doi:10.1093/comjnl/bxx114

MC64-Cluster: Many-Core CPU Cluster Architecture and Performance Analysis in B-Tree Searches

Francisco José Esteban[1*], David Díaz[2], Pilar Hernández[3], Juan Antonio Caballero[4], Gabriel Dorado[5†] and Sergio Gálvez[2†]

[1]*Servicio de Informática, Edificio Ramón y Cajal, Campus Rabanales, Universidad de Córdoba, 14071 Córdoba, Spain*
[2]*Dep. Lenguajes y Ciencias de la Computación, ETSI Informática, Campus de Teatinos, Universidad de Málaga, 29071 Málaga, Spain*
[3]*Instituto de Agricultura Sostenible (IAS-CSIC), Alameda del Obispo s/n, 14080 Córdoba, Spain*
[4]*Dep. Estadística, Campus Rabanales C2-20N, Universidad de Córdoba, 14071 Córdoba, Spain*
[5]*Dep. Bioquímica y Biología Molecular, Campus Rabanales C6-1-E17, Campus de Excelencia Internacional Agroalimentario (ceiA3), Universidad de Córdoba, 14071 Córdoba, Spain*
[*]*Corresponding author: fjesteban@uco.es*
[†]*Authors who contributed to the project leadership.*

The MC64-Cluster computer platform was designed, based on many-core CPU microprocessors: Tile64. MC64-Cluster architecture was outlined in terms of both hardware and software, including commands available to manage jobs and provided application programming interfaces to communicate and synchronize tiles, making this system easy to use. ~~Massively, concurrent searches~~ of keys in B-trees, which are used in many applications, including bioinformatics, were used. Remarkable performance improvements were obtained when the cluster resources were combined with those available in host machine (hybrid or heterogeneous environments). These results were even more outstanding when analyzed in terms of performance-per-watt, highlighting their green-computing advantages. Together with the cluster architecture, they represent the main contributions of this work. To our knowledge, this is the first cluster implementation of this kind being developed.

Keywords: parallel programming; high-performance computing; hardware architecture; hybrid computing; green computing; bioinformatics; B-tree

Received 29 November 2016; revised 19 September 2017; editorial decision 2 November 2017
Handling editor: Antonio Fernandez Anta

1. INTRODUCTION

Recently, we have introduced the cluster concept [1] of Tilera coprocessor cards, as a 'group of tightly- or loosely-coupled computers'. Thus, in our previous proposal for future developments [2], MC64-Cluster connected several coprocessors through their built-in 10 Gigabit (Gbit; Gbps) Ethernet interfaces. That represented a natural extension to overcome intrinsic limits of a single card, as observed during our previous developments. Such a cluster was composed by TILExpress-20 G cards, inserted as coprocessors into passive personal computer (PC)-hosts. Each card included an energy-efficient Tile64 microprocessor with 64 reduced instruction-set computing (RISC) tiles/cores. The new MC64-Cluster aims to take the concept of high-performance computing (HPC) clusters as a model to provide increased performance. That is accomplished by splitting computational tasks across several nodes. However, the term HPC is used here not to describe the MC64-Cluster, but just in a methodological way, in relation to the memory and network bandwidth available in our building blocks. These microprocessor architectures have a bright future ahead. Indeed, Tile64 microprocessor architecture has inspired new chips like Adapteva Epiphany III and IV (including floating-point support and a 28-nm technology at a much lower cost, albeit lower computing power) [3], as well as the recently announced Epiphany V, using 16 nm lithography. Close technologies, like ~~massively parallel~~ processor array (MPPA),

also appeared at the time, with microprocessors from companies like Ambric or PicoChip, and more recently with products like MPPA-256 (Andey) from Kalray [4] and newer MPPA2-256 (Bostan).

Mellanox Technologies purchased EZchip Semiconductor (which formerly acquired Tilera) to merge such technology with EZchip high-performance network processing. This will allow building new innovative chips to address many present and future computing challenges. Nonetheless, they are not the goal of this work, focusing instead on commercially available many-core central ~~processing units~~ (CPU). Likewise, other approaches in the field of specific computing, like field-programmable gate array (FPGA), general purpose graphics-processing units (GPGPU), cell broadband-engine (CBE), etc. fall outside our approach, because of the lack of flexibility caused by their coupling hardware–software model.

Other architectures, like the one implemented in Godson-T microprocessor [5] could be comparable. This is due to the use of million instructions per second (MIPS) and a matrix of cores, in a somewhat similar way as Tilera does. However, such former architecture, currently renamed as Loongson, has been implemented in multi-core CPU only, with a number of cores that ranges from two to eight.

MC64-Cluster was developed its performance tested when searching B-tree structures. This is an operation widely used in many computing areas, including, among others, general-database management and bioinformatics. The latter includes algorithms like the ubiquitous Basic Local-Alignment Search Tool (BLAST) [6, 7]. It is considered the most popular application by life-science researchers, in which the index usage and optimization are active-research areas [8]. A progressive-refine path was followed, with the aim to put all processing elements at work, and therefore reducing idle times as much as possible. Thus, from a first approach in which all tiles received the whole set of keys to search for, an optimized strategy was implemented, with a previous sort-and-split executed in the host machine. Such strategy has proved to be the most efficient approach to the problem. Other alternative approaches are also discussed below, like the ones using an additional thread to support the most demanding tasks. Execution of the same algorithm was also carried out in Intel Xeon E5405 microprocessor, which has four cores running at 2.0 GHz, with 8 GB of double-data rate type two (DDR2) memory. Additionally, many-core Intel Xeon Phi 31S1P with 57 cores at 1.10 GHz, with 8 GB Graphics double-data rate type-five (GDDR5) memory was also used. Both have Intel x86 architecture, which is the standard one in PC, and thus were used as reference microprocessors. For obvious reasons, benchmarking against any possible parallel architecture is beyond the purpose of this work. Our main contributions are the building of the cluster and its performance results (including performance-per-watt; PPW) for this particular problem.

2. RELATED WORK

The appearance of a microprocessor element boarded in a card, and acting more or less in conjunction with the host CPU, has its immediate antecessor in specialized microprocessors, like GPU. These systems have eventually evolved into general-purpose capable microprocessors, although maintaining their primary design, based in a great amount of ~~hierarchically organized~~ lightweight microprocessors [9]. Therefore, their programming techniques and optimizations are quite different and specific, in spite of recent efforts to unify CPU and GPU [10, 11]. Some clusters with this ~~kinds~~ of elements are described elsewhere [12–14], and they are even available as cloud solutions, like Amazon Elastic Computing Cloud (EC2) service <https://aws.amazon.com/ec2> [15]. Furthermore, tools to provide elastic virtualized clusters on top of these kind of platforms can be found [16, 17]. They offer unprecedented flexibility in the resources used, at the cost of extra computing-resource consumption, which is inherent to any virtualized solution.

On the other hand, Intel has introduced Many Integrated Core (MIC) architecture into Xeon Phi cards that, depending on the model, contain between 57 and 72 cores. Some works with multiple interconnected Xeon Phi systems have been described [18, 19]. Both Tile64 and Xeon Phi are boarded on Peripheral Component-Interconnect express (PCIe) cards that may work inside standard desktop PC hosts, so they are usually referred as coprocessors. It is remarkable that Intel Xeon Phi card requires a server or a very specific motherboard that fits with its specifications, being also less energetically efficient than Tilera one.

Clusters have been already applied to bioinformatics [20, 21], but it is difficult to find implementations in which many-core CPU (or even multicore) technologies are involved. There is a recent BLAST implementation in a Xeon Phi Cluster, called HPC-BLAST [22], being a general implementation of wavefront-algorithm suitable for many fields [23]. Heterogeneous clusters using Zynq boards from Xilinx have also been described to analyze sequencing data [24]. Our current study shares ideas with those three approaches, maintaining its uniqueness in the use of Tilera cards.

Xeon Phi performance [25] was near the same than that of Tile64 when only integers were involved. In particular, any difference in time execution of a launched program in these cards was not due to their complex instruction-set computing (CISC) vs. RISC architectures, respectively. Instead, they derived from different clock rates and number of tiles/cores used. Therefore, a rewriting of existing programs to take advantage of vectorization capabilities of such microprocessor is mandatory to take full advantage of Xeon Phi. Tile64 represents an interesting approach in the many-core CPU arena, mostly because of its well-established PPW goodness [26]. It should be taken into account that RISC is a proven

technology, used by modern mobile devices where PPW may be critical to prevent overheating and save battery life, like mobile phones and tablets. Besides, Tile64 has one of the highest core-counts per CPU (up to 72).

Second-generation Xeon Phi, known as Knights Landing (KNL), no longer consists of coprocessors, but regular microprocessors. It has great advantages over first generation Knights Corner (KNC) regarding memory management and speed access. However, Tilera vs. Intel coprocessor comparison is more balanced than that between a coprocessor and a pure microprocessor. KNL would be a mandatory reference if comparing 8- or 16-bit data (not just 32 bit). That is due to its brand new instruction set, albeit that does not fit into the scope of this work.

We have previously demonstrated the usefulness of Tile64 microprocessors when applied to common bioinformatics tasks [27–29]. Processing nucleic-acid or peptide (protein) sequences usually consists of read-assemblies to generate consensus contigs (both for de novo sequencing and re-sequencing), as well as pairwise and multiple alignments to compare sequences (in order to find differences, find identities, build dendrograms or phylogenetic tress, etc). As expected, the best results were achieved when both hardware and software peculiarities were taken into account, which highlights the critical relevance of such methodological considerations to optimize performance.

3. MC64-CLUSTER DESIGN AND ARCHITECTURE

Different hardware and software elements were developed to build the cluster system. The goal was to obtain a platform as transparent as possible, so that the users feel that they are dealing with a single computer. That is a key cluster characteristic. We have previously described the main guidelines to create the MC64-Cluster [2]. In the present work, it is implemented from both the hardware and software points of view.

3.1. Hardware-setup elements

The present work was carried out with Tile64 microprocessors from Tilera [30]. Each of them contains 64 cores, called tiles by the manufacturer, packed on a 90 nm System-on-Chip (SoC), operating at a work frequency from 500 MHz to 866 MHz. Each tile is based on RISC technology, with a 32-bit word size and no hardware-support for floating-point instructions. Base software for the microprocessor is a customized full-Linux kernel, so that any general-purpose application can be run unmodified in this platform. Additionally, specialized ~~application programming~~ interfaces (API) are provided to take full advantage of its intrinsic parallelism (see next section). In addition, tiles are internally connected by a low-latency network branded as intelligent Mesh (iMesh), with an aggregated bandwidth of 31 Terabits per second (Tbps). Tile64 microprocessor boarded in PCIe cards (TILExpress-20 G) were used with tiles running at 866 MHz, each including 8 GB of shared ~~random access~~ memory (RAM) and two 10GBase-CX4 Ethernet connectors. This kind of ports is internally composed by four coaxial wires, being the first commercially available copper-based 10 Gbit Ethernet implementation [31].

3.2. Software-setup elements

Available programming languages in the above-cited Linux implementation are C and C++. Development process was achieved via Tilera Multicore Development Environment (MDE) version 1.3.5. That is an x86 Eclipse-based platform, enriched with a cross-compiler into native Tile64 RISC code, integrated debugger and profiler. It also has a deploy utility for both Tile64 hardware and software emulator. Though new versions of MDE are currently available, they do not provide additional features useful to our developments.

Programming environment supports two API, allowing executing parallel algorithms and communicating concurrent processes. The first one (iLib) provides functions to start execution in any tile with its corresponding program, as well as several sets of functions to communicate tiles by using different approaches: shared memory, channels, message passing and program-execution coordination. The second one (NetIO) gives a low-level access to Ethernet ports, in order to take full advantage of their native speed. In the case of 10 Gbps, this implies a quick CPU-cycle pace between frame arrivals, requiring routines in this library to be fast enough to efficiently attend network events. It is also expected that a single tile could not manage the entire stream of data at this speed, so parallelism is required to reach wire speed.

Both libraries are specific developments from manufacturer, and, although they resemble concepts present in similar message passing or communication libraries, there is no standardization at all in them. Therefore, existing programs must be rewritten to be executed in this architecture. To complete these two libraries, a high-level ~~transmission control~~ protocol/internet protocol (TCP/IP) driver is also provided. Thus, a well-known network-socket programming may be used whenever a real-time response to the network events is not required.

A scalable many-core architecture for MC64-Cluster was built using 10 Gbps Ethernet ports available in TILExpress-20 G cards, allowing the 64 tiles of each card to be escalated to hundreds or even thousands of them. In order to achieve this goal, software components were developed to give the necessary global vision to the system. These components can be grouped into two main areas: (i) cluster administration, in which the most widely used commands for resource manager (RM) and advanced scheduler (AS) were included; and (ii) inter-process communication, in which the main functions

available in message-passing libraries were provided, along with native Tilera mechanism-extensions. For the first one, Terascale Open-Source Resource and the QUEue Manager (TORQUE) were used as a model, so the command-line oriented tools follow its terminology. The key-consideration was to control cluster resources, defining its geometry, knowing its state, executing jobs sent by users, distributing them through available cards/tiles and controlling their states and stages, providing at least one specific command to accomplish each of those tasks.

In order to get the least intrusive and most flexible approach, a client/server paradigm was chosen, outlined in Fig. 1. Using a set of command-line interface (CLI) tools, the user can specify the details for each request, and then send jobs to the master–server to be executed. A web interface is currently under development, in order to make cluster usage and management even easier. On the contrary, server cards of the cluster work in a master/slave configuration. Every server is initialized from internal configuration files, in which the initial geometry of the cluster is defined, including the number of available tiles in each server card. The server itself occupies a tile, and other eight tiles are reserved by the system for coordination and administrative tasks, including attending 10 Gbit ports, so the number of available tiles for user jobs is 55 per card. The types of jobs that can be launched in MC64-Cluster are listed in a properties file, so the system can be easily enhanced just by including and declaring new executable programs in such a file, keeping its modularity and minimal-intrusion features.

In the conversation that takes place in a typical command execution, the client sends command details to the first card in the server (acting as master–server). Then, such master–server communicates with remaining cards (slave–servers). Thus, the command gets executed in tiles controlled by each server (all master and slaves). Finally, command results are sent back to the client tool, so that it can deliver operation results to users. All communications between clients and servers are accomplished by means of TCP sessions, taking advantage of the built-in ~~native socket driver~~, in the cards. Although user-datagram protocol (UDP) would be available too, it is not usual in message-passing interface (MPI) implementations, due to the reliability required in MPI transactions. There are ongoing works showing alternatives to TCP in MPI implementations [32, 33], but their use is beyond the goals of

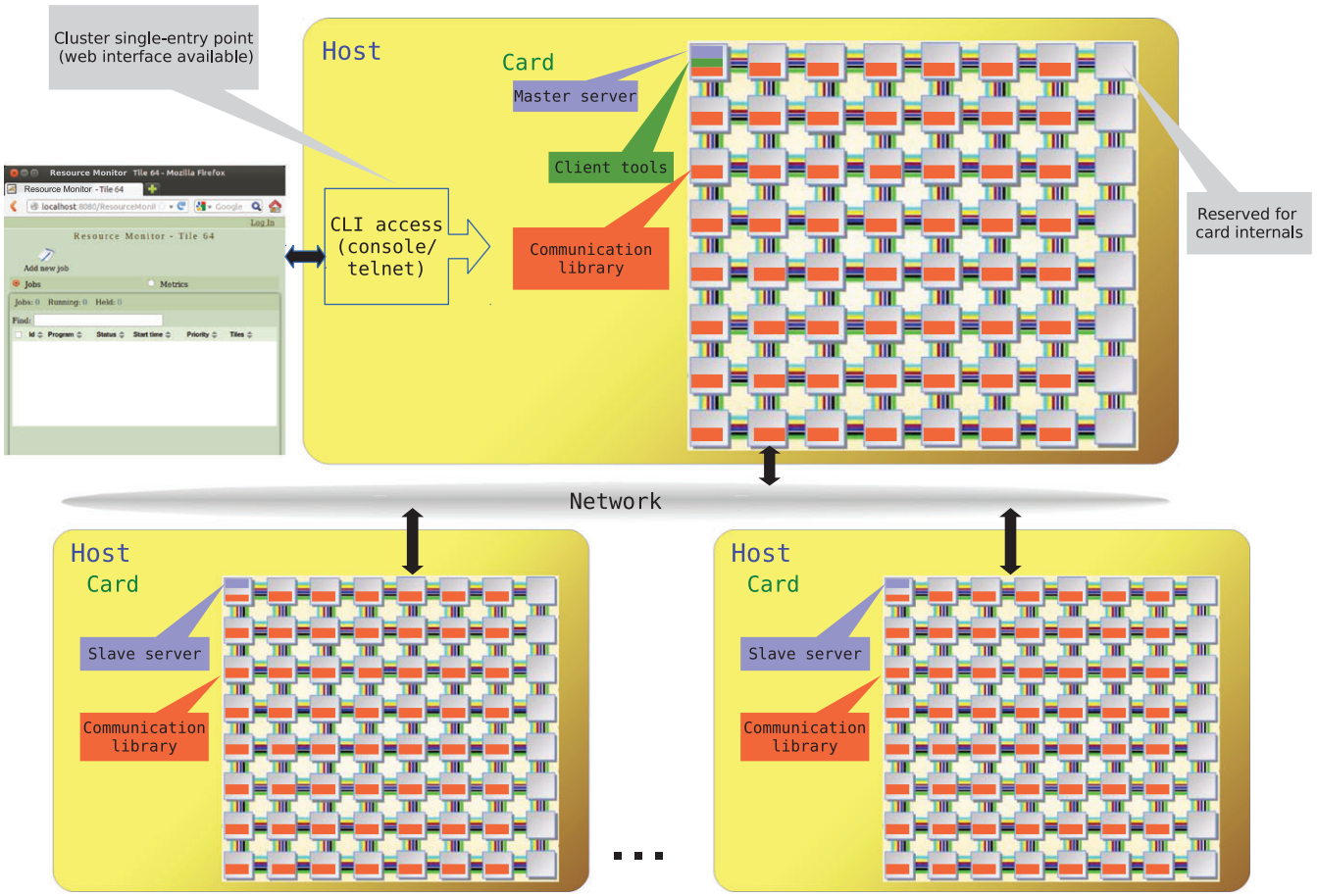

FIGURE 1. Cluster topology.

our work. Availability of several cards can contribute to avoid memory bottlenecks in memory-intensive executions, by means of instructing the job manager to distribute workload among as many different cards as possible.

A question/answer-like protocol was developed, in which a typical question contains a command code, followed by its applicable options and data associated with the question, if required. If these data are simple enough, then a single packet may be used to send them. Otherwise (i.e. when data are complex or have variable sizes), a further client/server conversation is established, in order to send them fragmented into the required number of packets. Along with this information exchange, a file exchange can eventually take place, in order to make data and executable files available to all cards participating in a job, or to retrieve back generated job results to the master card.

The inter-process communication component uses MPI specification as a model, and extends it with native iLib-channel features, to take the most out of the system. This task has been achieved by means of a newly developed library, in which a minimal set of MPI functions are offered. The rationale behind this strategy is to give an opportunity to existing MPI-based programs to be compiled and run unmodified in the MC64-Cluster, assuming that their complexity rests on the limits of our implementation. This may be considered also a contribution of our research and development work to Tile64 usability, since a native-MPI implementation is not provided by the manufacturer. Of course, other standard ~~communication mechanisms~~ like Unix sockets, TCP/IP sockets and threads are also available. But the advantage of using a cluster-specific library is abstraction of system topology. Thus, each participating process can communicate with each other, just by using the partner identifier (known as 'rank' in MPI terminology). The underlying library, developed in a modular fashion, encapsulates this communication in the available way: an iLib communication through iMesh internal network or a TCP session, through the 10 Gbps Ethernet inter-card network. Additionally, the presence of iLib-like channel-communication functions makes it easier to port existing one-card native Tile64 programs to the cluster.

Regarding API internals, the most complex task has been the development of TCP-session control code. Thus, when tiles participating in communications belong to the same card, a further call to the provided iLib-equivalent function allows almost all the job to be completed. Figure 2 graphically shows these different kinds of communications. They start when a client ~~execution command~~ arrives at the master–server process, by means of a TCP session. As with most server implementations, a new server process is forked in order to complete the request. This is a key feature in our case, because it will activate native iLib mechanisms (represented in the ~~figure~~ by the spawn calls), to launch the executables required by the user job. These spawns take place both in master and slave cards, after the needed TCP sessions to the slave cards are started. Once the job is running, forked processes are not needed, and therefore they are finalized, so that any further communication between tiles is directly delivered, either using low-level iLib API or by means of TCP functions, depending on participant locations.

These latter communications can imply additional requests to master–servers, in order to query available resources, or to coordinate executions. These different kinds of communications are carried out by our library internals, thus being fully transparent to the programmer. In addition to pure communication functions, MC64-Cluster API also provides the most-common utilities for process management and synchronization, allowing a complete ~~parallel environment~~.

4. B-TREES

Once MC64-Cluster internals were fully outlined as described above, the first complete software development for it is indicated below. This includes several implementations and a comparison of its performance to that of other many-core platforms.

Several approaches around B-trees data types are outlined below, so the set of tests presented can provide a wider outlook of the possibilities of the cluster from the performance point of view. Issues related to large-scale pairwise alignments, their relationships with B-trees and particularities of our implementation are also described in the next sections. These developments can be applied in a wide variety of areas.

4.1. General remarks

Alignment of two sequences/strings of letters is a process that provides a measure of their similarity (score). As we have previously described, it is one of the most ~~widely used~~ operations in bioinformatics, being the basic block of ~~higher level~~ tasks, as phylogenetic analyses [34], multiple alignments [35], chromosome assemblies [36] and comparative genomics [37]. In particular, an alignment can be carried out either (i) following an optimal approach (like the Smith-Waterman algorithm does), relying on time-consuming generation of a Dynamic-Programming Matrix (DPM); or (ii) using a heuristic strategy, like Fast Alignment Sequence Tools (FAST)-All (FASTA) and BLAST tools. Heuristics are common in second-generation sequencing (SGS) and third-generation sequencing (TGS) platforms of nucleic acids, sometimes known with the ambiguous name of next-generation sequencing (NGS), where the large amount of pairwise comparisons makes unfeasible the deployment of DPM-based methods.

The current SGS machines, like Roche 454 or Illumina NovaSeq cannot sequence full genomes (which may have milliards of base pairs) in a single or a few reads. So, many

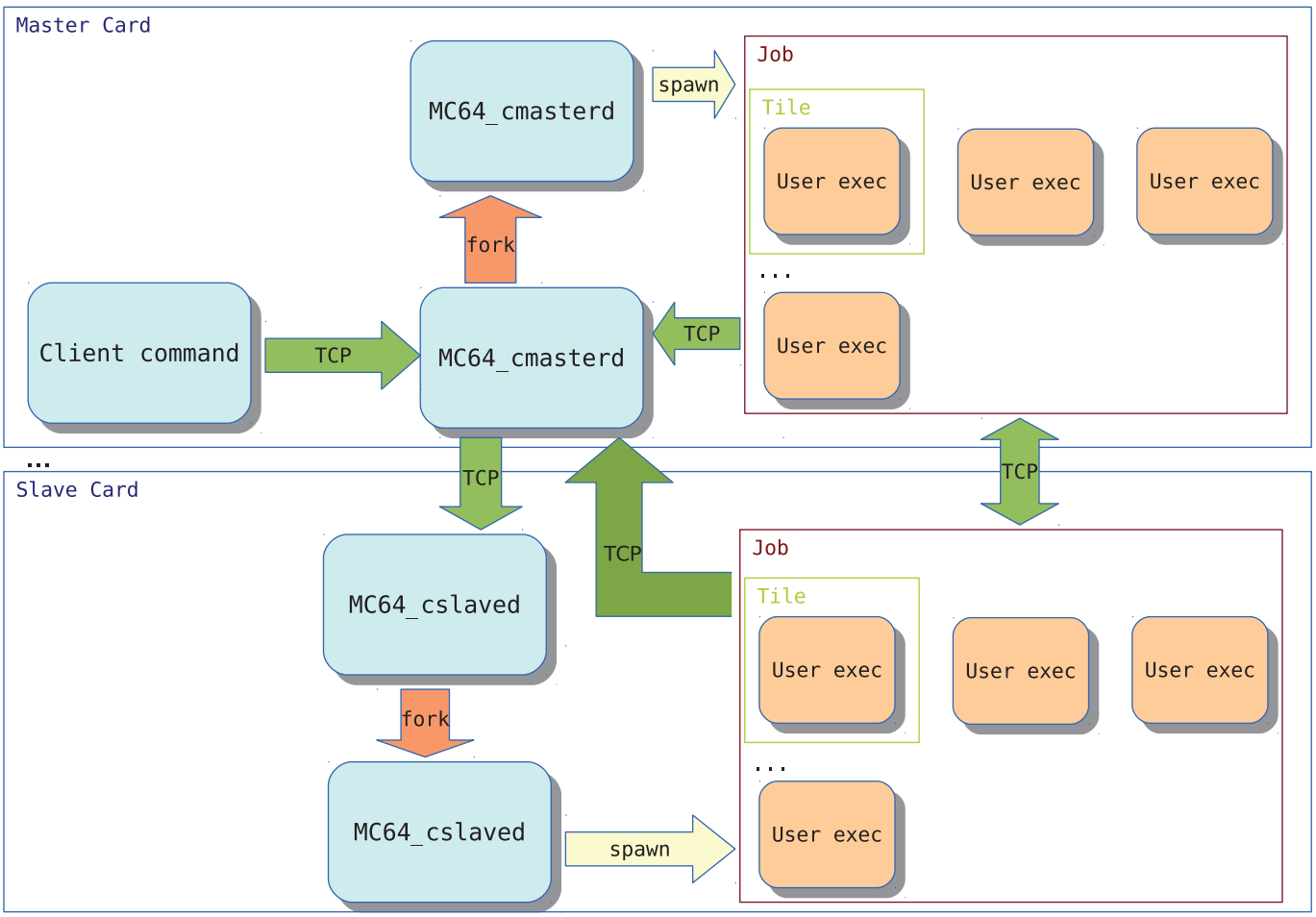

FIGURE 2. Cluster ~~internal architecture~~.

copies of the genome are broken down into small pieces and read with such sequencing platforms, which generate from about 50 bases (b) to one kilobase (kb) reads. Those milliards of short DNA reads (made of A, C, G and T nucleotide residues) may represent a daunting maze to assemble into contigs, chromosomes and genomes. This is particularly relevant for repetitive or homopolymer stretches. Different strategies have been devised to solve the problem of reconstructing original chromosomes/genomes—mostly in the absence of reference ones—using overlapping and redundant reads. Pairwise alignments allow discovering overlapping sequences; i.e. sequences whose similarity at the ends is greater than or equal to a threshold. This allows to align such reads into contigs, scaffolds, super-scaffolds and eventually full chromosomes/genomes.

Heuristic methods are employed when a sequence must be aligned against many others (e.g. several millions). They are usually based on particular database indexes, where sequences are stored. This process is used because alignments of interest are those whose score is greater than or equal to a minimum, and thus looking for such matches is similar to looking for them in an index. The techniques used to represent sequences and to create indexes depend on each tool used, but the most precise ones (like MegaBLAST) divide every sequence into *k*-mers, and store them in some type of index [6, 38].

Optimization of massive-search algorithms is a key area in many current scenarios. Concerning sequence alignments, there are several proposals related to such task in BLAST variants [39]. Although a suffix tree is the most prevalent approach when dealing with sequence data, due to its string nature, a B-tree could be a good alternative when numeric derivations are involved, like in distance-based approaches [40] or for hashing ones [41]. On the other hand, most existing assemblers use graph-based approaches for sequence assembly. But there are also promising developments based in massive local-searches [42].

To test the performance of our MC64-Cluster approach, a well-known search-tree type suitable for this purpose was implemented: B-tree. At the same time, the focus was on

main operations used in this particular problem; i.e. generation of a complete B-tree from scratch and usage of large-scale searches. It must be noted that insertions/deletions (InDels) are meaningless in this development, since the focus are searches. In particular, the uniform probabilistic distribution of keys is assumed. Indeed, it has been reported that distribution of nucleotides in DNA is usually quite uniform, with deviations under 5% from perfect uniformity [43], and therefore, the distribution of sequences to search for.

4.2. Implementation

B-tree search was selected as a proof-of-concept, where several ~~slightly different~~ tests can be run, in order to measure performance of MC64-Cluster under different circumstances. Furthermore, a split B-tree allows massive searches and scales almost linearly when the number of microprocessors is increased. Performance of this particular implementation of parallel B-tree searches in MC64-Cluster was evaluated. As said, Intel ×86 was used as reference microprocessor, on the same host hardware used to build the cluster. Such ×86 implementation was not parallelized because, in such a case, performance would depend on the particular number of cores (and multithreading capabilities) of the microprocessor used. In fact, its clock speed (2.0 GHz in our tests) must be considered for comparison purposes. In addition to this, our work was focused on a many-core CPU SoC environment, where threads are avoided and a process –developed in ANSI C– is executed by a dedicated CPU. Thus, a comparison with a GPU, FPGA or any hybrid environment is not relevant here, because of the rather different ~~processing units~~ and programming methodologies involved. On the other hand, Intel Xeon Phi boards are currently in mainstream, so they were also used as reference architecture.

B-tree data structure was first proposed by Bayer and McCreight [44], as a method to store pairs of key-data with logarithmic profile in time for any operation: insert, delete and search. This profile implies that the time required to find a given key is maintained between acceptable limits when the number of keys grows. Thus, it has been widely used in database deployments [45]. It takes also advantage of the fact that retrieving a single value from a magnetic storage disk takes basically the same time than retrieving the complete block of data where such single value is located. In the field of bioinformatics, different ~~tree structures~~ are widely used in heuristic ~~alignment methods~~ [46]. Other feature of B-tree structure that contributes to its high-performance is the fact that the tree keeps well-balanced when Indels are involved. This is accomplished by splitting full nodes when an additional insertion is required, as well as merging adjacent nodes when a deletion makes a node to contain fewer items than a predefined threshold.

Starting from this classic B-tree implementation, the basic ideas in B-tree structure were adapted in this work to the developed ~~parallel cluster~~ environment, establishing the following design criteria (Fig. 3 shows such a structure in detail):

(a) ~~*Equally sized*~~ *distribution.* In our parallelization, the list of keys is partitioned into a number of contiguous and equally sized sublists (with a difference of ±1) that matches the number of available tiles in the cluster. Each tile receives a different sublist, from which an independent B-tree is built. So, it is in charge of answering queries for values that range from its minimum to the minimum of next tile. In order to cover the whole ~~data range,~~ the first tile answers also from the absolute minimum up to its minimum, whereas the last tile answers also from its maximum up to the absolute maximum, so introducing the 'effective-range' concept. Numbers were chosen as keys instead of alphanumeric characters (as we can expect when dealing with nucleic-acid or peptide sequences) to make the 'ordered' approach clearer (this decision is irrelevant for the final outcome, but it was implemented here for a better understanding).

(b) *Null-pointer suppression at node and leaf levels.* In order to save memory, null pointers between values that have no descendants were suppressed, both at node and leaf levels, so valuable memory-space can be saved. The search algorithm was modified to implement such approach.

(c) *Filled-up nodes*. The strategy used by every tile to build its B-tree fills completely the nodes at the deepest levels. The result is a tree with minimum number of nodes. Therefore, a single operation of taking a new node from the main memory to cache due, for instance, to a fault, achieves maximum performance, because it retrieves as many keys as possible. This strategy guarantees that all nodes in a level are filled-up, except the last one. A tile does not create its tree by inserting keys one-by-one, but by partitioning its particular sequence of keys into equal slices, whose length is calculated in advance, using a simple calculus based on the order of nodes. No InDels are made after the tree has been built (this is the general case in many large-scale search algorithms).

(d) *Long order and binary search at node level.* A strategy for high-performance developments in Tile64 microprocessor is to take advantage of its cache structure, and to reduce node-cache faults to the minimum. As a consequence, our B-tree node is long enough to allow binary searches inside a node to be more efficient than usual sequential searches. This is due to the fact that the most populated nodes are in

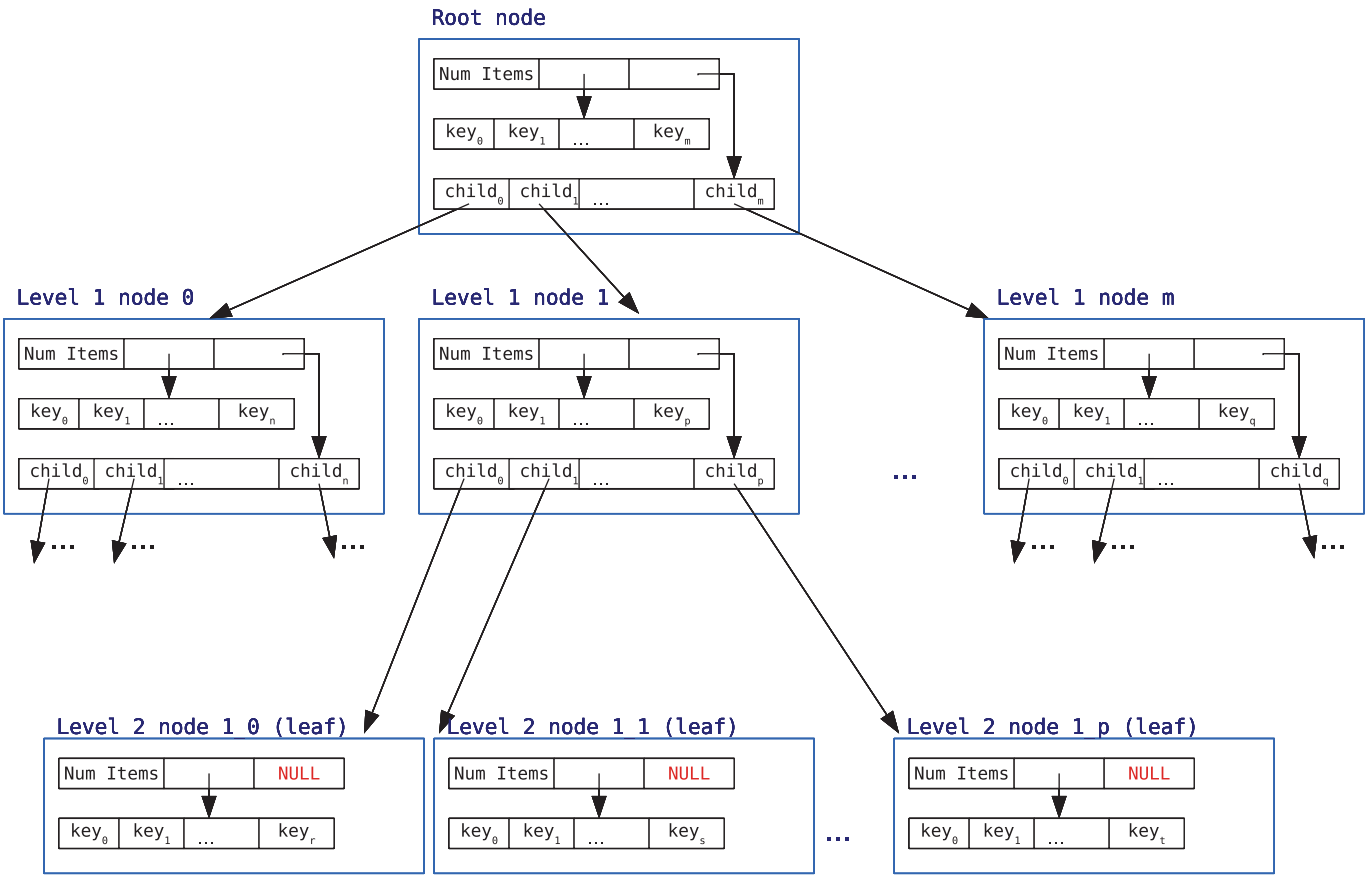

FIGURE 3. B-tree data structure used to test performance of MC64-Cluster. Note: In our implementation, $n = m = p = q =$ order*2.

the deepest level, so few intermediate nodes remain in cache for long periods of time.

(e) *No data are associated to a key.* In order to keep memory load at its minimum, the B-tree does not store pairs (key, value) but only keys to search for. Therefore, the work was carried out with a minimal B-tree implementation, where the hits should be managed in a second stage of any algorithm that uses the B-tree search.

Thus, these adaptations have been chosen taking into account Tile64 microprocessor internals; mainly, cache and share-memory limitations, and constitute the difference between our proposed strategy versus previous works. Therefore, the same performance results may not be expected when such an implementation (or any fine-tuned one) is executed in another hardware architecture different from many-core CPU SoC. For comparison purposes only, time executions obtained with Xeon Phi 31S1P cards were included, as described in Section 2 (Related work). In addition, this same ×86 implementation has been tested in a standalone PC, in order to obtain a reference ~~execution time~~ easily reproducible with standard hardware requirements.

In this implementation, keys are distributed among available cluster cards, by means of MC64-Cluster built-in job-execution system. It transfers the whole content of a given job directory to all participant cards in such a job. This task may be a bottleneck in TILExpress-20 G, as its native-socket implementation fails to get the most of available 10 Gbps. Probably due to this fact, other Tilera cards mount 10 ports at 1 Gbps each, instead of 1 port at 10 Gbps.

It should be noted that both the distribution of keys and that of values to search for are the same because, in a real situation, there is a database of nucleic-acid or peptide substrings as keys, and another set of DNA strings as values to search for. Although such a distribution is uniform, the approach used here can be applied to any other distribution, because the range of keys received by every tile contains the same amount of non-overlapping values. In other words, the ordered sequence of keys was not split into intervals of the same length, but into intervals with the same number of keys (these concepts match when uniform distributions are used). Therefore, if the set of values to search for follows the same distribution than the sequence of keys, then every interval will receive a similar amount of search operations. Figure 4 shows the general workflow chart, describing how the proposed strategy works.

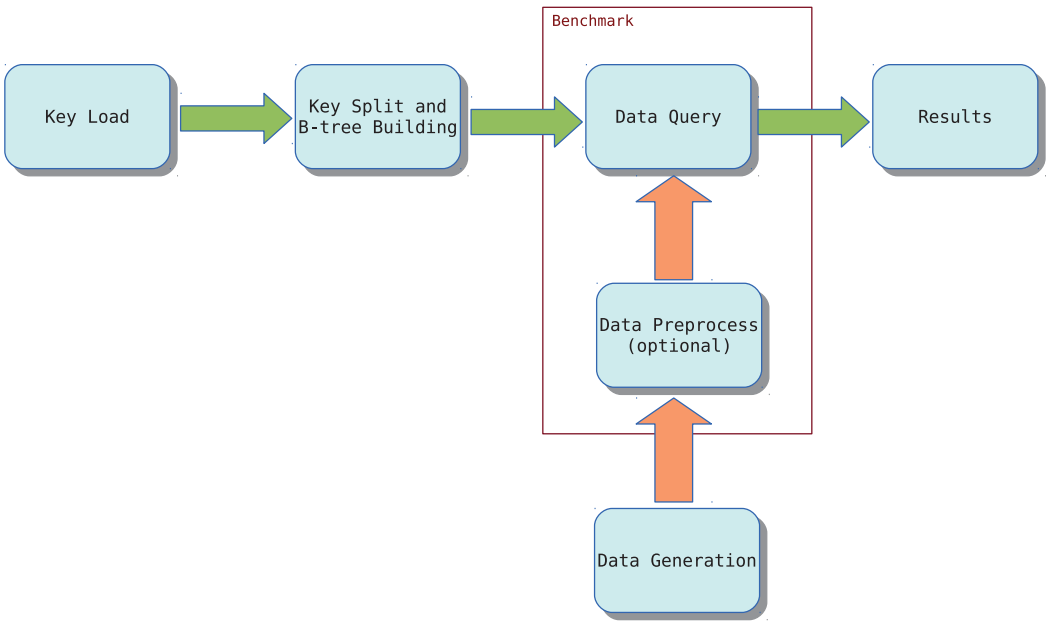

FIGURE 4. General benchmark workflow.

Tests have been performed in a two-card cluster, with a scheduling approach equally distributing load between them. This is the most efficient strategy in this given scenario, because of the minimum communication requirements between executables, further reducing memory requirements. In addition, this approach takes advantage of cache structure in Tile64. Of course, different strategies may be more appropriated in other situations. For instance, if the communications between processes are a key requirement, grouping the most-active communicating processes in the same card would offer better results. This is due to the above-mentioned fact that the internal iMesh network is by far faster (31 Tbps) than the 10 Gbps Ethernet inter-card network.

5. RESULTS AND DISCUSSION

MC64-Cluster was developed, performing two main sets of evaluation tests to obtain the corresponding benchmarks. In the first one, a data set of trees with sizes from 1 000 to 10 million keys was generated, with increments of one order of magnitude. As mentioned above, it has been considered that nucleotides in nucleic-acid strings follow a uniform distribution [43], so the keys were also uniformly distributed, by setting the distance between two consecutive keys as a random value between one and 49. With this technique, a set of keys that ranged from zero to ~25 × size of tree were obtained. Thus, the probability for any random value in such a range to be in the tree is 4%. In this scenario, the query data set consisted of a sequence of values generated on-the-fly, following the same uniform distribution. They ranged from zero to the highest expected value in the tree, being separated for a quarter of the maximum separation between keys. With this search strategy, the range in which data was expected to be located was fully covered, and both keys and queries were equally distributed across all available tiles. Thus, the effective key range had a similar size in all tiles, with the exception of the first and last ones. Xeon Phi 31S1P tests have been carried out both with sequential and binary intra-node searches. In addition, the full computing power of this card has been tested by launching 114 threads in every test; i.e. two threads per core (an execution with four threads per core had an inferior performance). As a reference, an identical B-tree implementation was used with the same order, keys and search data executed, in a workstation equipped with an Intel Xeon E5405, using a single core of such quad-core microprocessor in this reference ~~sequential test.~~ All results presented from this point on exclusively refer to the search operations.

Table 1 shows the obtained results. The optimized implementation had a better performance in Tilera than Intel from the very beginning. Besides, our cluster implementation scaled well as more tiles were added to calculi with only one exception, located around 110-tiles execution with short trees. These anomalies were associated to known memory-contention issues in the card, which tends to have less weight in overall result as the number of searches increases. For those larger-tree sizes and increasing number of searches, tests showed an almost proportional time-to-size behavior. Maximum speed-up was obtained with one-million-key tree, showing a 104.82x gain when all available tiles in cluster were used. ~~Figure 4~~ graphically shows these speedups. Xeon Phi 31S1P execution

TABLE 1. Execution times in fixed sequence. B-tree searches with values are generated on the fly. The number of performed searches is directly proportional to tree size in each test.

Execution times using a sequence between tree limits*									
Tree size	Processor elements								
	Intel Xeon E5405	MC64-Cluster (Tilera)							Intel Xeon Phi 31S1P
	1	2	4	8	16	32	64	110	57 × 2
1000	356	251	127	73	49	41	50	156	9
10 000	4731	3330	1579	770	328	213	124	148	101
100 000	86 161	39 499	18 895	9 013	4 405	2 026	1 286	822	1071
1 000 000	642 473	455 571	218 675	106 075	50 199	24 084	11 840	6839	11 618
10 000 000	7 659 499	5 541 621	2 682 674	1 290 884	579 550	278 227	133,402	75 826	121 255

*All times in microseconds.

times were, generally, slightly slower than those of Tile64 with similar number of cores in execution, in spite of having a higher clock speed (1.1 GHz vs. 866 MHz).

In a second set of tests, focus was set on input/output (I/O) performance. In this case, the query data set involved 10-million ~~randomly generated~~ integers stored in a shared file, searching for them in trees with different sizes. Once again, this key generation allowed a similarly effective-range size in all participant tiles, which is in the base of the scalability of this strategy. Three approaches to make every tile to read such a file were used. In the first one, a previous sort-and-split step was executed on file content. There were as many files generated as tiles, so that a given tile received only the file with values inside within its effective range. A well-known radix-sort algorithm was used to achieve this goal, executed in host machine (the same reference workstation cited above). This previous sorting had an additional advantage: it reduced possible ~~cache faults~~ during search phase, because of proximity of successive values to search for. Different ~~base values~~ were used to perform radix sorting, always choosing power-of-two (2^n) ones, in order to optimize the division phase of algorithm. As expected, our tests evidenced minimum sorting-times results from a base value of 4096 (2^{12}), because the size of integer was 2^{32}, and radix-sort algorithm could be executed with just three loops (32/12) and 'only' 4096 queues. It must be highlighted that the nature of the cluster makes relatively easy to carry out the different tests and approaches described, because they can be developed using standard methodologies of parallel programming and languages like C. This is one of the main advantages of MC64-Cluster when compared to many-core GPU ones.

Top section in Table 2 shows results obtained using this hybrid-computing approach; every execution time is divided into two parts: time taken to sort-and-split and time taken to search. Obviously, any attempt to improve searches in B-tree is limited by the sort-and-split phase. Indeed, this preprocessing step takes 6.65 s, in contrast to the best effective time taken to search: around 150 ms in MC64-Cluster and 83 ms in Xeon Phi 31S1P. However, this hybrid approach may provide the best performance, depending on characteristics of host and different implementations of sort-and-split step. In fact, when the same radix-sort implementation is executed on a PC with Intel Core i7-4820 K microprocessor at 3.70 GHz and 24 GB of RAM, execution time was reduced to 3 284 ms, and even to 1 609 ms (332 ms to read file + 336 ms to sort the set of values + 941 ms to write files), when a multi-threaded implementation was used. Furthermore, regarding Intel reference test, it is important to notice that the sorting step did not actually offer any performance improvement. In fact, as is shown in the next two approaches, when this phase was removed from the reference implementation, overall speed-up obtained by cluster execution was significantly increased ~~(Figure 5)~~. Q3

Other possible strategies have demonstrated to be less appropriated in this environment. Thus, in a second approach, searches where every tile reads the corresponding file in its entirety were performed, with a further selection of values in the range of interest by each tile. In this case, the read task showed a poor performance as compared to the search itself, even when carried out in a RAM-disk environment. Moving the I/O task into an additional thread (that reads keys and loads an internal buffer shared with the main thread) showed a slight benefit initially. But in a further third approach, when this strategy was used with big files and many tiles involved, performance was strongly penalized, since the same tiles got used for reading and searching tasks. Middle and bottom parts of Table 2 show results obtained by using these last two approaches (without sort-and-split-preprocess step). The additional thread technique was only implemented in MC64-Cluster and Xeon Phi 31S1P, so both Intel columns showed identical values. Xeon Phi 31S1P behaved as MC64-Cluster in both last approaches. In fact, speed-up of the third one was strongly dependent on I/O strategy and size of buffers used to read data (100 kB buffer size). However, when size of

TABLE 2. Execution times when reading an entire file in different situations.

Tree size	Processor elements								
	Intel Xeon E5405	MC64-Cluster (Tilera)							Intel Xeon Phi 31S1P
	1	2	4	8	16	32	64	110	57 × 2
	Execution times using 10-million random searches with sort-and-split preprocessing*								
1000	6650 + 3940	6650 + 1384	6650 + 679	6650 + 396	0 225	6650 + 326	6650 + 114	6650 + 160	6650 + 83
10 000	6650 + 6083	6650 + 1898	6650 + 942	6650 + 459	0 251	6650 + 135	6650 + 111	6650 + 119	6650 + 84
100 000	6650 + 6531	6650 + 2172	6650 + 1084	6650 + 536	0 322	6650 + 174	6650 + 110	6650 + 126	6650 + 83
1 000 000	6650 + 8620	6650 + 2458	6650 + 1238	6650 + 622	0 332	6650 + 202	6650 + 119	6650 + 131	6650 + 85
10 000 000	6650 + 9140	6650 + 2984	6650 + 1512	6650 + 749	0 401	6650 + 223	6650 + 184	6650 + 132	6650 + 83
	Execution times using 10-million random searches with a single thread*								
1 000	1006	1717	1228	1014	1 591	1248	2124	2899	3027
10 000	1003	2238	1483	1152	1 016	1260	2107	2919	3066
100 000	1017	2533	1613	1210	1 123	1285	2112	2930	3155
1 000 000	1187	2832	1768	1284	1 107	1320	2105	2938	3409
10 000 000	2922	3372	2036	1421	1 143	1315	2121	2978	3037
	Execution times using 10-million random searches with two threads*								
1 000	1006	1961	2216	2219	3 528	3644	3588	3931	6689
10 000	1003	1961	3802	3798	3 790	3048	3099	3927	6888
100 000	1017	5862	1960	3915	1 963	1961	2142	3925	6947
1 000 000	1187	1964	2744	2841	2 730	3679	3719	4676	7403
10 000 000	2922	1964	1961	4133	4 194	4354	3249	4828	7248

*All times in milliseconds.

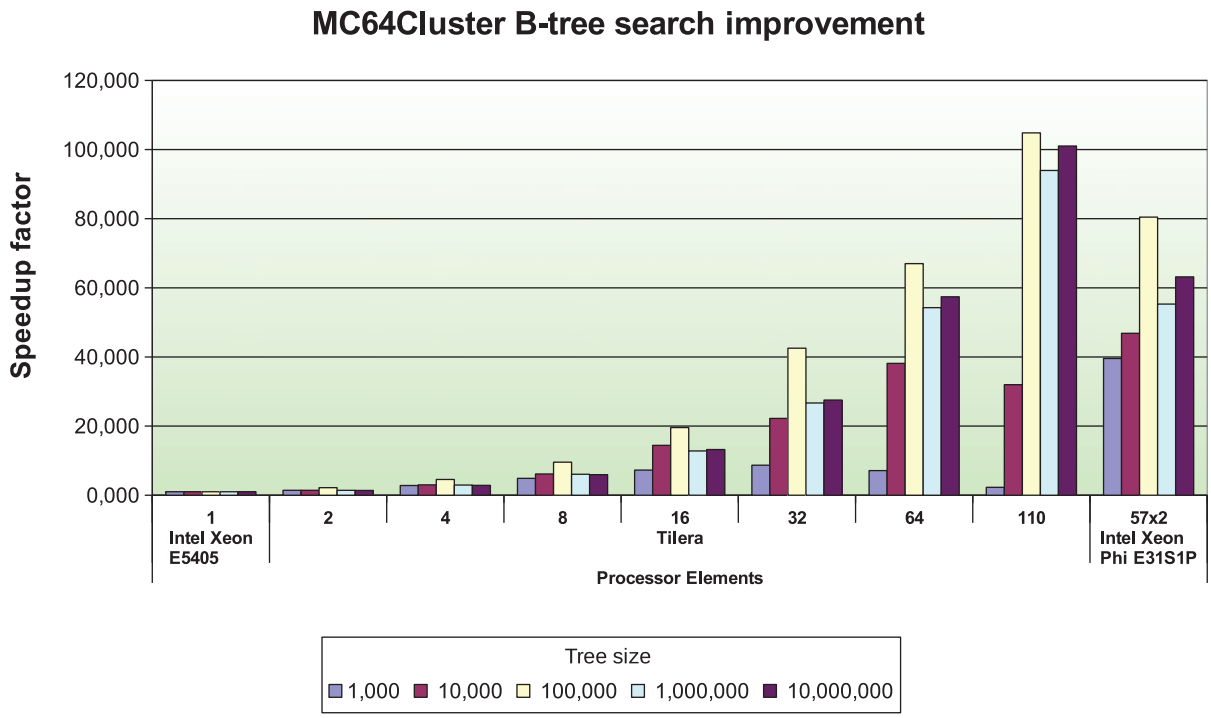

FIGURE 5. Speed-up obtained in fixed-sequence scenario.

buffer was reduced to one kB execution time increased to 586 seconds (~80 times slower).

To complete our analyses, a power consumption comparison was performed, showing that MC64-Cluster execution required 286 W, while Intel execution demanded 330 W. It must be considered that the cluster uses two PC hosts and Xeon Phi 31S1P uses only one. Therefore, using efficient hosts will be more than twice as favorable in cluster than in Xeon Phi 31S1P systems.

6. CONCLUSIONS AND FUTURE WORK

MC64-Cluster behaves as a high-performance and scalable system when solving a key task in many ~~general-purpose~~ applications, like the execution of massive searches against databases. Best results were achieved when used in a hybrid-computing environment, in which both host and many-core cards collaborated in calculi. Our code optimization has been focused on getting the most out of available resources in Tilera platform (mainly, its cache system), while avoiding less-optimal resources like solid-state drive storage systems, as well as network interfaces. It is remarkable in this sense that a significant improvement in cluster performance was achieved sorting the set of values to search for, while it had no effect when a single core was used. When compared to other many-core technologies (like GPU or FPGA), many-core CPU may be easier to deploy because they are programmed in standard C, without any drawback of hardware peculiarities. This fact has enabled to compare performance between MC64-Cluster with two cards and an Intel Xeon Phi 31S1P.

Although this cluster design can be expanded to a higher number of cards, a two-card setup has demonstrated to be large enough for this work, since each one supplied 55 worker-microprocessor elements (tiles or cores), out of the total of 64 cores per Tile64 microprocessor. Thus, with the smallest tree sizes and all tiles in the cluster working, times measured were in the range of a few milliseconds, which is near to the measurement-reliability limit. Cluster computing became even more attractive when PPW was considered (25 watts per Tile64 microprocessor), placing such cluster at the high-end of green-computing scale when appropriate hosts are used. To our knowledge, this is the first cluster implementation of this kind being developed, which, along with the results obtained, are our main contributions.

The above results allow the characterization of this particular many-core CPU system and its allocation in the HPC arena. Thus, it can provide a power-efficient solution to parallelizable problems that require a well-organized cache memory structure, with moderate communication system. In addition, a good performance is achieved by using standard and general-purpose hardware, avoiding particular *ad hoc* architectures [47]. On the downside, the number of programming languages and libraries available to the TILExpress programmer is very limited, which may reduce the scope of its application.

Regarding future work, improvements in ~~massively parallel~~ searches, along with our previous developments in dynamic-programming alignment methods [35] should allow further improvements of bioinformatics algorithms, effectively combining both approaches. Finally, a web manager is currently under development, in order to give non-programmer users (like most life-science researchers) a more-intuitive access to MC64-Cluster, including a predefined list of algorithms to run. This manager will graphically show the running status of cluster resources, allowing interaction with it, even without knowing syntax of command-line oriented tools.

ACKNOWLEDGEMENTS

We are grateful to Tilera <http://www.tilera.com> for providing hardware and software tools. We are also grateful to the Barcelona Supercomputing Center of 'Centro Nacional de Supercomputación', for granting access to its Xeon Phi-powered equipment. The preliminary version of the web interface was developed by José Carvajal, as part of his MS Thesis.

FUNDING

This work was supported by 'Ministerio de Economía y Competitividad' (MINECO grant BIO2011-15237-E); 'Instituto Nacional de Investigación y Tecnología Agraria y Alimentaria' (MINECO and INIA RF2012-00002-C02-02); 'Consejería de Agricultura y Pesca' (041/C/2007, 75/C/2009 and 56/C/2010); 'Consejería de Economía, Innovación y Ciencia' (P11-AGR-7322 and P12-AGR-0482) and 'Grupo PAI' (AGR-248) of 'Junta de Andalucía' and 'Universidad de Q4 Córdoba' ('Ayuda a Grupos'), Spain.

REFERENCES

[1] Cluster Encyclopedia <http://www.clusterbuilder.org/encyclopedia/alphabetized.php>.

[2] Esteban, F., Díaz, D., Hernández, P., Caballero, J., Dorado, G. and Gálvez, S. (2013) MC64-Cluster: A Many-Core CPU Cluster for Bioinformatics Applications. In Rocha, Á., Correia, A.M., Wilson, T. and Stroetmann, K.A. (eds), *Advances in Information Systems and Technologies*, V**ol. 206**, pp.819–825. Springer, Berlin Heidelberg.

[3] Inventing the future of computing <http://www.adapteva.com/>.

[4] de Dinechin, B.D. *et al*. (2013) A Clustered Manycore Processor Architecture for Embedded and Accelerated Q5 Applications. In *High Performance Extreme Computing Conference (HPEC), 2013*, IEEE Waltham, MA, USA, September 10–12, pp. 1–6. IEEE.

[5] Fan, D.-R. *et al* (2009) Godson-T: an efficient many-core architecture for parallel program executions. *J. Comput. Sci. Technol.*, **24**, 1061.

[6] Morgulis, A., Coulouris, G., Raytselis, Y., Madden, T.L., Agarwala, R. and Schaffer, A.A. (2008) Database indexing for production MegaBLAST searches. *Bioinformatics.*, **24**, 1757–1764.

[7] Altschul, S.F., Gish, W., Miller, W., Myers, E.W. and Lipman, D.J. (1990) Basic local alignment search tool. *J. Mol. Biol.*, **215**, 403–410.

[8] Jiang, X., Zhang, P., Liu, X. and Yau, S.T. (2007) Survey on index based homology search algorithms. *J. Supercomput.*, **40**, 185–212.

[9] Navarro, C.A., Hitschfeld-Kahler, N. and Mateu, L. (2014) A survey on parallel computing and its applications in data-parallel problems using GPU architectures. *Commun. Comput. Phys.*, **15**, 285–329.

[10] Arora, M. (2012) *The Architecture and Evolution of CPU-GPU Systems for General Purpose Computing*, **Vol. 27**. University of California, San Diego.

[11] Keckler, S.W., Dally, W.J., Khailany, B., Garland, M. and Glasco, D. (2011) GPUs and the future of parallel computing. *IEEE Micro*, **31**, 7–17.

[12] Shams, R. and Sadeghi, P. (2011) On optimization of finite-difference time-domain (FDTD) computation on heterogeneous and GPU clusters. *J Parallel Distrib Comput*, **71**, 584–593.

[13] Enos, J., Steffen, C., Fullop, J., Showerman, M., Shi, G., Esler, K., Kindratenko, V., Stone, J.E. and Phillips, J.C. (2010) Quantifying the Impact of GPUs on Performance and Energy Efficiency in HPC Clusters. In: *Proc. Int. Conf. Green Computing*, Chicago, IL, USA, pp. 317–324. IEEE Computer Society.

[14] Hamada, T. and Nitadori, K. (2010) 190 TFlops Astrophysical N-body Simulation on a Cluster of GPUs. In: *Proc. 2010 ACM/IEEE Int. Conf. for High Performance Computing, Networking*, Storage and Analysis New Orleans, LA, USA, pp. 1–9. IEEE Computer Society.

[15] Amazon EC2 Instances <http://aws.amazon.com/ec2/instance-types>.

[16] Caballer, M., de Alfonso, C., Alvarruiz, F. and Moltó, G. (2013) EC3: elastic cloud computing cluster. *Journal of Computer and System Sciences*, **79**, 1341–1351.

[17] Corradi, A., Foschini, L., Pipolo, V. and Pernafini, A. (2015) Elastic Provisioning of Virtual Hadoop Clusters in OpenStack-based Clouds. In: *2015 IEEE Int. Conf. Communication Workshop (ICCW)*, London, UK, June 8–12. IEEE.

[18] Cadambi, S., Coviello, G., Li, C.-H., Phull, R., Rao, K., Sankaradass, M. and Chakradhar, S. (2013) COSMIC: Middleware for High Performance and Reliable Multiprocessing on Xeon Phi Coprocessors. In: *Proc. 22nd Int. Symposium on High-performance Parallel and Distributed Computing New York*, New York, USA, pp. 215–226. ACM.

[19] Boisseau, J. (2011) Stampede: A Comprehensive Petascale Computing Environment. In: *IEEE Cluster—Special Topic*. Q6

[20] Schmidt, B. (2011) *Bioinformatics: High Performance Parallel Computer Architectures*. CRC Press. Q7

[21] Qiu, X., Ekanayake, J., Beason, S., Gunarathne, T., Fox, G., Barga, R. and Gannon, D. (2009) Cloud Technologies for Bioinformatics Applications. In: *Proc. 2nd Workshop on Many-Task Computing on Grids and Supercomputers*, Portland, Oregon, pp. 1–10. ACM.

[22] Sawyer, S.E., Rekepalli, B., Horton, M.D. and Brook, R.G. (2015) HPC-BLAST: Distributed BLAST for Xeon Phi Clusters. In: *Proc. 6th ACM Conf. Bioinformatics, Computational Biology and Health Informatics*, Atlanta, Georgia, pp. 512–513. ACM.

[23] Yi, L., Moretti, C., Emrich, S., Judd, K. and Thain, D. (2009) Harnessing Parallelism in Multicore Clusters with the All-pairs and Wavefront Abstractions. In: *Proc. 18th ACM Int. Symposium on High Performance Distributed Computing Garching*, Germany, pp. 1–10. ACM.

[24] Wang, C., Li, X., Zhou, X., Chen, Y. and Cheung, R.C.C. (2014) Big Data Genome Sequencing on Zynq based Clusters (abstract only). In: *Proceedings of the 2014 ACM/SIGDA international symposium on Field-programmable gate arrays*, Monterey, California, USA, pp. 247–247. ACM.

[25] Jeffers, J. and Reinders, J. (2013) *Intel Xeon Phi Coprocessor High Performance Programming*. Morgan Kaufmann Publishers Inc.

[26] Mastelic, T., Oleksiak, A., Claussen, H., Brandic, I., Pierson, J.-M. and Vasilakos, A.V. (2014) Cloud computing: survey on energy efficiency. *ACM Comput. Surv.*, **47**, 1–36.

[27] Gálvez, S., Díaz, D., Hernández, P., Esteban, F.J., Caballero, J.A. and Dorado, G. (2010) Next-generation bioinformatics: using many-core processor architecture to develop a web service for sequence alignment. *Bioinformatics.*, **26**, 683–686.

[28] Diaz, D., Esteban, F.J., Hernandez, P., Caballero, J.A., Dorado, G. and Galvez, S. (2011) Parallelizing and optimizing a bioinformatics pairwise sequence alignment algorithm for many-core architecture. *Parallel Computing*, **37**, 244–259.

[29] Esteban, F.J., Diaz, D., Hernandez, P., Caballero, J.A., Dorado, G. and Galvez, S. (2013) Direct approaches to exploit many-core architecture in bioinformatics. *Future Gener. Comput. Syst.*, **29**, 15–26.

[30] Bell, S. *et al*. (2008) TILE64 - Processor: A 64-Core SoC with Mesh Interconnect. In: *Solid-State Circuits Conference, 2008 ISSCC 2008 Digest of Technical Papers*, IEEE International San Francisco, CA, USA, February, pp. 88–598. IEEE.

[31] (2004) IEEE Standard for IT-Telecom. and Information Exchange Between Systems-LAN and MAN-Specific Requirements. Part 3: CSMA With Method and Physical Layer Specifications. Amendment: Physical Layer and Management Parameters for 10 Gb/S Operation, Type 10 GBASE-CX4. IEEE Std 8023ak-2004 (Amendment to IEEE Std 8023-2002 as amended by IEEE Stds 8023ae-2002, 8023af-2003 and 8023aj-2003):0_1.

[32] Kamal, H., Penoff, B. and Wagner, A. (2005) SCTP versus TCP for MPI. In: *Proc. 2005 ACM/IEEE Conf. Supercomputing*, Seattle, WA, USA, p. 30. IEEE Computer Society.

[33] Liu, X., Li, M., Lin, X. and Cheng, X. (2008) Implementation and Optimization of MPICH2 Multicast on Optical Fiber Network. In: *Network and Parallel Computing*, 2008 NPC 2008 IFIP International Conference on Shanghai, China, October 18–21, pp. 577–582. Springer.

[34] Peix, A., Ramirez-Bahena, M.H., Velazquez, E. and Bedmar, E.J. (2015) Bacterial associations with legumes. *CRC. Crit. Rev. Plant. Sci.*, **34**, 17–42.

[35] Diaz, D., Esteban, F.J., Hernandez, P., Caballero, J.A., Guevara, A., Dorado, G. and Galvez, S. (2014) MC64-ClustalWP2: a highly-parallel hybrid strategy to align multiple sequences in many-core architectures. *PLoS ONE*, **9**, e94044.

[36] Simpson, J.T., Wong, K., Jackman, S.D., Schein, J.E., Jones, S.J.M. and Birol, I. (2009) ABySS: a parallel assembler for short read sequence data. *Genome Res.*, **19**, 1117–1123.

[37] Hernandez, P. *et al* (2012) Next-generation sequencing and syntenic integration of flow-sorted arms of wheat chromosome 4A exposes the chromosome structure and gene content. *Plant J.*, **69**, 377–386.

[38] Zhang, J., Misra, S., Wang, H. and Feng, W.-C. (2016) muBLASTP: database-indexed protein sequence search on multicore CPUs. *BMC Bioinformatics*, **17**, 443.

[39] Morgulis, A., Coulouris, G., Raytselis, Y., Madden, T.L., Agarwala, R. and Schäffer, A.A. (2008) Database indexing for production MegaBLAST searches. *Bioinformatics*, **24**, 1757–1764.

[40] Tan, Z., , Cao, X., , Ooi, B.C., and Tung, A.K.H. (2003) The ed-tree: An Index for Large DNA Sequence Databases. In: *Proc. 15th Int. Conf. Scientific and Statistical Database Management*, Cambridge, MA, pp. 151–160. IEEE Computer Society.

[41] Ning, Z., Cox, A.J. and Mullikin, J.C. (2001) SSAHA: a fast search method for large DNA databases. *Genome Res.*, **11**, 1725–1729.

[42] Alba, E. and Luque, G. (2007) A New Local Search Algorithm for the DNA Fragment Assembly Problem. In: *Proc. 7th European Conf. Evolutionary Computation in Combinatorial Optimization Valencia*, Spain, pp. 1–12. Springer.

[43] Schwartz, S., Oren, R. and Ast, G. (2011) Detection and removal of biases in the analysis of next-generation sequencing reads. *PLoS ONE*, **6**, e16685.

[44] Bayer, R. and McCreight, E. (2002) Organization and Maintenance of Large Ordered Indexes. In Manfred, B. and Ernst, D. (eds) *Software Pioneers*, pp.245–262. Springer, New York.

[45] Comer, D. (1979) The ubiquitous B-tree. *ACM Computing Surveys*, **11**, 121–137.

[46] Cao, X., Li, S. and Tung, A.H. (2005) Indexing DNA Sequences Using q-Grams. In Zhou, L., Ooi, B. and Meng, X. (eds) *Database Systems for Advanced Applications*, **Vol. 3453**, pp.4–16. Springer, Berlin Heidelberg.

[47] Liu, P., Hemani, A., Paul, K., Weis, C., Jung, M. and Wehn, N. (2017) 3D-stacked many-core architecture for biological sequence analysis problems. *Int. J. Parallel Prog.*, 1–41.