Heterogeneous CPU+iGPU Processing for Efficient Epistasis Detection

Rafael Campos(✉), Diogo Marques, Sergio Santander-Jiménez, Leonel Sousa, and Aleksandar Ilic

INESC -ID, Instituto Superior Técnico, Universidade de Lisboa, Lisbon, Portugal
{rafael.campos,diogo.marques,sergio.jimenez,las,ilic}@inesc-id.pt

Abstract. Epistasis detection represents a fundamental problem in biomedicine to understand the reasons for occurrence of complex phenotypic traits (diseases) across a population of individuals. Exhaustively examining all possible interactions of multiple Single-Nucleotide Polymorphisms provides the most reliable way to identify accurate solutions, but it is both computationally and memory intensive task. To tackle this challenge, this work proposes a modular and self-adaptive framework for high-performance and energy-efficient epistasis analysis on modern tightly-coupled heterogeneous platforms composed of multi-core CPUs and integrated GPUs. To fully exploit the capabilities of these systems, the proposed framework incorporates both task- and data-parallel approaches specifically tailored to enhance single and multi-objective epistasis detection on each device architecture, along with allowing efficient collaborative execution across all devices. The experimental results show the ability of the proposed framework to handle the heterogeneity of an Intel CPU+iGPU system, achieving performance and energy-efficiency gains of up to 5× and 6× in different parallel execution scenarios.

Keywords: Epistasis detection · Heterogeneous computing · Integrated GPU+CPU platforms

1 Introduction

In the last decade, the increasing research focus on Genome-Wide Association Studies (GWAS) resulted in considerable developments in the understanding of human genomics [15]. These studies provide insight into the biological importance of Single-Nucleotide Polymorphisms (SNPs) and their k-order interactions,

This work was supported by Intel Corporation, the FCT (Fundação para a Ciência e a Tecnologia, Portugal) and the ERDF (European Regional Development Fund, EU) through the projects UIDB/50021/2020 and LISBOA-01-0145-FEDER-031901 (PTDC/CCI-COM/31901/2017, HiPErBio). Sergio Santander-Jiménez and Diogo Marques are supported by FCT fellowships under Grants SFRH/BPD/119220/2016 and SFRH/BD/136053/2018.

known as epistasis. As the variation of interacting SNPs is highly coupled with trait changes in the phenotype of each individual, epistasis analysis represents a fundamental tool to identify the relationship between particular genotypes and the risk of development of complex diseases. While some diseases are correlated to pairwise combinations of SNPs ($k = 2$) [2], other illnesses, such as Alzheimer's and type-2-diabetes, depend on higher-order epistasis ($k > 2$) [14,19]. Since the computational complexity increases exponentially with the interaction order k, due to the higher number of SNP combinations, achieving an efficient execution of this analysis is a challenging task. This is even more demanding when considering multi-objective optimization, which is increasingly being used in state-of-the-art works to improve the accuracy of epistasis detection [4,5,8].

To tackle this issue, the algorithms for epistasis detection can be deployed and optimized in modern processors, with powerful out-of-order cores [5]. On the other hand, the high computational complexity of high-order epistasis and the characteristics of modern processors may lead to a decreased energy-efficiency. An alternative is the utilization of low power devices, such as the integrated Graphics Processing Unit (iGPU) contained in the chip package of modern desktop processors [7]. However, the lower power consumption in these devices comes at the cost of reduced performance. To address this challenge, heterogeneous architectures constituted by multi-core Central Processing Units (CPUs) and an iGPU can be used to balance performance and energy-efficiency. However, it is not trivial to fully exploit the capabilities of these systems. CPUs and iGPU share part of the memory hierarchy, thus, the execution of one device may affect the performance of the other [7]. Furthermore, each device has distinct capabilities and, in order to achieve accurate load balancing and maximize the exploitation of parallelism, the data needs to be carefully partitioned across each device.

State-of-the-art works on epistasis are strongly focused on the performance boosting of optimization algorithms in several devices, such as CPUs [13,16,18], discrete GPUs [6,9], FPGAs [10] and co-processors [17]. Other works propose methods based on machine-learning and evolutionary algorithms [1,4]. Although these stochastic approaches allow high-order epistasis detection, they provide sub-optimal solutions to the problem. On the other hand, exhaustive methods have a deterministic nature that guarantees optimal solutions, at the expense of higher computational restrictions difficult to handle for higher-order epistasis. For these reasons, the work herein proposed focuses on addressing the challenges of exhaustive epistasis detection, in order to attain an efficient execution when targeting SNP interactions of order $k \geq 2$ on single and multi-objective search scenarios. To the best of our knowledge, there are no state-of-the-art works that explore heterogeneous computing at this level of integration (CPU+iGPU) to enhance the performance and energy-efficiency of epistasis detection.

To close this gap, this work proposes a modular and robust framework for single and multi-objective exhaustive epistasis analysis, targeting heterogeneous systems-on-chip with multi-core CPU and iGPU. Moreover, the efficiency of the proposed framework is experimentally assessed in an Intel CPU+iGPU platform, for a range of parallel and single-device execution scenarios, several data-sets with diverse characteristics and for different optimization goals, *i.e.*, performance, power consumption or energy-efficiency. In particular, this work includes the following contributions:

- Parallel algorithms for exhaustive epistasis detection on two different architectures, namely a modern multi-core CPU and a low-power iGPU;
- Configurable execution framework for collaborative single-objective and multi-objective epistasis detection on heterogeneous computing platforms;
- Insights into the performance, energy-efficiency and power consumption trade-offs when performing epistasis detection on modern CPU+iGPU systems.

This paper is structured as follows: Sect. 2 introduces the epistasis detection and its optimization challenges, while Sect. 3 provides algorithm and CPU+iGPU architecture overview. Section 4 describes the proposed framework and device-targeted optimizations, while Sect. 5 presents the experimental results. Section 6 concludes the paper and suggests future research directions.

2 Problem Formulation

The likelihood of certain phenotypic traits, e.g. diseases, is often governed by the joint interaction of different SNPs in the genome of a particular individual. The epistasis detection is targeted at identifying such interactions by processing genotypic information from a case-control data set D of size $N \times (M+1)$, where N is the number of individual samples and M the number of SNPs under study. Each entry $D[i,j], i \in \{1,...,N\}, j \in \{1,...,M\}$ displays the genotypic value observed at the j-th SNP from the i-th sample, represented as 0 (homozygous major allele), 1 (heterozygous allele), or 2 (homozygous minor allele). The disease status y for the i-th sample is stored in the last entry $(D[i, M+1])$, where $y = 0$ for control samples and $y = 1$ for case samples.

Epistasis procedures often implement optimization engines to identify the combination of k SNPs $x = [x_1, x_2, ..., x_k]$ best supported by some biological criteria, where $x_i \in \{1,...,M\}$. The number k of SNPs in the combination is designated as ‘interaction order’ and represents a key element from a computational perspective. When looking for higher-order epistatic interactions ($k > 2$, as in the case of complex diseases), the search space of all possible solutions increases exponentially according to the expression $\frac{M!}{k!(M-k)!}$ [12], thus impacting the times required to conduct accurate experimental campaigns in real-world scenarios.

The approach presented in this paper is aimed at dealing with such complexity issues by exploiting the heterogeneous computing capabilities of CPU+iGPU architectures. Several biological criteria, implemented as objective functions, can be adopted in our proposal. In the present work, two widely-used objective functions have been examined: Bayesian K2 score [11] and Mutual Entropy [3]. Built upon Bayesian network principles, the K2 score can be expressed as:

$$K2 = \sum_{i=1}^{I} \left(\sum_{b=1}^{r_i+1} \log(b) - \sum_{j=1}^{J} \sum_{d=1}^{r_{ij}} \log(d) \right), \quad (1)$$

where I is the number of possible genotypic combinations among k SNPs ($I = 3^k$), J the number of phenotypic states ($J = 2$ in case-control scenarios), r_i the frequency of a certain genotypic combination i at the evaluated SNPs $x = [x_1, x_2, ..., x_k]$, and r_{ij} the number of samples that satisfy the occurrence of the phenotypic state j with the genotypic combination i at the evaluated SNPs. Lower K2 score denote better solution quality.

The second objective function employs information theory concepts to quantify the quality of the evaluated SNP interaction. More specifically, the mutual entropy ME is the reciprocal of the mutual information $I(x, y)$:

$$I(x,y) = \sum_{i=1}^{I} \sum_{j=1}^{J} p(x[i], y[j]) log \frac{p(x[i], y[j])}{p(x[i])p(y[j])}, \tag{2}$$

where $p(x[i])$ is the probability of observing the genotypic combination i at x, $p(y[j])$ the probability of the phenotypic state j, and $p(x[i], y[j])$ the probability of j under the genotypic combination i. Similarly to the K2 function, candidate solutions with lower ME scores are preferred from a quality perspective.

The proposed heterogeneous strategies are aimed at allowing efficient epistasis detection supporting single-objective and multi-objective searches. While single-objective approaches seek a single optimal solution (attending to the chosen objective function, K2 or ME), a multi-objective search looks for a comprehensive set of non-dominated[1], Pareto-optimal solutions that represent the best trade-offs across the considered objectives (K2 and ME simultaneously).

3 Algorithm – Architecture Overview

This section is devoted to the description of the epistasis search workflow, design strategies, and characterization of the targeted CPU+iGPU architecture. In order to identify the SNPs interactions that best explain the traits in the input data set, a number of well-defined steps must be followed (Fig. 1(a)):

1. Generation of the combination of k interactive SNPs $x = [x_1, x_2, ..., x_k]$ subject to evaluation, given by an integer array for solution encoding purposes.
2. Identification of genotype frequencies across case-control samples in the data set, at the SNPs $x_1, x_2, ..., x_k$.
3. Scoring of the evaluated combination according to the considered objective function (single-objective search) or functions (multi-objective search).
4. Evaluation of the candidate solution:
   (a) Single-objective optimization: the currently evaluated interaction is retained in memory in case it improves the objective score of the best solution identified in previous iterations of the search.

[1] Given two solutions s_1 and s_2 and n objective functions $\boldsymbol{f}(s) = [f_1(s), ..., f_n(s)]$, s_1 dominates s_2 *iff* 1) $\forall\ i \in [1, 2, ..., n]$, $f_i(s_1)$ is not worse than $f_i(s_2)$ and 2) $\exists\ i \in [1, 2, ..., n]$ such that $f_i(s_1)$ is better than $f_i(s_2)$. Those solutions that are not dominated by any other candidate compose the Pareto-optimal set. The representation of this set in the objective space is commonly designated as Pareto front.

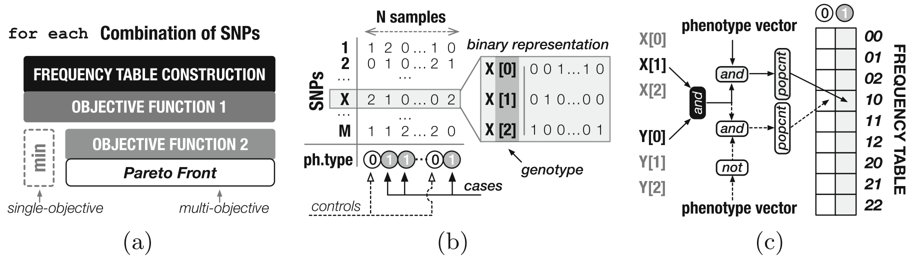

Fig. 1. General overview of epistasis detection: a) exhaustive search steps; b) binarized data set representation; c) frequency table construction.

(b) Multi-objective optimization: the currently evaluated interaction is stored in the Pareto set (front) in case it is not dominated by any other previous solution kept in the set. All the solutions in the set that are dominated by the new interaction are consequently removed.

These steps are performed for each possible combination of k sorted, non-repeated SNPs. In a single-objective search, the combination with minimal objective score represents the most supported solution. On the other hand, a multi-objective search returns the combinations that compose the Pareto-optimal set.

According to these steps, the representation and efficient processing of the input case-control data set represents a major concern to boost the performance of epistasis detection. The input data matrix can contain thousands of SNPs and samples, potentially resulting in high memory requirements. This can lead to an increased contention in shared memory resources of modern processors, thus reducing application performance. Since the range of genotypic values that a SNP can take is limited to three (0, 1 or 2), this issue can be mitigated by compressing the data. This is attained by binarizing the data matrix, *i.e.*, each element is codified in individual bits of a larger data element, *i.e.*, a 32-bit integer, as illustrated in Fig. 1(b). By using the binary encoding, each SNP is represented with three binary vectors (with indexes 0, 1 and 2), one for each genotypic value. Each bit in the vector reflects the presence (1) or absence (0) of a genotypic value the processed sample exhibits at the considered SNP. The disease status array is also binarized.

In the proposed work, the computations performed to calculate objective scores use a frequency table as support, where the instances of all possible genotypic combinations for the evaluated SNP interaction are accounted. This frequency table has 3^k rows, where k is the interaction order, and two columns, one per disease state, as presented in Fig. 1(c). To construct this table, the bitwise operations AND, NOT and POPCOUNT (POPCNT) are used to extract from the binary data the information required by each objective function. As shown in Fig. 1(c) for $k = 2$ and two SNPs X and Y, a bitwise AND is performed between the binarized data elements X[1] and Y[0] to calculate the observations of the genotypic combination '10' across samples. After this step, AND and

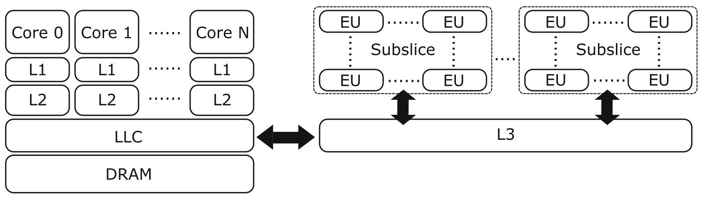

Fig. 2. General heterogeneous CPU+iGPU architecture.

AND+NOT operations are performed between this intermediate result and the disease status, resulting in two arrays that identify which observations belong to cases and which ones to controls. Finally, the POPCNT is performed over the two arrays and the frequency table is updated in the position corresponding to the considered genotype combination. Once the frequency table is filled, the objective scores can be computed by using the expressions in Eqs. 1 and 2.

In order to enhance epistasis searches through parallelism (namely targeting the parallel processing of independent SNP combinations and the data parallelism exhibited by frequency calculations), this work explores heterogeneous CPU+iGPU systems. Since each device has distinct hardware specifics, it is necessary to analyze their micro-architectures to understand the implementation scenarios that benefit the most from CPU and iGPU capabilities.

As shown in Fig. 2, modern processors are complex systems-on-chip containing several CPU cores and iGPU. In particular, CPU cores are equipped with powerful out-of-order engines that support diverse instruction types, including vector instructions to handle multiple computations per instruction, leading to higher performance. The vector length varies according to the micro-architecture, e.g., Intel processors may support 128-bit, 256-bit or even 512-bit AVX instructions. The CPU memory hierarchy usually encapsulates two private caches (L1 and L2), and shared memory levels: LLC and DRAM. Data sharing between cores is performed through the ring interconnect, which also provides a communication interface between the CPU LLC and the iGPU L3 cache, allowing data transfer between them.

The iGPU micro-architecture is organized in several slices, each formed by subslices, as illustrated in Fig. 2. The subslices consist on a set of Execution Units (EUs) containing the Arithmetic Logic Units (ALUs) and FP units that also perform vector instructions. These EUs handle several threads simultaneously, in order to exploit data-level parallelism. The subslices have access to the shared L3 cache, and also contain private L1 and L2 caches (reserved for sampling tasks, thus not used for general-purpose computing). The presence of an iGPU in the processor package and its sharing of the memory subsystem differentiates it from discrete GPUs. While discrete GPUs have its own dedicated memory, iGPUs share the memory subsystem with the CPU, which reduces the impact of data transfers between the two devices. When executing memory-intensive tasks, such as epistasis detection, this allows to fully extract the potential of the

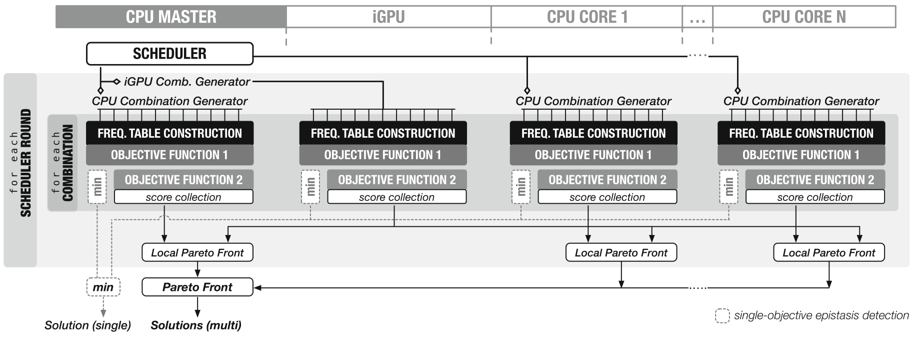

Fig. 3. Framework for heterogeneous exhaustive epistasis detection.

heterogeneous architecture. The application performance can be further improved by reducing its memory footprint (*e.g.* by binarizing the data set).

Due to the micro-architectural differences between CPU and iGPU, the heterogeneous implementation of epistasis detection algorithms should rely on different programming models that allow to fully exploit the properties of each device type, including their memory and compute resources. In the scope of this work, OpenMP is used for the CPU to efficiently distribute work across the multi-core processor, achieving task-level parallelism. For the iGPU, OpenCL was chosen, an often employed and highly optimized programming model that explores the compute abilities of these highly parallel devices. This can be explored to offload the iGPU kernels and exploit data-level parallelism in this application.

4 Heterogeneous Framework for Epistasis Detection

To achieve an efficient and collaborative search for optimal epistasis solutions in tightly-coupled multi-core CPU+iGPU systems, this work proposes a heterogeneous framework, presented in Fig. 3, that uses task and data-level parallelism as a mean to exploit the full potential of CPUs and iGPU, not only for performance, but also for power consumption and energy-efficiency.

In the proposed framework, the CPU master thread, *i.e.*, the scheduler, is responsible for assuring the load balancing by distributing different amount of combinations to be processed at the iGPU and all CPUs cores (including the scheduler). Besides the total number of combinations, each CPU core also receives the initial SNP combination number, which is used by the CPU combination generators to locally determine the next combination to be processed. At the CPU master, the iGPU kernel is enqueued and the iGPU combination generator creates a buffer containing the SNP combinations to be offloaded for the iGPU processing. While the iGPU performs these combinations, the scheduler enqueues the next iGPU kernel and refills the combination buffer with the next combinations to be processed. Some SNP combinations are also attributed to the scheduler, in order to maximize the utilization of the CPU cores. This process is repeated by the scheduler until all SNP combinations are exhausted, which means that all candidate solutions to the problem have been evaluated.

In the final step, the scheduler reports the solutions of the single-objective (the solution with minimum score) or multi-objective (Pareto front) approaches. It is worth to note that the local optimum solutions are kept at each device during the processing of SNP combinations, and the final solution(s) are produced at the CPU master only upon examination of all possible combinations. For the multi-objective approach, the local Pareto fronts are exclusively constructed at the CPU cores due to the low data-parallel potential of this algorithm. However, these local fronts include the scores computed at the iGPU, which are distributed across all CPU cores to ensure load balancing.

Due to its modular design, the proposed framework can be divided in distinct modules, each representing a processing element of the heterogeneous implementation of the algorithm presented in Sect. 3.

4.1 Multi-core CPU Parallelization and Vectorization

To exploit the task-level parallelism of multi-core CPUs, OpenMP is used to allow for the SNP combinations evaluation among the available threads. As shown in Fig. 3, after receiving the number of combinations and the initial SNP combination from the scheduler, each CPU core proceeds with the frequency table construction. To fully exercise the data-level parallel processing capabilities offered by modern CPUs, vectorization techniques are applied at this processing stage, which is accomplished by using the SIMD instructions supported by the CPU micro-architecture. For example, Intel 8th generation CPUs have AVX instructions, which can be employed to perform bitwise operations required by the algorithm, *i.e.*, AND and AND+NOT operations presented in Fig. 1(c) (see Sect. 3), using up to 256-bit wide vectors. In this case, the AND operation is performed with the `_mm256_and_si256` intrinsic, while AND+NOT operation is also implemented by a single intrinsic instruction, *i.e.*, `_mm256_andnot_si256`, allowing to compute AND+NOT for 256-bit wide data in a single clock cycle. As the POPCOUNT operation is not available as an intrinsic Intel 256-bit AVX instruction, it is implemented using the `_popcnt64` instruction on 64-bit data.

After the frequency table construction, the scores of the *Objective Function 1* (and *Objective Function 2* for the multi-objective algorithm) are calculated, as illustrated in Fig. 3. For the considered objective functions, *i.e.*, K2 score and Mutual Entropy, it is necessary to perform calculations based on the natural and base-2 logarithms, respectively. Since logarithm operations are computationally intensive, their repeated utilization may potentially lead to significant performance degradation (especially due to a high probability of recalculating the same logarithm value when evaluating a large set of SNP combinations). To overcome this issue, in the proposed work, look-up tables were employed to store and reuse the values for the logarithms. To reduce the memory impact of the look-up tables, their size is kept small, thus it is only needed to compute the values that surpass the maximum table size. Moreover, to increase the performance of the K2 score, the $\log(n!)$ function is replaced by a `gamma` function, a less computationally demanding alternative that removes the need to compute factorials. To limit the size of the look-up table while maintaining accuracy in the solutions, the gamma function is only used for higher values of n. For low n values, the look-up table is accessed to obtain $\log(n!)$.

After the score calculation, the best local solution or the partially composed Pareto set are stored locally in each thread, to minimize the communication costs between threads. This process is repeated until each CPU core has finished evaluating all the attributed SNP combinations. When all threads have finished this stage, the globally optimal solutions, i.e., a solution with the minimum score for the single-objective or Pareto-optimal set for the multi-objective approach, are constructed by the scheduler thread.

4.2 iGPU Implementation

The iGPU implementation explores the capabilities of the OpenCL programming model to offload work to the accelerator. The kernel function is defined to contain all operations related to the evaluation of a SNP combination through the building of the frequency table and calculation of objective function(s). The combinations to be evaluated by different work-items are provided in a buffer, which is created in the *iGPU Combinations Generator* module at the CPU master thread, as shown in Fig. 3. Each work-item in the iGPU independently processes the assigned SNP combinations, in parallel with the remaining work-items, in order to explore the data-parallel potential of this architecture.

Similarly to the CPU implementation, frequency table construction is implemented with vectorized instructions and specific OpenCL vector data types, thus employing the vector functional units contained in the iGPU EUs to maximize the overall performance. For example, each EU in Intel Gen9.5 iGPU supports 128-bit wide instructions [7], thus `uint4` data type is used, which corresponds to four 32-bit unsigned integers. By using these data types, the AND and NOT operators are vectorized, along with the POPCOUNT operation used in the proposed algorithm. After filling the frequency table, the scores of the objective functions are calculated by using the look-up tables and gamma function, similar to the procedure elaborated in the CPU implementation. The look-up tables are built by the CPU master and transferred to the iGPU before kernel execution.

For single-objective optimization, the minimum solutions are kept locally (in the iGPU L3 cache) on a per work-item basis, and communicated back to the CPU master only when all SNP combinations of the data-set are examined. This process is described in pseudo-code in Fig. 4. The combination (`comb`) to be processed by the specific work-item (identified by `id`) is obtained from the `combs` array and its score is computed with the `get_score()` function. This value is compared to the local best score and solution, which is updated if necessary. To compute the score, the kernel iterates through the patients (`n_patients` in total), filling the frequency table `ft` (with 3^k rows and 2 columns), using logical operations (AND, NOT and POPCOUNT) with the SNP and disease state (`state`). Following this, the final score for the combination is obtained by applying the objective function (represented by `obj_function()`).

```
kernel(combs, scores, solutions)
{
    id = work_item_id()
    comb = combs[id]
    score = get_score(comb)
    if(score < scores[id])
    {
        scores[id] = score
        solutions[id] = comb
    }
}
```

```
get_score(comb)
{
    for(j = 0:n_patients)
        for(i = 0:3^k){
            for(x = 0:k)
                res &= SNP[comb[x]][j]
            ft[i][0] += popc(res & ~state[j])
            ft[i][1] += popc(res & state[j])
        }
    return obj_function(ft)
}
```

Fig. 4. Pseudo-code for the iGPU kernel computing the score for a given combination of SNPs.

In the case of multi-objective optimization, the two scores are computed in the iGPU within each work-item and distributed to the CPU cores for the local Pareto front construction, as previously referred. It is worth emphasizing that the involvement of CPU cores for generating combinations, Pareto fronts and look-up table is crucial for attaining the high performance execution on iGPU, due to the low potential for data-level parallelization of these routines that involve a high amount of complex control statements, as it will be shown in Sect. 5.

4.3 Execution Orchestration for Multi-core CPU+iGPU Processing

As previously referred (see Fig. 3), the scheduler in the CPU master is responsible for distributing workload among the processing entities, *i.e.*, iGPU and CPU cores. Additionally, the scheduler also participates in the evaluation of SNP combinations, in order to fully exploit the multi-core CPU capabilities.

To attain an efficient data distribution across the devices, dynamic load balancing is applied with the goal of minimizing potential idle times at the CPU and iGPU sides. While the SNP combinations handled by the iGPU are performed in several rounds with the fixed amount of work-items per kernel invocation, the combinations evaluated in the CPU threads (in each scheduling round) is defined as $N_{CPU} = \frac{N_{iGPU}}{P_{iGPU}} \times P_{CPU}$, where N_{iGPU} is the number of SNP combinations assigned to the iGPU, while P_{iGPU} and P_{CPU} denote the measured performance of the iGPU and CPU, respectively (both assessed during the algorithm run-time to ensure self-adaptive nature of the proposed framework).

In the case of single-objective evaluation, the CPU master is responsible for performing the final reduction stage on the local optimum solutions encountered at each processing entity, thus determining the final optimal epistasis solution. For multi-objective optimization, this reduction stage consists on building the final Pareto front from the partial fronts built by the remaining CPU cores.

5 Experimental Results

In order to evaluate the benefits and drawbacks of the proposed heterogeneous framework for single and multi-objective epistasis detection on CPU+iGPU, its performance, energy-efficiency and power consumption needs to be compared and evaluated against different implementations and configurations. With this aim, four main comparative execution scenarios are considered, *i.e.*: the baseline CPU algorithm presented in Sect. 3 (Popcnt), its vectorized CPU implementation (Popcnt_Vec), execution of the search in the iGPU alone (iGPU), and execution in the iGPU with the CPU running only the scheduler thread (Sched).

The iGPU execution scenario was included to characterize the iGPU when working as a single device, which implied placing the operations necessary to iterate through combinations in an iterating kernel executing in the iGPU. Additionally, single-objective optimization is performed for the considered two objective functions, *i.e.*, K2 score and Mutual Entropy, separately.

5.1 Experimental Setup

The experimental platform involves an Intel i7-8700K processor and the Gen9.5 iGPU, following the architecture described in Sect. 3. This system has 32 GB of DRAM and runs Linux-based CentOS 7.5, with OpenCL 2.1 NEO drivers, version 19.34.13959. The CPU was kept at its base frequency of 3.7 GHz, and the number of OpenMP threads defined for execution was equal to the number of cores (6), and each thread was bound to a single core. The applications were compiled using ICC, version 19.0.5.281, with O3 optimization enabled. For the iGPU, the number of OpenCL work-items chosen was the same across all configurations of the framework, while the work-group dimension was defined as the maximum allowed in this iGPU, *i.e.*, 256 work-items. As work-items in the same group reside in the same subslice, the total number of work-items was defined as a multiple of $3 \times 256 = 768$. The value chosen was 76800 work-items, as this amount was experimentally determined to lead to better performance across the different test cases, *i.e.*, varying number of samples, SNPs and epistasis orders.

To perform an experimental evaluation for diverse amounts of SNPs and individual samples, three data sets from [5] are considered in this work, namely: small data set, with 23 SNPs and 10000 samples; medium data set, with 1000 SNPs and 4000 samples; large data set, with 31339 SNPs and 146 samples. Five individual tests were performed for each application implementation, for the epistasis orders $k = 2$ and $k = 3$.

Performance is reported as the amount of operations executed per second (OPS/s), while the energy-efficiency corresponds to the amount of operations per joule (OPS/J). For all the test cases considered in this work, the number of operations, OPS, is defined as $OPS = nCr(M, k) \times N$, where M is the number of SNPs, k the interaction order, N the number of individual samples, and $nCr(n, k)$ the number of k-combinations in a set of n items. The power consumption (W) and consumed energy are obtained through the RAPL interface, which allows the measurement of the energy consumption in the package (CPU and iGPU), iGPU and CPU cores. For all versions the package power was considered, except for the iGPU version, where the iGPU power is evaluated instead.

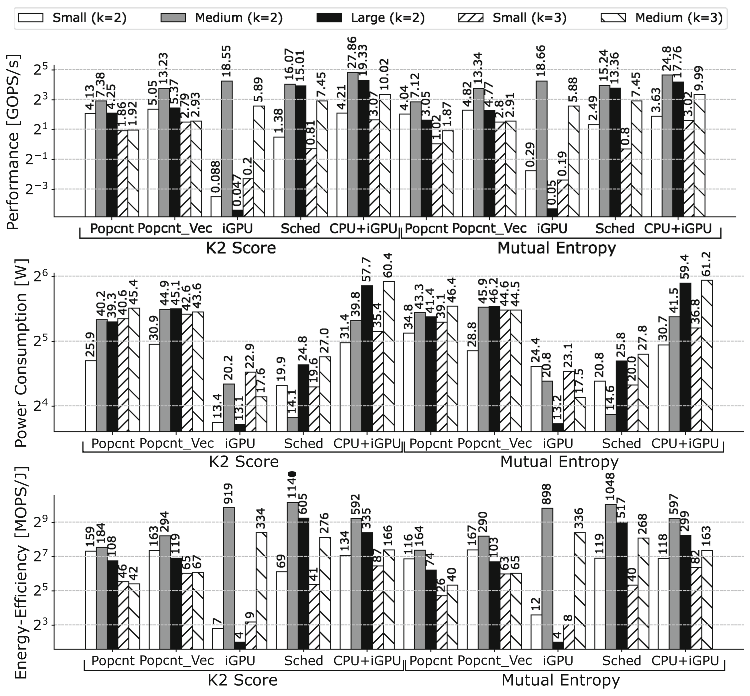

Fig. 5. Performance, power consumption and energy-efficiency results for single-objective configurations.

5.2 Single-Objective Evaluation

As a first step, the results obtained for the single-objective configurations are analyzed, aiming at determining the characteristics of each implementation depending on the execution scenario and input data set. Figure 5 contains the results for single-objective optimization. The performance and energy-efficiency units presented in the plots (OPS/s and OPS/J) were chosen to facilitate comparisons across different tests, as the time and energy results vary greatly, with the small data set having very short execution times and reduced energy consumption, and the larger tests executing in minutes, with large energy consumption. This is mainly due to the epistasis order, since the number of SNP combinations to be processed vastly increases with it, making the tests increase in complexity. For example, the number of SNP combinations for the medium data set increases from 4.99×10^5 with $k = 2$ to 1.66×10^8 with $k = 3$, which results in greatly increased average execution times (from 145 ms to 168 s) and average energy consumption (from 5 J to 6801 J). For the large data set (resulting in a total of 4.91×10^8 SNP combinations for $k = 2$), the average execution time was of 312 s, and the average consumed energy was 4295 J. The tests with the small data set have average execution time of 17 ms and average consumed energy of 383 mJ.

As can be observed, except for the simplest data set and epistasis order of $k = 2$, *i.e.*, small ($k = 2$), the CPU+iGPU architecture corresponds to the highest performance for the remaining test cases. For the small ($k = 2$) test, the amount of combinations to process is very small, which does not allow to fully exploit the data-level parallelism provided by the iGPU. For this test, Popcnt_Vec achieves the maximum performance for both objective functions across all execution scenarios. Moreover, by comparing Popcnt and Popcnt_Vec across all tests, it is possible to conclude that the use of vectorization leads to performance increases of up to 79%, in the case of the medium data set ($k = 2$).

On the other hand, executing solely on the iGPU leads to worse performance in all tests except for the medium data set. For this test, the iGPU scenario registered higher performance than Popcnt_Vec for $k = 2$ and $k = 3$. This behavior mainly occurs due to the execution time of the iterating kernel of the iGPU. According to the GPU Hotspots analysis in Intel VTune Amplifier, for $k = 2$, it takes 0.1 ms for the small data set (1 execution), 7 ms for the medium data set (7 executions), and 240 ms for the large data set (6395 executions). This indicates that the architecture of the iGPU is not suited for performing nested conditional statements and loops such as the ones necessary to iterate through combinations of SNPs, as the execution time grows exponentially with the number of SNPs. The number of iterating kernel executions also indicates that the medium data set makes full use of the iGPU data parallelism, while the small data set is not able to use all available work-items, as it only needs 1 execution to complete its work, resulting in lower performance.

Regarding the power consumption results, it can be observed that Popcnt_Vec has higher power consumption than Popcnt, due to the use of 256-bit vector execution units. The iGPU scenario reports the lowest power consumption values. This result is expected given the fact that this device operates in more strict power requirements than the CPU cores. Following it, the Sched scenario presents lower power consumption values than the remaining, which is explained by it only utilizing one CPU core with the scheduler thread and the iGPU. This same reason explains the highest power consumption registered for the CPU+iGPU fully operating simultaneously.

When comparing energy efficiency results, the Popcnt_Vec and Popcnt scenarios present higher efficiency than the iGPU operating by itself in all tests except for the medium data set, which is the same trend followed in the performance results, where the iGPU showed greater performance in these tests. This indicates that although the strategies operating in the CPU require more power, they compensate for it when it comes to energy efficiency, due to lower execution times. This correlation between high performance and high energy-efficiency is also observed when comparing just Popcnt_Vec and Popcnt, with the former having up to 60% higher energy-efficiency (medium data set, $k = 2$), despite the higher power-consumption values. Regardless, the energy-efficiency is highest for the Sched scenario, due to its high performance and especially low power consumption, whereas CPU+iGPU has lower energy-efficiency due to its high power values.

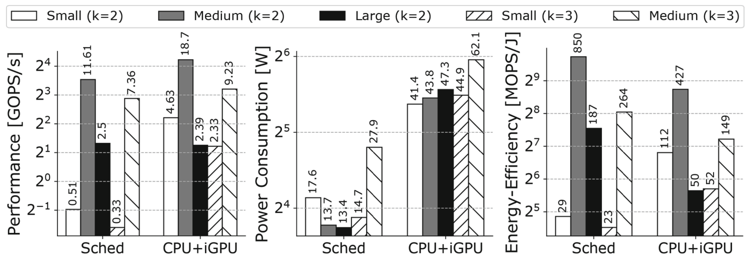

Fig. 6. Performance, Power consumption and energy-efficiency results for multi-objective heterogeneous configurations.

All performance, power or energy results are consistent through the two different objective functions, demonstrating that the differences between methods used to compute the K2 and ME scores are negligible for performance or energy.

5.3 Multi-objective Evaluation

The results obtained for the multi-objective configurations are presented in Fig. 6, focusing the comparisons with the Sched execution scenario. The average execution times for the medium data set are of 133 ms and 79 s, for $k = 2$ and $k = 3$ respectively. The same tests have energy consumption of 3 and 3280 J. For the large data set ($k = 2$), the average execution time is 18 s, and the average consumed energy is 374 J. Both tests with the small data have low execution times and energy consumption, in average 16 ms and 276 mJ, respectively.

Performance-wise, the CPU+iGPU proposal presents better results in all tests, similarly to the single-objective configurations. The Sched approach presents lower performance values, with the largest difference in performance being observed for the test with the large data set. This difference is due to the fact that the build of the Pareto front with the results from the iGPU is done only in the master thread in the Sched scenario, while in CPU+iGPU this task is divided across the processing threads, minimizing the workload and leaving the master thread available to send work for the iGPU as soon as it is available. The power and energy results herein shown are similar to the results for the heterogeneous single-objective versions, with the added CPU cores used for processing in the CPU+iGPU proposal increasing power consumption and decreasing energy-efficiency.

In summary, the attained results demonstrate the benefits in performance and energy efficiency when using heterogeneous strategies for exhaustive epistasis analysis. While the CPU or iGPU only provide advantage in specific tests, as is the case of the medium data set for the iGPU, the heterogeneous configurations, especially CPU+iGPU, show consistently better parallel performance in overall terms. The power consumption and energy-efficiency results are generally better for the Sched approach, as CPU+iGPU uses all CPU cores and the iGPU, while the Sched scenario uses only the iGPU and a single CPU thread, which in most cases leads to an energy efficiency advantage. These results are consistent when using a multi-objective method with the K2 and ME scores.

6 Conclusions

Epistasis detection is a fundamental GWAS research topic aimed at identifying interactive SNPs responsible for complex traits, with significant applications in biology and human health. In this context, exhaustive search methods are adopted to provide accurate solutions to the problem, at the cost of higher complexity, especially for high interaction orders. To tackle this issue, this work proposed a heterogeneous framework for efficient exhaustive epistasis detection in multi-core CPU+iGPU systems. The proposed framework considers both single and multi-objective optimization, based on the K2 score and Mutual Entropy objective functions. To fully exploit the potential of each device, OpenMP and OpenCL programming models were used and the adaptive scheduling was employed to attain efficient collaborative execution across several CPU cores and iGPU. The experimental evaluation shows that the proposed framework allows attaining performance and energy-efficiency gains of 5× and 6×, respectively, for different data sets and execution scenarios. Future directions involve the extension of the framework for other hybrid architectures, e.g., CPU+FPGA and/or different CPU+iGPU systems, as well as the evaluation of performance and energy-efficiency benefits when applying dynamic voltage and frequency scaling.

References

1. Che, K., et al.: Epistasis detection using a permutation-based gradient boosting machine. In: 2016 IEEE International Conference on Bioinformatics and Biomedicine (BIBM), pp. 1247–1252. IEEE (2016)
2. Dinu, I., et al.: SNP-SNP interactions discovered by logic regression explain Crohn's disease genetics. PloS One **7**(10), e43035 (2012)
3. Fan, R., et al.: Entropy-based information gain approaches to detect and to characterize gene-gene and gene-environment interactions/correlations of complex diseases. Genet. Epidemiol. **35**(7), 706–721 (2011)
4. Gallego-Sánchez, D., et al.: Parallel multi-objective optimization for high-order epistasis detection. In: Ibrahim, S., Choo, K.K.R., Yan, Z., Pedrycz, W. (eds.) ICA3PP 2017. LNCS, vol. 10393, pp. 523–532. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-65482-9_38
5. Gonçalves, F., et al.: Parallel evolutionary computation for multiobjective gene interaction analysis. J. Comput. Sci. **40**(101068), 1–15 (2020)
6. González-Domínguez, J., Schmidt, B.: GPU-accelerated exhaustive search for third-order epistatic interactions in case-control studies. J. Comput. Sci. **8**, 93–100 (2015)
7. Intel Corporation: The Compute Architecture of Intel$^{®}$ Processor Graphics Gen9 (2015)
8. Jing, P.J., Shen, H.B.: MACOED: a multi-objective ant colony optimization algorithm for SNP epistasis detection in genome-wide association studies. Bioinformatics **31**(5), 634–641 (2015)

9. Joubert, W., et al.: Attacking the opioid epidemic: determining the epistatic and pleiotropic genetic architectures for chronic pain and opioid addiction. In: SC18: International Conference for High Performance Computing, Networking, Storage and Analysis, pp. 1–57. IEEE (2018)
10. Kässens, J.C., et al.: High-speed exhaustive 3-locus interaction epistasis analysis on FPGAs. J. Comput. Sci. **9**, 131–136 (2015)
11. Li, X., et al.: Nature-inspired multiobjective epistasis elucidation from genome-wide association studies. IEEE/ACM T. Comput. Biol. (2018). https://doi.org/10.1109/TCBB.2018.2849759
12. Ritchie, M.D.: Finding the epistasis needles in the genome-wide haystack. In: Moore, Jason H., Williams, Scott M. (eds.) Epistasis. MMB, vol. 1253, pp. 19–33. Springer, New York (2015). https://doi.org/10.1007/978-1-4939-2155-3_2
13. Schüpbach, T., et al.: FastEpistasis: a high performance computing solution for quantitative trait epistasis. Bioinformatics **26**(11), 1468–1469 (2010)
14. Sun, J., et al.: Hidden risk genes with high-order intragenic epistasis in alzheimer's disease. J. Alzheimer's Dis. **41**(4), 1039–1056 (2014)
15. Visscher, P.M., et al.: 10 years of GWAS discovery: biology, function, and translation. Am. J. Hum. Genet. **101**(1), 5–22 (2017)
16. Wan, X., et al.: Boost: a fast approach to detecting gene-gene interactions in genome-wide case-control studies. Am. J. Hum. Genet. **87**(3), 325–340 (2010)
17. Weeks, N.T., et al.: High-performance epistasis detection in quantitative trait gwas. Int. J. High Perform. Comput. Appl. **32**(3), 321–336 (2018)
18. Yang, C., et al.: SNPHarvester: a filtering-based approach for detecting epistatic interactions in genome-wide association studies. Bioinformatics **25**(4), 504–511 (2009)
19. Yang, J.K., et al.: Interactions among related genes of renin-angiotensin system associated with type 2 diabetes. Diab. Care **33**(10), 2271–2273 (2010)