electronics

MDPI

Article

A Preliminary Empirical Study of the Power Efficiency of Matrix Multiplication

Fares Jammal [1,*], Naif Aljabri [1], Muhammad Al-Hashimi [1], Mostafa Saleh [2] and Osama Abulnaja [1]

[1] Department of Computer Science, Faculty of Computing and Information Technology, King Abdulaziz University, Jeddah 25732, Saudi Arabia; naifkau@gmail.com (N.A.); mhashimi@kau.edu.sa (M.A.-H.); abulnaja@kau.edu.sa (O.A.)
[2] Department of Information Systems, Faculty of Computing and Information Technology, King Abdulaziz University, Jeddah 25732, Saudi Arabia; msherbini@kau.edu.sa
* Correspondence: fjammal@kau.edu.sa

Abstract: Matrix multiplication is ubiquitous in high-performance applications. It will be a significant part of exascale workloads where power is a big concern. This work experimentally studied the power efficiency of three matrix multiplication algorithms: the definition-based, Strassen's divide-and-conquer, and an optimized divide-and-conquer. The study used reliable on-chip integrated voltage regulators for measuring the power. Interactions with memory, mainly cache misses, were thoroughly investigated. The main result was that the optimized divide-and-conquer algorithm, which is the most time-efficient, was also the most power-efficient, but only for cases that fit in the cache. It consumed drastically less overall energy than the other two methods, regardless of placement in memory. For matrix sizes that caused a spill to the main memory, the definition-based algorithm consumes less power than the divide-and-conquer ones at a high total energy cost. The findings from this study may be of interest when cutting power usage is more vital than running for the shortest possible time or least amount of energy.

Keywords: matrix multiplication; power-aware algorithms; Strassen's divide-and-conquer multiplication; running average power limit (RAPL); high-performance computing (HPC)

Citation: Jammal, F.; Aljabri, N.; Al-Hashimi, M.; Saleh, M.; Abulnaja, O. A Preliminary Empirical Study of the Power Efficiency of Matrix Multiplication. *Electronics* **2023**, *12*, 1599. https://doi.org/10.3390/electronics12071599

Academic Editors: Juan M. Corchado and David Defour

Received: 18 February 2023
Revised: 28 March 2023
Accepted: 28 March 2023
Published: 29 March 2023

1. Introduction

A frequent and computationally intensive operation in scientific computing and other high-performance applications is matrix multiplication. It is reasonable to expect it to be a significant portion of exascale workloads. Power consumption has been a major obstacle for developing exascale systems for many years [1] and will continue to be a concern in the future. A point of general motivation for this investigation is the notion that some algorithms may offer savings in power consumption that may be of value in systems that use them in a significant portion of typical workloads. In an exascale system, even small savings in power may be valuable, as the effects may accumulate and amplify in potentially massive workloads if sufficiently ubiquitous.

Previous work on the power characteristics of other commonplace computational tasks, such as sorting [2,3] and searching [4], ave provided further encouragement to examine matrix multiplication. Those studies showed that some methods, not necessarily the usual choices, could provide a better alternative for power-aware applications. In [3], experimental results showed that a natural unoptimized mergesort consumes less power than a highly optimized three-way quicksort. A later study by other researchers supported that result [5]. Similarly, in [4], a two-way binary search was more power efficient than its faster three-way counterpart. The researchers identified interesting time-power tradeoffs in cases of both mergesort and two-way binary search.

Moreover, work in [6,7] showed that the codes executed by a processor are correlated with the chip temperature. Codes generated under different conditions cause the chip to

run at different temperatures, which reflect the code and measures inside the processor designed to protect against overheating. The two chief influences on the code are the original algorithm and the compilation process. Compilers typically offer rich compilation options focused on execution speed and code efficiency, not power performance. The researchers experimented with matrix multiplication code. They proposed optimizations to reduce overheating based on monitoring the temperature reported by the chip. Their work served as further motivation for this work, which explores algorithmic aspects.

In this study, we describe a preliminary investigation of the energy and power characteristics of three matrix multiplication algorithms:

1. Definition-based matrix multiplication (will be referred to as D3.0 in this work).
2. Basic divide-and-conquer matrix multiplication by Strassen (D2.8).
3. Optimized divide-and-conquer multiplication (D2.4).

The first is simply the familiar pen–paper systematic row-by-column multiplications. The difference between the three methods is best illustrated by examining the 2-by-2 case. The definition-based algorithm performs eight basic multiplications to generate the elements of the product matrix. The other two perform only seven but combine the products in different ways. A recursive divide-and-conquer provides a succinct statement. It repeatedly replaces a product with a combination of seven half-dimension submatrices. The naming convention intends to remind the reader of an algorithm by its time complexity, which in each case is a polynomial term with a different degree. For example, the well-known time efficiency of brute force implementation of the definition of matrix multiplication is n^3, hence the D3.0 tag. For the other two, the degrees are either $\log_2 7 \approx 2.807$ [8] or $\approx$2.376 [9]. The programs used in the study were based on code from [10,11] adapted to the experimental environment. The study relied on internal sensors embedded in an HPC-class processor. Power, energy, and other microarchitectural activity measurements from those sensors were collected via the Intel running average power limit (RAPL) interface. A recent study showed that RAPL provided accurate power readings compared to plug power on the processor used for the experiment [12]. In that report, the terms power and energy were not used interchangeably.

The objectives of this study, within the general goal of discovering new ways to help build power-efficient HPC software, are threefold. First, to study the primary matrix multiplication algorithms that may be considered in high-performance applications and compare them based on their power efficiency on a credible HPC platform. Second, to quantify the power cost for each in an HPC context. Third, to help identify the best choices for applications where power efficiency is a priority. The organization of this paper is as follows. Section 2 reviews in chronological order interesting related research in the last ten years, focusing on the methods and the findings. Section 3 introduces the methods and materials used in this research. Section 4 presents and discusses the results. Finally, some conclusions and suggested future work are in Section 5.

2. Literature Review

Basmadjian et al. [13] performed redundant multithreading (RMT) on a two-way chip microprocessor (CMP) in gate-level mode. The failure rate of the method was estimated through the injection of the fault. The energy consumption and performance cost of the method were also studied. The simulation and examinations performed resulted in an RMT error (relative to 4% capacitance) of 91.7%. The RMT technology consumed 20 to 40 times more energy than conventional technology.

Khezripour et al. [14] implemented two different types of electronic refrigeration and cooling systems, exploring reducing the total energy consumption with a model that combines a microprocessor and a real cooling system. They utilized thermoelectric coolers (ETEC) to cool the cache, achieving a modest three development but with the benefit of integration. In addition, they utilized their cooling system for chip-level cooling, which saved 25% in energy in contrast to the non-cooled designs.

Partk and Yang [15] suggested a method to anticipate the number of cores needed to interconnect multicore processors using scheduling control information. The method was characterized by low energy consumption without deterioration. It was successfully applied to a 32-core processor.

Al-Hasib et al. [16] examined the energy competence impact of applying data reuse conversion to utilize time location on a multicore processor operation with a movement evaluation algorithm. This method increased the energy competence 5.5 times.

Kodaka et al. [17] comprehensively surveyed approaches for estimating the power consumption of single-core and multicore processors, virtual machines, and the entire server.

Dargie and Wen [18] examined a lightweight probability model to evaluate the power utilization for the entire CPU, network interface cards, and servers. They assessed the precision of the model via two criteria (custom-made and standard benchmarks) on two heterogeneous server platforms. The reported error related to the custom benchmark was minus 1%, while it was minus 12% for the standard scale.

Yuechuan et al. [19] provided an energy examination methodology for C resource programs. Building on the basic structure of the C process and the C program, this technique proposed classifying atomic processes and creating a database of power for atomic processes through experiments. For the sequential statement block, control blocks, and subblocks, three types of methodologies related to counting the energy were suggested. The entire C program was turned into a stream tree. The power prediction algorithm for C was advanced.

Hamady et al. [20] estimated the power used by multicore processor systems when operating diverse work burdens with a restricted numeral of cores. Their method developed energy competence by applying a single physical core with ultra-connectivity.

Poon et al. [21] presented an energy-conscious sorting algorithm applying networked computers with visual hierarchical mapping. This reduced the overall time and data flow, thus decreasing energy consumption.

Aliaga et al. [22] offered an energy-saving runtime application responsible for simultaneously executing ILUPACK on a multicore platform. The findings demonstrated that decreasing the idle time of the strands saved energy without degrading the performance.

Yildiz et al. [23] evaluated the power utilization of diverse I/O methods. Regardless of the system architecture or application-specific parameters, the model chose the best I/O method based on energy competence.

Cebrian et al. [24] applied microarchitecture to correspond to power limitations while minimizing the processor power consumption. They converted the decentralized CPU power consumption at the cycle and base block levels into icons to choose among various microarchitectural methods to save on energy consumption. The energy efficiency was increased by 11% for this method.

Lastovetsky et al. [25] suggested a new model founded on approaches and algorithms to reduce the computing time and effort for equal data applications implemented on a similar multicore mass (Intel E5-2670 Haswell Server). Compared to the customary balanced workload allocation, it also showed a significant increase in the two applications' average and maximum percent of performance and functionality. The findings demonstrated that performance improvement alone could significantly decrease energy consumption.

Abdel-Hafeez et al. [26] suggested a new mathematical algorithm for sorting and categorizing the correct elements of the input data on a y basis without a comparison. They used a custom built-in CMOS circuit (90 nm, Taiwanese semiconductor manufacturer) with a 1 V power supply to evaluate the work and contrasted it with other device-based enhanced hybrid classification designs; the findings demonstrated a 50% energy savings.

Gupta et al. [27] suggested a new methodology to distribute the best CPU and GPU power consumption to mobile platforms within a provided power financial plan. They assessed their work using simulations and trials on industry-standard advanced mobile platforms. The findings demonstrated the high productivity and efficient use of the avail-

able stagnated energy. They effectively achieved the objective of distributing the best power consumption for the CPU and GPU for the provided power.

Aljabri et al. [3] conducted a comprehensive empirical study on a high-performance quicksort against basic mergesort, paying careful attention to the experimental environment. They used a RAPL profiler on a light Linux setup to obtain measurements on an HPC-class Intel Haswell chip, which supported better instrumentation. They gathered energy (joules), power (watts), and cache miss information. The work was careful to focus readings on the experimental code and to eliminate those from other sources. The plain mergesort showed a clear advantage over the quicksort.

Haidar et al. [28] examined the performance of several kernels and the energy consumption of different hardware elements from a maximum power perspective. They utilized the PAPI instrument of power tuning to decrease the total power consumption by 30%, and there was no regression in performance in many cases.

Kondo et al. [29] suggested a venture under the name of "BomPP venture" and advanced numerous energy-saving methods within a specific energy financial plan. They also provided a recognized resource manager, energy performance simulation, and examined a framework for future supercomputer systems.

Chandra et al. [30] researched the impact of programming languages related to power consumption. They used a higher than one programming language (Java, Visual Basic, and C#) to implement different sorting algorithms (bubble, pick, insert, and quick sort) to find the most energy-efficient programing language. They discovered that Java was the most energy-efficient, and the least energy-efficient was Visual Basic.

Ozer et al. [31] applied a machine learning methodology to anticipate energy indicators and legacy guidelines with potential recurrence settings by utilizing information gathered at runtime. Then, they used degradation algorithms, creating preprocessed data and training algorithms to anticipate the best recurrence settings.

3. Methods and Procedures

The underlying objective of determining the power advantage that an algorithm for a basic computation process may have over its competitors is a critical driver of the methodology used in this research. As much noise as possible, i.e., power from environmental source,s had to be removed to obtain representative sensor values. Eliminating noise lends credibility to results. A considerable time was spent with the CPU environment to ensure removal of noise. For the best results, the experimental code was run on one core as a precaution to guarantee that readings were least affected by on-chip optimizations, internal power control mechanisms, or temperature effects from neighboring silicon. For a reliable estimate of the measured values, the experiment was allowed to run as much as needed until converging on an average value. In this environment, the average stabilized around 240 runs. Execution time was used as a control to confirm that the code worked as expected.

This section outlines the techniques and tools utilized in the research. It is divided into four parts: the environment of the experiments, the datasets used in the experiments, the characteristics of the executables that implemented the algorithms, and the profiling tool used to collect CPU data and related statistics.

3.1. Experimental Environment

This empirical study relies on an experimental setup originally developed for work described in [3]. The setup and the various measures and settings designed to eliminate as many factors as possible that could confound measurements in empirical studies, such as the ones described in this report, are detailed in [32]. This study contributes with a significant update, however. During the initial runs of the experiments, some environmental noise was detected in the results. The expected time complexity behavior was somewhat off. It was determined to be due to the data collection process thread running on multiple cores instead of the designated one, as had been anticipated. To fix this, we set the affinity for the OS thread and the process to run on core 0 only, thus eliminating another power

source that contaminates the measurements. It is vital to ensure the credible of the majority of the power readings that come from the running code to be able to discuss the underlying algorithms. This experience highlights the importance of collecting information about timing in these studies.

An Intel Xeon-class Haswell processor, a popular HPC part that provides credible readings from its internal sensors [12], was used. The addition of fully integrated voltage regulators (FIVR) to the Haswell platform provided the reliable, high-resolution measurements required for the experiments in this research. Table 1 lists the specifications of the machine used for running the experiments. The setup is shown in Figure 1.

Table 1. Experimental platform specifications.

Processor	Intel Xeon E5-2680/v3 2.50 GHz 12 cores
Cache	L1 data: 12 × 32 KB (8-way set associative)
	L1 instruction: 12 × 32 KB (8-way set associative)
	L2: 12 × 256 KB (8-way set associative)
	L3: 30 MB shared (20-way set associative)
Memory	8 GB
Operating System	Linux Ubuntu 16.04 64-bit
Compiler	GCC 7.5.0 (Ubuntu 7.5.0-3ubuntu1~18.04)

Figure 1. The experimental setup.

3.2. Test Dataset Generation

Square matrices, sized (dimension) from 50 × 50 to 1500 × 1500, were generated for the tests. They were filled by randomly generated 32-bit integer operands ranging in value from 0 to 9 (`int` in C++). A dataset was run three hundred times in each case, of which the first twenty were ignored to bypass the initial thermal state of the system and start from a consistent point. The remaining runs were enough to obtain a reliable average.

3.3. Executable Files

The algorithm codes, [10,11] for the Coppersmith–Winograd-based D2.4, were reimplemented in C++ to fit the experimental framework. Executables were produced by the GCC 64-bit compiler. Optimizations were disabled to minimize the role of compilation in shaping the code. With no optimization, the code was expected to be more faithful to the original algorithms. This measure was crucial for being able to argue about the underlying method. In addition, we specified in the code the affinity of the executables to processor core number 0 to avoid context-switching overheads when sampling the CPU sensors. In particular, the thread and the process were explicitly specified to run on core 0 only.

3.4. Profiling Tools

The Linux *perf* profiler was used to obtain the measurements. It is available from the kernel as a command. The powerful tool was developed to reliably read the CPU sensors based on the RAPL interface by Intel. It was configured to automatically run the executables from designated folders and systematically record the readings from the CPU.

4. Results and Discussion

Loading input datasets and reading sensor data while running the algorithm's code will consume some power that adds to the measurements. Those unwelcome additions, considered noise in this research, must be eliminated. To remove that noise, we configured the profiling tool to read sensor data right after running the executables. Furthermore, datasets were integrated directly into the executables to avoid the overheads associated with the loads. These simple measures resulted in measurements that better reflected the code. Table 2 presents the main results. They show the averages of three hundred executions for each set size. Randomized datasets ensured that the results were unaffected by anomalies of a set. A quick look reveals that there were no significant differences in energy or power consumption for very small matrix sizes. At those sizes, all methods would have very close runtimes. Time is the main factor affecting energy consumption in general, which seemed to be the case there. In addition, the readings at dimension 50 seemed off. Runtime at that point was very short and likely the most distorted by environmental factors. It may be ignored as an outlier.

Table 2. Average energy in millijoules (mJ), power in watts, and percentage difference of D2.4 relative to the other methods, where negative indicates better performance. The region of best power savings is marked.

Matrix Dimension	Energy			Power			% D2.4 Advantage			
							Energy		Power	
	D3.0	D2.8	D2.4	D3.0	D2.8	D2.4	D3.0	D2.8	D3.0	D2.8
50	39.6	47.2	53.5	13.2	11.8	10.7	35	13	−19	−9
100	132.3	136	141.9	14.7	13.6	12.9	7	4	−12	−5
150	385	402.3	394.8	15.4	14.9	14.1	3	−2	−8	−5
200	1333.8	1312	1271.7	17.1	16.4	15.7	−5	−3	−8	−4
250	3860.6	3698.4	3633.7	19.4	18.4	17.9	−6	−2	−8	−3
300	14,552	10,341.2	7819.5	21.4	20.6	19.5	−46	−24	−9	−5
350	50,020	29,106	18,144	24.4	23.1	22.4	−64	−38	−8	−3
400	166,123	79,679.8	37,861.2	27.1	25.4	23.4	−77	−52	**−14**	−8
450	559,056	256,522	85,106.8	30.4	29.2	26.3	−85	−67	−13	**−10**
500	1,771,599	755,158.6	183,804.8	32.1	30.7	28.4	−90	−76	−12	−7
550	5,644,914	2,231,517.6	380,553.6	34.1	32.4	29.4	−93	−83	**−14**	−9
600	18,422,747	6,576,116.8	833,593.6	37.1	34.1	32.2	−95	−87	−13	−6
650	58,690,200.6	19,493,061.4	1,775,916.8	39.4	36.1	34.3	−97	−91	−13	−5
700	184,560,201	58,058,035.2	3,738,227.2	41.3	38.4	36.1	−98	−94	−13	−6
750	587,196,378	169,003,328.4	8,201,318.4	43.8	41.4	39.6	−99	−95	−10	−4
800	1,825,939,422	493,783,689.6	17,065,369.6	45.4	43.2	41.2	−99	−97	−9	−5
850	5,815,657,278	1,449,803,759	35,870,412.8	48.2	45.3	43.3	−99	−98	−10	−4
900	8,591,024,333	3,852,873,000	75,220,172.8	50.4	47.9	45.4	−99	−98	−10	−5
1000	10,566,721,109	5,256,980,789	156,404,940.8	52.4	50.7	47.2	−99	−97	−10	−7
1100	12,542,732,410	6,813,082,979	327,390,003.2	54.4	52.1	49.4	−97	−95	−9	−5
1200	14,483,051,326	8,602,704,239	677,312,921.6	56.8	54.8	51.1	−95	−92	−10	−7
1300	18,771,993,578	14,159,804,022	2,046,518,886	60.1	71.2	77.2	−89	−86	28	8
1400	21,728,744,360	15,771,824,266	4,214,980,608	64.2	73.4	79.5	−81	−73	24	8
1500	24,178,945,769	19,860,856,069	8,493,583,565	66.8	76.1	80.1	−65	−57	20	5

The power consumption readings were the most interesting in this investigation for applications where the energy concern was not confined to total consumption. In mobile and exascale applications, the rate of energy expenditure is also a concern but for different reasons. For example, how quickly batteries drain is critical for mobiles. Optimizing for low power can allow making smaller and hence lighter, more practical batteries. Figure 2 tracks the power consumption that results from using the three methods under investigation as the computations move in the upper memory hierarchy. For sizes that fit in the cache memory regardless of the level, both the divide-and-conquer versions outperformed D3.0,

with the more time-efficient one, D2.4, being consistently better. It consumed up to 14% and 10% less power than D3.0 and D2.8, respectively, or a respectable average of 11% and 6%. At its best, D2.4 showed an average savings 4.817 W at dimensions 500–1200 (relative to D3.0). The trend reversed as D2.4, D2.8, and D3.0 progressively, in that order, moved to the main memory for larger matrix sizes. All cases were estimated to be out of the L3 cache by matrix dimension 1300, with D2.4 and D2.8 exiting sooner due to their cached recursion stacks. Power consumption increased in all cases, as expected. However, D3.0 not only consumed less power but continued to rise steadily, seemingly unaffected by the move out of the cache. It is worth noting that the high-performance CPU circa 2014 managed to hold fully in the cache computations of matrix dimensions 1200–1300, composed of 32-bit elements. It was also astonishing how much power was consumed at relatively moderate multiplication workloads, as much as a household lightbulb.

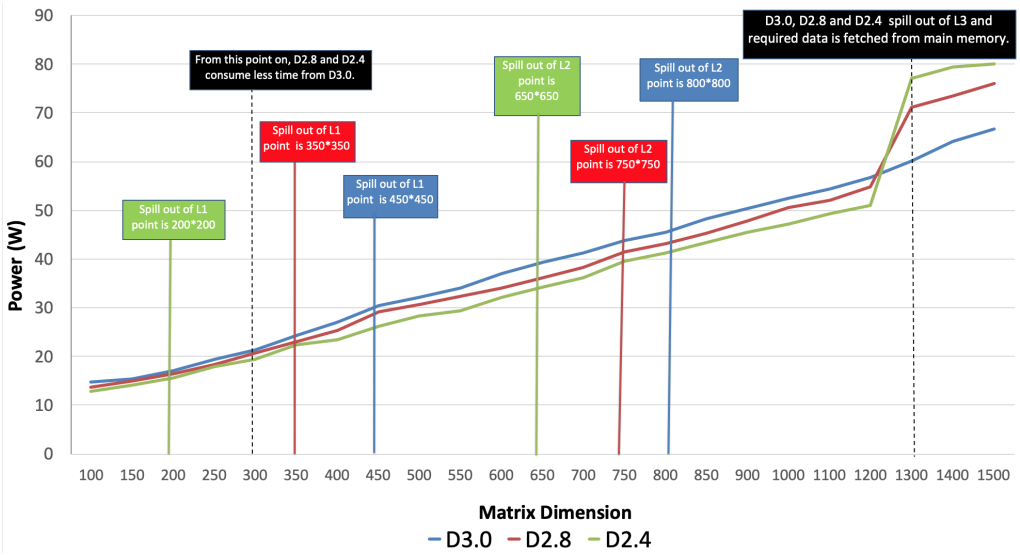

Figure 2. Power consumption in watts (W): estimated boundaries for L1, L2, and L3 caches marked for each case. Note the recursive divide-and-conquer methods spilled earlier due to increased stack storage overheads.

Figure 3 tracks the total energy used at each matrix dimension. It shows that D2.4 consumed much less energy than the computations based on the other two algorithms. The differences were drastic, averaging 73% and 65% better than D3.0 and D2.8, respectively. A closer look in Figure 4 reveals that the trend started early at matrix dimension 250. Both of the computations based on divide-and-conquer were decisively more energy-efficient. It is interesting to point out that energy consumption closely followed the runtime trend (see Figure 5), suggesting a close complexity behavior in each case. A closer look at run times (see Figure 6) confirms that the energy trend of D2.4 indeed started at matrix dimension 250. The figure seems to indicate that time efficiency had more influence on the overall energy consumption than being in the cache or the main memory. The transition to the main memory seemed smooth in all cases.

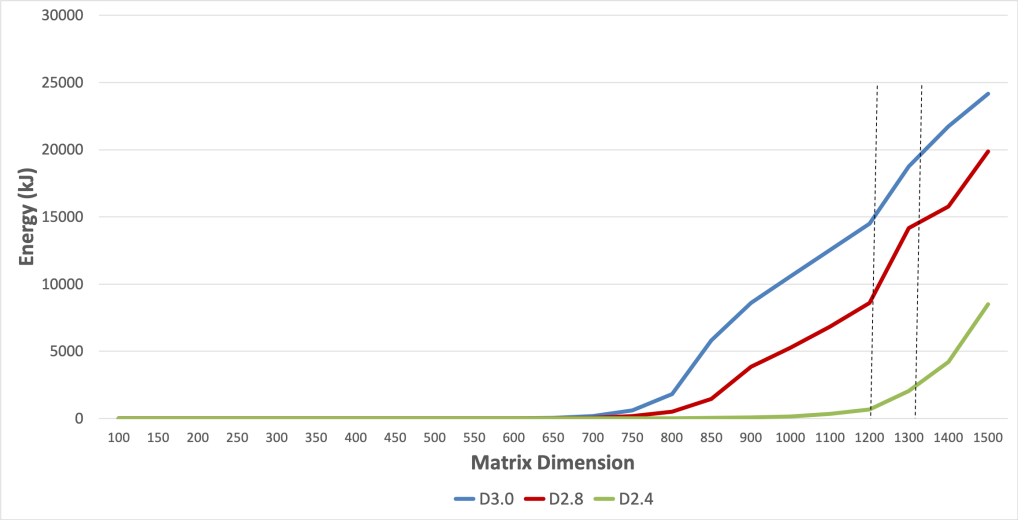

Figure 3. Total energy consumption in kJ, where the estimated points of spill out to main memory are marked.

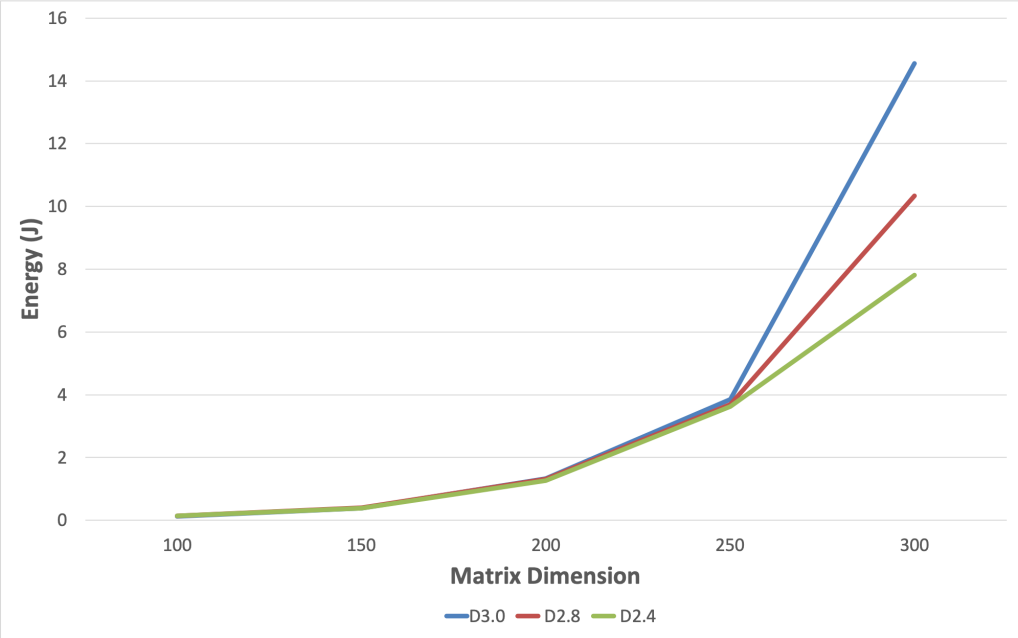

Figure 4. Total energy consumption in joule (J) for small matrix dimensions, where computation is estimated to be within L1 cache.

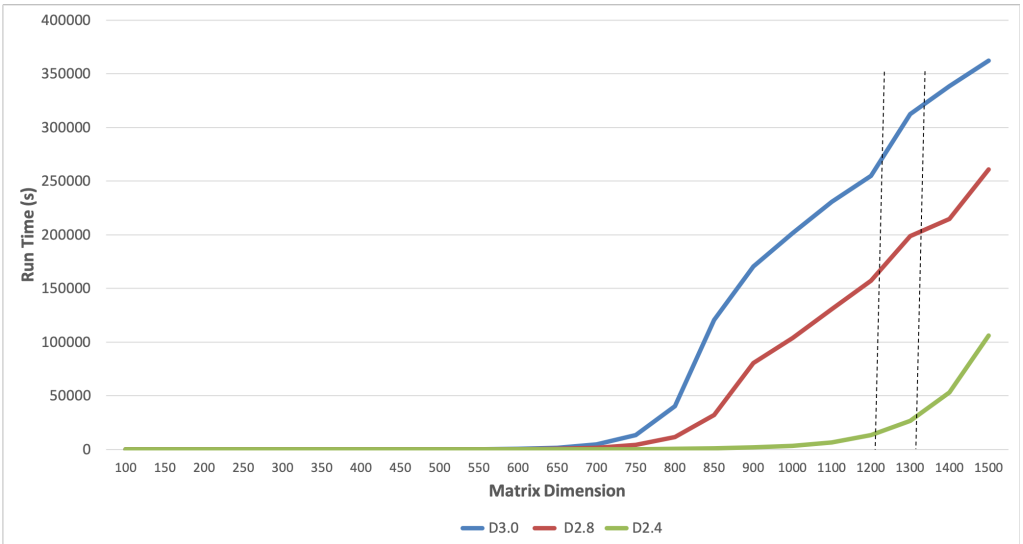

Figure 5. Execution time in seconds. Note the time trend seems to closely follow the total energy consumption.

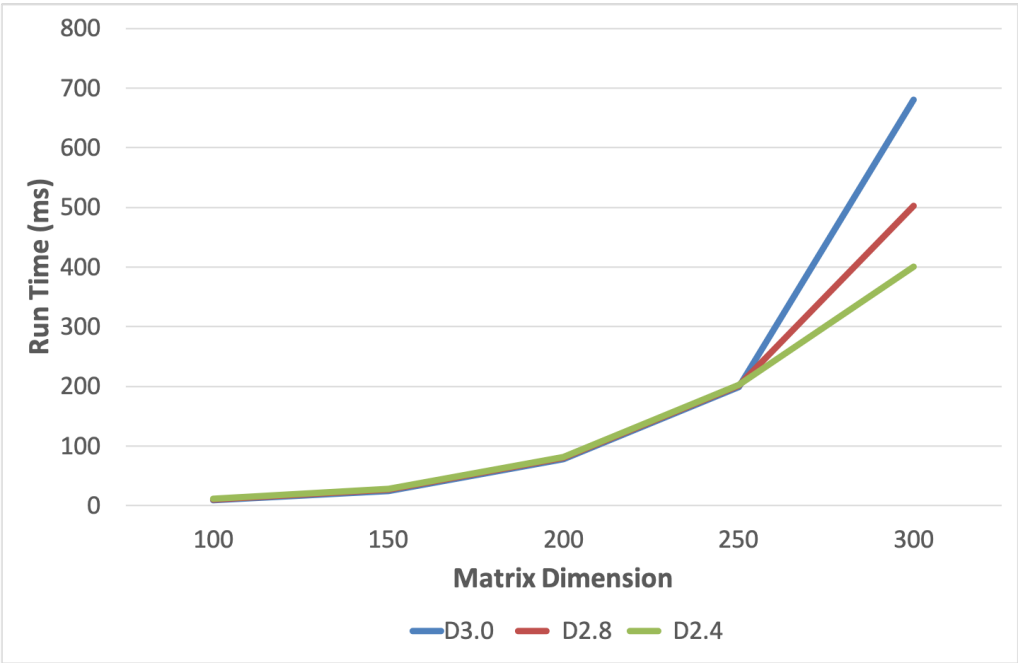

Figure 6. A detailed view of execution time (ms) for small matrix dimensions within estimated L1 cache boundary.

4.1. Miss Rate Analysis

Cache misses were expected to be the principal influence on power and energy characteristics. Thus, cache miss data were obtained to see the effects on power and energy usage. Moreover, the information helped track the computations across the various cache levels through to the main memory. The miss data could also help infer cache level and main memory boundaries. Table 3 shows the numbers of cache miss events reported by the internal counters in the CPU. We remind the reader that the matrix elements and stack frames, in the case of D2.4 and D2.8, were retained in the same cache. Therefore, the different algorithms should be expected to cause their computations to spill to the following level of the storage hierarchy at different matrix dimensions.

Table 3. Average cache miss from reported internal counter data.

Matrix	L1 Misses			L2 Misses			L3 Misses		
Dimension	**D3.0**	**D2.8**	**D2.4**	**D3.0**	**D2.8**	**D2.4**	**D3.0**	**D2.8**	**D2.4**
50	50,641	24,312	21,643	12,471	10,478	8741	24	17	15
100	48,531	24,781	22,314	12,781	10,241	8914	22	19	17
150	53,152	26,140	23,146	13,784	11,364	9246	24	21	19
200	53,941	27,140	24,691	17,425	12,634	10,656	25	23	20
250	54,631	28,631	25,631	18,421	13,847	11,634	27	27	23
300	55,981	29,140	26,147	22,641	14,852	12,647	29	28	24
350	56,910	30,147	27,931	30,145	15,362	14,654	31	30	26
400	57,931	31,651	28,146	33,652	16,324	15,698	33	31	27
450	59,713	32,950	30,147	41,320	17,422	16,874	35	33	28
500	60,235	33,165	31,460	50,361	21,632	21,698	38	34	30
550	75,321	33,714	32,785	55,617	23,547	22,948	41	36	33
600	79,310	34,601	33,147	70,142	29,841	25,478	43	38	34
650	85,312	33,631	33,910	82,156	35,261	33,695	45	39	39
700	87,932	35,489	34,942	83,149	39,475	35,954	46	41	40
750	91,324	36,631	35,147	90,145	44,361	41,658	49	43	41
800	95,312	37,326	36,147	95,961	55,641	50,647	48	46	43
850	98,123	38,971	37,120	97,447	62,145	59,841	50	49	45
900	99,145	39,361	28,147	99,147	70,456	63,587	52	50	47
1000	99,569	42,698	30,958	99,365	72,941	65,941	58	54	56
1100	914,320	714,327	678,910	916,347	578,912	469,820	60	59	58
1200	4,678,940	3,768,453	2,876,453	1,090,657	1,019,876	1,009,765	5698	4698	3548
1300	3,547,931	4,236,941	5,631,740	1,011,649	1,156,941	1,296,148	70,658	82,658	90,568
1400	7,890,147	8,316,740	12,321,945	1,260,478	1,340,658	1,345,964	150,968	192,689	210,658
1500	11,365,741	12,630,948	20,103,941	1,345,968	1,406,157	1,469,123	185,698	245,698	410,698

In particular, Table 3 gives cache misses averaged over multiple runs for each algorithm in each cache level. By carefully noting where the misses suddenly spike, estimating cache and main memory boundaries is possible. According to our analysis, the D3.0 spill point out of L1 was 200, L2 was 650, and L3 was 1300. D2.8 spilled out of L1 a bit earlier at dimensions 350, L2 at 750, and L3 at 1300. For D2.4, the points were L1 at dimensions 450, L2 at 800, and L3 at 1300. By matrix dimension 1300, all three were accessing the main memory. The estimated level boundaries on Figure 2 were made based on the previous analysis.

Hence, D2.4 had the power advantage when the computation was in the on-chip memory. The trend shifted in favor of D3.0 when accessing the off-chip memory (DRAM) at matrix dimension 1300. Only the expensive misses from the L3, which are satisfied from data kept in the main memory, had an impact on the power advantage, strongly indicating the DRAM effect.

Moreover, a closer inspection of the miss data table reveals that in the matrix dimensions from the smallest up to 1200, D3.0 displayed the sharpest increase in the cache miss rate, which could account for why it consumed the most power while there. Conversely, D2.4 had the lowest increase in the cache miss rate, which likely accounted for the lowest power consumption within the various cache levels. After the inflection point, the D3.0 algorithm, which had the highest power consumption when matrix dimensions were small, now has the lowest consumption because it had a lower increase in the cache miss rate. D2.4 had the most power consumption because it showed the highest rise in the cache miss rate. D2.8 sat in between the other two.

4.2. Main Memory Trends

The columns in Table 3 listing misses in the third level of the cache are the most relevant to behaviors related to the main memory. A high increase in misses is noted just before dimension 1300 in D2.4 and D2.8, compared with D3.0, signaling an early spill to DRAM. Subsequently, the earlier trend of power efficiency in the cache reversed in the main memory. Miss data suggest that 1300 was when D3.0 spilled. That shift associated

with the off-chip memory (DRAM) access should not be surprising. According to the literature [16], access to off-chip memory had a substantial impact on power consumption, more than on-chip memory. The divide-and-conquer access memory was heavier due to recursion overheads.

Hence, from matrix dimension 1300 onward, D3.0 commanded a significant edge in power draw over the time-optimized algorithms D2.4 and D2.8. The overall energy consumption, however, was significantly lower for the divide-and-conquer algorithms, signaling the clear advantage gained from their significant time efficiency and reflecting substantially lower runtimes. It was lower still in D2.4, which was the most aggressively time-efficient, suggesting that time efficiency is the dominant factor. From an energy budget viewpoint, D2.4 and D2.8 were the top performers. However, from a power draw perspective, D3.0 is more appealing.

4.3. Algorithm Behavior

The Strassen method and similar divide-and-conquer algorithms for matrix multiplication have significantly better time complexity than a definition-based one. They function as follows: a matrix is recursively broken into smaller submatrices and combined in specific ways to construct a product. The process takes $O(\log n)$ more memory for the bigger stack space [10], which is worse than a simpler nonrecursive D3.0. Generally, divide-and-conquer may be considered more efficient for small matrix sizes because it is faster, and the space overhead may be neglected. However, they are not space efficient for large matrices. Power and energy consumption depend on hardware usage. Hence, from a viewpoint based on hardware usage, divide-and-conquer may not necessarily be regarded as more efficient than D3.0 algorithms, especially for large datasets [31,33]. On the other hand, less runtime should lead to less energy consumption if the power draw (rate) is the same.

The results confirm the memory overhead of D2.8 and D2.4 as computations spill out of lower caches sooner than D.3.0. They show no significant differences for trivially small sizes that fit in the L1 cache. However, the faster D2.4 consumes considerably less energy, all the way through the DRAM. Figure 2 suggests that at least while in the cache, there is more than just lower runtimes behind the energy trend. D2.4 consistently drew less power despite needing more memory to complete, suggesting a more efficient use of the memory. Since the power overheads of the SRAM-based cache are generally similar and may be assumed to be relatively small, efficiency may be attributed to usage patterns that depend on how the computation was systemized. It is not entirely clear if the trend reversal in DRAM was due to nuances of the computations or solely due to the transition to reliance on DRAM.

According to [34], enlarging the cache size reduces those misses due to capacity. Therefore, power consumption should be less, and exit from cache memory would occur later. However, the trend of faster times and increased memory usage would remain the same. Results generally confirm this but also show that the trends are not solely dependent on being in the cache.

5. Conclusions

This study investigated a potential advantage that may be traced back to an algorithm for basic computation in terms of its overall energy cost and how quickly it spends that budget (power being the consumption time rate). The study compared two high-performance matrix multiplication algorithms (D2.8, D2.4) to a definition-based one (D3.0) as a baseline on an HPC-class platform. Misses in each cache level were studied to discern effects on consumption patterns and to examine their impact.

The main finding was an average 11% and 6% power advantage for D2.4 over D3.0 and D2.8, respectively, when the computation was generally within the boundaries of on-chip (SRAM) caches. It shifted just before all the cases spilled out to the off-chip memory (DRAM), where D3.0 gained the advantage. Total energy consumption mainly followed the runtime and seemed unaffected by the memory hierarchy. The most time-efficient,

D2.4, used drastically less energy, even in DRAM, where its power consumption (the rate) was the highest. Therefore, it was significant that the power trend reversal happened in the L3 cache. An effect of the shift to DRAM was evident, but there was a suggestion of a computation-related influence. This remains, however, inconclusive based on this one investigation.

In summary, for large matrix sizes relative to the cache configuration, a D3.0 algorithm could be interesting when the main concern is power, not time. However, D2.4 should be interesting when both power and total energy are the primary concern, particularly for computations that could fit in the caches. In the authors' opinion, the power-energy performance more than makes up for the functional complexity of the algorithm, even when compared to basic divide-and-conquer. Optimizing locality has been long known to result in better execution times. Optimizing the locality should also reduce power consumption, which our findings seem to support. There was no suggestion in this preliminary investigation of good opportunities to trade time for power savings, such as those reported for sorting and searching scenarios [2,4]. Although one could argue for trading runtime for low power by switching to D3.0 in DRAM, the time and total energy costs were too terrible to justify. In this study, there was no interplay between power and time on comparable energy budgets as in those studies. There is evidence to suggest that a larger cache was better for matrix multiplication in terms of power consumption. This could be a crucial design factor for high-performance processors routinely including that computation in their workloads. Finally, the study further highlights the importance of looking at power separately from energy in some cases. Some results may be of interest to mobile devices and exascale systems, where the power in watts (energy draw rate) is crucial. Exascale computing would also benefit from cumulative savings in their massive workloads, especially when the multiplication could fit in the caches and it is a significant fraction of the general workload (as much as 4.8 W average savings were measured for dimensions 500–1200).

The findings from this study should encourage looking into more algorithms, including those previously underutilized due to their time performance or complexity. It could lead to a finer understanding of the factors that affect their energy and power performance or uncover patterns that could help design new ones. Results from different hardware are needed to make a case for an algorithmic power advantage that should be demonstrable independently of hardware or at least for hardware in the same architectural class, particularly investigation of a more modern HPC-class processor platform. It would also be interesting to examine how each algorithm used the memory to understand and optimize the power behavior, perhaps by boosting locality. Future research may also devise a better experimental setup with a minimal system load to eliminate even more sources that contaminate measurements and amplify the signature of code implementing an algorithm. Further investigations along those lines may reveal natural algorithmic power efficiency, i.e., originating in the method.

Author Contributions: Conceptualization, M.S., F.J., N.A., M.A.-H. and O.A.; Data curation, F.J. and N.A.; Formal analysis, F.J., N.A. and M.A.-H.; Funding acquisition, M.A.-H., M.S. and O.A.; Investigation, F.J., N.A., M.A.-H., M.S. and O.A.; Methodology, N.A., M.A.-H. and F.J.; Project administration, N.A., M.A.-H. and O.A.; Resources, M.A.-H., M.S. and O.A.; Software, F.J. and N.A.; Supervision, M.A.-H.; Validation, F.J., N.A. and M.A.-H.; Visualization, F.J., N.A., M.A.-H., M.S. and O.A.; Writing—original draft, F.J. and N.A.; Writing—review and editing, M.A.-H., F.J. and N.A. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: Not applicable.

Conflicts of Interest: The authors declare no conflict of interest.

Abbreviations

The following abbreviations are used in this manuscript:

HPC	High Performance Computing
D3.0	The definition-based matrix multiplication
D2.8	Strassen's divide-and-conquer matrix multiplication
D2.4	An optimized divide-and-conquer matrix multiplication
CPU	Central Processing Unit
GPU	Graphics Processing Unit
EXE	Executable

References

1. Reed, D.A.; Dongarra, J. Exascale computing and big data. *Commun. ACM* **2015**, *58*, 56–68. [CrossRef]
2. Abulnaja, O.A.; Ikram, M.J.; Al-Hashimi, M.A.; Saleh, M.E. Analyzing power and energy efficiency of bitonic mergesort based on performance evaluation. *IEEE Access* **2018**, *6*, 42757–42774. [CrossRef]
3. Aljabri, N.; Al-Hashimi, M.; Saleh, M.; Abulnaja, O. Investigating power efficiency of mergesort. *J. Supercomput.* **2019**, *75*, 6277–6302. [CrossRef]
4. Al-Hashimi, M.; Aljabri, N. Exploring Power Advantage of Binary Search: An Experimental Study. *Int. J. Adv. Comput. Sci. Appl.* **2022**, *13*, 789–795. [CrossRef]
5. Dlamini, G.; Jolha, F.; Kholmatova, Z.; Succi, G. Meta-analytical comparison of energy consumed by two sorting algorithms. *Inf. Sci.* **2022**, *582*, 767–777. [CrossRef]
6. Shi, J.f.; Lin, Z.h.; Wang, J. Optimization of software codes for CPU Chip Reliability. In Proceedings of the 2010 Fifth International Conference on Frontier of Computer Science and Technology, Washington, DC, USA, 18–22 August 2010; IEEE: Piscataway, NJ, USA, 2010; pp. 595–599.
7. Wang, J.; Shi, J.F.; Lin, Z.H. Study on relationship between the usage rate of CPU Chip and its temperature. *Microelectron. Comput.* **2008**, *25*, 45–46.
8. Strassen, V. Gaussian elimination is not optimal. *Numer. Math.* **1969**, *13*, 354–356. [CrossRef]
9. Coopersmith, D.; Winograd, S. Matrix multiplication via arithmetic progressions. In Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing, STOC'87, New York, NY, USA, 25–27 May 1987; pp. 1–6.
10. Dasgupta, S.; Papadimitriou, C.; Vazirani, U. *Algorithms*, 1st ed.; McGraw-Hill Education: New York, NY, USA, 2006.
11. Khan, A.U.; Al-Mouhamed, M.; Fatayer, A.; Mohammad, N. Optimizing the Matrix Multiplication Using Strassen and Winograd Algorithms with Limited Recursions on Many-Core. *Int. J. Parallel Program.* **2016**, *44*, 801–830. [CrossRef]
12. Khan, K.N.; Hirki, M.; Niemi, T.; Nurminen, J.K.; Ou, Z. RAPL in Action: Experiences in Using RAPL for Power Measurements. *ACM Trans. Model. Perform. Eval. Comput. Syst.* **2018**, *3*, 9:1–9:26. [CrossRef]
13. Basmadjian, R.; De Meer, H. Evaluating and modeling power consumption of multi-core processors. In Proceedings of the 2012 Third International Conference on Future Systems: Where Energy, Computing and Communication Meet (e-Energy), Madrid, Spain, 9–11 May 2012; IEEE: Piscataway, NJ, USA, 2012; pp. 1–10.
14. Khezripour, H.; Pourmozaffari, S. Fault Tolerance and Power Consumption Analysis on Chip-Multi Processors Architectures. In Proceedings of the 2012 Seventh International Conference on Availability, Reliability and Security, Washington, DC, USA, 20–24 August 2012; IEEE: Piscataway, NJ, USA, 2012; pp. 301–306.
15. Park, W.H.; Yang, C.K.K. Effects of using advanced cooling systems on the overall power consumption of processors. *IEEE Trans. Very Large Scale Integr. Syst.* **2012**, *21*, 1644–1654. [CrossRef]
16. Al-Hasib, A.; Kjeldsberg, P.G.; Natvig, L. Performance and energy efficiency analysis of data reuse transformation methodology on multicore processor. In Proceedings of the European Conference on Parallel Processing, Rhodes Islands, Greece, 27–31 August 2012; Springer: Berlin/Heidelberg, Germany, 2012; pp. 337–346.
17. Kodaka, T.; Takeda, A.; Sasaki, S.; Yokosawa, A.; Kizu, T.; Tokuyoshi, T.; Xu, H.; Sano, T.; Usui, H.; Tanabe, J.; et al. A near-future prediction method for low power consumption on a many-core processor. In Proceedings of the 2013 Design, Automation & Test in Europe Conference & Exhibition (DATE), Grenoble, France, 18–22 March 2013; IEEE: Piscataway, NJ, USA, 2013; pp. 1058–1059.
18. Dargie, W.; Wen, J. A probabilistic model for estimating the power consumption of processors and network interface cards. In Proceedings of the 2013 12th IEEE International Conference on Trust, Security and Privacy in Computing and Communications, Melbourne, Australia, 16–18 July 2013; IEEE: Piscataway, NJ, USA, 2013; pp. 845–852.
19. Yuechuan, Y.; Guosun, Z.; Chunling, D.; Wei, W. Analysis method of energy for C source program and its application. In Proceedings of the 2013 IEEE International Conference on Green Computing and Communications and IEEE Internet of Things and IEEE Cyber, Physical and Social Computing, Beijing, China, 20–23 August 2013; IEEE: Piscataway, NJ, USA, 2013; pp. 1397–1402.
20. Hamady, F.; Kayssi, A.; Chehab, A.; Mansour, M. Evaluation of low-power computing when operating on subsets of multicore processors. *J. Signal Process. Syst.* **2013**, *70*, 193–208. [CrossRef]
21. Poon, P.; Stout, Q.F. Time-power tradeoffs for sorting on a mesh-connected computer with optical connections. In Proceedings of the 2013 IEEE International Symposium on Parallel & Distributed Processing, Workshops and Phd Forum, Cambridge, MA, USA, 20–24 May 2013; IEEE: Piscataway, NJ, USA, 2013; pp. 611–619.

22. Aliaga, J.I.; Barreda, M.; Dolz, M.F.; Martín, A.F.; Mayo, R.; Quintana-Ortí, E.S. Assessing the impact of the CPU power-saving modes on the task-parallel solution of sparse linear systems. *Clust. Comput.* **2014**, *17*, 1335–1348. [CrossRef]
23. Yildiz, O.; Dorier, M.; Ibrahim, S.; Antoniu, G. A performance and energy analysis of i/o management approaches for exascale systems. In Proceedings of the Sixth International Workshop on Data Intensive Distributed Computing, Vancouver, BC, Canada, 23–27 June 2014; pp. 35–40.
24. Cebrián, J.M.; Sánchez, D.; Aragón, J.L.; Kaxiras, S. Managing power constraints in a single-core scenario through power tokens. *J. Supercomput.* **2014**, *68*, 414–442. [CrossRef]
25. Lastovetsky, A.; Manumachu, R.R. New model-based methods and algorithms for performance and energy optimization of data parallel applications on homogeneous multicore clusters. *IEEE Trans. Parallel Distrib. Syst.* **2016**, *28*, 1119–1133. [CrossRef]
26. Abdel-Hafeez, S.; Gordon-Ross, A. An Efficient O (*N*) Comparison-Free Sorting Algorithm. *IEEE Trans. Very Large Scale Integr. Syst.* **2017**, *25*, 1930–1942. [CrossRef]
27. Gupta, U.; Ayoub, R.; Kishinevsky, M.; Kadjo, D.; Soundararajan, N.; Tursun, U.; Ogras, U.Y. Dynamic power budgeting for mobile systems running graphics workloads. *IEEE Trans.-Multi-Scale Comput. Syst.* **2017**, *4*, 30–40. [CrossRef]
28. Haidar, A.; Jagode, H.; Vaccaro, P.; YarKhan, A.; Tomov, S.; Dongarra, J. Investigating power capping toward energy-efficient scientific applications. *Concurr. Comput. Pract. Exp.* **2019**, *31*, e4485. [CrossRef]
29. Kondo, M.; Miyoshi, I.; Inoue, K.; Miwa, S. Power management framework for post-petascale supercomputers. In *Advanced Software Technologies for Post-Peta Scale Computing*; Springer: Berlin/Heidelberg, Germany, 2019; pp. 249–269.
30. Chandra, T.B.; Verma, P.; Dwivedi, A.K. Impact of programming languages on energy consumption for sorting algorithms. In *Software Engineering*; Springer: Berlin/Heidelberg, Germany, 2019; pp. 93–101.
31. Ozer, G.; Garg, S.; Davoudi, N.; Poerwawinata, G.; Maiterth, M.; Netti, A.; Tafani, D. Towards a Predictive Energy Model for HPC Runtime Systems Using Supervised Learning. In Proceedings of the Euro-Par 2019: Parallel Processing Workshops: Euro-Par 2019 International Workshops, Göttingen, Germany, 26–30 August 2019; Revised Selected Papers; Springer: Berlin/Heidelberg, Germany, 2019; pp. 626–638.
32. Aljabri, N.; Abulnaja, O. Build Power Profiling Tool for Modern CPUs. *JKAU Comp. IT Sci.* **2019**, *8*, 11–18.
33. David, H.; Gorbatov, E.; Hanebutte, U.R.; Khanna, R.; Le, C. RAPL: Memory power estimation and capping. In Proceedings of the 2010 ACM/IEEE International Symposium on Low-Power Electronics and Design (ISLPED), Austin, TX, USA, 18–20 August 2010; IEEE: Piscataway, NJ, USA, 2010; pp. 189–194.
34. Javaid, Q.; Zafar, A.; Awais, M.; Shah, M.A. Cache memory: An analysis on replacement algorithms and optimization techniques. *Mehran Univ. Res. J. Eng. Technol.* **2017**, *36*, 831–840. [CrossRef]