The Journal of Supercomputing

On solving the unrelated parallel machine scheduling problem: active microrheology as a case study

--Manuscript Draft--

Manuscript Number:	
Full Title:	On solving the unrelated parallel machine scheduling problem: active microrheology as a case study
Article Type:	S.I. : High Performance Computing in Science and Engineering – CMMSE-2019
Keywords:	Parallel scheduling; heterogeneous cluster; unrelated machines; genetic algorithm
Corresponding Author:	Gloria Ortega, PhD University of Almeria Almeria, Spain SPAIN
Corresponding Author Secondary Information:	
Corresponding Author's Institution:	University of Almeria
Corresponding Author's Secondary Institution:	
First Author:	Francisco Orts
First Author Secondary Information:	
Order of Authors:	Francisco Orts
	Gloria Ortega, PhD
	Antonio Manuel Puertas
	Inmaculada García Fernández
	Gracia Ester Martín Garzón
Order of Authors Secondary Information:	
Funding Information:	Spanish Science and Technology Commission (CICYT) (RTI2018-095993-B-I00) — Not applicable
	Spanish Science and Technology Commission (CICYT) (FIS2015- 69022-P) — Dr. Antonio Manuel Puertas
	Junta de Andalucía (P12-TIC-301) — Not applicable
	Spanish Science and Technology Commission (CICYT) (TIN2015-66680) — Not applicable
Abstract:	Modern computational platforms are characterized by the heterogeneity of their processing elements. Additionally, there are many algorithms which can be structured as a set of procedures or tasks with different computational cost. Balancing the computational load among the available processing elements is one of the main keys for the optimal exploitation of such heterogeneous platforms. When the processing time of any procedure executed on any of the available processing elements is known, this workload balancing problem can be modelled as the well-known scheduling on unrelated parallel machines problem. Solving this type of problems is a big challenge due to the high heterogeneity on both, the tasks and the machines. In this paper, the balancing problem has been formally defined as a global optimization problem which minimizes the makespan (parallel runtime) and a new heuristic based on a Genetic Algorithm, called Genetic Scheduler (GenS), has been developed to solve it. In order to analyze the behavior of GenS for several heterogeneous clusters, an example taken from the field of statistical mechanics has been considered as a case study: an active microrheology model. Given this type of problem and a heterogeneous cluster, we

Powered by Editorial Manager® and ProduXion Manager® from Aries Systems Corporation

	seek to minimize the total run-time to extend and analyze in depth the case of study. In such context, a task consists of the simulation of a tracer particle pulled into a cubic box with smaller bath particles. The computational load depends on the total number of the bath particles. Moreover, GenS has been compared to other dynamic and static scheduling approaches. The experimental results of such a comparison show that GenS outperforms the rest of the tested alternatives achieving a better distribution of the computational workload on a heterogeneous cluster. So, the scheduling strategy developed in this paper is of potential interest for any application which requires the execution of many tasks of different duration (a priori known) on a heterogeneous cluster.

Powered by Editorial Manager® and ProduXion Manager® from Aries Systems Corporation

Manuscript Click here to access/download;Manuscript;template.pdf

Click here to view linked References

Noname manuscript No.
(will be inserted by the editor)

On solving the unrelated parallel machine scheduling problem: active microrheology as a case study

F. Orts · G. Ortega · A.M. Puertas · I. García · E.M. Garzón

Received: date / Accepted: date

Abstract Modern computational platforms are characterized by the heterogeneity of their processing elements. Additionally, there are many algorithms which can be structured as a set of procedures or tasks with different computational cost. Balancing the computational load among the available processing elements is one of the main keys for the optimal exploitation of such heterogeneous platforms. When the processing time of any procedure executed on any of the available processing elements is known, this work load balancing problem can be modeled as the well-known *scheduling on unrelated parallel machines* problem. Solving this type of problems is a big challenge due to the high heterogeneity on both, the tasks and the machines. In this paper, the balancing problem has been formally defined as a global optimization problem which minimizes the makespan (parallel runtime) and a new heuristic based on a Genetic Algorithm, called Genetic Scheduler (GenS), has been developed to solve it. In order to analyze the behavior of GenS for several heterogeneous

F. Orts
Informatics Department, University of Almería, ceiA3, Carretera Sacramento s/n, Almería, Spain
Tel.: +34950214393
E-mail: francisco.orts@ual.es

G. Ortega
Computer Architecture Department, Campus Teatinos, Universidad de Málaga, Málaga, Spain

A.M. Puertas
Department of Applied Physics, University of Almería, Carretera Sacramento s/n, Almería, Spain

I. García
Computer Architecture Department, Campus Teatinos, Universidad de Málaga, Málaga, Spain

E.M. Garzón
Informatics Department, University of Almería, ceiA3, Carretera Sacramento s/n, Almería, Spain

clusters, an example taken from the field of statistical mechanics has been considered as a case study: an active microrheology model. Given this type of problem and a heterogeneous cluster, we seek to minimize the total run-time to extend and analyze in depth the case of study. In such context, a task consists of the simulation of a tracer particle pulled into a cubic box with smaller bath particles. The computational load depends on the total number of the bath particles. Moreover, GenS has been compared to other dynamic and static scheduling approaches. The experimental results of such a comparison show that GenS outperforms the rest of the tested alternatives achieving a better distribution of the computational workload on a heterogeneous cluster.So, the scheduling strategy developed in this paper is of potential interest for any application which requires the execution of many tasks of different duration (a priori known) on a heterogeneous cluster.

Keywords Parallel scheduling · heterogeneous cluster · unrelated machines · genetic algorithm

1 Introduction

Modern computational systems consist of heterogeneous clusters which are composed by the interconnection of Processing Units (PUs) with different computational power, such as CPU-cores, GPUs and so on [1]. Algorithms developed for this kind of platforms have to treat such heterogeneity to efficiently exploit the different resources on modern computers. To this effect, the programmer is responsible for explicitly selecting the devices and mapping the tasks among PUs. So, scheduling techniques become one of the most challenging problems, having a tremendous impact on performance. Many examples of parallel applications consist of a set of independent tasks, with different computational cost, which have to be scheduled on a set of heterogeneous processing units in an optimal way. This problem can be modeled as a *scheduling tasks on unrelated parallel machines* problem, which is NP-complete [2] and very well-known in the field of operational research [3].

There are two different approaches for the scheduling problems, dynamic and static. The dynamic one is based on the definition of a global queue of tasks from which every available PU picks a new task up. A dynamic scheduling does not need any a priori information and usually is the best option when the tasks load is unpredictable. However, dynamic scheduling can produce non-optimal solutions when the tasks runtime is strongly heterogeneous. Several dynamic interfaces have emerged in the last few years to face scheduling in heterogeneous clusters. For instance, StarPU [4], Qilin [5] and Scout [6] offer different methods to map tasks to CPU and GPU. The disadvantage of these paradigms are that they require the programmers to rewrite their codes using a new programming language in the case of StarPU or Scout or using specific APIs in Qilin [7].

On the other hand, static approaches are useful when it is possible to have an estimation of the runtime of the tasks a priori. In such cases, they can provide results near optima since they consider the problem from a holistic view. This

paper is focussed on such context. Although this kind of problems is challenging, it can be efficiently solved if an a priori knowledge about the runtime of every task on every machine is considered. The proposed scheduling is of potential interest for any problem that meets the above mentioned premises.

In this work an Active Microrheology Model (AMM) in hard colloids has been selected as a case study to illustrate the scheduling of simulations of bulk systems on heterogeneous platforms. From the computational point of view, simulations of bulk systems have huge requirements and can be structured as a set of independent tasks with different computational loads (simulations of systems with different sizes).

Here, a finite size analysis is used to extrapolate the results from a finite system to an infinite one, requiring simulations of systems with different (large) sizes. In active microrheology, the mechanical and flow behavior of a complex fluid is studied at the microscopic level[11,12]. Therefore, in order to compute the microviscosity for a bulk system, it is necessary to run simulations of systems with different sizes, and extrapolate to the infinite system relying on the model. Note that for every system size, many tracer trajectories must be evaluated (typically 500 in this work) to obtain a good estimation of the average tracer velocity. In the context of AMM simulations, it is feasible to have a good a priori estimation of the simulation time on different processing units.

So, a static scheduling based on a global analysis is an appropriate option to optimize the parallel execution of such simulations. In this paper, the scheduling strategy is formally defined as a global optimization problem which minimizes the makespan (parallel runtime of the simulation processes). Then, a new heuristic based on a Genetic Algorithm is developed to solve the scheduling on unrelated parallel machines. Hereinafter, it is referred to as the Genetic Scheduler (GenS). Other scheduling approaches (two dynamic and two static strategies) are revised and comparatively evaluated with respect to GenS. Our results show that GenS outperforms the other scheduling methods in terms of makespan using the paradigmatic case study.

The main contributions of the paper can be summarized as follows: (1) a new scheduling heuristic based on a Genetic Algorithm to efficiently distribute the heterogeneous tasks on heterogeneous resources has been designed and comparatively evaluated; (2) this scheduling makes feasible to solve computationally harder simulations of the active microrheology case study; (3) a scheduling software to efficiently distribute a set of independent tasks with different costs on heterogeneous processing units, called GenS, is provided (https://github.com/2forts/GENS). Thus, this software can be useful for all problems which can be modeled by scheduling on unrelated parallel machines beyond the case study of this paper.

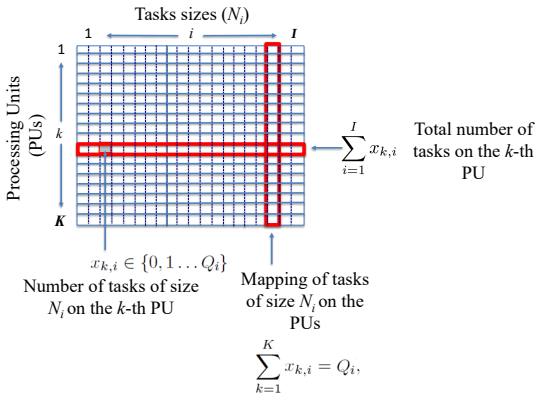


Fig. 1: Matrix X defines the assignment of tasks to PUs.

2 Scheduling problem on unrelated parallel machines

Let assume that a cluster has K PUs (Processing Units), that is, for example the total number of available CPU-cores plus GPUs. Let $\{R_m\}$ be the set of tasks that defines the model, with $m = 1, \dots, M$ and $M = \sum_{i=1}^{I} Q_i$ represents the total number of tasks to compute, I denotes the number of different system sizes N_i, with $1 \leq i \leq I$, and Q_i represents the number of tasks with system size N_i.Then, the goal is to find a scheduling that minimizes the makespan, C_{max}

$$
\begin{aligned}
&\text{Find:} && X && \text{to} \\
&\text{minimize:} && C_{max} \\
&\text{subject to:} && t_k = \sum_{i=1}^{I} x_{k,i} t_{k,i} \leq C_{max}, 1 \leq k \leq K \qquad (1) \\
& && \sum_{k=1}^{K} x_{k,i} = Q_i, 1 \leq i \leq I \\
& && x_{k,i} \in \{0, 1, \dots, Q_i\}, 1 \leq k \leq K; 1 \leq i \leq I
\end{aligned}
$$

where $x_{k,i}$ represents the number of tasks of size N_i assigned to the k-th PU; $x_{k,i}$ is an element of the matrix, X, that defines the assignment of tasks to PUs (machines); $t_{k,i}$ represents the runtime to compute a task of system size N_i on the k-th PU. The constraints for X mean that every task is computed on a single PU and every set of Q_i tasks with the same size is distributed among all the PUs. The k-th row of X defines the set of tasks assigned to k-th PU and the i-th column establishes the distribution of the tasks of size N_i among the K PUs (See Fig 1).

The scheduling problem defined by Eq. 1 includes the runtime of every task at every PU, $t_{k,i}$. The estimation of $t_{k,i}$ can be accurately and fast computed

a priori, because the $K \times I$ matrix $T = (t_{k,i})$ includes a high percentage of identical rows related to the same kinds of PUs. The runtime of the tasks can be characterized by a matrix $\mathcal{T}$ of reduced dimensions $S \times I$ where S represents the number of different kinds of PUs.

The scheduling on unrelated parallel machines is a challenge because of the heterogeneity of both, the required tasks and cluster architecture. Then, it is necessary to define an appropriate task scheduling to obtain the optimal parallel performance.

3 Scheduling approaches

According to the formalism introduced, the static methodology to optimize the task distribution among the heterogeneous PUs consists of three stages: (1) Profiling stage, which estimates the values of every element of the matrix T, according to the different sizes of the systems involved in the analysis and the number of Processing Units, PUs; (2) Optimal scheduling estimation to identify the set of tasks which every PU should compute, bearing in mind the a priori knowledge provided by the profiling stage, so a parallel runtime can also be estimated; and (3) Parallel execution of all simulations on the heterogeneous PUs of the cluster according to the scheduling defined in the second stage.

The optimal scheduling estimation (stage 2) could be simpler if there was a single type of tasks and PUs, since we could easily reach a good solution using a homogeneous distribution. However, in general, this estimation is more complex, even with a single type of PU, because the parallel software may include tasks with different computational loads. Of course, the computational complexity of the optimal scheduling estimation increases for high values of I, M and K. The scheduling on a heterogeneous cluster is NP-complete [14]. Our goal is to apply a heuristic which provides near optimal solutions for the scheduling problem [14].

3.1 Genetic Scheduler (GenS)

There are previous works where Genetic Algorithms (GAs) are used to solve scheduling problems according to a static scheme [15]. In this work, a GA is customized for solving the unrelated parallel machine scheduling on heterogeneous clusters.

A GA works with a set of individuals which represents possible solutions of the scheduling policy problem (*population*). It is an iterative procedure which starts with a random set of individuals, $P(0)$, and at every iteration, *iter*, a selection mechanism and genetic operators are applied to the population, $P(iter)$. Thus, the population is constantly evolving. The selection mechanism allows the individuals of the next generation to be closer to the optimal solution (see Algorithm 1).

To apply the Algorithm 1 to the problem of finding a near optimal scheduling, it is necessary to specify the following concepts: individual, fitness function,

Algorithm 1 Genetic Algorithm

1: $iter \leftarrow 0$
2: Initialize random population $P(0)$
3: Evaluate the fitness for the population $P(0)$
4: **while** termination condition is not true **do**
5: $iter \leftarrow iter + 1$
6: Select $P(iter)$ from $P(iter - 1)$
7: Apply genetic operators (crossover and mutation) to $P(iter)$
8: Evaluate the fitness for $P(iter)$
9: **end**

and genetic operators (crossover and mutation). We propose to adapt a GA to the aforementioned scheduling problem, resulting the GenS algorithm.

Bearing in mind the formalism introduced above, every individual in $P(iter)$ is represented by a $K \times I$ matrix X which defines the assignment of tasks to the PUs according to the definition in Eq. 1 (and Fig 1). After the evaluation of the fitness function (UB) for the whole population, individuals are ordered according to their fitness. Thus, the individuals with smaller UB will be selected while the GA advances.

At every iteration of Algorithm 1, two operations are applied to the population to promote the evolution. Firstly, a random set of pairs of individuals (parents) is defined and then, new individuals (children) are produced by the crossover operator. Then, the well-known single point crossover operator is applied. Fig 2 describes how the crossover is applied. A random column is selected to split the matrices of both parents and new individuals are generated swapping the four sets of columns. In this scheme, the children can be considered as valid solutions since the constraints for the columns of their matrices are verified. After the crossover, the mutation operator acts on every descendant and it can alter the distribution of every column (with a probability of 1%). It is a random exchange of tasks of the same size between a pair of PUs, i. e. elements in the same column of the corresponding matrix interchange their tasks partially. Every iteration starts with the same population size (PS). The selection phase only consists of choosing/ keeping the best PS individuals since the population has been previously ordered according to the fitness, UB. The procedure stops when the UB over 10 iterations does not change for the 30% of best individuals. Summarizing, if $\{t_{k,i}\}$ with $1 \leq k \leq K$ and $1 \leq i \leq I$ is known, GenS is able to identify a near optimal distribution of tasks among the set of PUs.

3.2 Additional static approaches

An example of static approach is the Polynomial Time Approximations Scheme (PTAS) algorithm [14]. The PTAS algorithm can give a good estimation of the optimal scheduling, with the additional advantage that it is possible to estimate theoretically the ratio to the optimal solution. However, PTAS has a high computational overhead due to the large amount of information that it

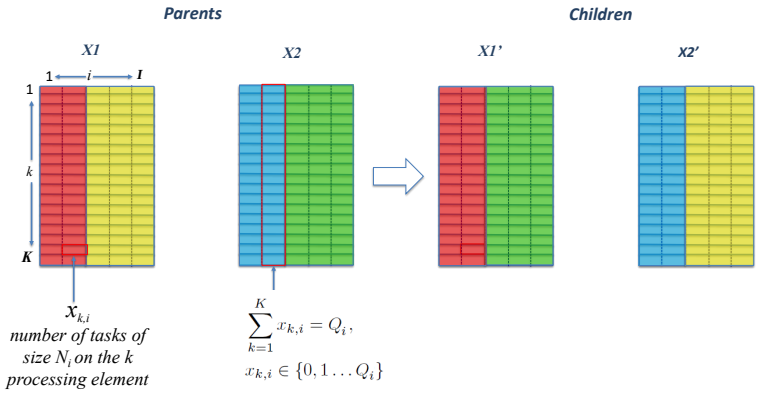


Fig. 2: Crossover procedure to produce new individuals in the population.

is necessary to store. This is the reason PTAS has not been included in the comparative study of GenS.

An alternative approach to GenS consists of a cyclic distribution of the tasks over the set of PUs.First of all, the tasks are ordered according to their computational load. Afterthat, they are distributed in a cyclic order among the PUs. In this way, every PU computes similar percentages of tasks of different costs. Hereinafter, this scheme will be referred to as Cyclic. It is probable that it achieves a near optimal schedule in homogeneous clusters.

A greedy heuristic following the scheme considered in [16] can also be defined to solve the scheduling problem (Greedy). For a given system size, (N_i), the a priori estimation of the runtime allows us to identify the slowest PU, and the acceleration factor of the remaining PUs with respect to it. These factors define the percentage of tasks of a specific size that will be executed in every PU. So, every set of Q_i tasks with the same work-load are distributed among the PUs. The PUs with less computational power compute fewer simulations and PUs with more power will compute the percentage of tasks defined by the corresponding acceleration factor. This procedure is repeated for every subset of tasks, obtaining a near optimal distribution in every case, with the aim of obtaining a global optimal solution.

3.3 Dynamic approaches

Several kinds of scheduling policies without a priori estimations of the tasks runtime can be defined. However, our interest is focussed on two dynamic approaches that partially use this information, since the starting point of both is an ordered tasks queue according to their computational load, N_i.

The Simple Tasks Queue (STQ) solves the problem dynamically, managing a single queue. Each PU will compute a task from this queue and when it finishes, it takes the new task from the head of the queue.

Another dynamic approach is to use a Double-Ended Queue in combination with a classification of the devices in two categories, slow and fast devices. In this scheme slower devices takes the lighter tasks of the queue, and faster devices the heavier ones. It will be referred to as Double-Ended Tasks Queue (DETQ) and it considers a priori information about the loads of tasks and the power of machines.

4 Active Microrheology Model (AMM) as a case study

As mentioned above, AMM is considered here as case study because from the computational of view it can be seen as a set of independent tasks of several different loads which can be executed on heterogeneous clusters. In active microrheology, the mechanical and flow behavior of a complex fluid is studied at the microscopic level [12,17]. For this, an intruder particle, typically of colloidal size, is introduced and pulled through the system, and its dynamics is monitored. In particular, the microviscosity can be computed from the stationary tracer velocity at long times.

In our case study, the host fluid is modeled as Brownian quasi-hard spheres, mimicking hard colloids. Brownian motion is described by the Langevin equation [18], which for particle j reads:

$$m_j \frac{d^2\mathbf{r}_j}{dt^2} = \sum_i \mathbf{F}_{ij} - \gamma_j \frac{d\mathbf{r}_j}{dt} + \boldsymbol{\eta}_j(t) \ \ + \mathbf{F}_{\text{ext}}\delta_{j1} \tag{2}$$

where the terms in the r.h.s. are the interaction forces ($\sum_i \mathbf{F}_{ij}$), friction with the solvent ($-\gamma_j \frac{d\mathbf{r}_j}{dt}$), random force ($\boldsymbol{\eta}_j(t)$), and external force ($\mathbf{F}_{\text{ext}}\delta_{j1}$), respectively; γ_j is the friction coefficient with the solvent, which is related to the random force via the fluctuation dissipation theorem [18], and depends linearly on the particle radius. The external force, $\mathbf{F}_{\text{ext}}$, which acts only onto the tracer, labeled by $j = 1$, is constant in our model (this fact is expressed by the well-known Kronecker delta, denoted by δ_{j1}). The interaction forces are derived from the interparticle potential $V(r_{ij}) = k_{\text{B}}T(r_{ij}/d_{ij})^{-36}$, where r_{ij} is the center to center distance, and $d_{ij} = (a_i + a_j)/2$, where a_i is the radius of particle i.

The simulations are run in a cubic box, with N particles and periodic boundary conditions. The bath particles and tracer have radii a_b and a_t, respectively, and all particles have the same mass, m. Details of the features of the simulations can be found in [20]. Fig 3 presents a snapshot of a system with N=15625 particles.

In the simulations, the tracer particle is pulled at a constant force, and its trajectory is recorded. The effective friction coefficient of the tracer with the bath, γ_{eff}, is obtained from the average tracer velocity using the stationary state relation: $\mathbf{F}_{\text{ext}} = \gamma_{\text{eff}}\langle\mathbf{v}\rangle$. A large number of trajectories is therefore needed to obtain reliable values of γ_{eff}. However, the tracer distorts the bath as it displaces, and since it is much larger than the bath particles, FSE can be

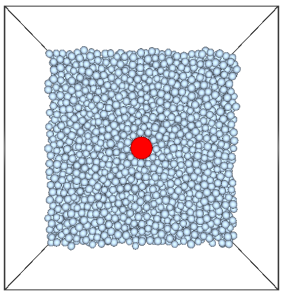


Fig. 3: Snapshot of the system with $N = 15625$ particles. The tracer, three times larger than the bath particles, $a_t = 3a_b$, is marked in red, and the particles in front of it have been removed.

present. In fact, due to the periodic boundary conditions, an array of particles is simulated with the lattice sparing equal to the box size. Starting from the Navier-Stokes equation, Hasimoto [21] showed that the effective friction coefficient measured by an array of particles in an incompressible Newtonian fluid depends on the lattice spacing, L, as:

$$\frac{1}{\gamma_{\text{eff}}} = \frac{c}{L} + \frac{1}{\gamma_\infty} \tag{3}$$

where c is a constant that depends on the structure of the array, and γ_∞ is the effective friction coefficient measured by an isolated particle [21]. Following this theoretical result, γ_∞ can be obtained running simulations with different system sizes, L, in order to obtain $\gamma_{\text{eff}}(L)$, and extrapolate linearly to $1/L \to 0$. Note that changing the system size implies changing the number of particles because the volume fraction is constant. Fig 4 shows the results of γ_{eff} for seven system sizes, with the number of particles ranging from $N = 216$ to 15625. The inverse friction coefficient is indeed linear for small systems, but deviates for $1/L \to 0$, due to the approximations in the theoretical model.

The full analysis of the finite size effects in the system necessitates a large number of simulations or tasks of *i*) systems with different number of particles, N_i with $1 \leq i \leq I$, and *ii*) a large number of trajectories (Q_i) for every system size (N_i), requiring *iii*) solving N_i equations of motion repeatedly for each trajectory. Therefore the computational requirements of AMM models are very high which are provided by modern multi-GPU clusters. The model exhibits several parallelism levels, which allows the appropriate exploitation of such heterogeneous clusters. Previous works focused on accelerating the computation of a single tracer trajectory (bottom parallelism level) on the GPU [22,23]. However, to advance in this kind of models, it is necessary to run efficiently many simulations in parallel on heterogeneous clusters (the highest parallelism level).

This way, the model defines a set of $M = \sum_{i=1}^{I} Q_i$ tasks which compute every tracer trajectory. So, tracer trajectories can be computed in parallel on

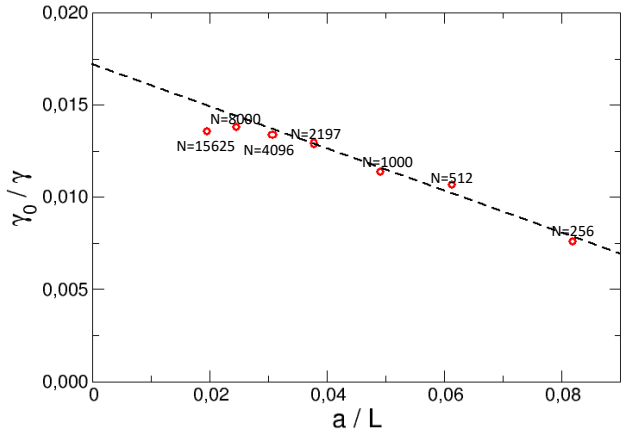


Fig. 4: Inverse friction coefficient vs. inverse length of the simulation box for a system with a volume fraction of $\phi = 0.50$, and a tracer of size $a_t = 3a_b$ pulled with a force $F = 2.5\, k_{\mathrm{B}}T/a_b$. The labels indicate the number of particles used in every simulation. The number of trajectories analysed for every point is 500.

the CPU-cores and GPUs of a cluster. Every CPU-core (GPU) can execute the sequential code (CUDA code) to compute one tracer trajectory, and the set of tasks can be computed with the collaboration of all processing units (PUs) of the cluster, CPU-cores and GPUs (PUs). Consequently, to get an optimal exploitation of heterogeneous clusters of models AMM is necessary to solve the scheduling problem on unrelated parallel machines defined in Sec. 2.

5 Results

In this section, the above mentioned strategies for load balancing (GenS, Cyclic, Greedy, STQ and DETQ) the case study in AMM are evaluated on a wide variety of heterogeneous clusters. For all the estimations and tests, the same problem is used: System sizes of N_i= 216, 512, 1000, 2197, 4096, 8000 and 15625, with 250 trajectories of 500 time units (corresponding to 10^6 time steps). Four kinds of PUs have been considered:

$Core_1$: 1 core of Bullx R424-E3 Intel Xeon E5 2650 with 8GB RAM
GPU_1 : NVIDIA Tesla M2070 GPUs (Fermi)
$Core_2$: 1 core of Bullx R421-E4 Intel Xeon E5 2620v2 with 64GB RAM
GPU_2 : NVIDIA Kepler GK210 (NVIDIA K80)

The characteristics of the GPU devices are given in Table 1. From these PUs, seven test clusters have been defined (five heterogeneous clusters and two homogeneous ones) to evaluate the scheduling methods (see Table 2).

Two implementations have been considered to simulate every tracer trajectory: a sequential CPU version coded in Fortran and a GPU version implemented in ANSI C and CUDA. Moreover, a Python's Multiprocessing module has been used to code the schedulers considered in the experimental evaluation.

Firstly, focusing the attention on the static policies the profiling stage has been carried out. Hence, an estimation of the runtime for all system sizes involved in the case study and the four kinds of PUs (CPU-cores and GPUs) was obtained (matrix $\mathcal{T}$) and shown in Table 3. AF stands for the acceleration factor of each kind of device versus the slowest device for each system size. These values are used in the Greedy strategy, as it was mentioned in the previous section. Let us remark that the execution time increases with N_i. Moreover, the use of GPU computing is not beneficial to accelerate microrheology problems when N_i is low. However, when $N_i \geq 1000$, GPUs increase the performance.

Secondly, focusing our attention on the second stage of our methodology, Table 4 shows the estimated parallel runtime (C_{max}) in hours. Analyzing the homogeneous platforms (F and G), it is observed that GenS achieves the best makespan, equalling or improving the other strategies by 3%. So, for these platforms, the advantages of GenS are maintained although they are not very relevant. But for all the heterogeneous platforms (A - E), the experimental results of GenS are significantly better than the other approaches, and this improvement is more evident as the heterogeneity and size of the platform increase. The Cyclic strategy always has the worst runtime by far, STQ obtains reasonable runtime close to the DETQ (the second best one), and the Greedy, although it improves the cyclic one, has large makespan.

Table 1: Characteristics of the GPU devices.

	M2070 (GPU_1)	GK210 (GPU_2)
Peak performance (double prec.) (TFlops)	0.51	2.91
Peak performance (simple prec.) (TFlops)	1.03	8.74
Device memory (GB)	5.2	24
Clock rate (GHz)	1.2	0.82
Memory bandwidth (GBytes/s)	150	480
Multiprocessors	14	13
CUDA cores	448	2496
Compute Capability	2.0	3.7

Table 2: PUs provided for every test cluster (A–F).

	$Core_1$	GPU_1	$Core_2$	GPU_2	K
A	14	2			16
B	28	4			32
C	28	8			36
D	56	8			64
E	56	8	10	2	76
F		8			8
G	64				64

Table 3: Total execution time (in seconds) for seven tasks sizes (N_i). $t_{GPU1,i}$ ($t_{GPU2,i}$) and $t_{CPU1,i}$ ($t_{CPU2,i}$) columns identify the runtime to compute a single trajectory on a GPU of kind 1 (2) and a CPU-core of kind 1 (2). AF_s, with $1 \leq s \leq 4$, are the acceleration factors of every kind of device versus the slowest one for each N_i.

	$s=1$	$s=2$	$s=3$	$s=4$				
N_i	$t_{GPU1,i}$	$t_{CPU1,i}$	$t_{GPU2,i}$	$t_{CPU2,i}$	AF_1	AF_2	AF_3	AF_4
216	1580	790	1406	101	1,0	2,0	1,1	15,6
512	1785	1860	1714	507	1,0	1,0	1,1	3,7
1000	2240	3715	2030	2319	1,7	1,0	1,8	1,6
2197	2930	8710	2112	5315	3,0	1,0	4,1	1,6
4096	4450	18065	3235	10465	4,1	1,0	5,6	1,7
8000	7650	43080	4587	24427	5,6	1,0	9,4	1,8
15625	12050	113940	10043	63788	9,5	1,0	11,3	1,8

Table 4: Makespan, in hours, for each strategy for cases exposed in Table 2. The scheduling scheme that obtains the best performance is marked in bold.

	Heterogeneous					Homogeneous	
	A	B	C	D	E	F	G
STQ	489,6	244,8	184,8	129,6	108,0	**283,2**	211,2
DETQ	412,8	223,2	141,6	122,4	103,2	**283,2**	211,2
Greedy	504,0	261,6	177,6	136,8	112,8	290,4	211,2
Cyclic	844,8	422,4	369,6	211,2	211,2	290,4	211,2
GenS	**410,4**	**206,4**	**139,2**	**102,5**	**79,2**	**283,2**	**206,4**

It is relevant to underline that due to the non-deterministic behavior of GenS, it has been executed 10 times in order to check its robustness and the dispersion of the results has been less than $0,15\%$. Therefore, we can remark the high robustness of the GenS solution.

To demonstrate that a simple random search is not competitive with respect to GenS, it has been executed during a time interval significantly greater than the GenS runtime to solve the same problem. Results have shown that the GenS overcomes the random search in terms of makespan. For the sake of clarity, this study has not been included.

Let us now focus our attention on the heterogeneous cluster with more resources, E. Fig 5 (a–e) shows the runtime for every device from each strategy and (f) shows the percentage of task sizes in every platform scheduled by the GenS (the colors show the task size, as given in the legend). In the STQ strategy (a), large tasks that consume a lot of time are computed on the CPU-cores meanwhile the GPUs are inactive, causing important imbalances with large makespans. In the DETQ strategy (b), the number of heavy tasks that come to the CPU-cores is not so important (the $Core_2$ does not take any, for example) and therefore the makespan is reduced, but there are still large imbalances. The Cyclic strategy (d), as the distribution is made without taking into account the heterogeneity of the platform, is the worst one. The Greedy strategy is also far from the optima. GenS tries to fit the heterogeneity of the tasks and

Table 5: Estimated (stage 2) and experimental makespan (stage 3), in hours, obtained by GenS scheduling, for heterogeneous clusters D and E from Table 2.

	Estimated		Experimental	
	D	E	D	E
C_{Max}	102,5	79,2	105,4	82,5
C_{Min}	101,8	76,4	104,4	79,4

the hardware of the platform, so its evolution makes the most powerful PUs compute the largest tasks (see that the $Core_2$ has more large tasks than $Core_1$) and being the less powerful devices those that are in charge of the light ones Fig 5(f). If we analyze the unbalance among the different platforms in Fig 5(e), we can conclude that the GenS solution is not far from the optimum since all devices finish almost at the same time (a makespan of 79,2 hours).

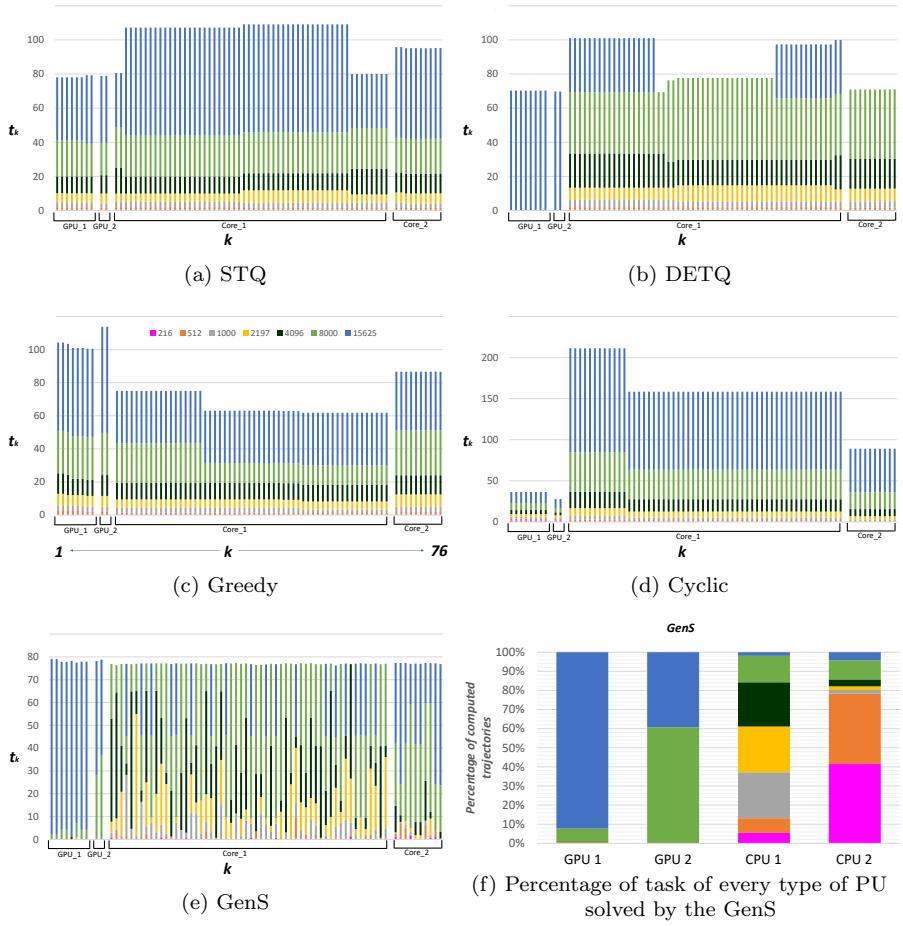
So, the GenS scheduling obtains the minimum estimation of parallel runtime. Then, the next step is to analyze the parallel executions on the test clusters (stage 3) to verify that GenS estimations are realistic. Real executions on the clusters D and E (the clusters which have the largest number of devices) have been tested. Table 5 shows the estimated makespan by GenS, in hours, in comparison with executions using GenS scheduling on both clusters. Analysing the execution time, it can be concluded that the estimation of the GenS is close to the makespan of the real experimentation. Experimental runtime is a little larger than the predicted one because GenS estimation does not take into account the runtime to prepare and to send a task to the corresponding machine and also the contention among the PUs in the clusters.

Therefore, using GenS to schedule a model composed by heterogeneous tasks on a heterogeneous cluster has resulted in an important reduction of the impracticable runtime of the previous versions of such model. For the study case, if the set of simulations are computed on a $Core_1$ the estimated sequential runtime would be $13205,6$ hours. Then, an acceleration factor of $\times 129$ ($\times 167$) is achieved on cluster D (on cluster E) using the GenS scheduling.

6 Conclusion

In this work, the scheduling of heterogeneous tasks on unrelated parallel machines has been studied. An approach for distributing the workload in a near-optimal way based on a Genetic Algorithm (GenS) has been analyzed. GenS has been comparatively evaluated with respect to other schedulers using a real problem from the field of statistical mechanics (Active Microrheology Model) as a case study. The goal of such model is the computation of the effective friction coefficient of complex fluids where Finite Size Effects are dominant. The computational cost for these models is huge because they are based on statistical analysis of the dynamics of a tracer particle for several system sizes. Therefore, the use of appropriate scheduling approaches on

Fig. 5: Fig (a–e) show the runtime, in hours, for all the devices of every strategy on cluster E. Each column k, shown as several stacked bars, corresponds to a device. A stacked bar represents the time spent by device k to compute the tasks of size i assigned to the device, so the entire column represents its total runtime. Fig (f) shows the percentage of the sizes of the tasks scheduled by GenS on each platform.



(a) STQ

(b) DETQ

(c) Greedy

(d) Cyclic

(e) GenS

(f) Percentage of task of every type of PU solved by the GenS

heterogeneous clusters has been a key to strengthen the applicability of these models.

Experimental results have shown that GenS achieves a near-optimal load balance, even when the cluster supplies a large and heterogeneous set of processing units, outperforming other studied strategies. GenS improves the performance with respect to the second fastest scheduling (DETQ) up to $23,56\%$ on the cluster E (the highest heterogenous one). Thus, the advantages of GenS are more relevant as the cluster heterogeneity increases.

Only the evolution of GenS has allowed to define the assignment task/processing-unit according to the load-of-task/computational-power optimally for highly heterogeneous tasks and processing units. This way, all processing units finish their computation almost simultaneously. A suitable definition of the operators and individuals involved in the Genetic Algorithm has been relevant to achieve these scheduling results.

The main contribution of this work has been to design and to provide a scheduling software for efficiently distributing a set of independent tasks varying in cost on heterogeneous processing units (https://github.com/2forts/GENS). Thus, this software can be useful for all problems which can be modeled by scheduling on unrelated parallel machines beyond the case study.

Acknowledgements This work has been supported by the Spanish Science and Technology Commission (CICYT) under contracts TIN2015-66680, RTI2018-095993-B-I00 and FIS2015-69022-P; Junta de Andalucía under contract P12-TIC-301 in part financed by the European Regional Development Fund (ERDF). G. Ortega is a fellow of the Spanish 'Juan de la Cierva Incorporación' program.

References

1. Hennessy JL, Patterson DA. Computer Architecture: A Quantitative Approach. Morgan Kaufmann; 2011.
2. Lenstra JK, Shmoys DB, Tardos E. Approximation Algorithms for Scheduling Unrelated Parallel Machines. Math Program. 1990;46(3):259–271.
3. Shmoys DB, Tardos E. An Approximation Algorithm for the Generalized Assignment Problem. Math Program. 1993;62(3):461–474. doi:10.1007/BF01585178.
4. Augonnet C, Thibault S, Namyst R, Wacrenier P. StarPU: A Unified Platform for Task Scheduling on Heterogeneous Multicore Architectures. Concurr Comp-Pract E. 2011;23(2):187–198.
5. Luk C, Hong S, Kim H. Qilin: Exploiting Parallelism on Heterogeneous Multiprocessors with Adaptive Mapping. In: Proc. of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture. MICRO 42. New York, NY, USA: ACM; 2009. p. 45–55.
6. McCormick Pea. Scout: a data-parallel programming language for graphics processors. Parallel Comput. 2007;33(10):648 – 662.
7. Chend Q, Guo M. Task Scheduling for Multi-core and Parallel Architectures: Challenges, Solutions and Perspectives. Springer; 2017.
8. Privman V. Finite Size Scaling and Numerical Simulation of Statistical Systems. Springer; 1998.
9. Guyon E, Huling JP, Petit L, Mitescu CD. Physical hydrodynamics. Oxford University Press; 1994.
10. Landau LD, Lifshitz EM. Fluid Mechanics. Butterworth-Heinemann; 1987.
11. Cicuta P, Donald AM. Microrheology: a review of the method and applications. Soft Matter. 2007;3:1449–1455.
12. Puertas AM, Voigtmann T. Microrheology of colloidal systems. J Phys Condens Matter. 2014;26(24):243101.
13. Wang T, Liu Z, Chen Y, Xu Y, Dai X. Load Balancing Task Scheduling Based on Genetic Algorithm in Cloud Computing. In: Proc. of the 2014 IEEE 12th International Conference on Dependable, Autonomic and Secure Computing. DASC '14. IEEE Computer Society; 2014. p. 146–152.
14. Gehrke JC, Jansen K, Kraft SEJ, Schikowski J. A PTAS for Scheduling Unrelated Machines of Few Different Types. In: SOFSEM 2016: Theory and Practice of Computer Science. vol. 9587 of Lecture Notes in Computer Science. Berlin, Heidelberg: Springer; 2016. p. 45–55.

15. Sels V, Coelho J, Dias AM, Vanhoucke M. Hybrid tabu search and a truncated branch-and-bound for the unrelated parallel machine scheduling problem. Computers & Operations Research. 2015;53:107 – 117. doi:https://doi.org/10.1016/j.cor.2014.08.002.
16. Woodside CM, Monforton GG. Fast allocation of processes in distributed and parallel systems. IEEE Transactions on Parallel & Distributed Systems. 1993; 2: 164 – 174.
17. Waigh TA. Advances in the microrheology of complex fluids. Rep Prog Phys. 2016;79(7):074601.
18. Dhont JKG. An Introduction to Dynamics of Colloids. Studies in Interface Science. Elsevier Science; 1996.
19. Paul W, Yoon DY. Stochastic phase space dynamics with constraints for molecular systems. Phys Rev E. 1995;52:2076–2083.
20. Orts F, Ortega G, Garzón EM, Puertas AM. Finite size effects in active microrheology in colloids. Comput Phys Commun. 2019;236(1):8–14.
21. Hasimoto H. On the periodic fundamental solutions of the Stokes equations and their application to viscous flow past a cubic array of spheres. J Fluid Mech. 1959;5:317–328.
22. Ortega G, Puertas AM, de Las Nieves FJ, Garzón EM. GPU Computing to Speed-Up the Resolution of Microrheology Models. In: Algorithms and Architectures for Parallel Processing: Proc. of ICA3PP Conference. Cham: Springer International Publishing; 2016. p. 457–466.
23. Ortega G, Puertas AM, Garzón EM. Accelerating the problem of microrheology in colloidal systems on a GPU. J Supercomput. 2017;73(1):370–383.