Azerbaijan Journal of High Performance Computing, Vol. 2, Issue 2, 2019, pp. 122-140
https://doi.org/10.32010/26166127.2019.2.2.122.140

The PARSEC Machine: a Non-Newtonian Supra-linear Supercomputer

Ulviya Abdulkarimova[1,2], Anna Ouskova Leonteva[1], Christian Rolando[3], Anne Jeannin-Girardon[1], Pierre Collet[1]

[1]Strasbourg University, France, ulviya.abdulkarimova@icube.unistra.fr, leopard152015@gmail.com, christian.rolando@univ-lille1.fr, Pierre.Collet@Lsiit.U-Strasbg.Fr
[2]France UFAZ - Azerbaijan State University of Oil and Industry, Baku, Azerbaijan
[3]University of Lille, France

*Correspondence:
Pierre Collet, Strasbourg University, France, Pierre.Collet@Lsiit.U-Strasbg.Fr

Abstract

This paper describes how transfer-learning can turn a Beowulf cluster into a full super-computer with supra-linear qualitative acceleration. Harmonic Analysis is used as a real-world example to show the kind of result that can be achieved with the proposed supercomputer architecture, that locally exploits absolute space-time parallelism on each machine (SIMD parallelism) and loosely-coupled relative space-time parallelization between different machines (loosely coupled MIMD).

Keyword: Beowulf cluster, relative space-time, supra-linear acceleration, qualitative acceleration, GPGPU, loosely coupled machines, artificial evolution, transfer learning, harmonic analysis, super-resolution, non-uniform sampling, Fourier transform.

1. Introduction

Years 2005 and 2006 have been a turning point in computer science. Before 2005, parallel machines were not widely available even though they had been the object of intensive research with, for example, the Connexion Machines CM-1 and CM-2 (Lippert, Schilling & Ueberholz, 1993), but also on a smaller scale transputers (Hull, Sweeney & Crookes, 1994) and vectorial multi-processor machines such as the 1980-90 era CRAY X-MP, Y-MP supercomputers, whose MIMD capabilities were not really exploited, for lack of efficient generic parallel algorithms, which will be the object of this paper. Typically, 8-vectorial processor CRAY Y-MP machines were used as 8 independent vectorial computers.

1.1. A bit of history on the development of parallel hardware and software

In 2005, a first physical barrier was hit for air-cooled personal computer CPUs, with 115W 3.8GHz Intel Pentium 4 Prescott. Because consumption (and therefore heat production) increases as the cube of the clock frequency, the only way to improve on CPU performance was to develop multi-core processors (only twice the consumption for twice the computing power, compared to 8 times the consumption for twice the clock frequency). The first dual-core Intel processor was the 130w 3.2GHz Pentium D Smithfield that came out in May 2005 (reduced clock frequency

for higher consumption) (Krazit, 2005, August, 17). It is since that time that all CPUs are now multi-core CPUs more or less capped at 4GHz (typically only on one core of all the cores of the CPU), which created software-related problems:

- the computing power for highly-demanding algorithms is not increasing anymore for sequential algorithms (99% of all algorithms) and no real speed increase perspective for the future (Intel Core I9 clock speed is still in 5Ghz in 2020),
- the only possible way to exploit the "new" multi-core architecture that is now standard since 2005-2006 requires the development of parallel algorithms, which unfortunately are quite difficult to program efficiently.

1.2. Parallelizing on GPGPU cards

The above observation applies for "standard" algorithms that do not parallelize well. However, graphic algorithms used for 3D rendering through vertex shaders (for games such as Doom) and pixel-based imaging (raster graphics algorithms for software such as photoshop) do not fall in this category as they are inherently (embarrassingly) parallel. Due to the huge gaming market ($150 billion in 2019, larger than the CPU market because there are more gamers than computer scientists) chip manufacturers rapidly created dedicated graphic cards that implemented hard-wired specialized Graphic Processing Units (GPU) for vertex and raster algorithms. However, it is only in 2006 that the computing power of the GPU chips became large enough to be able to produce a fluid real-time rendering of both kinds using software algorithms. Because raster and vertex rendering algorithms are very different, chipmakers had to create General Purpose Graphic Processing Units (GPGPUs) whose cores were de facto fully-fledged, i.e., able to implement any kind of standard algorithms as CPUs would, only with a slower clock-rate but... the first GPGPU card with a dedicated open development environment (NVIDIA GeForce 8800 GTX + CUDA) boasted in 2006 128 cores, compared to the two cores of the Yonah Intel Dual-Core Pentium.

This meant that the first GPGPU card was a massively parallel processor, however with a catch: its architecture made it an SPMD (Single Program Multiple Data) processors composed of SIMD (Single Instruction Multiple Data) multiprocessors, i.e., vector processors not unlike those of the Cray supercomputer series of the 1980s. In practice, this meant that in order to fully exploit the 8800GTX card as a multiprocessor, it was necessary to be able to parallelize an algorithm not only in 128 independent tasks but many more, because of a smart, lightweight system that used a very large number of registers to switch between threads in the same context, as NVIDIA GPGPU cards implement Spatio-temporal parallelism: spatial parallelism over the 128 cores of the GPGPU chip, and temporal parallelism thanks to pipelining. The amazing result is seen in Fig. 1, is that when parallelization is well done on an NVIDIA GPGPU card, evaluation time is constant up to 2048 threads. Then, a transitory phase is observed as the scheduling mechanism starts to deal with more than 16 threads per core, until 3072 parallel evaluations are required, after which the card is fully loaded. Computation time is linear again with the sequential and parallel parts of the code (Maitre, Baumes, Lachiche, Corma, &

Collet, 2009, July).

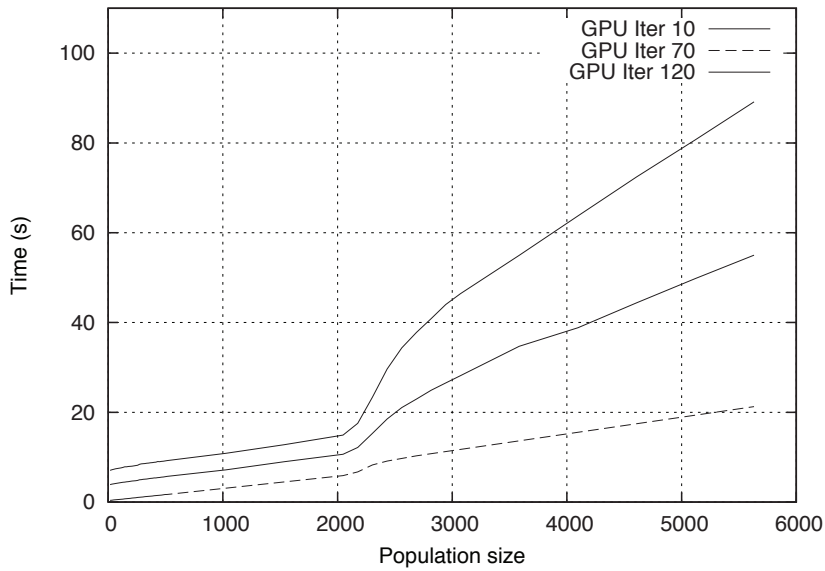

Fig 1: Parallelization of the evaluation of a Weierstrass benchmark function for 10, 70, and 120 iterations on a 128 cores GeForce 8800GTX GPGPU card, for a growing number of individuals of a genetic algorithm. Only the evaluation of individuals is parallelized, so what is seen until Population size 2048 is the increase of the sequential time of the evolutionary algorithm + one evaluation time for 10, 70, 120 iterations of the Weierstrass function. After 3072 individuals total time increase is linear again as the card is fully loaded, with sequential + parallel evaluations adding up.

As shown above, the real number of threads that need to be launched in parallel to fully exploit the scheduling capacity of a GPGPU card is much larger than its number of cores, meaning that for a current RTX 2080 TI card with 4352 cores, it is necessary to parallelize code over several tens of thousands of independent threads in order to be able to load the card correctly. This poses the question of the kind of algorithm that may need/exploit such an extraordinary number of independent threads, and be generic enough so that it can be used to solve nearly any kind of problem.

1.3. Parallelizing on computers / Philosophical introduction to absolute vs. relative space and time

The problem here is quite different as parallelization over several machines requires data to be exchanged over a network, which has its limitations.

In order to facilitate the mathematical description of motion (and more broadly Newtonian Physics) Sir Isaac Newton proposed in Philosophiæ Naturalis Principia Mathematica (Newton, 1687) that (cf. citation in annex):

- space and time be considered as absolute measures,

• time flows regularly and continuously in one direction.

The incredible success of Newtonian physics led most of the physicists to firmly adopt these postulates, even though Leibniz was a ferocious opponent to absolute space and time, as seen in his 1715-16 correspondence with Samuel Clarke (Clarke, Leibniz, 1717), a supporter of Newton. It is not before Einstein's famous June 30th, 1905 paper Zur Elektrodynamik bewegter Körper (On the Electrodynamics of Moving Bodies) (Einstein, 1905) that Newton's absolute definition of space and time were shown as being based on wrong assumptions, by a pure thought experiment (Gedankenexperiment) in a way not dissimilar to how Galileo disproved Aristotle's wrong theory of gravity.

What made physicists validate Newtonian physics is that it is a very good approximation of what can be measured at a mesoscale. This was also true of Aristotle's theory of falling bodies, which more or less corresponds to what is observed if a body falls in a fluid (there was no known way to create a void in the time of Aristotle). If Aristotle's physical theories are not taught anymore, Newton's physics corresponds so well to what is observed in everyday life that it is still taught everywhere in high-schools. It is only at University level that Einstein's theories of special and general relativity are taught, and only to students who study physics, meaning that even though GPS satellites heavily rely on it, most adults still consider Newtonian's physics as true, even though it is only an approximation. This has consequences in many fields, one of which is supercomputing.

In this paper, we will describe the implications of Einstein's special and general relativity theories on supercomputing. We will present a way to address this problem by parallelizing on loosely coupled computers, with algorithms that can exploit knowledge transfer to achieve linear and supra-linear acceleration to solve any kind of continuous, discrete and combinatorial problems. An example will be given with harmonic analysis on a simple system made of 4 computers that exhibit supra-linear acceleration.

2. Relative space and time concepts applied to supercomputing

Absolute space and time are so well anchored into everyone's head that when designing supercomputers, computer scientists naturally followed these principles in designing their distributed algorithms, and therefore the machines to implement them. The problems are that as Einstein explained in (Einstein, A., 1905) (and as before him, Leibnitz had understood), time is not absolute, but relative to each physical entity.

What probably makes humans (and therefore computer scientists) ignore this fact is that there is such an incredible difference between the speed of light (that leads to the Einsteinian notion of locality) and the human perception of time (1/10s) that for most human-scale applications, time (and space) can be considered as being absolute.

However, since 2005, air-cooled processors run at around 4GHz (and much faster in the case of liquid nitrogen cooled supercomputers) meaning that for scalar processors, the execution of one instruction takes around 25 nanoseconds, i.e. the time for a light-speed traveling photon to cover a distance of about 7.5 cm only.

If this distance is of the order of magnitude of the diagonal of a CPU chip, one understands that any supercomputer made of distinct machines whose CPUs or GPUs are more than 7.5 cm away is directly confronted to the principle of locality of Einstein's Special Theory of Relativity described in the famous paper (Einstein, Podolsky and Rosen, 1935), because the fastest high-speed network that can be created cannot transfer data faster than the speed of light.

Then, even if superluminal transmission speed could be achieved between independent processors, there is no common clock binding the different machines of a supercomputer together. Thence, it follows that absolute time is a concept that cannot be used as a basis to design super-computer distributed software:

Absolute and perfect synchronization between machines is hopeless.

In parallel computing, linear acceleration means that using ten computers would allow a 100% parallelizable program to run ten times faster. While it is possible to obtain linear (or even super-linear) acceleration in certain conditions (after data is locally transferred on the correct nodes), this can only be achieved for a small period (transient mode). In continuous operation, there is simply no way for an operator that adds or multiplies two floating-point numbers in .25ns to obtain its operands from distant computers at exactly the right time when it is needed.

Therefore, linear acceleration is simply impossible to achieve in super-computers in continuous operation.

Such perfect synchronization is needed because traditional supercomputers expect to exchange data needed for small grain operators such as an addition of a multiplication. The question posed in this paper is whether exchanging higher-level information could make it possible to alleviate the very strong time constraints of data-exchange.

Indeed, in computer science, a distinction is traditionally made between:

- data: 20,
- information: 20 kg, 20 km, 20C, 20K, ..., and
- knowledge: 20 km is a long walking distance, 20K is very cold for a human.

The problem with processing data is that data carries nearly no meaning, so it is necessary to process millions of pieces of data to extract knowledge out of data. A 10-megapixel photo may contain 30 megabytes if each color pixel is coded on 3 bytes. Each byte of each pixel is a piece of data that does not represent much. However, assembled, the 30 megabytes may represent the photo of a Burmese cat, watching a canary in a red cage in an empty room with white walls.

Somehow, the 80 letters A Burmese cat, watching a canary in a red cage in an empty room with white walls represent knowledge, that can be considered as an approximation of the 30 megabytes of data contained in the photo.

Could exchanging knowledge between computers solve the problem of linear acceleration, i.e., getting ten computers to go ten times faster than one computer to execute one task, on generic problems?

2.1. Transfer Learning

In traditional machine learning (that is, the field of Artificial Intelligence that automatically processes data to extract meaningful relationships between data to

make predictions about unseen data), Transfer Learning is a widely used method that consists, basically, in transferring portions of a trained model (trained on a source domain) to an untrained model (to be trained on a target domain). Transfer Learning is mainly used to cope with some lack of data in the target domain: if the source and target domain are related, it Transfer Learning postulates that it should be possible to reuse the knowledge learned by the source model. For instance, if there exists a model that can predict the risk of developing heart diseases in adults, some of the knowledge learned by could be reused in a model to be trained to predict the same risk but in the case of children (the two domains are different but related). The main challenge is to tackle the differences between the source and the target domains to generalize well on the target domain.

In practice, in the case of Deep Machine Learning, Transfer Learning consists in reusing the parameters learned at the layer levels of Deep Neural Networks. Given the hierarchical nature of Deep Neural Networks (DNN), it has been empirically shown (Yosinski, Clune, Bengio, Lipson, 2014) that transferring the layers closest to the input (low-level features) in trained model A lead to faster learning performance in the training of the untrained target model B. Higher level features (in the layers closer to the outputs and specific to the task at hand) is too specific to be transferable without considerable modifications. Even though this empirical study gave a precious insight regarding what portions of a model can be transferred, it should be noted that, so far, the actual success of a transfer (i.e., whether model is more accurate after the transfer than it would have been without) can only be evaluated with trial and error (i.e., after transferring some parameters, checking the accuracy, transferring different parameters, checking the accuracy, etc.) and this process is extremely time-consuming.

So Transfer Learning is about transferring knowledge rather than data, as what is transferred (parameters of a trained neural network) are representations of the data processed by the trained algorithm A. In other words, the first layers of the trained Neural Network represent an alternative version of the data, most often reduced, expressing the –complex– relationships between the attributes of the data).

Transfer learning can also be applied in the context of Artificial Evolution, as presented in the next sections of this paper.

3. Artificial evolution and evolutionary algorithms

Evolution is a generic solver in the sense that beyond biological evolution (that resulted in creating organisms and animals adapted to their ecosystems), its computer science embodiment makes it capable of finding good solutions to difficult continuous, discrete and combinatorial problems. Evolutionary algorithms are very old. They are among the first ones that were tried on the very first computers (Fraser, 1958), but it is not before the 1970s and 1990s that they started to provide really interesting results. Since 2000, Genetic Programming (a branch of artificial evolution) routinely produces human-competitive results.

Artificial evolution regroups many different kinds of evolutionary algorithms, among which evolution strategies, genetic algorithms, evolutionary programming and genetic programming (Rechenberg, 1965, Rechenberg, 1973, Schwefel, 1965,

Schwefel, 1981, Koza, 1992).

Evolutionary algorithms are very well described in the computer science literature, so they will only be shortly described below. After having decided on a digital representation of a potential solution to a problem (called an individual), the following steps are implemented:

1. Initialize a large number of individuals (called a population)
2. Evaluate the quality of the individuals (called the fitness). Evaluation turns the individuals into potential "parents."
3. For nbGen generations, do:

(a) Create a population of children by:

i. Selecting parents among the best individuals of the population

ii. Creating one child by using variation operators (typically crossover and mutation)

(b) Evaluate the population of children

(c) Reduce the population of individuals back to its initial size by selecting individuals among the best for the next generation

(d) If a termination condition is met (valid solution, computing time constraint, ...), stop the algorithm and return the best individual of the population, otherwise return to step 3.

3.1. Massive parallelization over GPGPU cards

What is very nice with evolutionary algorithms is that they are embarrassingly parallel: suppose that the population size is 1000 individuals and that at every generation, 1000 children are created, then after the initialization step, all initial individuals can be evaluated independently. At each generation, all newly created children can also be evaluated independently.

In the case of evolution strategies or genetic algorithms, the evaluation (fitness) function is identical for all individuals, meaning that these algorithms are perfectly suited for the SIMD parallelism of GPGPU cards (Tsutsui & Collet (Eds.)., 2013).

The EASEA (EAsy Specification of Evolutionary Algorithms) platform was created in 1998 in order to allow non-computer scientists (mathematicians, physicists, chemists, biologists, engineers) to use artificial evolution to solve their difficult problems. It was first described in a paper in 2000 (Collet P., Lutton E., Schoenauer M., Louchet J., 2000). In 2006, the first NVIDIA General Purpose Graphic Processing Unit came out in the 8800GTX graphics card, along with the CUDA development environment. A Ph.D. (Maitre, 2011) was then started to efficiently and automatically parallelize the algorithms implemented by the EASEA platform and the first papers started to come out in 2009. Acceleration factors of more than x100 vs. a sequential version on CPU are regularly achieved on most kinds of problems.

What is wonderful in the parallelization of evolutionary algorithms is that:

1. Using thousands of individuals makes it possible to have a near-perfect SIMD parallelization of the completely sequential fitness function as it is not the code of the function that is parallelized. However, the execution of the same sequential fitness function launched in parallel with thousands of different parameters (individuals).
2. All the steps of the evolutionary algorithm can be parallelized (not only the

evaluation part).

So Artificial Evolution is a way to efficiently parallelize sequential code on potentially thousands of cores (in fact, no parallelization is needed: thousands of sequential evaluations are launched in parallel). Evolution takes care of efficiently exploiting the sequential computations done in parallel.

In the example used below to show how PARSEC computers are working, the parallel evaluation was performed on equations potentially modeling Fourier Transform Ion Cyclotron Resonance (FT-ICR) spectrometry data. The sequential evaluation of one population generation on an Intel Core i7-9700K CPU overclocked to 4.6GHz necessitated 177s. Parallel execution of the Parallel evaluation of the same population generation on an NVIDIA GEFORCE RTX2080 TI GPGPU necessitated 0.98s, for an obtained speedup of .

This was done by using standard SIMD parallelization techniques that are described in (Maitre, Baumes, Lachiche, Corma & Collet, 2009, July).

3.2. Parallelizing over several computers via transfer learning (island model)

As was described in section 2.1, when parameters of a trained neural network are transferred to another neural network, it is not data that is transferred, but knowledge. Indeed, the parameters of the source trained network can be regarded as a model of the data on which the neural network was trained.

Supposing now that several machines try to optimize the same problem. Individuals that are evolved in these machines correspond to potential solutions to the problem, so the individuals carry knowledge about the problem. Therefore, exchanging individuals between different machines running a similar evolutionary algorithm can be considered as Transfer Learning.

In Artificial Evolution, this type of parallelization is called island parallelization, as a reference to the ecological niches found by Darwin on different Galapagos islands during his famous trip on HMS Beagle. The interesting evolutionary feature of evolving different populations on different islands is that the populations are kept distinct thanks to the distance separating the islands. Only once in a while, during a storm, can the wind blow a bird from an island to another island. This is exactly what is done when implementing the island model in artificial evolution.

Once in a while, an individual carrying knowledge is transferred from island to island . If the individual from island is not as good as the population in island, it will not be chosen for breeding and will not be selected to survive to the next generation. It will quickly disappear from the population of island .

On the contrary, if the individual from island is better than the best individuals of island, then it will be elected for breeding and will bring genetic diversity to island, preventing it from prematurely converging towards a local optimum.

Therefore, implementing an island model preserves diversity and the efficiency of crossover when creating children (cf. experiments below).

4. Introducing PARSEC machines

PARSEC stands for PARallel System for Evolutionary Computing. A PARSEC machine is made of a number of computers that are interconnected with a low-

speed network (typically a cheap Ethernet network). The main difference with a Beowulf cluster (Becker, Sterling, Savarese, Dorband, Ranawak & Packer, 1995, August) is that PARSEC machines can use heterogeneous machines, running at a different speed, and even heterogeneous operating systems. The other difference with a Beowulf cluster is that in order to overcome the heterogeneity of the machines, a PARSEC machine must run collaborative software that can exchange knowledge, not data.

The specificity of PARSEC machines is that they are loosely coupled. In contrast, standard supercomputers are made of tightly coupled machines interconnected with very expensive high-speed fiber-optic networks that represent a big fraction of the cost of a supercomputer. Another problem with standard supercomputers is that if one part of the hardware breaks down (the power supply of one of the computers that are part of the supercomputer, for instance), the whole computation must stop until the repair is done.

When sharing knowledge using the Island model, all computers are independent and continue trying to solve the problem on their own. If one machine breaks down, computing continues on the other machines. The only implication of having one less machine is that the other machines will not be able to benefit from its computing power. When the broken down machine is repaired, and up again, computing can be restarted from scratch on this machine only. Very soon, the newly restarted machine will start receiving the latest solutions meaning that it will recover and be operational again on the cutting edge of the computation within seconds. Thus, PARSEC machines are inherently robust.

5. Experimenting with a PARSEC machine on the harmonic analysis problem

5.1. Description of the harmonic analysis problem

Harmonic analysis is a mathematical method used to express signals as sums of sines. Several methods have been proposed for the analysis of harmonic signals, Fast Fourier Transform (FFT) being the most widely used one. Nowadays, with advances in technology, obtaining large and complex data has become very common. When the amount of data is large and these data are noisy, the FFT method performs poorly, especially in finding the phase parameter of sinusoidal functions. Standard methods use denoising algorithms to remove the noise from the signals before applying the Fourier Transform. The number of denoising algorithms has been developed (Chiron, van Agthoven, Kieffer, Rolando & Delsuc, 2014) but computation time is very long when the data is large and complex and even with denoised signals, the Fourier Transform does not determine the phase parameter correctly.

In this paper, we present an artificial evolution approach run on a PARSEC machine in order to determine the harmonic components of a noisy signal produced by an FT-ICR mass spectrometer. FT-ICR mass spectrometers are used to measure the weight of molecules rotating in an electromagnetic field. As molecules rotate, a sinusoidal image is produced and the resulting signals can be modeled using a sum of sine waves, as given in the following formula:

$$\text{Signal}\,[i] = \sum_{\text{peaks}} A \times \sin(\omega \times i + \phi) + \eta \quad (1)$$

where A is the amplitude, ω is the frequency, ϕ is the phase, and η is noise.

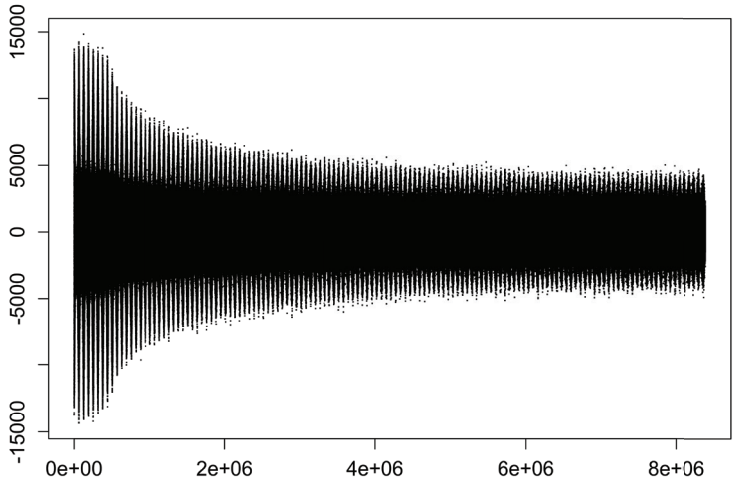

Fig. 2: Complete data for Substance P is obtained from the FT-ICR spectrometer of MSAP (Miniaturization, for Synthesis, Analysis, and Proteomics CNRS USR 3290 laboratory, Lille University). The vertical lines are beats due to the constant interval between the isotopic peaks of a pure substance.

The evolutionary algorithm has been previously applied in a few numbers of publications for analyzing harmonic signals. Choy & Sanctuary, (1998) used a genetic algorithm method for harmonic analysis of NMR (Nuclear Magnetic Resonance) signals with the low signal-to-noise ratio. It should be noted that the NMR spectrum is much smaller in size compared to the FT-ICR spectrum: they used a small data set with only 128 points on a Pentium 200MHz PC in their analysis. However, this method was costly in time and thus, it was recommended to use it only when the signal-to-noise ratio is low since conventional methods perform poorly in this case. Zamanan, Sykulski & Al-Othman, (2006, September) applied a real coded genetic algorithm for harmonic analysis and used simulated data to test their method. They concluded that a signal with a sum of 16 sinusoids could be estimated with only 20 points using the evolutionary algorithm method, whereas 200 000 points are needed if the FFT method is used.

5.2. Description of the data

Fig. 2 shows the data output from the FT-ICR machine for Substance P. The generated dataset contains 8 million points on one dimension. As observed in Fig. 2, the signals are damped all along with the acquisition, as molecules are interfering with the imperfect vacuum in the machine. In order to find a sum of sines (that has a constant amplitude), genetic programming was used to find the damping equation

matching the energy loss of the molecules. The function is then used to normalize the data by straightening the signal.

6. Obtained results

The first point to discuss is whether artificial evolution is capable of finding a good solution to the harmonic analysis problem.

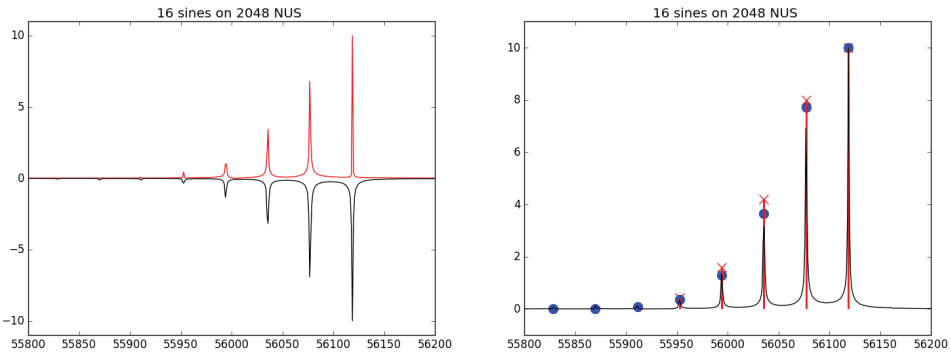

Fig. 3: Spectrum of the evolved functions (red). The 6th peak can be seen.

Only a very small part of the signal has been used (2048 points) randomly extracted from a larger set (16384 points) out of the 8M points (Non-Uniform Sampling). The genetic algorithm breaks the Nyquist constraint which states that the acquisition should comprise at least two complex signal periods, meaning that evolutionary algorithms could be used for faster acquisition of ion cyclotron resonance mass spectrometry. The precision obtained on peak position is equivalent to the one obtained by FFT, whereas the accuracy on peak intensity is better. Furthermore, the same algorithm can deconvolute non-uniform sampling data obtained by skipping points in the acquisition which is a major improvement in 2D spectroscopies.

The presented method is not costly in time as its execution time is linearly proportional to the sampling size and the number of sines in the signal. So it can be used in the processing of large datasets such as those produced in FT-ICR mass spectrometry where a single spectrum may contain 16M points. The application to real-world complex mixtures which may comprise up to 100,000 sines is under investigation and to 2D FT-ICR mass spectrometry which is currently limited by the computational power (Smith, Podgorski, Rodgers, Blakney & Hendrickson, 2018).

6.1. Discussion on the computation time between the island and isolated runs

In this section, two main sets of experiments are presented, in which harmonic analysis is performed by using:

• an isolated artificial evolution algorithm with 262 144 individuals.

• 4 islands with 65 536 individuals each, loosely coupled over an Ethernet network, exchanging individuals every second.

Individuals are composed of the parameters for 10 sines, i.e. 30 single precision real values (120 bytes), so the total load on the network between the 4 machines is to transfer 480 bytes per second, without any synchronization between the machines (incoming individuals are integrated at the next new generation, where they replace a bad individual in the accepting island).

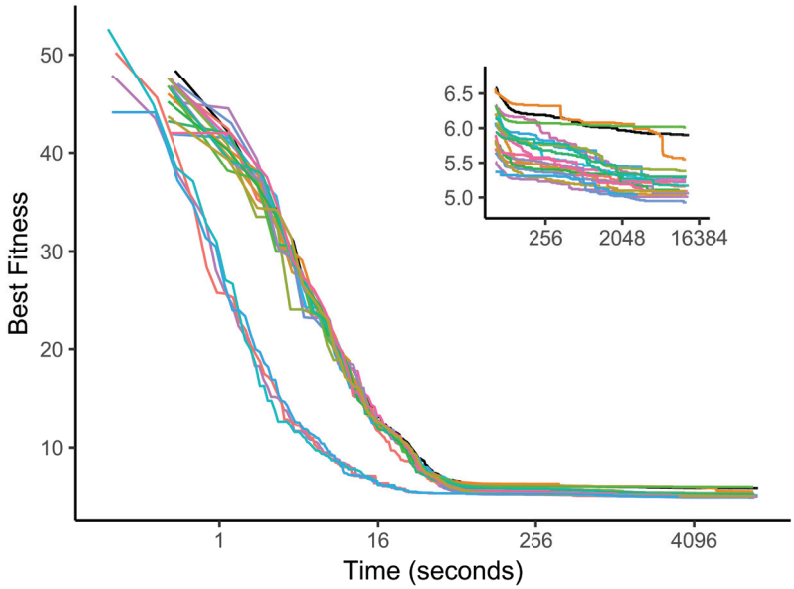

Fig. 4: Evolution of the fitness of the island model (left curves) vs isolated runs (right curves). Lower values means better data fitting. Nb: the x-axis is given in log scale.

6.1.1. Fitness evolution of the island model vs isolated runs for similar end results

What can be seen in Fig. 3 is that the best fitness of the islands (that represents the error between the evolved sum of sines and the obtained data that must be minimized) improves much faster than the fitness of the isolated runs.

This is expected as 4 machines have 4 times the computing power of one machine. The big question is whether loosely coupled machines can cooperate well enough to be able to exhibit linear acceleration.

Because artificial evolution is a stochastic algorithm that contains a random part, it is important to show boxplots over several experiments. Fig. 5 shows the boxplots of the best fitness on 18 isolated runs (red) and the 4 islands (blue). One can see that variability is much more important in the 18 262144 isolated independent runs than on the island model, where standard deviation is negligible. In fact, in order to find results that are comparable in quality to the island model (below value 5), it was necessary to

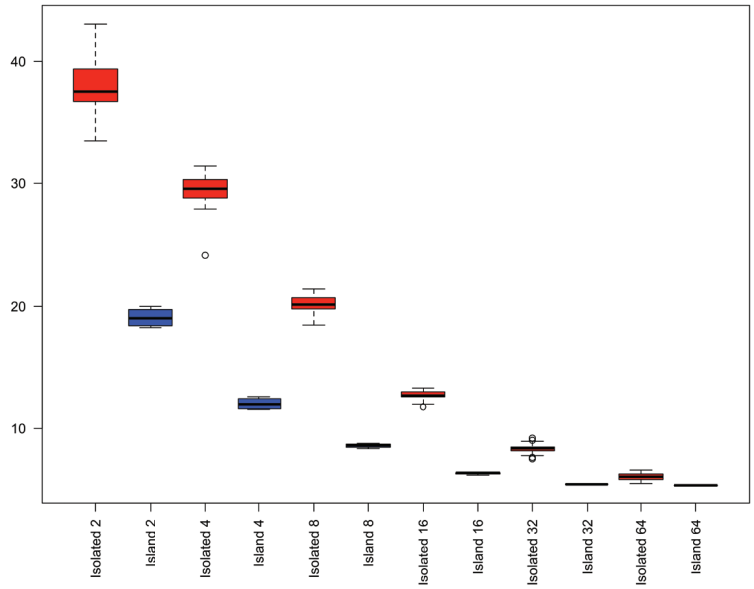

Fig. 5: Boxplots comparing the obtained error values for isolated and 4-island runs after 2, 4, 8, 16, 32 and 64 seconds (once more a log scale).

launch 18 independent isolated runs, meaning that the runtime for isolated runs should in effect be multiplied by 18 to obtain comparable results than with the island model.

7. Defining qualitative acceleration

Usually, the maximum acceleration that can be obtained by parallelizing an algorithm is described in terms of Amdahl's law (Amdahl, 1967, April):

$$A = \frac{s+p}{s+p/N} \tag{2}$$

with A the maximum expected acceleration for N the number of processors, s the sequential time on one processor and p the sequential time of a perfectly parallelizable piece of code.

Gustafson's law shows how this equation is too restrictive, as what must be taken into account is proportional acceleration, and not absolute acceleration:

$$A_p = \frac{s\prime+p\prime N}{s\prime+p\prime} \tag{3}$$

with A_p being Gustafson's proportional acceleration, N the number of processors and s' and p' the sequential and parallel time taken on the parallel system (Gustafson, 1988).

However, both acceleration metrics refer to the $number$ of instructions executed per second in sequential or parallel systems. We will say that these metrics define quantitative acceleration.

However, on real-world problems, what is interesting is not the number of instructions performed per second, but the quality of the results obtained for a given run time.

Because the island model presented in this paper to interconnect different machines is a complex system with emergent properties, we are interested in measuring acceleration obtained by the island model vs the isolated model not in terms of number of instructions per second but in term of necessary time to obtain a similar quality, as measured by the error to be minimized between the acquired signal and the sum of sines that is evolved to model the signal.

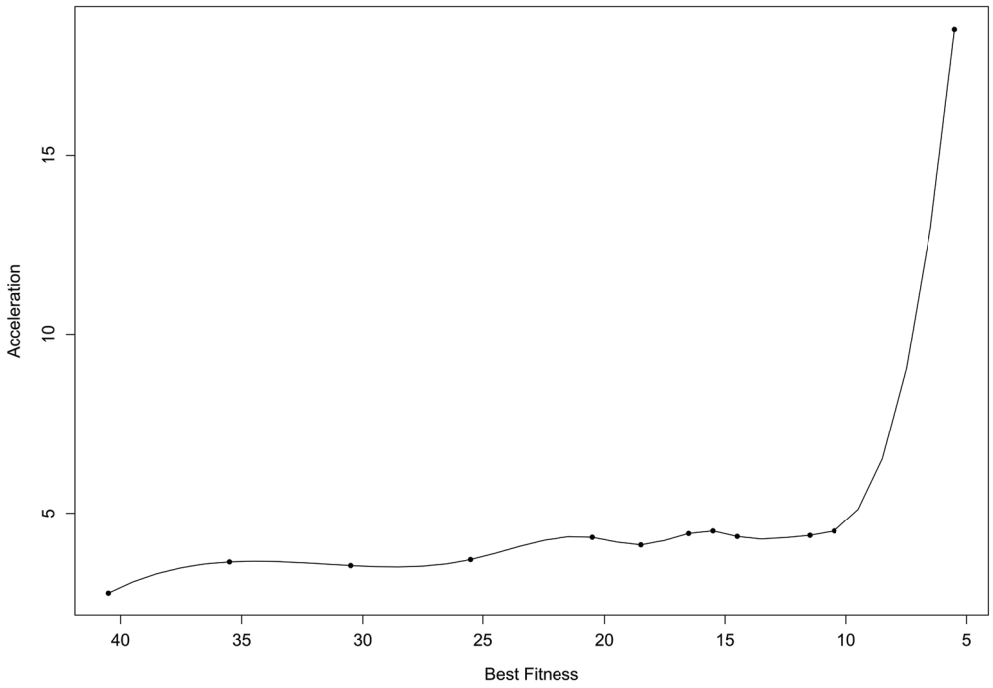

Fig. 6: Acceleration plot of island runs with respect to single runs.

We therefore propose a new acceleration metrics that we call qualitative acceleration, defined by the ratio between the time needed for the island model to obtain an error value over the time needed for an isolated algorithm to obtain the same error value:

$$A_q = \frac{t_{im}(\epsilon)}{t_{is}(\epsilon)} \tag{4}$$

With A_q the quantitative acceleration, $t_{im}(\epsilon)$, the time for the island model to reach error value ϵ and $t_{is}\epsilon$ the time for the isolated algorithm to reach the same ϵ error value.

Qualitative acceleration can be plotted by taking slices from the Best Fitness over

Time curve (Fig. 3) and having on the x axis the values attained by both experiments and in y axis the time ratio between isolated model (reference) and island model.

The result is Fig. 5, where we can see the time ratio between both models. The time to find value 40.5 is roughly similar for both models. however, it is about 2.5 faster to find value 35.5 with the island model than with the isolated model. This represents an infra-linear acceleration as 4 machines are only 2.5 times faster to find the same result.

However, things get interesting as it becomes more and more difficult to find low error values. From error value 20 to 10, linear acceleration (≈ 4) is achieved.

Then, something really nice happens: 4 machines obtain value 5.5 nearly 20 times faster than one single machine, with an identical population size.

Values 5.5 was chosen as the lowest value to compare both models because it was the only value found by at least 5 isolated algorithms (the 13 other could not find this error value).

7.1. Defining supralinear acceleration

Linear acceleration is defined as obtaining and $\times N$ speedup with N machines. On Fig. 5, this would appear as a horizontal line with y value 4.

Super-linear acceleration would be represented by a horizontal line with a y value greater than 4. However, this is not what is observed on Fig. 5.

Beyond value 10 (when it becomes really difficult to find better results), the island model becomes much faster as its individuals still benefit from efficient crossover operators, because the rare individual migrations between the islands prevented their population from converging to a local optimum, therefore maintaining genetic diversity between the individuals. Indeed, after a population algorithm has converged (i.e. when all the individuals are clones, stuck in a local optimum, sharing identical genes), crossover is ineffective as children will be identical to the parents. This is what is happening in the panmictic isolated 262144 individuals islands, that can only rely on mutation to find better results below error value 10.

We therefore define supralinear acceleration as a non-constant positive acceleration evolution, which keeps improving well beyond superlinearity. Indeed, observed acceleration on the shown example stops at $\approx 20\times$ with only 4 machines, but this is only due to the fact that we stopped qualitative comparison at value 5.5.

Supralinear acceleration will still increase until the island model finds values that cannot be obtained by isolated panmictic runs, in which case acceleration will de facto increase to infinite values.

8. Discussion on PARSEC machines vs. standard supercomputers and conclusion

Standard supercomputers are nowadays often made of many independent computers hosting one or several GPGPU cards. Whereas parallelization on the GPGPU cards is difficult to do efficiently for standard algorithms (because they require

to be able to identify tens of thousands of independent threads in the algorithm that must run in SIMD mode), the very high clock frequency speed shows achieved nowadays by CPUs show that it is impossible to perfectly synchronize different machines due to Einstein's principle of locality exposed in his 1905 and 1935 papers.

Attempting to obtain linear accelerations with the number of computers for continuous operation is, therefore, a hopeless quest that is bound to fail.

Evolutionary algorithms are generic solvers that can tackle all kinds of difficult problems, be they continuous, discrete or combinatorial. They are embarrassingly parallel, meaning that they parallelize perfectly on SIMD GPGPU cards, which more or less impose that all cores execute the same instruction at the same nanosecond, over tens of thousands of threads. , requires absolute time that is achievable inside a single GPGPU chip, but unachievable in between CPUs.

However, because evolutionary algorithms can use Transfer Learning, exchanges between CPU is not limited to meaningless data, that requires millions of bytes to describe a simple cat (photo). Exchanging individuals between islands run on different CPU makes it possible to achieve not only linear or super-linear acceleration, but supra-linear acceleration on several machines, by exchanging but a few bytes per second (480 in the presented example).

This makes it possible to create efficient loosely-coupled PARSEC supercomputers for the only cost of computing power i.e. a fraction of the cost of a standard supercomputer. Then, the island model makes PARSEC computers robust to failures, meaning that no expensive infrastructure is needed to run them.

In conclusion, the presented experiments show that very good quality results can be obtained on a state of the difficult art problem (harmonic analysis) by using loosely coupled PARSEC machines, without the use of an expensive high-speed fiber-optic network to interconnect them. × 180 acceleration was achieved on SIMD parallelization of the 262144 individuals of the algorithm which had to be multiplied by > 20 to take into account the acceleration obtained on four island-interconnected machines, amounting to a $\times 3600$ acceleration factor.

Results obtained in 1 minute on this 4-machine PARSEC system were, therefore, equivalent to 60 hours run for a sequential algorithm on a CPU.

We believe cheap PARSEC machines could be a good alternative to current million dollars super-computers.

Acknowledgements

We would like to thank the French-Azerbaijani UFAZ University for supporting the PhD of Ulviya Abdulkarimova.

Annex

Quotation of Scholium (page 5) of chapter 1 (Definitions) of Newton's MATHEMATICAL PRINCIPLES OF NATURAL PHILOSOPHY:

Hitherto I have laid down the definitions of such words as are less known, and explained the sense in which I would have them to be understood in the following discourse. I do not define time, space, place and motion, as being well known to all. Only I must observe, that the vulgar conceive those quantities under no other notions but from the relation they bear to sensible objects. And thence arise certain prejudices, for the removing of which, it will be convenient to distinguish them into absolute and relative, true and apparent, mathematical and common.

I. Absolute, true, and mathematical time, of itself, and from its own nature, flows equably without regard to anything external, and by another name is called duration: relative, apparent, and common time, is some sensible and external (whether accurate or unequable) measure of duration by the means of motion, which is commonly used instead of true time; such as an hour, a day, a month, a year.

II. Absolute space, in its own nature, without regard to anything external, remains always similar and immovable. Relative space is some movable dimension or measure of the absolute spaces; which our senses determine by its position to bodies; and which is vulgarly taken for immovable space; such is the dimension of a subterraneous, an aereal, or celestial space, determined by its position in respect of the earth.

References

Lippert, T., Schilling, K., & Ueberholz, P. (1993). Science on the Connection Machine: Proceedings of the First European CM Users Meeting. (pp. 1-238).

Hull, M. E., Sweeney, P. J., & Crookes, D. (1994). *Parallel Processing; The Transputer and Its Applications.* Addison-Wesley Longman Publishing Co., Inc..

Krazit T. (2005, August, 17). *First Dual-Core Pentium 4 a Rush Job, Intel Says.* Retrieved from: https://www.pcworld.com/article/122236/article.html

Maitre, O., Baumes, L. A., Lachiche, N., Corma, A., & Collet, P. (2009, July). Coarse grain parallelization of evolutionary algorithms on GPGPU cards with EASEA. In *Proceedings of the 11th Annual conference on Genetic and evolutionary computation* (pp. 1403-1410). ACM.

Newton, I. (1687). Philosophiae naturalis principia mathematica. Royal Society, London.

Clarke, S. Leibniz, G. W. (1717) A Collection of Papers, which passed between the late Learned Mr. Leibniz, and Dr. Clarke, In the Years 1715 and 1716. James Knapton, London.

Einstein, A. (1905). On the electrodynamics of moving bodies. Annalen Phys. 17, 891–921.

Einstein A., Podolsky B. and Rosen N. (1935). Can Quantum-Mechanical Description of Physical Reality Be Considered Complete?, in Phys. Rev., vol. 47, p. 777-780

Yosinski, J., Clune, J., Bengio, Y., Lipson, H. (2014). How transferable are features in deep neural networks? Advances in Neural Information Processing Systems. NIPS Foundation. 27, 3320–3328.

Fraser A.S. (1958). “Monte Carlo analyses of genetic models”. Nature. 181 (4603): 208-209. doi:10.1038/181208a0

Tsutsui, S., & Collet, P. (Eds.). (2013). *Massively parallel evolutionary computation on GPGPUs* (Vol. 453). Heidelberg: Springer.

Rechenberg, I. (1965). Cybernetic solution path of an experimental problem. Royal AircraftEstablishment, Farnborough p. Library Translation 1122.

Rechenberg, I. (1973). Evolutionsstrategie: Optimierung Technischer Systeme Nach Prinzipien Der Biologischen Evolution. Fromman-Hozlboog Verlag, Stuttgart, (1973) (in German)

Schwefel, H.-P. (1965). Kybernetische evolution als strategie der experimentellen forschung in der stroemungstechnik. Dipl.-Ing. thesis (in German)

Schwefel, H. P. (1981). *Numerical optimization of computer models.* John Wiley & Sons, Inc.

Koza, J. (1992). GP: On the programming of computers by means of natural selection.

Collet P., Lutton E., Schoenauer M., Louchet J. (2000) Take It EASEA. In: Schoenauer M. et al. (eds) Parallel Problem Solving from Nature PPSN VI. PPSN 2000. Lecture Notes in Computer Science, vol 1917. Springer, Berlin, Heidelberg

Maitre O. (2011). GPGPU for Evolutionary Algorithms, PhD thesis.

Maitre, O., Baumes, L. A., Lachiche, N., Corma, A., & Collet, P. (2009, July). Coarse grain parallelization of evolutionary algorithms on GPGPU cards with EASEA. In *Proceedings of the 11th Annual conference on Genetic and evolutionary computation* (pp. 1403-1410). ACM.

Maitre, O., Lachiche, N., Clauss, P., Baumes, L., Corma, A., & Collet, P. (2009, August). Efficient parallel implementation of evolutionary algorithms on GPGPU cards. In *European Conference on Parallel Processing* (pp. 974-985). Springer, Berlin, Heidelberg.

Becker, D. J., Sterling, T., Savarese, D., Dorband, J. E., Ranawak, U. A., & Packer, C. V. (1995, August). BEOWULF: A parallel workstation for scientific computation. In *Proceedings, International Conference on Parallel Processing* (Vol. 95, pp. 11-14).

Chiron, L., van Agthoven, M. A., Kieffer, B., Rolando, C., & Delsuc, M. A. (2014). Efficient denoising algorithms for large experimental datasets and their applications in Fourier transform ion cyclotron resonance mass spectrometry. *Proceedings of the National Academy of Sciences, 111*(4), 1385-1390.

Flynn, M. J. (1972). Some computer organizations and their effectiveness. *IEEE transactions on computers, 100*(9), 948-960.

Choy, W. Y., & Sanctuary, B. C. (1998). Using genetic algorithms with a priori knowledge for quantitative NMR signal analysis. *Journal of Chemical Information and Computer Sciences, **38***(4), 685-690.

Zamanan, N., Sykulski, J. K., & Al-Othman, A. K. (2006, September). Real coded genetic algorithm compared to the classical method of fast fourier transform in harmonics analysis. In *Proceedings of the 41st International Universities Power Engineer-*

ing Conference (Vol. 3, pp. 1021-1025). IEEE.

Gustafson, J. L. (1988). Reevaluating Amdahl's law. *Communications of the ACM, 31*(5), 532-533.

Amdahl, G. M. (1967, April). Validity of the single processor approach to achieving large scale computing capabilities. In *Proceedings of the April 18-20, 1967, spring joint computer conference* (pp. 483-485). ACM.

Smith, D. F., Podgorski, D. C., Rodgers, R. P., Blakney, G. T., & Hendrickson, C. L. (2018). 21 tesla FT-ICR mass spectrometer for ultrahigh-resolution analysis of complex organic mixtures. *Analytical chemistry, 90*(3), 2041-2047.

Submitted 10.07.2019
Accepted 22.10.2019