Evaluating the Energy Efficiency of OpenCL-accelerated AutoDock Molecular Docking

Leonardo Solis-Vasquez*, Diogo Santos-Martins†, Andreas Koch*, and Stefano Forli†
*Embedded Systems and Applications Group
Technische Universität Darmstadt, Darmstadt, Germany
Email: {solis, koch}@esa.tu-darmstadt.de
†Center for Computational Structural Biology
The Scripps Research Institute, La Jolla, CA, United States
Email: {diogom, forli}@scripps.edu

***Abstract*—AUTODOCK is a molecular docking application that consists of a genetic algorithm coupled with the Solis-Wets local-search method. Despite its wide usage, its power consumption on heterogeneous systems has not been evaluated extensively. In this work, we evaluate the energy efficiency of an OpenCL-accelerated version of AUTODOCK that, along with the traditional Solis-Wets method, newly incorporates the ADADELTA gradient-based local search. Executions on a Nvidia V100 GPU yielded energy efficiency improvements of up to 297x (Solis-Wets) and 137x (ADADELTA) with respect to the original AUTODOCK baseline.**

***Index Terms*—Energy efficiency, power profiling, OpenCL, molecular docking, AutoDock, gradients**

I. INTRODUCTION

Energy efficiency is becoming increasingly important in the design of computer systems. In fact, the development of future systems (e.g., exascale) will be constrained by their power consumption [1]. As indicated in [2], [3], the upcoming trend is the replacement of architectures based on homogeneous CPUs with heterogeneous accelerators. This emerging scenario has motivated large investments in accelerator technologies by cloud and data center companies (e.g., Amazon, Microsoft, etc), as well as porting efforts of scientific applications to heterogeneous High Performance Computing (HPC) [4], [5].

Molecular docking (MD) is a widely-used computational method in HPC. Basically, it aims to predict the interaction between a small molecule (ligand) and a macromolecular target (receptor). One of the most popular MD tools is AUTODOCK [6], [7]. The embarrasing parallelism from its genetic-algorithm engine has made it suitable for different acceleration approaches [8]–[10]. Concurrently to efforts to speed-up MD processing times, several studies have been aiming to improve the MD quality of results by using more efficient search methods [7]. One of these are local-search methods based on gradients that have enabled significant enhancements over traditional ones [11], [12].

In a previous study [13], we incorporated the gradient-based ADADELTA method [14] as local search into an OpenCL-accelerated version of AUTODOCK. Besides achieving high speedups (max. 399x on GPUs) with respect to the AUTODOCK baseline, our work demonstrated that ADADELTA is able to find chemically stronger ligand-receptor systems, and thus produces higher-quality MD results than the legacy Solis-Wets method. In order to efficiently deploy a parallel application on HPC systems, it is important to understand – besides its performance gains – its power draw, and to quantify the energy gains achieved by hardware acceleration. This becomes even more important, when the application is used at scale, such as the massive drug discovery use-case MD is often used in practice.

To this end, this work extends our previous studies [9], [13] by providing a detailed energy-consumption analysis of our OpenCL-AUTODOCK implementation on different devices. Concretely, this paper makes the following contributions:

1) Investigation of correlation between power consumption and performance of OpenCL-AUTODOCK, comparing the Solis-Wets and ADADELTA local-search methods.
2) Evaluation of the impact of the molecular complexity on the energy efficiency and execution performance on recent GPU- and CPU-based accelerator devices.

II. AUTODOCK MOLECULAR DOCKING

In AUTODOCK [15], ligand-receptor interactions are described by the spatial arrangement or pose adopted by the flexible ligand upon binding on a given region on the rigid receptor. Such a pose is encoded with translational (x, y, z), rigid-body rotation (ϕ, θ, α), and N_{rot} rotatable-bond (ψ_1, …, $\psi_{N_{\text{rot}}}$) variables. Each pose is quantified with a score, which represents the binding energy[1]. MD simulations produce solutions Ω (poses) aiming for optimized scores.

A. Lamarckian Genetic Algorithm

AUTODOCK performs a series of independent Lamarckian Genetic Algorithm (LGA) runs (Algorithm 1), each combining two methods: a genetic algorithm (GA), followed by a local search (LS). The GA generates populations of solutions via genetic operations (e.g., crossover, mutation, selection), while LS performs a local optimization on a population subset. Each solution is encoded as a `genotype`, which is composed

[1]Binding energy (kcal/mol) refers to the strength of the molecular interaction, and should not be confused with the compute energy (J).

of N_{genes} genes (i.e., x, y, z, ϕ, θ, α, ψ_1, ..., $\psi_{N_{\text{rot}}}$). The LGA-run duration is determined by the current number of score evaluations and generations, i.e., an LGA-run terminates when any of its limits is reached, which are by default: $N^{\text{MAX}}_{\text{score-evals}} = 2\,500\,000$, and $N^{\text{MAX}}_{\text{gens}} = 27\,000$.

Algorithm 1: Lamarckian Genetic Algorithm (LGA)

```
Function AutoDock
    /* High-Level Parallelism */
    for each LGA-run do
        while (N_score-evals < N^MAX_score-evals) and (N_gens < N^MAX_gens) do
            /* Medium-Level Parallelism */
            GA (population)
            /* Medium-Level Parallelism */
            for solution in random-subset (population) do
                LS (get-genotype (solution))
```

The scoring function (SF) calculates the binding energy of a given pose. Its computation (Algorithm 2) is composed of:

- A pose calculation, that inputs a solution (`genotype`) and outputs a set of three-dimensional coordinates for all ligand atoms, iterating over all $N_{\text{pose-rot}}$ rotation items.
- Inter (ligand-receptor) and intramolecular (ligand-ligand) interactions that iterate over all N_{atom} ligand atoms, and all $N_{\text{intra-contrib}}$ intramolecular contributors, respectively.

Algorithm 2: Scoring Function (SF)

```
/* Low-Level Parallelism */
Function SF (genotype)
    for each rot-item in N_pose-rot do
        PoseCalculation
    for each lig-atom in N_atom do
        InterInteraction
    for each intra-pair in N_intra-contrib do
        IntraInteraction
```

B. Local Search methods

After GA has processed the entire population of solutions, LS optimizes a random population-subset (default: 6%). In AUTODOCK, the LS method traditionally employed is that of **Solis-Wets** (SW) [16], which generates new solutions either by adding or subtracting small variations to the initial value. New solutions are stored if their scores are lower (better) than those of the initial solution. In contrast to Solis-Wets, the alternative **ADADELTA** (AD) utilizes a more complex update-rule based on *gradients* of the SF as well as a *history* of past gradient and update vectors [14].

The gradient calculation (GC) in ADADELTA is derived from SF, and as such, both functions share a similar code structure. Basically, GC (Algorithm 3) starts computing the pose exactly as in SF. Then, the gradients of inter- and intramolecular components are calculated with respect to the variables representing the three-dimensional coordinates of all ligand atoms. At this point, the gradients are expressed in the atomic space. However, in order to incorporate the GC into ADADELTA local search, gradients must be expressed in the genetic space. Therefore, additional conversion functions, specific to each gene type (translational, rigid-body rotation, and rotatable bonds) are required. Such conversion relies on mathematical operations described in detail in [13].

Algorithm 3: Gradient Calculation (GC)

```
/* Low-Level Parallelism */
Function GC (genotype)
    /* Gradients in atomic space */
    for each rot-item in N_pose-rot do
        PoseCalculation
    for each lig-atom in N_atom do
        InterGradient
    for each intra-pair in N_intra-contrib do
        IntraGradient
    /* Conversion into genetic space */
    Gtrans // Translational gradients
    Grigidrot // Rigid-body rotation gradients
    Grotbond // Rotatable-bond gradients
```

C. OpenCL parallelization

Our data-based parallelization assigns AUTODOCK functions (GA, LS, solutions) to OpenCL processing elements (kernels, work-groups, work-items). The assignment is determined by the parallelism level present in AUTODOCK, i.e., high, medium, low (Algorithms {1 to 3}). The GA and LS functions are mapped onto Krnl_GA and Krnl_LS kernels. For keeping most of the processing units on the accelerator busy, solutions from *different* LGA runs are processed *simultaneously*. For that purpose, each solution is mapped onto a *work group*, while the fine-grained tasks required for either generating and scoring solutions are executed by *work items*.

Furthermore, two main design aspects were considered:

Arithmetic precision: instead of using double-precision floating-point (FP) as in the original AUTODOCK, single-precision FP is employed in our OpenCL version. This is because lower precision FP calculations result in faster program executions without deteriorating the MD quality [8], [9].

Work distribution: the total number of work-items executed within each kernel (NDR_{size}) depends on: the number of LGA runs (R), the population size (P), LS rate (lsrate), and work-group size (WG_{size}):

$$\text{NDR}^{\text{Krnl\_GA}}_{\text{size}} = \{R \times P \times \text{WG}_{\text{size}},\ 1,\ 1\} \quad (1)$$

$$\text{NDR}^{\text{Krnl\_LS}}_{\text{size}} = \{R \times P \times \text{lsrate} \times \text{WG}_{\text{size}},\ 1,\ 1\} \quad (2)$$

All experiments were executed with R = 100, P = 150, lsrate = 100%. Similar to our previous study [9], the WG_{size}s chosen here lead to faster executions in most cases, and depend on the device type: $\text{WG}^{\text{GPU}}_{\text{size}}$ = 64, and $\text{WG}^{\text{GPU}}_{\text{size}}$ = 16.

III. EVALUATION

A. Experimental setup

1) Dataset: twenty molecular inputs were chosen from: [17] (`1u4d`, `1xoz`, `1yv3`, `1owe`, `1oyt`, `1ywr`, `1t46`, `2bm2`, `1mzc`, `1r55`, `1kzk`), [18] (`3s8o`, `1hfs`, `1jyq`, `2d1o`), and [19] (`5wlo`, `5kao`, `3drf`, `4er4`, `3er5`).

2) Hardware and software: for the baseline test, i.e., the measurement of execution times and power draws of the original AUTODOCK4.2.6 (*single-threaded*, implementing *only* Solis-Wets as LS method), we used an Intel Xeon E5-2666 CPU core. For parallel executions, accelerators based on commercial GPUs/CPUs were selected (Table I). Initial development was carried out targeting an AMD Vega 56 GPU, whereas cloud-based platforms were used as *porting* targets.

TABLE I
CHARACTERISTICS OF SELECTED ACCELERATORS INCLUDING THERMAL DESIGN POWER (TDP) AND OPENCL COMPUTE UNITS (CUS).

Device Name	Instance Type	GB/s GFLOP/s	TDP (W)	# CU
AMD Radeon RX Vega 56 GPU	On-premise	410 10 566	210	56
Nvidia Tesla V100 GPU	AWS `p3.2xlarge`	900 15 700	300	80
Intel Xeon E5-2666 v3 @2.6 GHz 18-core CPU	AWS `c4.8xlarge`	136 1500	135	36
Host	*Compiler*: g++ 5.4.0, *flags*: -O3			
Device	*OpenCL flags*: none			

3) Power measurements: while external meters can provide higher accuracy with complex measurement setups, internal power sensors accessed through software-based meters are convenient for monitoring power in-situ [20]. Therefore, for the Vega 56, we used a proprietary AMD software tool, while for the V100 and E5-2666, we employed publicly-available utilities like Nvidia-SMI [21] and Turbostat [22], respectively.

B. Performance analysis

1) Execution profiling: Table II reports four profiling metrics of Krnl_LS measured on the Vega 56.

Total time reports the fraction of the total execution time that is spent overall in Krnl_LS. For both Solis-Wets and ADADELTA, this metric is at least 99%, which confirms the fact that LS has the largest occupation in execution time.

 # Calls represents the number of times Krnl_LS is enqueued for execution. Executions using Solis-Wets require more enqueues than those of ADADELTA. Since the overall program must reach the $N^{\text{MAX}}_{\text{score-evals}}$ limit regardless of the LS method chosen, Krnl_LS – running Solis-Wets – must be enqueued more often than when it runs ADADELTA.

Avg. time is the mean elapsed time (ms) of a single Krnl_LS execution. In all cases analyzed, single executions of this kernel running ADADELTA are longer than those of Solis-Wets, which is attributed to its higher calculation complexity and larger number of LS iterations performed. The overall program duration can be calculated closely as *# Calls* $\times$ *Avg. time*.

Occupancy measures how efficiently GPU resources are used during execution. It is calculated as the number of *in-flight* GPU *wavefronts*. A wavefront is a block of work-items that are executed together, whereas one or more wavefronts conform a work-group. Values of 20% (Solis-Wets) and 10% (ADADELTA) indicate a low utilization efficiency.

2) Preliminary power profiling: Fig. 1 depicts the power consumption over time on the Vega 56. For all inputs, profiles

TABLE II
PROFILING METRICS OF LOCAL-SEARCH METHODS ON THE VEGA 56 (R = 100 LGA RUNS) USING INPUT ID: `3s8o`.

Krnl_LS	Solis-Wets	ADADELTA
Total time (%)	99	99
# Calls	120	46
Avg. time (ms) [per kernel enqueue]	225	5526
Occupancy (%)	20	10

for both Solis-Wets and ADADELTA are characterized by transitions between low and high power draws, ranging between 100 ... 220 W. These frequent power swings correspond to the switching between host-side and kernel (a sequence of Krnl_GA and Krnl_LS) executions.

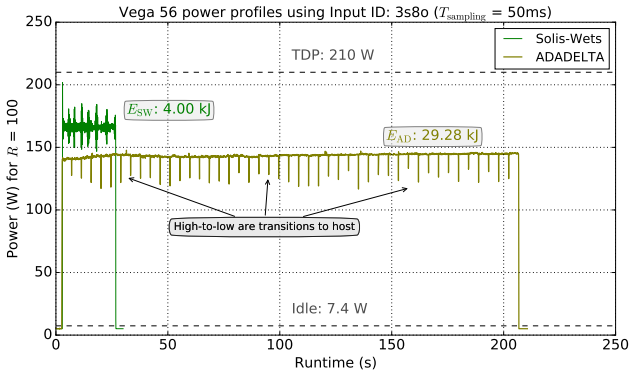

Fig. 1. Power measurements of OpenCL-AUTODOCK on the Vega 56.

Since Solis-Wets has many more *# Calls* to LS, the frequency of power transitions is also higher in Solis-Wets when compared to ADADELTA. Table II corroborates this, e.g., using `3s8o` as input results in 120 (Solis-Wets) and 46 (ADADELTA) Krnl_LS enqueues. In Fig. 1, it is even possible to count the 46 power transitions for ADADELTA.

Moreover, power draws on the Vega 56 were mostly around $\sim$170 W (Solis-Wets), and $\sim$140 W (ADADELTA). Even with ADADELTA performing more complex computations for gradients, and thus taking longer to complete, its kernel occupancy drops down to 10% due to the serialization required for correct partial derivatives (Section II-B). The lower occupancy of Krnl_LS implies that some Vega 56 block units are not utilized, and hence, are automatically turned off by the GPU (confirmed by discussion with the GPU vendor). This would explain the lower power draws of ADADELTA vs. Solis-Wets.

C. Power profiling on cloud devices

An initial sampling period T_{sampling} = 50 ms was chosen, as that was the shortest interval supported by the Vega 56. However, for the V100 and E5-2666, it is possible to use even shorter T_{sampling} values. The purpose of this section is to determine if such values provide any practical advantage.

1) V100 GPU: Fig. 2 depicts the V100 power profiles obtained for different configurations of LGA runs (R) = {50, 10} and T_{sampling} = {20, 10} ms. Profiles from different inputs are very similar, so only the case of `3s8o` is plotted.

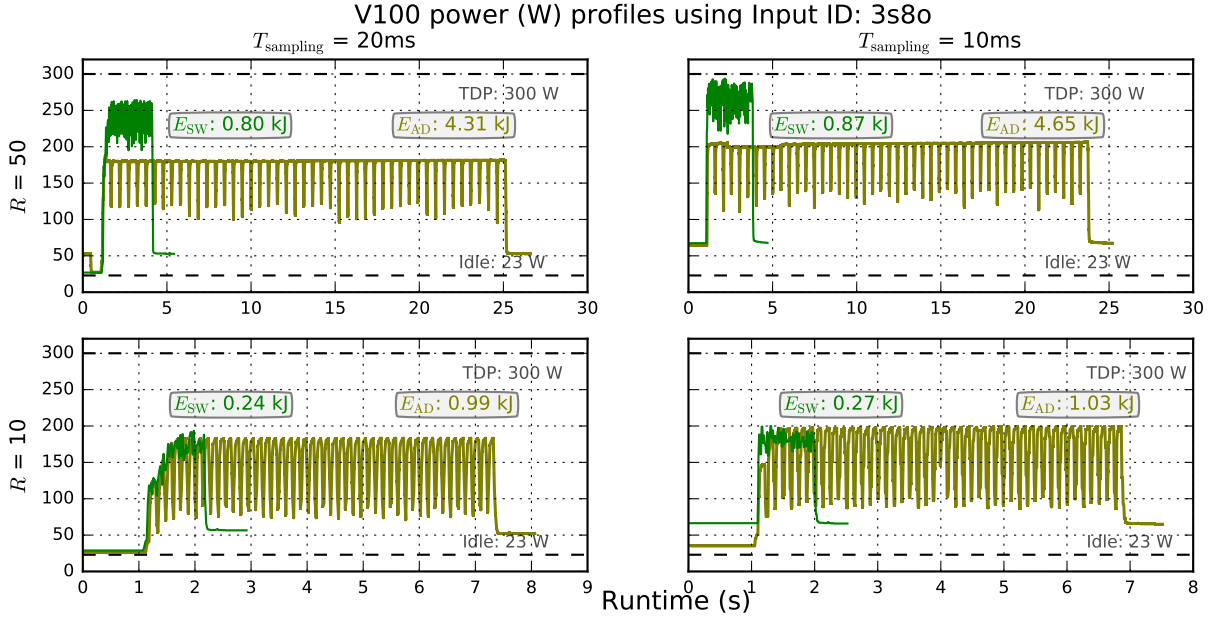

Fig. 2. Power measurements of OpenCL-AUTODOCK on the V100 (T_{sampling} = {20, 10} ms, R = {50, 10} LGA runs).

For the V100, T_{sampling} does not affect the overall profile shape. In fact, although not shown in Fig. 2, profiles obtained with T_{sampling} = 50 ms are very similar to those depicted. Observing more closely, it is possible to notice differences in the total runtimes. For instance, ADADELTA profiles (R = 50) for T_{sampling} = 20 ms have slightly longer executions ($>$ 25 s) than those for T_{sampling} = 10 ms ($<$ 25 s). Furthermore, MD log files tracking the entire execution for T_{sampling} = {50, 20, 10} ms indicate similar total program durations of ∼28.2 s, which in turn would correspond to a total number of {564, 1410, 2820} power samples, respectively. However, the power-sampling log files – from which power profiles were plotted – instead have {559, 1331, 2520} power samples. This suggest a loss of samples that accounts for {0.8, 5.6, 10.6}%, meaning that more samples are *lost* when using shorter T_{sampling} periods.

2) E5-2666 CPU: in contrast to the V100, where decreasing T_{sampling} and R values resulted in still consistent power profiles, we observed that for T_{sampling} = 10 ms and R = {1, 10} LGA runs, power-sampling log files showed erroneous measures (e.g., only ∼10 W during most of the execution). We believe this is due to a limitation also observed in other software-based power meters (e.g., RAPL-based ones [23]).

CPU power profiles were characterized by draws *higher* than the CPU thermal design power (TDP). As indicated in [24], this is due to the *turbo mode* feature, which makes chips run at higher frequencies and exceed the TDP under full load. On the E5-2666, all executions of the OpenCL program were characterized by a 100% utilization of all 36 CUs.

Finally, for each device, there is a slight difference in energy (in the range of some 0.1 kJ) when comparing measurements sampled at 20 ms vs. those at 50 ms. While this is not critical, increasing the sampling frequency also increases the number of samples lost during measurement, introducing larger inaccuracies for larger molecules. Therefore, the next experiments are all performed using T_{sampling} = 50 ms.

D. Energy efficiency

Fig. 3 depicts the speed and energy efficiencies achieved for each molecule in our dataset. This is used for understanding the correlation between both metrics on GPUs and CPUs.

On the V100, running Solis-Wets results in *higher* efficiencies with *larger* molecules (larger N_{rot} and N_{atom}) vs. the AUTODOCK baseline, with max. gains of ∼400x (speed) and ∼297x (energy) using `3drf`. ADADELTA results in *slower* executions compared to Solis-Wets in all cases. The highest gains of ∼112x (speed) and ∼137x (energy) are achieved using `1u4d` as input.

On the E5-2666, efficiencies of both LS methods tend to *decrease* with *growing* molecular complexity. Solis-Wets executions are more efficient than those of ADADELTA. The Solis-Wets method achieves min. gains of ∼3.5x (speed) and ∼2.0x (energy) when using `3er5`. ADADELTA executions achieve their max. gains of ∼11x (speed) and ∼7.0x (energy) when using `1u4d`. However, while ADADELTA speedups were at least 1.5x (using e.g., `3er5`), their corresponding energy-efficiency gains are lower than *one* (i.e., efficiency losses) with the largest inputs (e.g., `1jyq`, `4er4`, `3er5`). Lower efficiencies of ADADELTA are due to the GC (Algorithm 3), which becomes more time-consuming for larger molecules.

Table. III reports the geometric mean of energy-efficiency gains for the entire dataset. Although power draws of up to ∼300 W were observed on the V100 (i.e., higher than on the other devices), due to its much shorter runtimes, it yields higher energy gain factors over the baseline: ∼297x (Solis-Wets) and ∼137x (ADADELTA). The advantage of the V100 over the Vega 56 – quantified here as a ratio of geo. mean gains

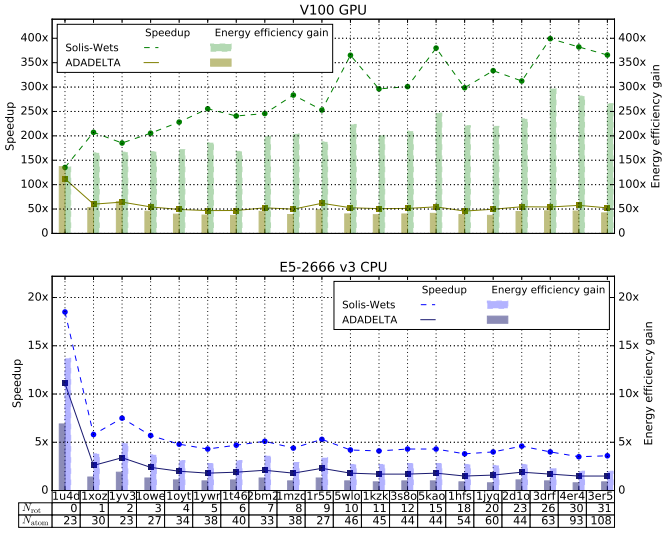

Fig. 3. Efficiency gains of OpenCL-AUTODOCK vs. baseline (R = 100).

of 2.3x (= $\frac{203.7}{88.6}$, Solis-Wets) and 3.3x (= $\frac{45.6}{13.8}$, ADADELTA) – come at higher economic costs, currently by a factor of 20x at street prices (mid 2019). Moreover, Table III indicates that the E5-2666 was the *least-efficient* device. Compared to the V100, its ratios of geo. mean gains are 0.015x (Solis-Wets) and 0.026x (ADADELTA). Such ratios are much lower than the hourly-price ratio of 0.475x (= $\frac{\$1.817/h}{\$3.823/h}$) when using their respective AWS instances[2] [25]. This means, the V100 offers higher energy gains than the E5-2666, for the same price.

TABLE III
ENERGY EFFICIENCIES OF OPENCL-AUTODOCK VS. BASELINE (R = 100).

Energy efficiency	**Solis-Wets**			**ADADELTA**		
	Vega 56	***V100***	***E5***	***Vega 56***	***V100***	***E5***
Geo. mean	88.6	203.7	3.1	13.8	45.6	1.2

Although the reported energy gain factors (Fig. 3) show a significant advantage of Solis-Wets over ADADELTA, the longer runtimes and higher energy-consumptions of ADADELTA result in higher-quality dockings in *many* cases [13]. This is a Solis-Wets vs. ADADELTA trade-off, where for molecules with few rotatable bonds ($N_{\text{rot}} < 8$), Solis-Wets could lead to sufficiently good results, and thus, spending more computing resources running ADADELTA is not worth it. However, for molecules where $N_{\text{rot}} > 11$, ADADELTA is likely to find *better* solutions, even in cases where Solis-Wets is simply not able to find any solution at all.

IV. CONCLUSIONS

Regarding energy savings of our OpenCL-AUTODOCK implementation, the V100 GPU was the *most* efficient device, achieving max. gains of ∼297x (Solis-Wets) and ∼137x (ADADELTA). On the other hand, the E5-2666 CPU was the *least* efficient one, yielding max. gains of ∼13x (Solis-Wets) and ∼7x (ADADELTA). The device-specific mapping of OpenCL constructs has a considerable impact on performance. The higher energy efficiencies achieved on GPUs are attributed to the fine-grained parallelization employed, which is more suitable to the underlying many-core GPU architecture.

[2]Charges comprise only compute capacity (Frankfurt region).

ACKNOWLEDGMENT

This work was supported by the German Academic Exchange Service (DAAD) and the Peruvian National Program for Scholarships and Educational Loans (PRONABEC), as well as by AMD Inc. and AWS Credits for Research program.

REFERENCES

[1] "Energy efficiency in high performance computing: The montblanc project." [Online]. Available: https://www.montblanc-project.eu/wp-content/uploads/2017/12/Teratec-presentation-tafani.pdf
[2] H. Anzt et al., "On the performance and energy efficiency of sparse linear algebra on gpus," *J. High Perf. Comput. Applications*, 2017.
[3] [Online]. Available: https://www.top500.org/green500/lists/2018/11
[4] M. A. S. Netto et al., "Hpc cloud for scientific and business applications: Taxonomy, vision, and research challenges," *J. ACM Comput. Surv.*, 2018.
[5] Q. Wu et al., "A heterogeneous platform with gpu and fpga for power efficient high performance computing," in *ISIC*. IEEE, 2014.
[6] L. G. Ferreira et al., "Molecular docking and structure-based drug design strategies," *J. Molecules*, 2015.
[7] U. Yadava, "Search algorithms and scoring methods in protein-ligand docking," *J. End. & Metabolism*, 2018.
[8] I. Pechan et al., "Molecular docking on fpga and gpu platforms," in *FPL*. IEEE, 2011.
[9] L. Solis-Vasquez et al., "A performance and energy evaluation of opencl-accelerated molecular docking," in *IWOCL*. ACM, 2017.
[10] E. Mendonça et al., "Accelerating docking simulation using multicore and gpu systems," in *ICCSA*. Springer, 2017.
[11] J. Fuhrmann et al., "A new method for the gradient-based optimization of molecular complexes," *J. Comput. Chem.*, 2009.
[12] J. Tavares et al., "On the efficiency of local search methods for the molecular docking problem," in *EvoBIO*. Springer, 2009.
[13] D. Santos-Martins et al., "Accelerating autodock4 with gpus and gradient-based local search," *ChemRxiv*, 2019.
[14] M. D. Zeiler, "ADADELTA: an adaptive learning rate method," *arXiv*, vol. abs/1212.5701, 2012.
[15] G. M. Morris et al., "Autodock4 and autodocktools4: Automated docking with selective receptor flexibility." *J. Comput. Chem.*, 2009.
[16] F. J. Solis et al., "Minimization by random search techniques," *J. Mathematics of Operations Research*, 1981.
[17] M. J. Hartshorn et al., "Diverse, high-quality test set for the validation of protein-ligand docking performance," *J. Med. Chem.*, 2007.
[18] Y. Li et al., "Comparative assessment of scoring functions on an updated benchmark: 2. evaluation methods and general results," *J. Chem. Information and Modeling*, 2014.
[19] H. M. Berman et al., "The protein data bank," *J. Nucleic Acids Research*, 2000.
[20] R. A. Bridges et al., "Understanding gpu power: A survey of profiling, modeling, and simulation methods," *J. ACM Comput. Surv.*, 2016.
[21] [Online]. Available: https://developer.nvidia.com/nvidia-system-management-interface
[22] [Online]. Available: https://www.linux.org/docs/man8/turbostat.html
[23] [Online]. Available: https://software.intel.com/en-us/articles/intel-power-governor
[24] D. Rohr et al., "Lattice-csc: Optimizing and building an efficient supercomputer for lattice-qcd and to achieve first place in green500," in *ISC*. Springer, 2015.
[25] [Online]. Available: https://aws.amazon.com/ec2/pricing/on-demand