Many-Core Processor Bioinformatics and Next-Generation Sequencing

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

[1] Servicio de Informática, Campus Rabanales, Universidad de Córdoba, 14071 Córdoba (Spain)
[2] Dep. Lenguajes y Ciencias de la Computación, Boulevard Louis Pasteur 17, 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, 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)

{fjesteban,ma1camoj,bb1dopeg}@uco.es,
{david.diaz,galvez}@lcc.uma.es, phernandez@ias.csic.es

Abstract. The new massive DNA sequencing methods demand both computer hardware and bioinformatics software capable of handling huge amounts of data. This paper shows how the many-core processors (in which each core can execute a whole operating system) can be exploited to address problems which previously required expensive supercomputers. Thus, the Needleman-Wunsch/Smith-Waterman pairwise alignments will be described using long DNA sequences (>100 kb), including the implications for progressive multiple alignments. Likewise, assembling algorithms used to generate contigs on sequencing projects (therefore, using short sequences) and the future in peptide (protein) folding computing methods will be also described. Our study also integrates the last trends in many-core processors and their applications in the field of bioinformatics.

Keywords: Parallel Computing, Grid and Cloud Computing, Biotechnology, Agrifood and Agribusiness.

1 Introduction

After fifty years of using computers in order to solve scientific problems, computing limits are still a real challenge in many knowledge fields. During the first three decades of this period, high-performance computing was accomplished via computers equipped with specialized vector processors, where advances in performance were obtained by increasing the number of transistors and clock frequency, using greater

---

* Corresponding author.
** Authors who contributed to the project leadership.

M.L. Reyes et al. (Eds.): IT Revolutions 2011, LNICST 82, pp. 172–188, 2012.

amounts of memory and making them capable of dealing with larger data sets by means of increasing their bus size. This scenario has been exploited to solve scientific problems by the development of faster algorithms and by using increasingly sophisticated programming languages, which are based on sequential processes. Yet, the processor performance cannot be indefinitely increased by simply rising its transistors amount and clock frequency, due to physics law limits, like the power consumption associated to the silicon geometry used in the chips.

With the popularization of computers in the late 80's decade, followed by the advances in computer networks, more affordable computing power was possible. Indeed, complex scientific problems could be resolved at the cost of developing a new paradigm: parallel computing, where many tasks can be executed simultaneously, instead of sequentially. From this point on, the parallelization methods have evolved continuously, involving: i) implementing more than one Central Processing Unit (CPU) into a computer, using Symmetric MultiProcessor (SMP) systems; ii) assisting the main processing with coprocessors specialized in mathematics or graphics tasks, using ASymmetric MultiProcessor (ASMP) systems; iii) increasing the integration levels, using more than one processor in the same silicon die (multicore systems); iv) using coprocessors in general and specially Graphic Processing Units (GPU), which have eventually left their specificity to become general-purpose processors; v) using Accelerated Processing Units (APU) which represent a CPU-GPU hybrid for High-Definition (HD) imaging, Three-Dimensional (3D) rendering and data-intensive workloads in a single-die processor; and vi) using many-core systems capable of running a whole operating system in each core.

Our research group has previously demonstrated the usefulness of the Tile64 processor (from Tilera <http://www.tilera.com>) with 64 tiles (cores) in the field of bioinformatics [1]. In the present work, we deal with pairwise alignments, multiple alignments and sequence assembly of DeoxyriboNucleic Acids (DNA), Ribonucleic Acids (RNA) and peptides (proteins). The application to protein folding is also considered, as well as other academic and commercial initiatives and many-core products that can be of relevance for life-science researchers.

2 Tile64 Architecture

The Tile64 architecture consists of a Reduced Instruction Set Computing (RISC)-based processor which contains 64 general-purpose processor cores, connected by a low-latency network (with more than a terabit/sec of interconnect bandwidth) called Intelligent Mesh (iMesh), in which the core geometry does not affect the performance obtained. Each tile runs at 500-866 MHz, executing a customized Linux operating system in it. A few dedicated tiles may be needed for coordinating tasks, so that the number of available tiles to optimally run applications is always less than 64. As an example, five tiles may be needed for administrative and coordinating tasks (one for communications with the host, three shared for internal operations and one to distribute the jobs amongst the worker tiles), reducing the available tiles to run applications to 59 [1]. The processor is boarded on a Peripheral Component

Interconnect Express (PCIe) card, along with a given amount of Random Access Memory (RAM) and a set of Ethernet ports. We have used the TILExpress-20G card, with 8 GB RAM and two 10GBase-CX4 Ethernet connectors. In this card, the tiles of Tile64 run at 866 MHz.

Regarding the software, the programs to be run on the Tile64 processor can be developed with the Eclipse-based Tilera's Multicore Development Environment (TileMDE), which includes a C/C++ cross-compiler, so native RISC code for the processor can be generated and later deployed to the card via the PCIe interface. This tool can be used in any standard Linux distribution, although Red Hat or CentOS are the recommended distros.

In order to exploit parallelism, two main Application Programming Interfaces (API) are provided, named iLib and NetIO. The former allows the programmer using iMesh to provide common parallel functionality: task processing, message passing, channel communication and shared memory. The latter gives access to the Ethernet ports. In both cases, no widely used implementation like Message Passing Interface (MPI) or Parallel Virtual Machine (PVM) is followed, which leads to the fact that, although a direct porting is possible when using existing C/C++ sources, rewriting at least some code is always mandatory to allow it running in parallel.

3 Sequence Alignments

The goal of a pairwise alignment algorithm is to identify similar or discrepant regions of a DNA, RNA or peptide sequence when comparing it to just another sequence. On the other hand, multiple sequence alignments allow to find similarities and differences in a set of two or more sequences. This way, evolutionary relationships can be established, including the generation of phylogenetic trees (dendrograms). Likewise, the polymorphisms in the sequences can be identified, which can be useful, for instance, to design specific molecular markers for DNA, RNA or peptide fingerprinting. We have used such approach to identify Single Nucleotide Polymorphisms (SNP) in DNA, in order to enforce quality control and prevent fraud of olive oil [2].

3.1 Pairwise Alignments

From a programmer's point of view, a sequence is represented by a string of symbols from a reduced alphabet. Each symbol represents the molecular element relevant for the study (nitrogenous base residues for RNA and DNA or amino acid residues for peptides). A pairwise alignment is a matching between two sequences without altering the order in their elements (residues), taking into account inserted or deleted residues, which leads to the presence of extra residues or gaps in one or another sequence, respectively. Depending on the alignment nature, a global alignment can be considered if the sequences are completely compared, and an alignment is considered to be local when it focuses on the sequence stretch where the sequences show the maximum similarity. There are several methods to calculate pairwise alignments, both local and global, like the global aligner Needleman-Wunsch (NW) algorithm [3], the first suitable to be obtained by a computer, proposed by such authors in 1970. The local alignment concept was introduced by Smith and Waterman (SW) [4] later in 1981. During the following years, further improvements were made, like Gotoh's [5], Hirschberg's [6] or FastLSA [7]. For our study, we have programmed from scratch a new implementation of these algorithms for the Tile64 platform, called MC64-NW/SW [1].

The Needleman-Wunsch algorithm uses dynamic programming to maximize scores in a matrix, whose dimensions are the respective sizes of the two sequences being compared. The scores are calculated taking into account the residue changes, insertions and deletions, so each cell of the matrix stores the best score to align the elements of both sequences up to this point. To compute this score, a reference table or substitution matrix is used, considering also a gap insertion or deletion penalty cost. Several substitution matrices have been proposed over the years, all of them experimentally obtained. The Dynamic Programming Matrix (DPM) is calculated from left to right and from top to bottom, initializing the first row and column and getting the remaining positions from its upper, left and upper-left neighbor. Once the entire matrix is calculated, the global alignment is obtained in a backward stage, from the bottom right corner, following the maximum score path in horizontal, vertical or diagonal ways.

Yet, the original algorithm described above lacked precision, because it severely penalized longs gaps caused by insertions or deletions, a circumstance that can arise in nature from mutation events. To avoid such issue, Smith and Waterman introduced the concept of "local alignment", which means finding the longest pair of segments from each sequences to align, so that there is no other one with greater similarity. To implement this idea, three changes were included in the Needleman-Wunsch algorithm: i) recalculating the penalty cost in each position from the dynamic programming submatrix, already calculated up to this point; ii) suppressing negative values in the initialization; and iii) saving the highest score achieved during the matrix calculations. Thus, the backward stage starts from this highest score node, finishing when the first zero value is reached. In order to avoid the higher time complexity in the new algorithm, getting it back from $n^2 \times m$ to $n \times m$, Gotoh introduced the idea of "affine gap", using two additional matrices instead of recalculating the gap penalty for each cell [5]. In order to avoid memory limitations, a first "divide and conquer" approach to this problem was proposed by Hirschberg, obtaining the results in linear space and double quadratic time by means of a double calculation of the DPM [6]. A further improvement by the FastLSA proposed to save temporal values from the first stage in a grid cache, so when the backward stage is executed, only the matrices in the optimal path are recalculated.

To estimate the potential of the many-core processor, we have implemented a parallel version of the FastLSA in the Tile64 architecture, both for the global Needleman-Wunsch and the local Smith-Waterman aligners. We have run several comparatives and benchmarking tests, optimizing the performance of the algorithms to exploit the Tile64 features. We decided to write the MC64-NW/SW as an entirely new development, instead of porting an existing FastLSA implementation. Thus, at the cost of a greater development effort, updates, adding functions and specially optimizing code fragments (our main goal) were easier to accomplish. As stated before, the most time-consuming task in our algorithm was to calculate the DPM, being the task that should be distributed among the available tiles. The way to do this in FastLSA is using a wavefront parallelism, where sub-matrices can be calculated progressively, once their upper-left elements are available. Thus, at the time that a sub-matrix is available, its bottom-right elements become the initial data to the next calculus, repeating this process until the last row and column are reached. With a controller-worker scheme, a master tile is assigned with the role of getting the initial data, passing them to the available slave tiles (assigned with the role of matrix calculations), and finally collecting results to make the overall work progress.

With this general approach, which offered impressive results from the first implementation due to its inherent high parallelization factor, several variants were used, effectively exploiting the potential of the TILExpress-20G hardware. The Figure 1 shows the MC64-NW/SW version history and the performance when aligning a given sample pair of sequences of about 360 kilobases (kb) in the same hardware environment and with all possible executing parameters alike.

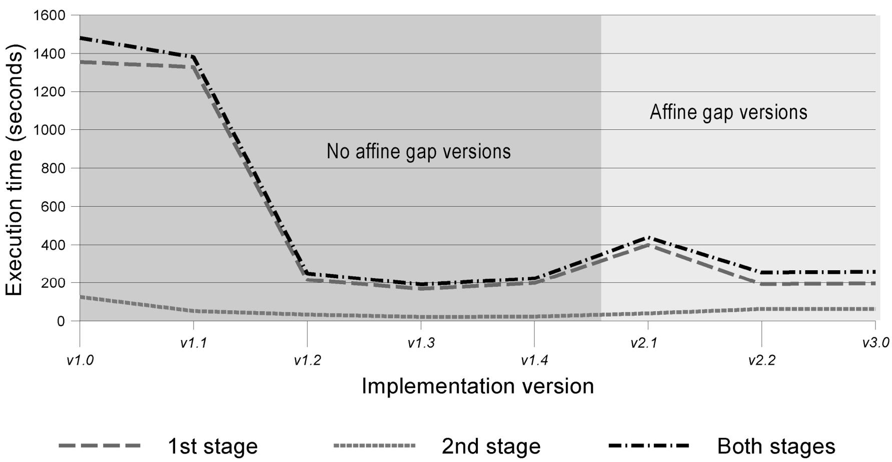

Fig. 1. MC64-NW/SW version history and performance

The key change to improve our implementation's performance occurred with the arrival of version 1.2, transferring some data from shared memory to the local memory of the tile requiring it. This amazing speedup factor was reached because a tile can access data much faster when it is allocated into its local memory space. The conclusion is that the Tile64 can be an effective environment to execute bioinformatics applications, when parallel implementations are well structured and fine-tuned to meet the architecture peculiarities. Our work in progress under this development includes a new version, where intermediate results are stored in a host database, freeing the memory resources onboard the card, and thus allowing to align sequences up to 16 megabases (Mb).

3.2 Multiple Alignments

Once the usefulness of the Tile64 platform in pairwise alignment was demonstrated, the next natural step was to complete this work adding the feature of multiple sequence alignments. In fact, that can be derived from a triangular matrix of distances, obtained from the full pairwise alignments between the sequences. In this scenario, the pairwise alignments derived from the MC64-NW can have the additional advantage of being optimal, in the sense that no heuristics are used to speed up the alignment process.

The most widely used multiple alignment tool throughout the last years has been Clustal W [8]. Its source code has been ported to many platforms, entire rewritten in C++ to give it a modern interface [9] and being slightly parallelized using MPI [10].

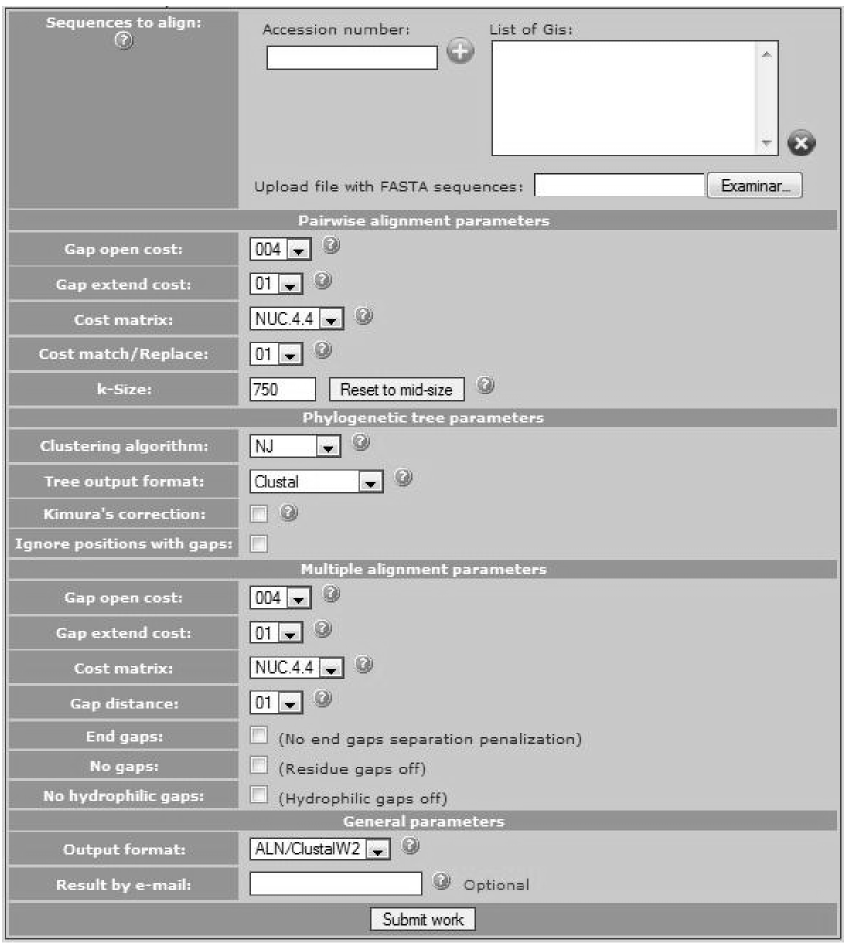

Fig. 2. Clustal W parameter form at our Web page http://www.sicuma.uma.es/manycore/index.jsp?seccion=m64cl/run

The algorithm has three stages, all of them allowing variations. First of all, all possible pairs of sequences are aligned, building a matrix representing the degree of divergence between sequences by means of scoring or "Weighting" (the "W" stands for it) the quality of the pairwise alignments. In the second stage, a guide tree is derived from the distance matrix, in order to simplify the generation of the final alignment, making it computationally affordable. Finally, a progressive alignment is run using a clustering method. The first and last stages are the most computationally demanding. Regarding to the variants, any of the available pairwise alignment methods can be used in the first stage, either heuristic or fully dynamic programming, with the above mentioned advantage in accuracy in the second option. In the second stage, the Neighbor-Joining (NJ) [11] or Unweighted Pair Group Method Average (UPGMA) [12] approaches are usually employed.

A preliminary parallel implementation of the Clustal W for the Tile64 (MC64-ClustalW) is already available. We have replaced the Clustal W's first stage of pairwise alignments with our MC64-NW/SW local aligner. Thus, for 10 sequences of approximately 100 kb, the 45 elements of the distance matrix are calculated in 860 seconds (less than 15 minutes) instead of 8,067 seconds (more than two hours) in the non-parallelized first stage of Clustal W in a quad-core Xeon microprocessor. The MPI Clustal W version in the same quad-core system completes this same task in 3,121 seconds (about 52 minutes). The Figure 2 shows the parameter page available at our web site <http://www.sicuma.uma.es/manycore/index.jsp?seccion=m64cl/run> to access this preliminary MC64-ClustalW implementation.

4 Sequence Assembly

A single chromosome from higher organisms (eukaryotes) may have about 100 to 200 million base pairs. Current technologies do not allow to "read" full chromosomes from one end to the other on a single sequencing reaction. The different sequencing platforms can only read from about 30 to 1,000 bases per reaction (depending on the technology used). Thus, sequencing full genomes is time-consuming and expensive. Some of the current limitations stem from the fact that, to sequence a chromosome, not a single molecule but millions of them may be required. Such DNA is randomly broken into small pieces, generating a huge library of redundant and overlapping fragments, which are subsequently sequenced by "brute force".

The bioinformatics tools are then required to sort such complex maze and generate contiguous overlapping fragments ("reads") known as contigs (name derived from "contiguous"), with enough redundancy to overcome possible sequencing errors. Some reads may be small, may contain homopolymeric stretches (eg., tens of T bases in a row) and repetitive sequences (eg., AC many times in a row). Such kinds of sequences may exist on many locations on the genome, which may prevent proper contig generation. As expected, these issues are more significant for technologies generating small reads (eg., 30 bases), being therefore more difficult to assemble into contigs. Consequently, gaps may remain in the sequence, requiring further efforts to fill using other sequencing approaches. Actually, most of the times such gaps are not even filled, due to the cost involved, and the genome is assembled using a reference one if available. Indeed, virtually all published genomes to date include many unsequenced stretches or gaps.

There are two main scenarios when assembling genomic sequencing reads into contigs: i) de novo sequencing, when such species genome is being sequenced for the first time; and ii) resequencing, when the species genome has been previously sequenced (eg., from a different individual). In the latter case, the previous sequenced genome is used as reference, to align the other from the new sequencing data. Sequencing-data assembly has been tackled by different bioinformatics strategies [13], but it has been a fully computational task only for the last years. This has been a consequence of the huge amount of sequencing data generated by the so-called "next-generation" sequencing technologies. The first approach to the problem consists of finding the shortest common superstring (contig) of a set of sequences. This task can be efficiently accomplished by a "greed" algorithm, starting by finding all possible overlaps between the initial samples. Later on, a score is assigned to each potential overlap, and finally the samples are iteratively merged into a single contig, by combining the ones with the highest scores. The process ends when no more samples can be merged (albeit, leaving gaps in some instances, as previously explained). As examples, TIGR Assembler [14], Phrap [15] and CAP3 [16] follow this paradigm. However, this approach has one big problem: it is useful only for short genomes, due to its large RAM requirements, so its applicable field is limited to prokaryotes (bacteria) and a few lower (single-celled) eukaryotes.

Three other work lines can solve the limitations of a greed algorithm: the DeBruijn graph [17] representation, the prefix-tree approach, and the overlap-layout-consensus. The first work on short reads by de novo assembly was a pioneering DeBruijn graphs-based implementation by Prevzner [18] in the first year of this century. Velvet [19], EULER-SR [20] and ALLPATHS [21] also follow this paradigm, which is the most recurrent in scientific literature. The SSAKE [22] by Warren was the first prefix-tree approach, followed by the VCAKE [23] and SHARCGS [24]. Finally, the Edena [25] represents the last step in the overlap-layout-consensus tradition.

The parallelism has come late to this field. The first algorithm capable of distributing a DeBruijn graph across a network of computers has been ABySS [26], breaking its inherent memory usage limit and providing the potential ability to assemble genomes of virtually any size. This brand new parallel implementation in a new bioinformatics field (like the de novo assembly of full genomes from short sequencing reads) was really interesting, so we tried to develop a Tile64 ABySS implementation of it. In this case, we started from the C++ code freely available at the author's web site. To get it run in parallel, a message passing technique was used in the original source code, by means of exploiting the widely used MPI library.

Due to the lack of this library in the Tilera's software environment, we carried out a previous study of MPI usage in ABySS, analyzing its communication behavior. We found it rather simple: only a small subset of MPI functions was used, consisting of "blocking send" to exchange data messages, "synchronous blocking send" to exchange control messages, and "non-blocking receive" to answer both kinds of messages. Moreover, byte is the only data type used in any call. The control flow is always between a master node and the others, whereas the data flow can be exchanged between any two nodes. Additionally, the barrier mechanism was used to control the algorithm progress. The code was also well structured, with all the communication stuff concentrated in a single module, which contains most of the MPI calls.

In this situation, our first step was to analyze the similarities and differences between the MPI and the message passing capabilities of the Tilera's iLib library. We encountered two major differences regarding the absence of distinction between synchronous send and "normal" blocking send and, on the other hand, the absence of a function to open a receive call without identifying the expected sender. The first issue can be overcome by considering the iLib blocking send implementation as synchronous, and thus introducing the concept of a router tile to simulate "normal" MPI blocking send, according to the blocking send definition in the MPI specification. For the second issue, taking into account that the participant tiles were established at the very first stage of execution, we made a separate receive call for each other participant tile to deal with control messages, whereas the router tile concept solved this problem inherently for the data messages exchanges. Therefore, the router tile must issue a non-blocking receive call to the rest of participant tiles, and the other tiles must issue only a non-blocking receive call to the router tile. In this final situation, the message format needs a slight modification, adding a source/destination header in order to keep the communication participants identified.

With the above constraints, we have carried out a parallel ABySS porting to the Tile64 platform. As a reference for our porting, we have run the original algorithm in an Intel quad-core Xeon workstation, equipped with the same amount of RAM present in the TILExpress-20G cards (8 GB), in which we installed the MPI distribution recommended by the original algorithm's authors (OpenMPI). As we were interested in evaluating the Tilera's platform performance, we conducted our tests only with the same synthetic data used by the authors, checking the accuracy of our solution first, which could be consistently established. Thus, with the same input data, results from our implementation and from the Xeon reference were always the same, resulting in the same assembled sequences in every execution.

However, the performance results were not satisfactory. From the first tests we could see that, despite of the larger number of tiles of the Tile64 platform, they could not beat the execution times offered by the four Xeon cores, nor even approaching them. The reason for this poor performance remained in the fact that the distribution of the DeBruijn graph among the participant tiles resulted to be a more time-consuming task than the actual assembly. Thus, the iMesh behavior with a router tile could not compete this time with the fine-tuned mechanism used by OpenMPI when distributing processes inside the quad-core processor. Given this situation, we introduced a change in our communication strategy, dedicating more tiles to the router role than to execute the assembling algorithm itself. The remaining tasks alternate its router in a round-robbing fashion. An optimal execution time could be obtained fine-tuning the number of routers and assembler tiles, given a number of samples to assemble. The Table 1 shows these times, with the same consideration that they are far from the Xeon platform scores.

Table 1. MC64-ABySS best execution times

Samples	MPI execution time (seconds)	Tile64 execution time (seconds)	Working tiles	Router tiles
1,000	1,170	6,930	31	32
10,000	4,460	19,090	47	16
100,000	41,780	167,620	31	32
200,000	79,610	287,180	31	32
500,000	198,900	711,570	47	16
1,000,000	493,700	1676,810	47	16

Finally, we made a second implementation based upon the iLib channel communication paradigm, maintaining the idea of the router tile in order to meet the MPI requirements. With this, we tried to emulate most of the MPI interface in order to get a small MPI implementation in the Tilera platform. The performance of this second implementation was equally unsatisfactory. As conclusion for this section, we can see that a partial porting of an existing algorithm may not yield better performance on some many-core platforms. Thus, if the parallelization factor cannot be exploited or other constraints arise, results could be frustrating. As we could see, this particular algorithm was best suited to multicore platforms (instead of current many-core ones), where less, yet more powerful and better-communicated processors can beat many, less powerful and worse-communicated ones.

5 Protein Folding

The central dogma of molecular biology states that "DNA makes RNA makes protein". Once a sequence is obtained, it may be useful to determine its 3D structure. This process is usually less relevant in DNA, somewhat useful in RNA and most important in peptides (like proteins). In fact, a wrong protein folding may generate severe pathologies like the Alzheimer disease or the Creutzfeldt-Jakob disease (similar to the "mad cow" disease). Interestingly, some proteins can auto-fold spontaneously, but others require the participation of specialized cellular machinery like the chaperons and chaperonins.

The 3D structure of molecules can be experimentally obtained after crystallization and further X-ray diffraction. Yet, such approach may be difficult, time consuming and expensive in some instances. An alternative approach is to computationally simulate the molecule folding by means of prediction or modeling techniques. However, this process presents vast computationally challenges, that may be even higher than the experimental ones. For instance, protein folding processes typically take only tens of microseconds in the cells, but a computer simulation on a single current computer may take one day to simulate just a single folding nanosecond. So, the parallel and distributing computing is the only way to tackle such challenge nowadays.

Many approaches have addressed the protein folding in silico. Perhaps, the most widely known project is Folding@home [27] at Stanford University, and its homonyms Predictor@home and Rosetta@home, all of them relaying in a principle already used in previous projects: to harvest unused personal computer (PC) cycles distributed in the Internet to develop protein folding. The amazing amount of published results based on this lose-connected medium-powered computer cluster (see, for example [28]) widely demonstrates the usefulness of this paradigm for this bioinformatics challenge. Other approaches in this field fall in the subject of ab initio prediction of RNA [29] and protein [30] folding structure or protein structure comparison [31], whose parallelization has been afforded with the omnipresent MPI. A general algorithm review can be found at [32], compiling the most usefulness and the state of the art in parallelization, exploiting already existing grid infrastructures.

In this scenario, multi-core and General-Purpose GPU (GPGPU) architectures usually serve as building blocks, to increase the computing power of the overall distributed or grid system within a single node. As an example, the Table 2 shows the number of clients in the Folding@home project and its computer power. This implies the cost of adapting grid-enabled software to exploit this first-order parallelism, using the appropriate technique like the Compute Unified Device Architecture (CUDA) for the nVidia GPU, and multiprocessing or multithreading for SMP. Another alternative consists of using a parallel-capable middleware. Again, as an example, the MPI is used to build the SMP-ready clients at Folding@home.

Table 2. Client statistics in the Folding@home project. Source:<http://fah-web.stanford.edu/cgi-bin/main.py?qtype=osstats> as of 28th October, 2010

Operating system	TFLOPS	x86-equivalent TFLOPS	Active CPU	Total CPU
Windows	212	212	223,062	3,494,452
Mac OS X (PPC)	4	4	4,466	141,183
Mac OS X (Intel)	22	22	7,192	133,576
Linux	62	62	36,354	532,490
ATI GPU	655	691	6,419	138,782
NVIDIA CPU	1,037	2,188	8,716	221,109
Cell B3	785	1,656	27,833	1,030,352
Total	**2,777**	**4,835**	**314,046**	**5,691,944**

Regarding the many-core architectures (like Tilera), as we have demonstrated before, we can surpass other implementations in a single PC when addressing bioinformatics problems with medium to high computational challenges and medium to low communication tightness. Thus, a cluster of the Tilera cards could offer good performance for some of these problems, if they can be re-scaled to the dimension of the cluster. Another strategy is building a regular client architecture. In this scenario, each tile could be an independent client, given its capability to execute a whole operating system, accessing the grid through the card built in Ethernet interfaces.

6 Future Many-Core Trends

To conclude this work, we present a short review of the available (or announced) many-core products, as well as their possible usefulness in bioinformatics.

Intel has released a research program called Single-chip Cloud Computer (SCC) [33], developing a 48-core chip currently tested by academic and research groups. This product is in the line with other manufacturers like Tilera, with each core having the ability to run an entire operating system and with hardware message-passing support, so meeting the requirements of nowadays widely available parallel implementations. The Figure 3 shows the SCC die. The first pre-commercial bundle of such architecture has been a PCIe card (codenamed "Knights Ferry") with a 32-core Aubrey Isle die in it. Future plans in this line includes 80- and 100-core single-chip developments at up to 5.7 GHz for CPU clock and 1.81 TFLOPS of performance [34]. As we have demonstrated with the Tilera's cards, this architecture can be exploited for bioinformatics applications [1]. Intel may have some advantages in this scenario, including its capability to use floating point instructions in hardware, its use of the well-known x86 architecture and its efficient use or power consumption, with the built-in capability of independently regulating such parameter in each individual core.

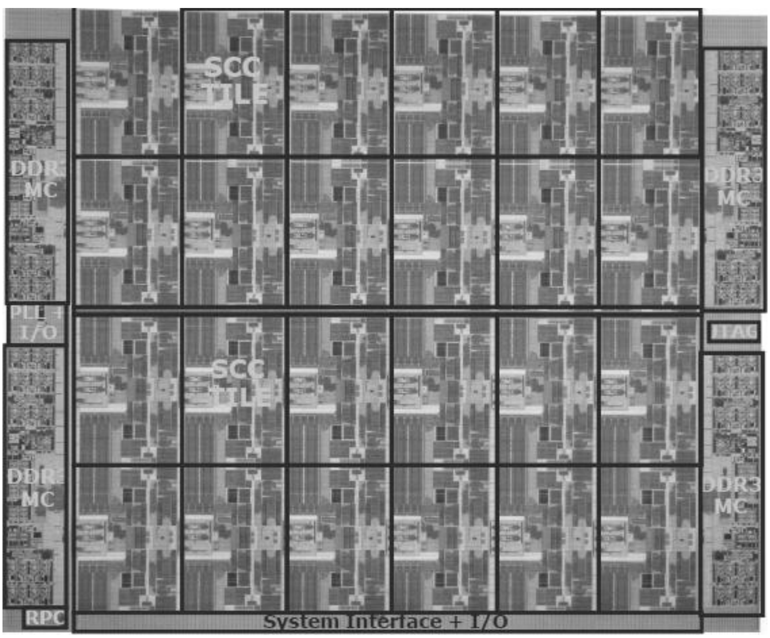

Fig. 3. Intel SCC die. Source: Intel SCC symposium (2010)

On the other hand, AMD has changed its strategy after acquiring the GPU manufacturer ATI. Its future plans are centered on its Fusion platform, [35] announcing the idea of the Accelerated Processing Unit (APU). The rationale behind this concept is the integration of traditional x86 CPU and many GPU cores in the same die, adding also critical elements like memory controllers, specialized video decoders, display outputs and bus interfaces. The idea resembles the System on Chip (SoC) products, with a difference in application. Thus, the goal of AMD Fusion is not making a specialized limited system, but a general-purpose high-performance one, capable of combining scalar or serial computing abilities (present in CPU), as well as parallel or vector computing features (associated with GPU) in an heterogeneous blend. At the time of writing this paper, no Fusion product was commercially available. However, two processors have been announced for 2011: one oriented to netbooks and laptops (codenamed "Llano") and another for desktops (codenamed "Zambezi"). The Figure 4 shows a Llano die. Although the programming methods are not known at the moment, no change of paradigm is expected.

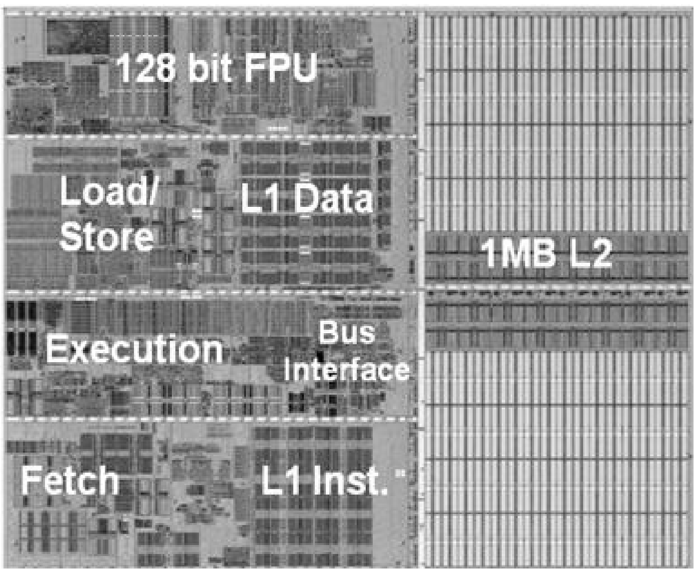

Fig. 4. AMD Fusion "Llano" die. Source: ISSCC 2010

In the pure GPU manufacturer's side, nVidia continues its GeForce/Tesla development. Its strategy is far from other manufacturers: its CUDA technology is based in distributing massively parallel threads of execution among the available cores, specialized in this scheme and so not interested in running an entire operating system. At present, its top of the line product in the GPGPU brand is the Tesla C2070 PCIe card [36]. Its 448 cores are far away from any other current product, but this leadership could be conditioned by the specialization of cores above mentioned. Thus, when operations per second are used as the measuring unit, its performance is in the same range than the remaining products in this revision. In the pure GPU arena, its recent GeForce GTX 580 announcement [37] makes a step forward, with 512 cores and an innovative vapor chamber design for more efficient heat dissipation. It is important to note that a strength in this solution is the full floating point operations hardware support. In the downside, the use of this platform requires a complete rethinking of algorithms, in order to maximize the number of executing threads, and therefore limiting its usefulness for general bioinformatics purposes.

Finally, Tilera has announced the first 100-tile many-core processor, named Tile-Gx [38]. Besides the number of tiles, many other features have been improved from the Tile64 processor: faster CPU clock, 64 bit processors, more cache memory and much better interfaces, including eight 10 GB Ethernet ones and three proprietary 20 GB/s connectors for twin card interconnection. Besides, it maintains the previous philosophy (a whole Linux operating system running in each tile). The power efficiency has been improved too, with a power consumption of 10 to 55 watts. A downside is that it lacks floating point operations by hardware, which can be a barrier for some scientific applications. But, fortunately, this is not the case for many bioinformatics tasks, as we have previously shown [1]. The Figure 5 illustrates a Tile64 processor die shot. Finally, the Table 3 shows a comparison amongst the main features of the products described in this review.

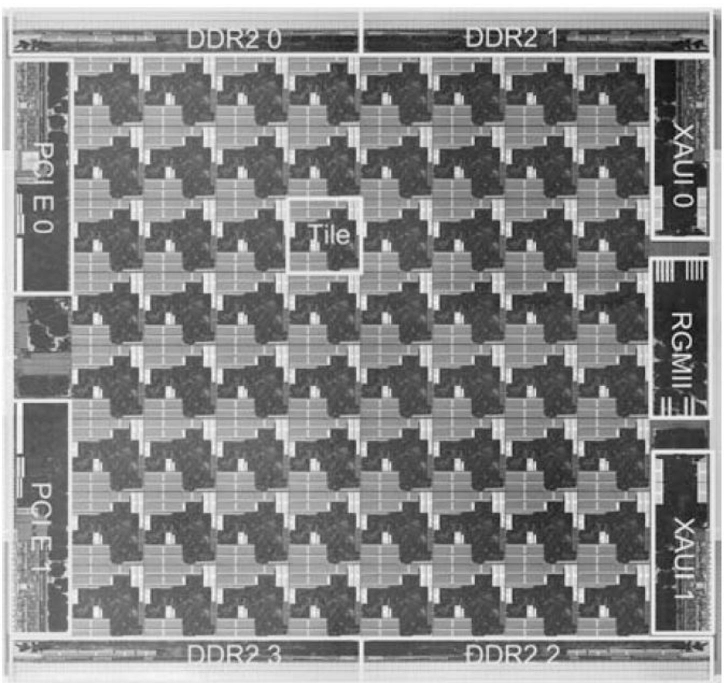

Fig. 5. Tile64 processor die shot. Source: ISCC 2008

Table 3. Many-core platforms

Manufacturer	Product	Cores	Speed	Cache	Memory	Power (W)
Intel	SCC	48	1-2 GHz CPU clock	16 + 16 kB L1 256 kB L2	4 DDR3 controllers	24
AMD	APU (Fusion "Llano")	Up to 4 CPU + GPU	30 GFLOPS	1 MB L2	1 DDR3 1600 controller	14
nVidia	Tesla C2070	448	1.15 GHz CPU clock 515 GFLOPS	n/a	6 GB	238 (entire board)
nVidia	GeForce GTX 580	512	1.54 GHz CPU clock 515 GFLOPS	n/a	n/a	244 (entire board)
Tilera	Tile-Gx	100	1.0-1.5 GHz CPU clock 750 GOPS	32 + 32 kB L1 256 kB L2	4 64-bit DDR3 controllers	10-55

7 Conclusions and Future Work

After more than two years applying the first commercially available real many-core architecture –in the sense that each core or tile can run a whole operating system– to the field of bioinformatics, we have demonstrated its potential when addressing some challenges, like optimal local or global pairwise alignments [1]. Furthermore, our studies reveal that the GPU architectures can also yield good results in specific situations, but that they do not apply to most bioinformatics scenarios, due to the in-depth redesigning required to adopt the CUDA parallelization.

We have also demonstrated that these architectures are not yet capable to address other challenges in its current state of software development, where further methodological programming improvements are needed. These new methodologies should establish the way an algorithm should behave, in order to meet its special capabilities. They include a large amount of available processors, a moderate individual computing power and a medium to high performance interconnection network. Our work in progress is currently investigating solutions in this field, by the analysis of current parallelization techniques and requirements. Our latest reviews also show that the market trends are following this many-core line as a way to improve performance, being the energy consumption one of the biggest challenges to address.

Acknowledgments. We are grateful to Tilera for providing hardware and software tools <http://www.tilera.com>. This work was supported by "Ministerio de Ciencia e Innovación" [MICINN-FEDER grants BIO2011-15237 and AGL2010-17316]; "Consejería de Agricultura y Pesca" of "Junta de Andalucía" [041/C/2007 75/C/2009 & 56/C/2010]; "Grupo PAI" [AGR-248]; and "Universidad de Córdoba" ["Ayuda a Grupos"], Spain.

References

1. Gálvez, S., et al.: Next-Generation Bioinformatics: Using Many-Core Processor Architecture to Develop a Web Service for Sequence Alignment. Bioinformatics 26(5), 683–686 (2010)
2. Castillo, A., et al.: Genomic approaches for olive oil quality control. In: Plant Genomics European Meetings (Plant GEM 6), Tenerife, Spain (2007)
3. Needleman, S.B., Wunsch, C.D.: A general method applicable to the search for similarities in the amino acid sequence of two proteins. J. Mol. Biol. 48(3), 443–453 (1970)
4. Smith, T.F., Waterman, M.S.: Identification of common molecular subsequences. J. Mol. Biol. 147(1), 195–197 (1981)
5. Gotoh, O.: An improved algorithm for matching biological sequences. J. Mol. Biol. 162(3), 705–708 (1982)
6. Hirschberg, D.S.: A linear space algorithm for computing maximal common subsequences. Commun. ACM 18(6), 341–343 (1975)
7. Driga, A., et al.: FastLSA: A Fast, Linear-Space, Parallel and Sequential Algorithm for Sequence Alignment. Algorithmica 45(3), 337–375 (2006)
8. Thompson, J.D., Higgins, D.G., Gibson, T.J.: CLUSTAL W: improving the sensitivity of progressive multiple sequence alignment through sequence weighting, position-specific gap penalties and weight matrix choice. Nucleic. Acids Res. 22(22), 4673–4680 (1994)
9. Larkin, M.A., et al.: Clustal W and Clustal X version 2.0. Bioinformatics 23(21), 2947–2948 (2007)
10. Li, K.-B.: ClustalW-MPI: ClustalW analysis using distributed and parallel computing. Bioinformatics 19(12), 1585–1586 (2003)
11. Saitou, N., Nei, M.: The neighbor-joining method: a new method for reconstructing phylogenetic trees. Mol. Biol. Evol. 4(4), 406–425 (1987)
12. Sneath, P.H.A., Sokal, R.R.: Numerical Taxonomy. The Principles and Practice of Numerical Classification (1973)
13. Pop, M., Salzberg, S.L., Shumway, M.: Genome sequence assembly: Algorithms and issues. Computer 35(7), 47–48 (2002)
14. Sutton, G.G., et al.: TIGR Assembler: A New Tool for Assembling Large Shotgun Sequencing Projects. Genome Science & Technology 1(1), 11 (1995)
15. Green, P.: Phrap Documentation: Algorithms. Phred/Phrap/Consed System Home Page (2002), `http://www.phrap.org` (cited October 31, 2010)
16. Huang, X., Madan, A.: CAP3: A DNA sequence assembly program. Genome. Res. 9(9), 868–877 (1999)
17. De Bruijn, N.G.: A Combinational Problem. Koninklijke Nederlandse Akademie v. Wetenschappen 49, 758–764 (1946)
18. Pevzner, P.A., Tang, H.X., Waterman, M.S.: An Eulerian path approach to DNA fragment assembly. Proceedings of the National Academy of Sciences of the United States of America 98(17), 9748–9753 (2001)
19. Zerbino, D.R., Birney, E.: Velvet: algorithms for de novo short read assembly using de Bruijn graphs. Genome. Res. 18(5), 821–829 (2008)

20. Chaisson, M.J., Pevzner, P.A.: Short read fragment assembly of bacterial genomes. Genome. Res. 18(2), 324–330 (2008)
21. Butler, J., et al.: ALLPATHS: de novo assembly of whole-genome shotgun microreads. Genome. Res. 18(5), 810–820 (2008)
22. Warren, R.L., et al.: Assembling millions of short DNA sequences using SSAKE. Bioinformatics 23(4), 500–501 (2007)
23. Jeck, W.R., et al.: Extending assembly of short DNA sequences to handle error. Bioinformatics 23(21), 2942–2944 (2007)
24. Dohm, J.C., et al.: SHARCGS, a fast and highly accurate short-read assembly algorithm for de novo genomic sequencing. Genome. Res. 17(11), 1697–1706 (2007)
25. Hernandez, D., et al.: De novo bacterial genome sequencing: millions of very short reads assembled on a desktop computer. Genome. Res. 18(5), 802–809 (2008)
26. Simpson, J.T., et al.: ABySS: a parallel assembler for short read sequence data. Genome. Res. 19(6), 1117–1123 (2009)
27. Shirts, M., Pande, V.S.: COMPUTING: Screen Savers of the World Unite! Science 290(5498), 1903–1904 (2000)
28. Marianayagam, N.J., Fawzi, N.L., Head-Gordon, T.: Protein folding by distributed computing and the denatured state ensemble. Proc. Natl. Acad. Sci. USA 102(46), 16684–16689 (2005)
29. Ding, F., et al.: Ab initio RNA folding by discrete molecular dynamics: from structure prediction to folding mechanisms. RNA 14(6), 1164–1173 (2008)
30. Ding, F., et al.: Ab initio folding of proteins with all-atom discrete molecular dynamics. Structure 16(7), 1010–1018 (2008)
31. Shah, A.A., et al.: Parallel and Distributed Processing with Applications. In: Proceedings of the 2008 International Symposium on Parallel and Distributed Processing with Applications, pp. 817–822 (2008)
32. Shah, A.A., Barthel, D., Krasnogor, N.: Grid and Distributed Public Computing Schemes for Structural Proteomics: A Short Overview. In: Thulasiraman, P., He, X., Xu, T.L., Denko, M.K., Thulasiram, R.K., Yang, L.T. (eds.) ISPA Workshops 2007. LNCS, vol. 4743, pp. 424–434. Springer, Heidelberg (2007)
33. Intel, The SCC Platform Overview (2010), Web: `http://techresearch.intel.com/spaw2/uploads/files/SCC-Overview.pdf` (cited October 31, 2010)
34. Intel, Intel's Teraflops Research Chip (2010), Web: `http://download.intel.com/pressroom/kits/Teraflops/Teraflops_Research_Chip_Overview.pdf` (cited October 31, 2010)
35. Brookwood, N.: AMD Fusion™ Family of APUs: Enabling a Superior, Immersive PC Experience (2010), Web: `http://sites.amd.com/us/Documents/48423B_fusion_whitepaper_WEB.pdf` (cited October 31, 2010)
36. nVidia, Tesla C2050 and Tesla C2070 Computing Processor Board Specification (2010), Web: `http://www.nvidia.com/docs/IO/43395/BD-04983-001_v04.pdf` (cited October 31, 2010)
37. nVidia, GeForce GTX 580 Specification (2010), Web: `http://www.geforce.com/#/Hardware/GPUs/geforce-gtx-580/specifications` (cited October 31, 2010)
38. Tilera, Tile-Gx Processor Family Product Brief, Web: `http://www.tilera.com/sites/default/files/productbriefs/PB025_TILE-Gx_Processor_A_v3.pdf` (cited October 31, 2010)