Performance-energy investigation of selected applications using a parallel multi-GPU genetic algorithm under power capping

Filip Magdziak[1][0009−0008−7641−2876] and Paweł Czarnul[1][0000−0002−4918−9196]

Faculty of Electronics, Telecommunications and Informatics,
Gdańsk University of Technology, Narutowicza 11/12, 80-233 Gdańsk, Poland
filip0magdziak@gmail.com

Abstract. In this article we demonstrate performance-energy optimization of multi-GPU genetic algorithm execution using power capping. Firstly, we outline elements concerning the design and implementation of a multi-GPU framework for the execution of a genetic algorithm, allowing the application of a solution to a variety of problems. Secondly, the implementation of three algorithms is proposed and discussed: Traveling Salesman, Knapsack, and Partition. Then, we present a testbed environment with a high-performance computing node with 2 multi-core Intel Xeon CPUs and 8 NVIDIA Quadro RTX 6000 GPUs as well as a Yokogawa WT-310E power meter. Finally, we describe and discuss the optimization results of the implementations using 1, 2, 4, and 8 GPUs under different power caps imposed by NVIDIA NVML. We show the scalability of the solution in terms of fitness versus the number of GPUs used and analyze execution times and energy consumption of various cases under various power caps. We demonstrate that for 8 GPUs, using the power cap of 140W per GPU, we can obtain considerable energy savings of over 17.93% for Traveling Salesman, 15.88% for Knapsack, 21.97% for Partition, with small increases of execution time: 0.89% for Traveling Salesman, 1.41% for Knapsack, and 14.64% for Partition, versus the results for the default power cap of 260W per GPU.

Keywords: CUDA, power capping, multiple GPUs, genetic algorithm, performance-energy optimization

1 Introduction

Nowadays, the reduction of the execution time of applications is possible thanks to the utilization of high-performance computing systems allowing parallel computing [13], at the level of many: nodes, computing devices, and cores [23]. Evolutionary Algorithms (EAs) [12] are frequently used for finding solutions to a variety of problems, especially those for which deriving accurate, analytical models is difficult. At the same time, energy consumption has become a very important factor. In this paper, we contribute[1] by:

[1] The data and code supporting the results of this study are openly available in the GitHub repository at https://github.com/k0n0di0d4/MultiGPU_RP

ICCS Camera Ready Version 2025
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-97554-7_12

- Implementing 3 parallel multi-GPU genetic algorithms Traveling Salesman problem, Knapsack problem, Partition problem.
- Integrating simulation of the former with energy measurements using the professional meter Yokogawa [1].
- Demonstrating multidimensional optimization using various numbers of GPUs and power caps showing the scalability of the solution and potential to obtain higher percentage energy gains than percentage performance losses, using power capping [14,15], compared to the results for the default power cap per GPU.

The article is organized in the following manner. In Section 2 we present selected existing works on parallel implementation of genetic algorithms (GAs) and energy-aware EAs. In Section 3 we present a detailed description of the three different genetic algorithms implemented for the three problems – Traveling Salesman problem, Knapsack problem, and Partition problem. Section 4 details our experimental approach, metrics, results, and analysis of the collected data, highlighting key findings related to the impact of power capping and GPU utilization on algorithm efficiency. Section 5 presents a concise recap of the study's main findings and contributions.

2 Related work

2.1 Overview of Parallel Genetic Algorithms

There are several review/survey papers available in the literature, focusing on various aspects of parallel genetic algorithms (PGAs). In paper [21] authors discuss: master-slave, fine-grained, multiple-population parallel GAs. Hierarchical parallel algorithms are discussed with: a multi-deme GA at the upper level and a fine-grained GA at the lower level; multi-deme and master-slave; or multi-deme GAs at both these levels. Speed-up curves versus demes are outlined. In paper [3] authors introduced the taxonomy of search techniques, including parallel GAs, with coarse and fine-grained parallelism, heterogeneous and homogeneous versions. Paper [10] presents a taxonomy and discussion of PGAs in terms of: (parallel programming) APIs, software: libraries, programming languages, hardware (including GPUs, CPUs, FPGAs, cloud, grid), implementations (global, island, cellular, hybrid), problem domain and applications. In paper [5] authors considered choosing parameters that affect the execution and quality of a PGA – specifically sizing populations, migration rates, and topologies, that can be of interest from the practical point of view. As outlined in the survey paper [6], parallelizing GAs using GPUs poses several important challenges, including: exposing sufficient parallelism, optimization of instruction execution, and memory access. Data layout in memory as well as block and grid configurations are important.

It has been demonstrated that parallelizing GAs or using parallel algorithms with GAs with GPUs can bring considerable speed-ups over the CPU-based implementations, both sequential and parallel. In paper [16], using an island

ICCS Camera Ready Version 2025
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-97554-7_12

model executed on a GPU and a system with an Intel i7-4770K 3.5 GHz CPU, 16 GB memory and NVIDIA Titan Black GPU, running Linux, for the TSP benchmark, acceleration ratios of GAs on the GPU over those on the parallel CPU ranged up to even approx. 160, depending on the population size (512 to 8192 tested) and the number of nodes (16 to 256 tested). The solution presented in paper [4] takes into account a parallel genetic algorithm approach, utilizing GPUs to solve the Traveling Salesman Problem. It focuses on the Lightweight Island Model (LIM) that aims to implement the concept of persistent threads in the island model of the genetic algorithm. The results show that the new approach can increase the speed-up to 27x over the serial CPU and 4.5x over the Traditional Island Model (TIM). To achieve these results authors used an Intel Core i5-10 2.5GHz CPU and Nvidia RTX 3050Ti GPU. Article [11] explores the acceleration of genetic algorithms on GPUs using CUDA to solve the N-Queens problem. It presents a modified genetic algorithm that leverages GPU parallelism by allocating multiple threads per individual. This approach proves that the computational speed can be increased significantly, achieving over 24 times faster results compared to CPU implementations for single-island setups, and substantial improvements in multi-island scenarios. Intel i7 3770 and an Nvidia GTX1060 GPU were used. In paper [24] authors presented an FFEAT library that aimed at simplification of the implementation of parallel EAs using GPUs, based on Torch. The authors used two systems for the comparison of the solution using GPUs and CPUs. The same code implemented with Python and PyTorch was used with different hardware layers. For the GPU tests, a computer with an Intel Xeon Gold 5128 (16 cores/32 threads) with 192 GB RAM and an NVidia Tesla T4 was used. For CPU tests a computer with an AMD Epyc 7452 processor (32 cores/64 threads)and 256 GB of RAM was used. Examining the running time of the GA for the SAT problem with various numbers of literals (100-2000) and different population sizes (32-32768), one can observe that the GPU version requires a large enough population size to offer a smaller execution time. For the population sizes of 2048 and more the GPU version is better by an order of magnitude for this application.

Selected, available research works concerning energy consideration and energy efficiency of the evolutionary algorithms themselves, are summarized below. Authors of paper [19] compared the energy consumption of various tools, including bun version 0.5.8, deno version 1.32.1 and node.js version 18.5.0, for running EAs[2]. They investigated the use of several tools for energy consumption such as pinpoint (using the RAPL interface and NVIDIA registers), perf, and likwid-powermeter. The authors concluded that for EA workloads, bun turned out to be the best in terms of energy consumption and running time; followed by deno, except for a specific case (2048) being better than node.js ranking the third. Paper [7] investigates the energy consumption in the context of batch runs of evolutionary algorithms, focusing on the impact of introducing rest times between runs to mitigate CPU overheating due to hysteretic effects. The study demonstrates that inserting short pauses (e.g., 100 seconds) can reduce energy

[2] https://github.com/JJ/energy-ga-icsoft-2023

ICCS Camera Ready Version 2025
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-97554-7_12

consumption by 5%-8% for various benchmark functions. While the paper focuses on CPU-based implementations, it underscores the broader importance of energy-aware strategies in evolutionary algorithms, aligning with our investigation of power capping on multi-GPU systems.

Further tests using various versions of the solutions [18] indicate that experimentation shall be performed as a valid indicator for given versions of the virtual environments in order to assess their relative capabilities in terms of energy consumption. In paper [9] authors consider power and energy consumption when running GAs. Specifically, they investigate the impact of population size on fitness as well as energy consumed. Two measurement perspectives are considered in the analysis: running the algorithm within a certain time frame (300 seconds assumed in the experiments) and running the algorithm until a given fitness of the number of generations has been reached. LID and ORDER Tree benchmarks were used. Three different platforms were analyzed: a Raspberry Pi 3 Model with a Quad Core 1.2GHz Broadcom BCM2837 CPU and 1GB RAM; a laptop with an Intel i5-2450M 2.5 GHz, 4 cores, and 8GB RAM running Ubuntu 14.04.1 LTS; a desktop PC with an Intel i5-4430 processor at 3 GHz, 4 cores, 32GB RAM running Ubuntu 14.04.1 LTS. Yokogawa WT-310E (laptop and Raspberry Pi) and APPPowerMeter (via Intel RAPL for the PC) were used for energy consumption measurements. The authors found that the function of energy consumption versus population size is non-linear. Additionally, by imposing a time limit they noticed that increasing the population size can result in worse fitness values. Depending on the problem and platform, the best energy consumption and fitness can occur either for the smallest tested population size (256) or in between population sizes. Depending on the configuration, sizes between 25 and 8192 were tested. For the fitness level bound, the best fitness on the PC platform was obtained for the 1024 size and largest 8192 size for LID and ORDER respectively with energy consumption smallest for the smallest population sizes. Additionally, the authors concluded that larger population sizes resulted in higher numbers of cache misses, contributing to energy consumption. In paper [2] authors studied which operations, and to what degree, contribute to the energy consumption of genetic algorithm execution – both sequential GAs and distributed versions (dGAs). For dGAs, both synchronous and asynchronous communication schemes were tested. Algorithms were implemented with C++, with MPI for dGAs. Energy measurements were taken with Intel RAPL. Several benchmark problems were considered, including bit-counting, multimodal problem generator, error-correcting code design, minimum tardy task problem, an instance of MAXSAT, maximum cut of a graph, frequency modulation sounds, and massively multimodal deceptive problem. Tests were averaged over 30 runs using a machine with an Intel Xeon E5-2620 v4, Linux 14.04.5, and 64 GB RAM. The population size in a sequential GA was 100, and the subpopulation in the dGA was 50, 32 islands. For a sequential GA, fitness and genetic operators consumed most of the energy. Mutation was the most energy-requiring component for 4 problems and required more energy than crossover for all problems. For the dGA, the asynchronous version resulted in higher (time and energy) efficiency

ICCS Camera Ready Version 2025
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-97554-7_12

with the optimal energy consumption for the number of islands equal to the number of cores.

2.2 Research Gap and Motivation

Despite significant advancements, existing research primarily focuses on consumer-grade hardware or older HPC systems. Moreover, while some studies explore energy efficiency, they often neglect the trade-offs between performance, accuracy, and actual deployment constraints. Our work aims to bridge this gap by evaluating PGAs on modern HPC clusters, analyzing versatility and energy efficiency across different parallelization strategies, and integrating hardware acceleration techniques refined for recent architectures.

3 Implementation of algorithms

3.1 Algorithm Design

We implemented our framework using CUDA 11.8 and OpenMP 4.5. The population size on each GPU is set to 65,536, and during initialization, it is randomly generated on each GPU using cuRAND [20] the latter also used for mutation. The population evolves over 500 generations, with a migration interval of 5 generations. The islands (GPUs) are interconnected in a circular topology for communication. Each GPU can directly access the memory of its neighboring GPU using CUDA's Peer-to-Peer (P2P) functionality. The communication follows a sequence that progresses from the GPU with the lowest ID to that with the highest ID. The transfer between GPUs is achieved asynchronously using cudaMemcpyAsync, allowing for computation and communication to overlap.

For the Traveling Salesman problem, we have generated a dataset that contains the distances between each pair of cities beforehand. The dataset comprises 100 cities and is saved in a two-dimensional array of size 100 by 100. We allocate a separate thread for each individual in the population to manage its computations. Aiming at the shortest route between cities, we want to minimize the total distance traveled. We employ tournament selection to choose the fitter individual out of two randomly selected members of the population. For the crossover operation, we utilize the one-point crossover method. The mutation comes after the crossover, and there is a 0.01 probability of mutation to occur for each member. If a mutation occurs, a randomly chosen pair of cities will be swapped in a member's DNA.

For the Knapsack problem, we use a pre-generated dataset that consists of 1000 different items with corresponding values and weights. In the code, we maximize the total value of items of a member. However, if the generated member's weight is greater than the knapsack's capacity, the fitness is set to 0. For the selection process, we picked the tournament type, which chooses between 2 members of the tournament, which one is more fit to be sent to the next generation. For the crossover operation, a one-point crossover method is applied. Mutation occurs with a 0.01 probability.

ICCS Camera Ready Version 2025
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-97554-7_12

For the Partition problem, in our dataset, there are 8000 numbers with values ranging from 1 to 300. The objective of our algorithm is to find a solution where the total value of each of the three parts is equal. In the code, we minimize the difference between the largest and smallest partitions' sums.

For each generation, after defining our starting population, the selection phase follows, in which, we pick 5 members of the population to participate in a tournament. Only the participant with the best fitness wins, and will move on to the next generation. Crossover is then performed using the one-point method. The probability of mutation is 0.01. If a member mutates, one value of the member's DNA is chosen, and a partition that does not currently own it becomes the new owner of that value. Since the partition problem has three different parts, if the previous partition that owned it was numbered 0, the algorithm randomly changes its ownership to 1 or 2.

3.2 Pseudo-Code Overview

The PGA for TSP, KP, and PP is summarized in pseudo-code shown in Listing 1.1.

Listing 1.1. Parallel Genetic Algorithm Pseudo-Code

```
# Constants
POPULATION_SIZE = 65536
GENERATIONS = 500
MAX_WEIGHT = 500

FUNCTION genetic_algorithm(algorithm_type, num_gpus, problem_data)
    FOR each GPU in parallel
        INITIALIZE random number generator
        ALLOCATE memory for population and fitness
        COPY problem_data to GPU
        FOR each individual in a population
            IF algorithm_type = TS
                GENERATE random permutation of cities
                APPLY mutation with probability 0.01
                CALCULATE ts_fitness equal to path length
            ELSE IF algorithm_type = KP
                SELECT items randomly until weight <= MAX_WEIGHT
                APPLY mutation with probability 0.01
                CALCULATE kp_fitness as backpack value
            ELSE IF algorithm_type = PP
                ASSIGN values to random partitions
                APPLY mutation with probability 0.01
                CALCULATE pp_fitness as max(partition sums) - min(partition sums)
        END FOR
        COPY population and fitness to backup
        TRACK best fitness (minimize for TS/PP, maximize for KP, generation = 0)
    END FOR
    FOR generation = 1 to GENERATIONS
        FOR each GPU in parallel
            FOR i = 1 to POPULATION_SIZE / 2
                SELECT two parents via tournament selection (size 5)
                PERFORM one-point crossover to produce two offspring with recalculated fitness
                APPLY mutation with probability 0.01
            END FOR
            IF generation % 5 = 0
                EXCHANGE half of a population with neighboring GPU
            TRACK the best fitness (minimize for TS/PP, maximize for KP, generation)
            UPDATE population and fitness backup
        END FOR
    END FOR
    COLLECT best fitness from all GPUs
    WRITE results to file
    FREE GPU memory
END
```

ICCS Camera Ready Version 2025
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-97554-7_12

4 Experiments

4.1 Methodology

In our experimental methodology, we establish a fixed maximum of 500 generations per trial, regardless of the GPU count involved. Each GPU employs 512 blocks with 128 threads per block, totaling 65,536 threads. This approach ensures consistency across all tests and allows for a fair comparison of results. We assess the effectiveness of our configurations by monitoring the improvement in fitness over and after the elapsed generations. Given the variability of genetic algorithms, we conduct 100 runs for each configuration to account for the stochastic nature of the process, and note the average outcome. We run all configurations for various numbers of GPUs available for the same number of generations. For consistency, in the following figures and the analysis, the fitness values for the traveling salesman and partition problem are presented as `1/ts_fitness` and `1/pp_fitness` while for the knapsack problem fitness is shown as `kp_fitness` – to align with the convention of higher values indicating better solutions (see Listing 1.1, `pp_fitness` was greater than 0 in simulations). Consequently, we expect that average and median fitness values will grow with an increasing number of GPUs, justifying parallelization. Apart from the fitness, we assess the quality of each configuration also by taking into consideration:

- Average energy spent per run – expecting an increase for a larger number of GPUs involved.
- Average execution time – expecting slightly larger execution times for a larger number of GPUs but only due to synchronization/communication costs. In the latter aspect, the computational time for each GPU shall be the same.

Additionally, we assess all of the aforementioned metrics: fitness, energy, and time for various power caps, applied on the GPU(s) involved in computations, to see which power cap setting would yield more desired trade-off of fitness, energy, and time. In case more than 1 GPU is involved, the same power cap is applied for all the GPUs used. Specifically, we are interested in: verification of the speed-up, i.e., an increase in fitness versus the number of GPUs, impact of power capping on energy consumption, impact of power capping on execution time, searching for a configuration resulting from power capping that provides larger energy gains percentage-wise than execution time increase percentage-wise, for the best performing configuration, presumably using 8 GPUs.
We shall note that in the paper we are intentionally focusing on the exploration of the aforementioned metrics with fixed GA parameters such as population size, probability of mutation, crossover method, etc.

4.2 Testbed environment

For our experiments, we used a high-performance computing server, with RAM having a total size of 394 GB, featuring two Intel Xeon 4210 CPUs. The server hosts 8 NVIDIA Quadro RTX 6000 GPUs.

ICCS Camera Ready Version 2025
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-97554-7_12

To measure the energy consumption [8] accurately, we utilized a professional grade Yokogawa WT-310E power meter, which has a power measurement accuracy of 0.1% of the value reading, and a sampling rate of 100kS/s (100000 samples per second). We used yokotool which allowed us to collect time and power usage during the run, of the whole machine. The data was measured at a frequency of 10Hz allowing for an accurate measurement of power and conversion to energy. The aggregated power was averaged by dividing by the total number of trials which amounted to 100.

4.3 Results

For the benchmarking runs, the sizes of the given problems are as follows. The traveling salesman problem consists of 100 cities, with distances between cities ranging from 5 to 300. Each city has stored the distance to every other city, and if a city's index matches the index of the city in the saved distances, the distance is 0, reflecting that there is no distance to itself. The knapsack problem is initialized with 1000 items, where each item has its own randomly assigned value and weight. The values are drawn from a uniform distribution within the range of 1 to 100, while the weights are similarly chosen between 1 and 50. The Partition problem comprises 8000 numbers, ranging from 1 to 300. The data for each problem consists of integers and was pre-generated and saved in a separate array using the Mersenne Twister (MT19937)[17] pseudo-random number generator.

Figures 1 through 3 present the average fitness of the genetic algorithm for the Traveling Salesman, Knapsack, and Partition problems using specified power caps of 100W, 140W, 180W, 220W, and 260W respectively, across GPU counts of 1, 2, 4, or 8. Values shown in the figures denote proper median values for the given number of GPUs used along with boxplots and outliers.

Figures 4 through 6 depicts average energy (along with standard deviation) spent per run by genetic algorithm for the Traveling Salesman, Knapsack, and Partition problems using specified power caps of 100W, 140W, 180W, 220W, and 260W respectively, across GPU counts of 1, 2, 4, or 8.

Figures 7 through 9 display average time (along with standard deviation) spent per run by genetic algorithm for Traveling Salesman, Knapsack, and Partition problems using specified power caps of 100W, 140W, 180W, 220W, and 260W respectively, across GPU counts of 1, 2, 4, or 8.

ICCS Camera Ready Version 2025
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-97554-7_12

Fig. 1. Fitness for Traveling Salesman problem with power capped at 100W, 140W, 180W, 220W, and 260W, median values are shown under the bars.

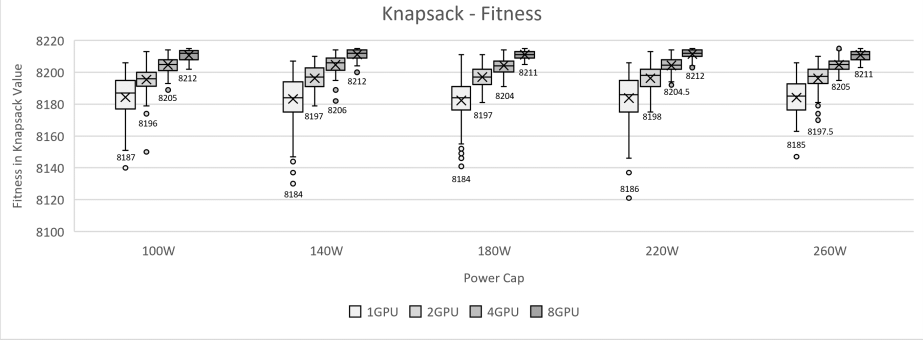

Fig. 2. Fitness for Knapsack problem with power capped at 100W, 140W, 180W, 220W, and 260W, median values are shown under the bars.

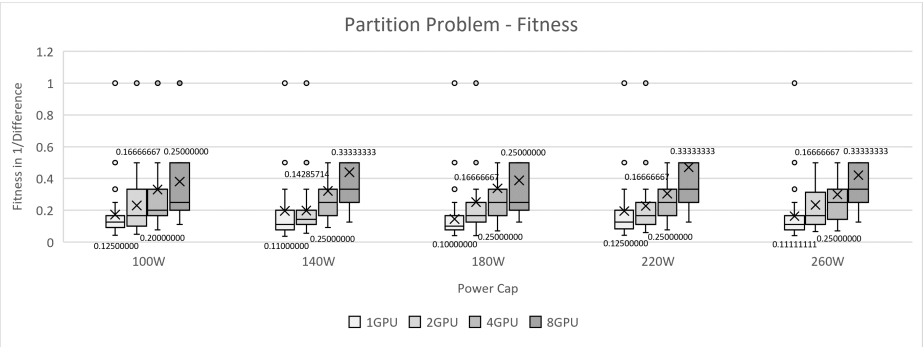

Fig. 3. Fitness for Partition problem with power capped at 100W, 140W, 180W, 220W, and 260W, median values are shown under the bars.

ICCS Camera Ready Version 2025
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-97554-7_12

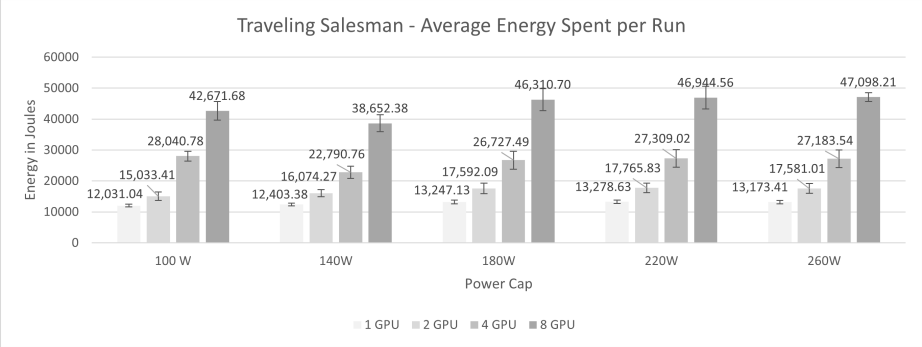

Fig. 4. Total energy spent for Traveling Salesman problem with power capped at 100W, 140W, 180W, 220W, and 260W

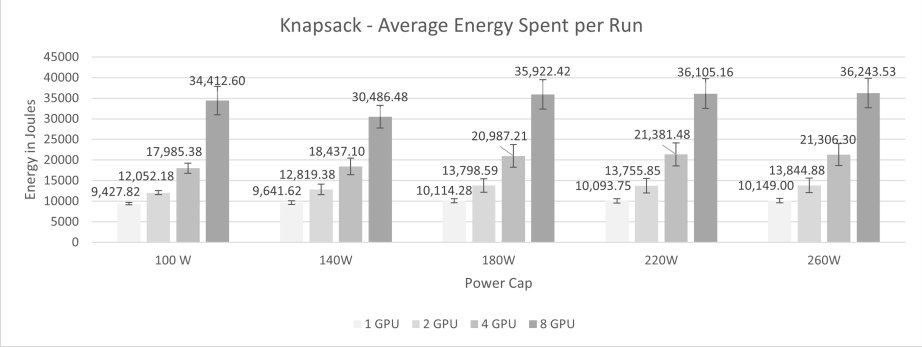

Fig. 5. Total energy spent for Knapsack problem with power capped at 100W, 140W, 180W, 220W, and 260W

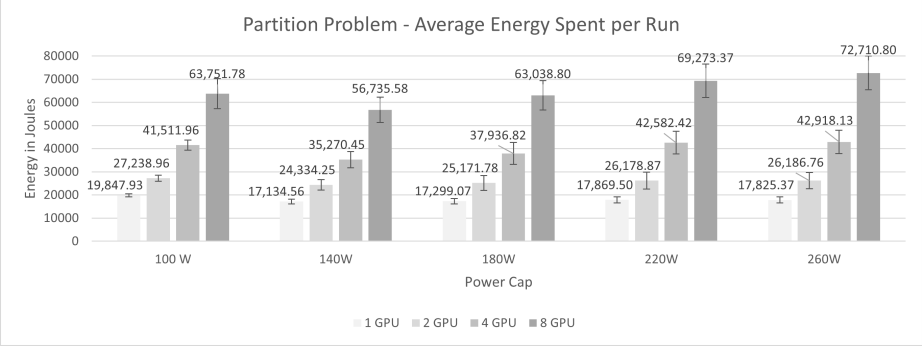

Fig. 6. Total energy spent for Partition problem with power capped at 100W, 140W, 180W, 220W, and 260W

ICCS Camera Ready Version 2025
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-97554-7_12

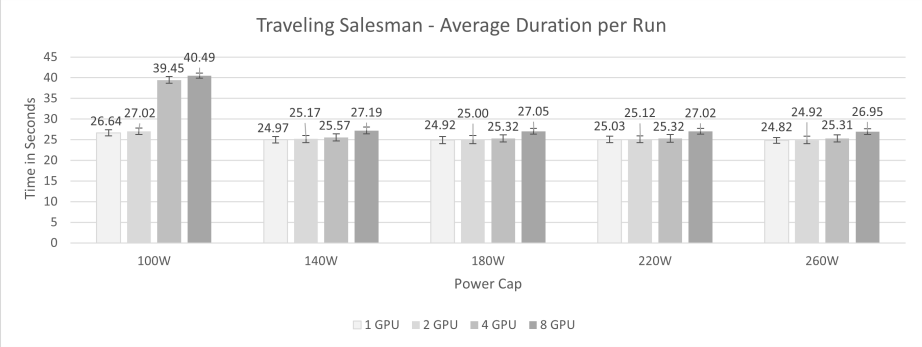

Fig. 7. Total time spent for the execution of Traveling Salesman problem with power capped at 100W, 140W, 180W, 220W, and 260W

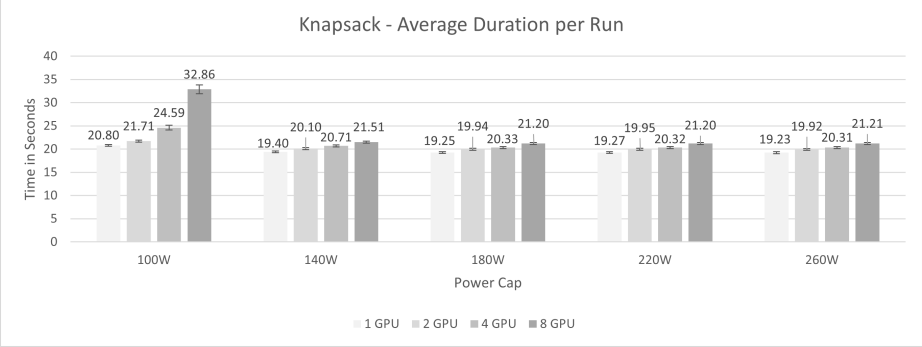

Fig. 8. Total time spent for the execution of Knapsack problem with power capped at 100W, 140W, 180W, 220W, and 260W

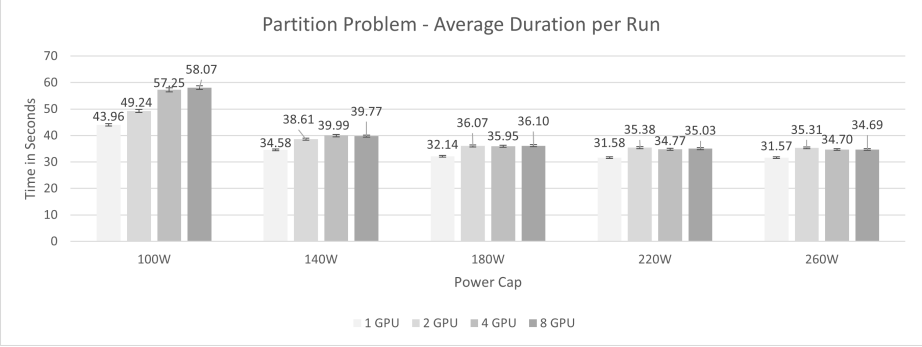

Fig. 9. Total time spent for the execution of Partition problem with power capped at 100W, 140W, 180W, 220W, and 260W

ICCS Camera Ready Version 2025
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-97554-7_12

4.4 Discussion

In terms of fitness, we see that fitness levels for a particular number of GPUs stay at similar levels for any of the power cap settings. This is expected, as power capping settings shall affect execution performance rather than affect changes in the genetic algorithm although it might affect e.g. ratio of host-device or device-device communication to computation times. For the desired configuration with 8 GPUs (because it allows the best fitness levels), differences between medians across all power caps tested are: for the Traveling Salesman (TS) between 0.00102669 and 0.00106895, for the Knapsack Problem (KP) between 8211 and 8212 and for the Partition Problem (PP) between 0.25000000 and 0.33333333. These minor variations in fitness values are primarily attributed to the inherent stochasticity of the genetic algorithm, such as random initialization and mutation, which are slightly amplified by power capping's effect on inter-GPU communication timing. This confirms that power capping primarily impacts execution time and energy consumption, with minimal influence on fitness performance. We note that for the latter a difference of just 1 in the sums of partitions causes a reasonably large percentage difference in fitness due to the construction of the fitness function mentioned before. For the latter, we can also see that the upper bound of the boxes reaches the same level of 0.5 for all power caps.

At the same time, we see improvements when engaging more GPUs for the same number of generations, for each of the algorithms. Specifically, average improvements versus 1 GPU are, comparing average fitness across all power caps tested for a given number of GPUs (gains for different power cap values are listed in the parentheses), for:

- TS: 10.55% for 2 GPUs (9.29, 12.78, 8.85, 9.54 and 12.28%), 18.63% for 4 GPUs (17.02, 21.08, 15.67, 19.32 and 20.04%) and 24.94% for 8 GPUs (26.33, 24.13, 25.49, 22.09 and 26.65%);
- KP: 0.15% for 2 GPUs (0.11, 0.16, 0.16, 0.15 and 0.15%), 0.24% for 4 GPUs (0.22, 0.27, 0.24, 0.23 and 0.24%) and 0.32% for 8 GPUs (0.31, 0.34, 0.33, 0.32 and 0.32%) – note that the relatively small percentage numbers result from an already high absolute value of fitness for 1 GPU but there is very consistent improvement from a larger number of GPUs;
- PP: 42.64% for 2 GPUs (33.33, 29.87, 66.67, 33.33 and 50%), 112.45% for 4 GPUs (60, 127.27, 150, 100 and 125%), and 163.94% for 8 GPUs (100, 203.03, 150, 166.67 and 200%).

We shall note that in all cases an increase in fitness with an increasing number of GPUs is clearly visible and a relative increase depends on how the fitness is defined and what the absolute level of the reference fitness for 1 GPU is.

Power capping does affect execution times and energy consumption values. We see that for power caps between 140W and 260W, for each application there is a slight but similar growth in execution time between 1 and 8 GPUs. This is expected and results from the additional inter-GPU communication times for a larger number of GPUs. However, percentage increases are different for various algorithms which results from various compute to communication times.

ICCS Camera Ready Version 2025
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-97554-7_12

Additionally, a different behavior is observed for the power cap of 100W – here we see much larger growth in time for 4-8 GPUs, due to the apparent slowing down of the communication.

When it comes to energy consumption, we see growing energy consumption for an increasing number of GPUs for each application and each power cap, due to higher power consumption of more engaged GPUs and slightly increased execution time because of increased communication due to a larger number of GPUs. What is important is that, especially for 8 GPUs, across power caps tested, we see the smallest energy consumption for the power cap of 140W which is a very valuable, practical result. This indicates that this setting shall be preferred for energy consumption minimization with 17.93% for TS, 15.88% for KP, 21.97% for PP, compared to the power cap of 260W. This comes at the cost of only 0.89%, 1.41%, and 14.64% penalty of execution time, compared to 260W and 8 GPUs, for TS, KP, and PP respectively. Again, this proves to be a very valuable result because it allows us to achieve considerable energy savings at a small and percentage-wise much smaller increase in execution time.

5 Summary and future work

In this article, we presented a comprehensive study on the implementation and performance-energy analysis of a multi-GPU genetic algorithm framework, focusing on the impact of power capping. Our investigation included the implementation and evaluation of three distinct problems: the Traveling Salesman Problem, the Knapsack Problem, and the Partition Problem. The experimental setup consisted of a high-performance computing node equipped with dual Intel Xeon CPUs and eight NVIDIA Quadro RTX 6000 GPUs, paired with a Yokogawa WT-310E power meter for precise energy consumption measurements. Our findings showed that including more GPUs led to improved fitness levels of the algorithms, indicating enhanced solution quality with increased parallelism. We demonstrated that the application of non-default power cap settings affected execution times and energy consumption without significantly affecting the algorithm's fitness values. Notably, a power cap of 140W (versus the default power cap of 260W per GPU) in combination with 8 GPUs emerged as the best setting in terms of energy efficiency. This configuration provided a balanced compromise, achieving considerable energy savings with minimal impact on execution time and algorithm performance. These results emphasize the importance of power capping as a valuable tool for optimization. Looking forward, we propose several paths for future research. Scaling the framework to accommodate a larger number of GPUs through adding more nodes, preferably based on existing frameworks such as KernelHive [22], could further clarify the scalability and efficiency of parallel genetic algorithms. The other path we could take could be applying the framework to other real-time and complex problems in fields such as bioinformatics could show the practical utility and versatility of our framework. Additionally, extended power capping aware analysis could include consideration

ICCS Camera Ready Version 2025
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-97554-7_12

of more dimensions including various parameters such as population sizes, GPU grid sizes, etc., per implementation of a given problem.

Acknowledgments

The research was supported in part by the project "Cloud Artificial Intelligence Service Engineering (CAISE) platform to create universal and smart services for various application areas", No. KPOD.05.10-IW.10-0005/24, as part of the European IPCEI-CIS program, financed by NRRP (National Recovery and Resilience Plan) funds.

References

1. WT310E Digital Power Meter User's Manual
2. Abdelhafez, A., Alba, E., Luque, G.: A component-based study of energy consumption for sequential and parallel genetic algorithms. J. Supercomput. **75**(10), 6194–6219 (oct 2019). https://doi.org/10.1007/s11227-019-02843-4
3. Alba, E., Troya, J.M.: A survey of parallel distributed genetic algorithms. Complex. **4**(4), 31–52 (mar 1999)
4. Alraslan, M., Alkurdi, A.H.: A lightweight island model for the genetic algorithm over gpgpu. International Journal of Electrical and Computer Engineering Systems **14**(7), 753 – 763 (2023). https://doi.org/10.32985/ijeces.14.7.3
5. Cantú-Paz, E., Goldberg, D.E.: Efficient parallel genetic algorithms: theory and practice. Computer Methods in Applied Mechanics and Engineering **186**(2), 221–238 (2000). https://doi.org/https://doi.org/10.1016/S0045-7825(99)00385-0
6. Cheng, J.R., Gen, M.: Accelerating genetic algorithms with gpu computing: A selective overview. Computers Industrial Engineering **128**, 514–525 (2019). https://doi.org/https://doi.org/10.1016/j.cie.2018.12.067
7. Cotta, C., Martínez-Cruz, J.: Energy consumption analysis of batch runs of evolutionary algorithms. In: Proceedings of the Genetic and Evolutionary Computation Conference Companion. p. 87–88. GECCO '24 Companion, Association for Computing Machinery, New York, NY, USA (2024). https://doi.org/10.1145/3638530.3664093
8. Czarnul, P., Proficz, J., Krzywaniak, A.: Energy-aware high-performance computing: Survey of state-of-the-art tools, techniques, and environments. Scientific Programming **2019**, 8348791 (Apr 2019). https://doi.org/10.1155/2019/8348791
9. Díaz-Álvarez, J., Castillo, P.A., de Vega, F.F., Chávez, F., Alvarado, J.: Population size influence on the energy consumption of genetic programming. Measurement and Control **55**(1-2), 102–115 (2022). https://doi.org/10.1177/00202940211064471
10. Harada, T., Alba, E.: Parallel genetic algorithms: A useful survey. ACM Comput. Surv. **53**(4) (aug 2020). https://doi.org/10.1145/3400031
11. Janssen, D.M., Liew, A.W.C.: Acceleration of genetic algorithm on gpu cuda platform. In: 2019 20th International Conference on Parallel and Distributed Computing, Applications and Technologies (PDCAT). pp. 208–213 (2019). https://doi.org/10.1109/PDCAT46702.2019.00047

ICCS Camera Ready Version 2025
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-97554-7_12

12. Katoch, S., Chauhan, S.S., Kumar, V.: A review on genetic algorithm: past, present, and future. Multimedia Tools and Applications **80**(5), 8091–8126 (Feb 2021). https://doi.org/10.1007/s11042-020-10139-6
13. Krzywaniak, A., Czarnul, P.: Parallelization of selected algorithms on multi-core cpus, a cluster and in a hybrid cpu+xeon phi environment. In: Borzemski, L., Świątek, J., Wilimowska, Z. (eds.) Information Systems Architecture and Technology: Proceedings of 38th International Conference on Information Systems Architecture and Technology – ISAT 2017. pp. 292–301. Springer International Publishing, Cham (2018)
14. Krzywaniak, A., Czarnul, P.: Performance/energy aware optimization of parallel applications on gpus under power capping. In: Wyrzykowski, R., Deelman, E., Dongarra, J., Karczewski, K. (eds.) Parallel Processing and Applied Mathematics. pp. 123–133. Springer International Publishing, Cham (2020)
15. Krzywaniak, A., Czarnul, P., Proficz, J.: Dynamic gpu power capping with online performance tracing for energy efficient gpu computing using depo tool. Future Generation Computer Systems **145**, 396–414 (2023). https://doi.org/https://doi.org/10.1016/j.future.2023.03.041
16. Li, C.C., Lin, C.H., Liu, J.C.: Parallel genetic algorithms on the graphics processing units using island model and simulated annealing. Advances in Mechanical Engineering **9**(7), 1687814017707413 (2017). https://doi.org/10.1177/1687814017707413
17. Matsumoto, M., Nishimura, T.: Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator. ACM Trans. Model. Comput. Simul. **8**(1), 3–30 (Jan 1998). https://doi.org/10.1145/272991.272995
18. Merelo-Guervós, J.J., García-Valdez, M., Castillo, P.A.: Energy consumption of evolutionary algorithms in javascript. In: Villani, M., Cagnoni, S., Serra, R. (eds.) Artificial Life and Evolutionary Computation. pp. 3–15. Springer Nature Switzerland, Cham (2024)
19. Merelo-Guervós, J.J., García-Valdez, M., Castillo, P.: An analysis of energy consumption of javascript interpreters with evolutionary algorithm workloads. In: Proceedings of the 18th International Conference on Software Technologies - Volume 1: ICSOFT. pp. 175–184. INSTICC, SciTePress (2023). https://doi.org/10.5220/0012128100003538
20. NVIDIA Corporation: curand - cuda random number generation library. NVIDIA Developer Documentation, https://developer.nvidia.com/curand
21. Paz, E.C.: A survey of parallel genetic algorithms. Calculateurs Paralleles, Reseaux et Systems Repartis **10**(2), 141–171 (1998)
22. Rościszewski, P., Czarnul, P., Lewandowski, R., Schally-Kacprzak, M.: Kernelhive: a new workflow-based framework for multilevel high performance computing using clusters and workstations with cpus and gpus. Concurrency and Computation: Practice and Experience **28**(9), 2586–2607 (2016). https://doi.org/https://doi.org/10.1002/cpe.3719
23. Sato, Y., Hasegawa, N., Sato, M.: Gpu acceleration for sudoku solution with genetic operations. In: 2011 IEEE Congress of Evolutionary Computation (CEC). pp. 296–303 (June 2011). https://doi.org/10.1109/CEC.2011.5949632
24. Valkovič, P., Pilát, M.: Implementing and evaluating parallel evolutionary algorithms in modern gpu computing libraries. In: Proceedings of the Genetic and Evolutionary Computation Conference Companion. p. 506–509. GECCO '22, Association for Computing Machinery, New York, NY, USA (2022). https://doi.org/10.1145/3520304.3529000

ICCS Camera Ready Version 2025
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-97554-7_12