to the number of available CPU cores, ensuring optimal resource utilization. At the end of the execution, results from all threads are aggregated, yielding the total number of attempts, coins found, and an overall measure of search efficiency.

This multi threaded implementation demonstrates the effectiveness of using thread-specific IDs to partition the computational workload. By preventing redundant work, it maximizes CPU utilization and achieves significant speedups, making it highly efficient for high-performance DETI coin mining.

D. WebAssembly

The WebAssembly search function, in file *deti_coins_webassembly_search.c*, brings DETI coin mining to the web and cross-platform environments by leveraging the lightweight and portable nature of WebAssembly. This approach allows the mining process to be executed efficiently in web browsers or environments that support WebAssembly, without relying on native CPU or GPU-specific instructions.

The function initializes a coin template with mandatory fields alongside dynamically updated sections for unique candidate generation. The *MD5* hash is computed using a custom implementation embedded directly in the WebAssembly module, ensuring compatibility with the runtime environment.

A key feature of the implementation is its systematic exploration of possible coin values using the *inc()* function. This ensures exhaustive and unique candidate generation while staying within the *ASCII* printable

range. For each generated coin, the function computes the *MD5* hash and checks if it meets the criteria for a valid DETI coin (at least 32 trailing zeros in the hash). Valid coins are output in hexadecimal format, and counters for attempts and coins found are updated.

The WebAssembly implementation is designed to maximize portability and enable mining across diverse platforms. Although it lacks the advanced parallelism of *SIMD* or multi threaded implementations, it offers a straightforward and accessible way to mine DETI coins in environments where more specialized approaches may not be feasible. This makes it a versatile addition to the set of mining strategies, bridging the gap between high-performance computing and platform-independent execution.

E. CUDA

The CUDA implementation is divided into two key components: the kernel function and the host-side search logic.

E.1 Kernel Function (*deti_coins_cuda_kernel_search.cu*)

The kernel is designed to perform parallel computations of DETI coin candidates across thousands of GPU threads. Its main tasks are as follows:

- **Generate DETI Coin templates:** Each thread creates a unique DETI coin template by modifying specific fields based on the thread ID and custom words provided by the host.
- **MD5 Hash Computation:** Each thread computes the MD5 hash of its coin template. This hash computation is implemented using a predefined macro from *md5.h*.
- **Validation:** The kernel checks if the computed hash meets the DETI coin requirement (last 8 hexadecimal characters of the hash are zero, i.e., 32 trailing zero bits).
- **Check Results:** Valid DETI coins are stored in a global buffer on the device.

E.2 Host Code (*deti_coins_cuda_search.h*)

The host-side code manages the CUDA execution and coordinates between the CPU and GPU.

- **Memory Management:** The host code allocates device memory for input and output buffers, including DETI coin templates, hash storage, and result buffer.
- **Customizable parameters:** The search is parameterized by three custom words (*custom_word_1*, *custom_word_2*, and *random_word*) that define the search space. These words are iteratively updated during the search.
- **Kernel Execution:** The *deti_coins_cuda_search.h* function repeatedly launches *deti_coins_cuda_kernel_search.cu*. It configures the kernel with: A grid size of (1 « 20) / 128 blocks and 128 threads per block and

device memory parameters and custom words passed as arguments.

- Statistics: At the end of the execution, the host should print statistics about the number of attempts, the coins found, and the efficiency of the search.

Due to unexpected errors, the CUDA implementation remains unfinished. However, the provided code closely follows the described methodology and successfully translates the intended functionality into CUDA.

F. Special search

Another CPU-based search was implemented, seen in file `deti_coins_cpu_special_search.h`, but with a twist. It had a special format on top of the regular "DETI coin" format. We chose to search for coins with the surname of author 1 embedded in it. We managed to find 3 coins, seen in figure 1, with this format, in 1 hour.

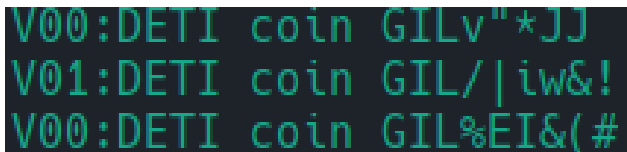


Fig. 1 - Special format coins

III. Results

The performance of DETI coin mining algorithms was evaluated on three *CPU*s using four approaches: sequential *CPU*-based mining, *AVX*, *AVX2*, and multi threaded *POSIX* implementations. Results showed significant differences in throughput and coin discovery, influenced by the algorithm and *CPU* capabilities.

Sequential *CPU*-based mining was the slowest, with low calculation rates and coin discovery. *AVX* instructions nearly tripled performance over the *CPU*-only approach, while *AVX2* doubled *AVX* performance through wider vectors and better resource utilization. The *POSIX* multi threading approach achieved the highest performance, leveraging parallelism to process multiple candidates simultaneously. On the i7-10870H, *POSIX* reached a calculation rate ten times higher than the *CPU* approach and significantly outperformed *AVX2*.

Among *CPU*s, the i7-10870H delivered the best results, benefiting from its higher core count and clock speed, achieving 376 coins with *POSIX*. The i7-4558U lagged behind due to fewer cores, but *AVX2* still showed strong improvement over *AVX*. The i7-8750H showed steady gains across methods, with *POSIX* providing five times the calculations of *AVX2*, demonstrating the effectiveness of *multithreading*.

These results highlight the impact of advanced methods and hardware on performance. Vectorized instructions like *AVX* and *AVX2* offer significant improvements, while multi threading with *POSIX* maximizes

efficiency, making it the most effective approach for DETI coin mining, among the ones tried.

The bar chart in Figure 2 illustrates calculation rate improvements, with *POSIX* achieving the highest rates and the i7-10870H outperforming other *CPU*s due to its parallel processing capabilities. Figure 3's stacked bar chart shows total coins discovered, emphasizing the dominance of advanced methods like *AVX2* and *POSIX*, with *POSIX* providing the largest boost. These findings underscore the importance of multi threading, vectorization, and hardware capabilities in mining efficiency.

IV. Conclusion

In conclusion, this report demonstrates the significant impact of computational methods and hardware capabilities on the performance of DETI coin mining. The results show that while sequential *CPU*-based mining offers limited performance, the use of vectorized instructions like *AVX* and *AVX2* substantially increases throughput. The most notable improvement comes from the *POSIX* multi threading approach, which leverages parallel processing to achieve the highest calculation rates and coin discovery across all tested *CPU*s.

Among the *CPU*s tested, the i7-10870H outperformed the others, benefiting from its higher core count and clock speed, while the i7-4558U, with fewer cores, showed reduced performance despite the use of *AVX2*. The i7-8750H demonstrated steady improvements with each method, with *POSIX* providing the greatest boost.

Although the results demonstrate significant improvements using *AVX*, *AVX2*, and *POSIX*, it is worth noting that a *CUDA*-based implementation would likely have been the most efficient, potentially leading to even greater performance gains. Unfortunately, due to errors in the developed code, we were unable to test this approach within the available time frame. Nonetheless, these findings underscore the importance of modern computational techniques, such as vectorization and multi threading, and the hardware's capabilities in optimizing performance for DETI coin mining. The *POSIX* multi threading approach, in particular, proves to be the most effective method, maximizing the potential of available resources and achieving significant performance gains.

References

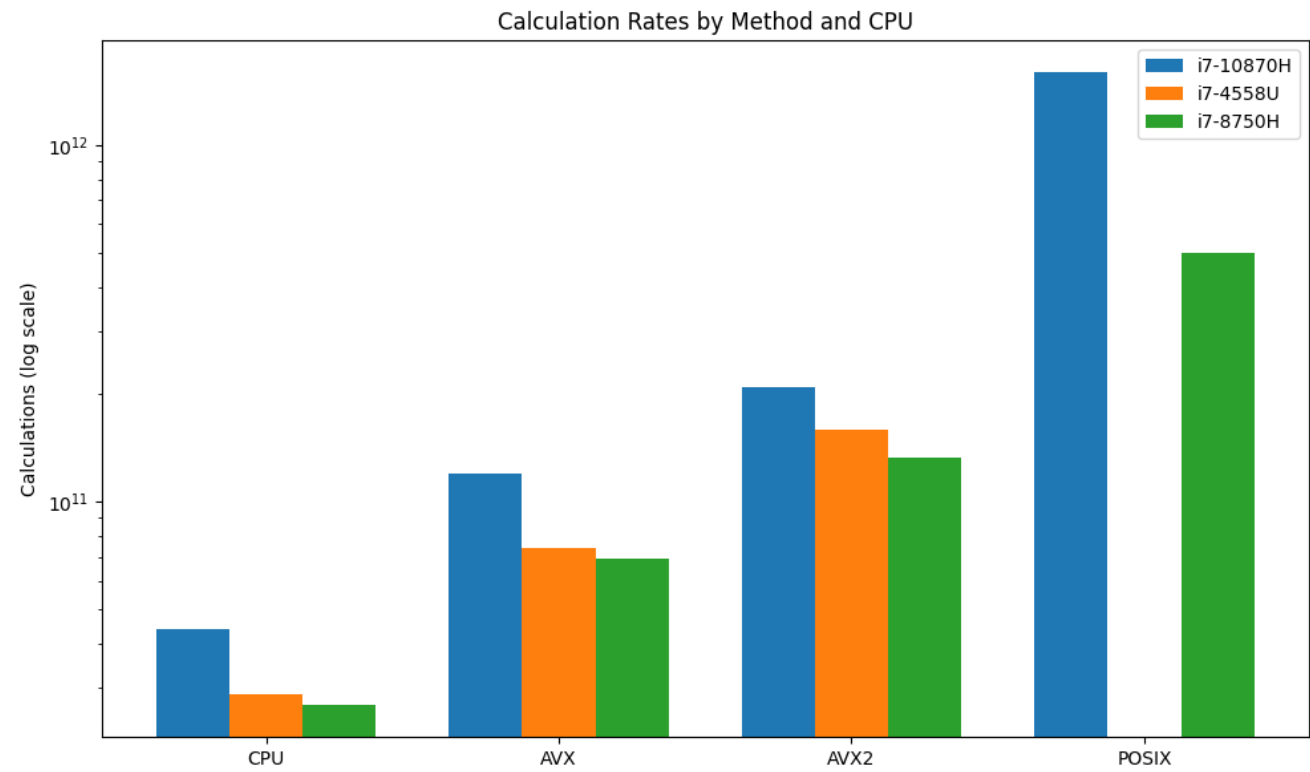


Fig. 2 - Calculation Rates by Method and CPU

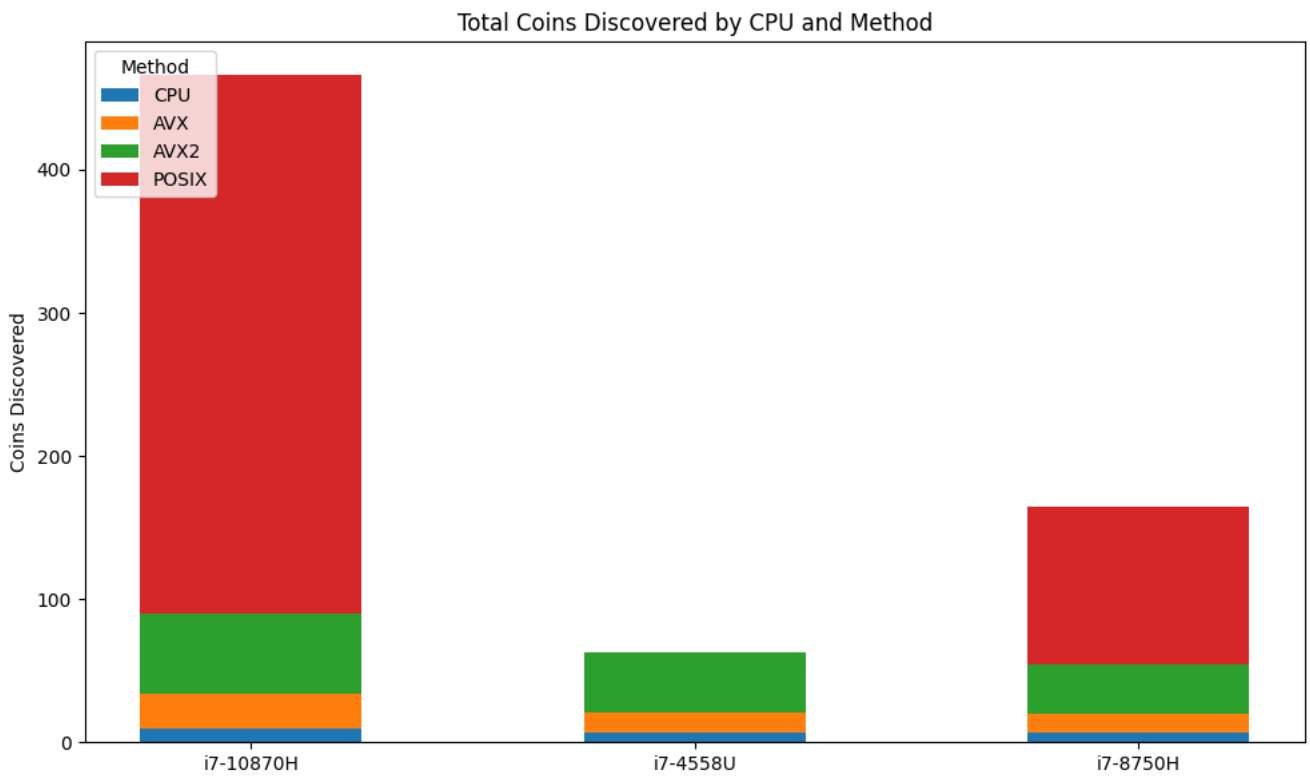


Fig. 3 - Total coins by Method and CPU