Olympiads in Informatics, 2012, Vol. 6, 133–147

REPORTS

An Experience on the Organization of the First Spanish Parallel Programming Contest

Francisco ALMEIDA[1], Vicente BLANCO PÉREZ[1], Javier CUENCA[2], Ricardo FERNÁNDEZ-PASCUAL[2], Ginés GARCÍA-MATEOS[3], Domingo GIMÉNEZ[3], José GUILLÉN[4], Juan Alejandro PALOMINO BENITO[4], María-Eugenia REQUENA[4], José RANILLA[5]

[1]*Departamento de Estadística, I.O y Computación, Universidad de La Laguna 38201 Tenerife, Spain*

[2]*Departamento de Ingeniería y Tecnología de Computadores, Universidad de Murcia 30071 Murcia, Spain*

[3]*Departamento de Informática y Sistemas, Universidad de Murcia Campus de Espinardo, 30071 Murcia, Spain*

[4]*Centro de Supercomputación, Fundación Parque Científico, Ctra. Madrid km. 388, Complejo Espinardo, 30100 Murcia, Spain*

[5]*Departamento de Informática, Universidad de Oviedo Campus de Viesques, 33204 Gijón, Spain*

e-mail: {falmeida,vicente.blanco}@ull.es, jcuenca@um.es, r.fernandez@ditec.um.es, {ginesgm,domingo}@um.es, {jguillen,jpalomino,mrequena}@parquecientificomurcia.es, ranilla@uniovi.es

Abstract. The first Spanish Parallel Programming Contest was organized in September 2011 within the *Jornadas de Paralelismo*, in La Laguna, Spain. The aim of the contest is to disseminate parallelism among the participants and Computer Science students who can use the material generated in the contest for educational purposes. The contest is similar to other sequential and parallel programming contests in which teams participate by solving a set of problems in a given time. But the Spanish contest has characteristics which distinguish it from other contests: an automatic tool (Mooshak) is used to validate the solutions and the tool has been modified to send the solutions to a cluster of nodes and to obtain the classification based on the speed-ups achieved; candidates can participate both in situ and online; a classification with the records for each problem is maintained on the web page of the contest, with explanations and codes of the record solutions so that the page can be used for educational purposes. This paper summarizes the experience and perspectives of the contest.

Key words: programming contests, online judge, parallel programming.

1. Introduction

With the evolution of technology parallel computing is becoming increasingly popular. At present, the basic computing systems are parallel (laptops and desktops are dual, quadcore or even hexacore, possibly with hyperthreading and programmable graphics cards) and so are clusters and supercomputers, which are composed of multicore nodes. This situation has led to a continuous increase in parallel computing courses in computing curricula (Meder *et al.*, 2008), and to some initiatives of the IEEE Technical Committee on Parallel Processing (IEEE TCPP, 2011), such as a proposal of parallel computing curriculum with a number of topics to be adopted in Computer Science studies.

In this context, the first Spanish Parallel Programming Contest (SPPC) was organized in September 2011 in the *Jornadas de Paralelismo* (JP, 2011). Our goal is to spread parallelism and at the same time that the material generated in the contest can be used for educational purpose in parallel computing courses.

Computing competitions can be organized in different ways, in a variety of fields and with different goals (Dagienė, 2010; Hakulinen, 2011). In particular, programming contests are useful for increasing the students interest for programming and for enhancing their programming abilities. So, there are a number of Computer Olympiads all over the world, with the main events being the International Olympiad in Informatics (IOI, 2011) and the ACM Programming Contest (ACM ICPC). Furthermore, contests are successfully used for educational purposes in programming courses (Fernández Alemán, 2011; García-Mateos and Alemán, 2009; Lawrence, 2004).

There are also a number of tools for the organization of programming contests (Kolstad, 2009). Some of them maintain a set of problems to practice with (UVa Online Judge) or allow the creation of contests (Leal and Silva, 2003).

Associated to the increasing popularity of parallelism, a number of competitions devoted to the topic have emerged in the last years, for example the Student Cluster Competition (SCC) and the Marathon of Parallel Programming (MPP). The Spanish Parallel Programming Contest follows this line, but it has some distinguishing features, which we explain below:

- The teams can participate in two ways (in situ and online) and there are two different classifications. This allows the teams to participate without needing to physically attend the *Jornadas de Paralelismo*.
- The tool Mooshak (Leal and Silva, 2003) has been modified to send the solutions provided by the teams to a cluster of four nodes, each with eight cores, and to obtain the classification considering the speed-ups achieved.
- A classification of records for all the problems of the different editions is maintained on the web of the contest (SPPC), with the explanation and code of the best solutions. In this way the page can serve as an educational resource for parallel programming courses.

The rest of the paper is organized as follows. Section 2 details the general organization of the contest. The problems in the first edition are commented on in Section 3. Section 4 describes how the first edition of the contest ran. Finally, future perspectives are discussed in Section 5.

2. General Organization

The contest has some organization aspects similar to those of other sequential and parallel programming contests:

- As in other contests, teams of three students and a teacher who acts as a coach are encouraged to participate. In the ACM Programming Contest (ACM ICPC) and the Marathon of Parallel Programming (MPP) the teams are of three components, and in the Student Cluster Competition (SCC), with a longer running time, the teams have six components and are supported by a computing company.
- As in the ACM competition and the Marathon, the Spanish contest lasts a short time (four hours), and the teams must solve a number of programming problems in that time.
- In our case (and also in the Marathon), a sequential solution of each problem is provided, and the teams generate parallel solutions with the aim of reducing the execution time of the corresponding sequential solution. The programs are developed in C, and the parallel environments OpenMP (Chandra *et al.*, 2001) and MPI (Snir and Gropp, 1998) can be used to develop the corresponding shared-memory and message-passing versions. MPI and OpenMP can also be combined to further reduce the execution time using hybrid parallelism.
- The problems and the sequential solutions are selected to cover different algorithmic paradigms and a variety of computational costs. The task selection is explained in the next section.

The main difference of the Spanish contest with respect to others is the evaluation system, which is done automatically and in real time, and allows in situ and online participation.

The cluster Arabí of the Supercomputing Centre of the Scientific Park Foundation of Murcia (SCC) is used for the evaluation of the solutions. The cluster comprises 102 nodes, each with eight cores, and four nodes (a total of 32 cores) are used in the contest. The SCC facilitates the occasional use of this subcluster for training, and it is used in the contest for the preparation and evaluation of the solutions provided to the teams, for a warm-up session and for the celebration of the contest. A job queue system is used to ensure only one program runs in the subcluster at a particular moment, which is necessary for the calculation of speed-ups and the classification.

The tool Mooshak (Leal and Silva, 2003) is installed in a virtual machine from which the solutions generated by the teams are sent to the queue in the cluster. It has been necessary to make some modifications in Mooshak to adapt it to the characteristics of the contest: Mooshak is used in conjunction with the subcluster of Arabí, and a new form of obtaining the classification based on speed-ups has been added to Mooshak.

The teams send their solutions to Mooshak, and it connects to a host associated to Arabí, where the programs are compiled and sent to the queue, from where the jobs are sent to the part of the subcluster specified in the submitted job. The solution is validated in the host by comparing it with the solution given by the sequential program provided by the organization. Finally, the host sends back to the Mooshak acknowledgement of the

correctness of the solution. In case of error, extra information is provided (compilation or execution error, too many resources required, etc).

The classification is computed based on the speed-ups: the execution time of the sequential program divided by that of the parallel program ($S_p = t_s/t_p$) (Grama *et al.*, 2003). In our case t_s is the time obtained for the test input with the sequential solution provided by the organization, and t_p the execution time for the same input with the program sent by the contestants. In that way the speed-up would be one for solutions which do not improve the sequential program. For each problem, the mark assigned to a correct solution could be $\max\{S_p - 1, 0\}$, so that solutions which do not reduce the sequential execution time have no positive mark, and positive marks begin from zero. The teams can send a maximum of ten solutions to each problem without penalization. After that, each additional submission means one point penalization, and the mark for the problem is $\max\{\max\{S_p\} - 1 - \max\{s - 10, 0\}, 0\}$, where s stands for the number of submissions for that problem and $\max\{S_p\}$ represents the maximum speed-up achieved in the s submissions. For a problem for which some team has a mark higher than 15, the marks of each team are linearly scaled so that the maximum mark is 15. This is to avoid very high marks (which could be obtained with an efficient use of the system combined with an improvement of the sequential solution) and to avoid some problems having an excessive weighting in the final score (problems with different algorithmic complexity have different parallelisation complexity). The final score for each team is obtained by adding up the marks in the problems for which they have provided some valid solution.

For each problem a brief description of the problem together with an example input and an execution scheme and the sequential solution are provided. The input provided is similar in number and form of the entries and in the execution time to that of the input used for automatic validation and scoring. So, the contestants can use this entry to evaluate their solutions in their laptop or in a parallel system to which access is provided by the local organization. The execution scheme is a C program (file `scheme.c`) which can not be modified. The I/O are performed via this program, which has a limit for the execution time and generates the solution in a file which is compared for validation with the output of the sequential program. The scheme is compiled and linked by the system with the file with the sequential function (`sec.c`). The resulting executable is run through MPI, with only one MPI process and one OpenMP thread, so it is considered the sequential version to compare against. The file `sec.c` has a heading of the form:

```
/*
CPP_NUM_CORES = 1
CPP_PROCESSES_PER_NODE 1
CPP_PROBLEM=mm
*/
```

where `CPP_NUM_CORES` establishes the number of cores to use (maximum 32), `CPP_PROCESSES_PER_NODE` the number of MPI processes to run on each node, and `CPP_PROBLEM` the name of the problem. The example corresponds to the heading of the sequential program, and so the number of cores is 1 and the number of

processes per node is 1. The teams modify the sequential function to make it parallel and send the file with the new function and the modified heading to use more cores and MPI and/or OpenMP. The number of nodes reserved to run the program is $NUM\_NODES = \lfloor (CPP\_NUM\_CORES - 1)/8 \rfloor + 1$, and the number of MPI processes $NUM\_PRO = NUM\_NODES * CPP\_PROCESSES\_PER\_NODE$. Inside each MPI process, the number of OpenMP threads (NUM_THR) is set with the function $set\_omp\_num\_threads$. So, a maximum of 32 cores can be used by a parallel program, and it is possible to use message-passing parallelism ($CPP\_NUM\_CORES > 1$ or $CPP\_PROCESSES\_PER\_NODE > 1$ or both, and $NUM\_THR = 1$), shared-memory parallelism ($CPP\_NUM\_CORES \leqslant 8$ and $NUM\_THR > 1$), or hybrid MPI+OpenMP parallelism. Several combinations can be used to attempt to achieve the highest speed-up.

3. Problems in the First Spanish Parallel Programming Contest

When designing a programming contest it is necessary to carefully select the problems to work with. There are some papers dedicated to tasks selection in computing competitions (Burton and Hiron, 2008; Vasiga *et al.*, 2008; Hakulinen, 2011). For a parallel programming contest the problems selection has some particularities that differentiates it from other computing contests. In this section the problems used in the First Spanish Parallel Programming Contest are shown and the criteria for problem selection are discussed, comparing them with the recommendations in the literature.

Five problems were proposed to be solved in four hours (to generate parallel solutions and adapt them to the computational cluster). They can be found on the web page of the contest (SPPC). Next we enumerate and discuss the problems:

A Multiplication of matrices with rectangular holes: Two square real matrices are multiplied. The matrices have rectangles of zeros. The rectangles can overlap.
The sequential solution provided uses the zeros structure of the matrices to accelerate the computation. The contestants can parallelize that sequential version or develop the parallel program from a different sequential version. For example, they could use a dense or a sparse matrix multiplication version, but the matrices are not dense or sparse, and with those approaches the parallel version may be far from satisfactory speed-ups. Furthermore, the sequential version provided does not optimize memory access, and in a multiplication AB, matrix B is accessed by columns, which can be improved just by transposing matrix B and accessing it by rows.

B Live game with variable neighborhood: This is a live game where the value in each position depends on the values in the neighboring positions in the previous generation, but in different generations the neighborhood varies, with the neighbors being the positions at a given Manhattan distance.
The sequential program follows an iterative scheme, and parallelisation can be achieved only inside each iteration. The computational cost and the memory ac-

cess in each iteration have order $O\left(n^2\right)$, which makes it difficult to obtain highly efficient parallel versions.

C Obtain values in given positions after sorting: An array of integers is given, together with a set of positions. The problem is to obtain the values in these positions when the values in the array are sorted.
The sequential solution sorts initially the positions, and then obtains the values in those positions by applying a partition scheme recursively. The quicksort pivoting strategy is applied to obtain the element in the middle position, and then the same method is applied to the left and right parts of the integer values and with the left and right parts of the positions. Obviously, it is possible to obtain a parallel solution just by sorting the array of integers, but the execution time would be higher than that of the solution provided, and consequently the speed-up achieved (if any) would not be very high.

D Multiply four dense square matrices.
This is the easiest problem in the contest. Three typical and naive matrix multiplications are performed. It is possible to optimize the memory access as indicated for problem A, and the parallelization of the matrix multiplication gives high speed-up (the computational cost is $O\left(n^3\right)$ and the access cost is $O\left(n^2\right)$). Furthermore, two of the multiplications can be done in parallel, which would allow a better use of the cluster, and consequently higher speed-up.

E Knapsack problem with affinities: We have a number of knapsacks with a certain capacity each, and a set of objects with a certain weight and with affinities between the objects. The objective is to obtain the assignation of objects to the knapsacks with the highest total affinity. The weight restrictions must be fulfilled, and the total affinity is the sum of the affinities between objects assigned to the same knapsack.
The sequential solution follows a backtracking scheme. It is possible to obtain better sequential solutions, with a better backtracking or with branch and bound algorithms, but the best solution depends on input, and the entries to be solved are not very large because it would produce a very long execution time. So, possibly the best approach is to parallelize the backtracking algorithm by generating a set of subproblems with all the possible assignations of some objects, and to assign a number of subproblems to different processes or threads.

The five problems follow well known algorithmic schemes and can be parallelized with parallel schemes which are explained in parallelism books (Almeida *et al.*, 2008; Grama *et al.*, 2003; Quinn, 2004). The targeted contestants (final years undergraduate and master and doctoral students) should know the sequential schemes and the basic parallel algorithms. So, the parallelization is not a big problem ... if they did not have a time limit of four hours. Furthermore, access to internet was allowed in the contest for a number of reasons: the possibility of participation online, the use of the tool Mooshak through internet, the need for access to a remote parallel system and because at present most of the bibliography is consulted on internet. So, the problems should not be typical problems or they can be well known problems but that should be modified to achieve the maximum performance in the system where the contest runs. The solutions must be

Table 1

Estimated maximum speed-ups achievable, the maximum with sequential, message-passing and shared-memory optimization, and the records in the contest and at February 8, 2012

	A	B	C	D	E
Sequential	3	1	1.2	4	2
message-passing	3	1.5	1.5	3.5	3.5
shared-memory	6	6	4	7	6
maximum estimated speed-up	54	9	7.2	98	42
Record in the contest	17.09 seq. shared	2.68 shared		25.88 seq. message shared	
Record at February 8, 2012	19.5 message shared	6.23 seq. shared	2.55 shared	45.13 seq. message shared	5.9 message shared

correct (they are checked automatically), but the goal is to achieve a high speed-up, and for that it is necessary not only to solve the problems in parallel, but also to optimize the sequential program and to adapt the parallel program to the computational system. Even though all the sequential programs follow well known algorithmic schemes, the solutions for A, B and D use a regular scheme (a number of loops) while the solutions of C and E have a more complex structure, and consequently it should be more difficult to obtain a parallel program for them.

Roughly speaking, we can estimate the ease of parallelism of each problem (the maximum expected speed-up) by multiplying the speed-up expected by sequential optimization, by the use of multiple nodes with message-passing and by the use of all the cores in a node. The maximum estimated speed-up is shown in Table 1. The values in the table represent estimations based on the empirical knowledge of the members of the organizing committee (they are not experimental speed-ups obtained by running efficient programs). The record speed-up for each problem in the contest and at February 8, 2012 are also shown, together with the combination of optimizations used in each record. Due to the difference between the estimated speed-ups and those of the records, there is space for further improvement. The different sources for improvement are commented:

- The sequential speed-up corresponds to improvements in the sequential program. As mentioned, the matrix multiplications (problems A and D) can be improved by changing the data access, transposing one of the matrices or designing an algorithm by blocks. In problem A the matrices are not dense, and so the estimated improvement is lower. In problem C the level of recursion can be changed, and so slightly reduce the execution time. For problem E the source of improvement is non predictable, because it depends on the input, but some changes can be done in the backtracking, or alternative methods can be used, and so a value of 2 is assigned to the sequential speed-up.

- The maximum message-passing speed-up is 4, because the cluster comprises 4 cores which work together with message-passing. Of course, the complete system (32 cores) can be used with MPI processes and message-passing, but the speed-up obtained with the use of cores in the same node is included in the shared-memory speed-up.
  Problems A, D and E have the highest computational cost, and so the highest speed-up is assigned to them. The value assigned is less than 4 due to the cost of communications. Problem A is a matrix multiplication, which is easily parallelizable, but the structure of the matrices, with rectangles of zeros, reduces the computational cost and hence the achievable speed-up. Problems B and C have lower costs, and it will be more difficult to obtain high speed-up with message-passing programs.
- The number of cores per node is 8, and this is the maximum shared-memory speed-up.
  The speed-up in shared memory is relatively easier to obtain, particularly for the dense matrix multiplication (problem D). The lowest achievable speed-up has been assigned to problem C due to its low computational cost.

Next we comment on some of the tasks generation recommendations in Burton and Hiron (2008) and indicate how they apply to the problems in the contest:

- The problem statements are short and easy to understand, so that the participants can concentrate on the solution (the parallelization) of the problem. Some of the statements are very short, as for example that of problem D: "Multiplication of four dense square matrices."
- The problems are modifications of classical problems, and the sequential solutions provided follow typical sequential algorithmic schemes, but to obtain a highly efficient parallel solution it may be necessary to modify the sequential program and to design the parallel version bearing in mind the target computational system.
- For each problem there are different possibilities of parallelization, of varying difficulty and efficiency. And for some problems satisfactory solutions can be easily programmed or taken from internet (this is specially true for problem D). So, it should be easy to gain some marks in some problems, and more difficult to obtain high speed-up.
- There are no official solutions apart from the sequential ones, which follow well known schemes.
- The problems on which the proposed problems are based are well known, and also the schemes solving them, but the best solution is not clear, and modifications of the basic schemes and adaptation to the computational system are needed.

4. The Competition

The contest was celebrated in the *Jornadas de Paralelismo* in September 2011 in La Laguna, Tenerife. The *Jornadas de Paralelismo* is an annual event which attracts most of the people working in parallel computing in Spain, so this is an appropriate framework

Fig. 1. Participants in the First Spanish Parallel Programming Contest.

for a parallel programming contest. The contest is conceived for undergraduate students in their last years or for master or doctoral students, and some of them participate in the *Jornadas de Paralelismo* by presenting their initial research results.

To facilitate participation, the teams can participate in situ or online, with two classifications: one for teams participating in the *Jornadas de Paralelismo* and the other for all the participants (in situ and online participants). Eight groups from six universities participated (Fig. 1, four in situ and four online). The number of teams is not large, but we consider it satisfactory for the first edition, moreover considering La Laguna is far from most of the Spanish Universities. Anyway, our main goal with the contest is to spread parallelism, and with this first edition we hope to have prepared the base for a higher participation in successive editions.

Before the contest there was a warm-up session in July, to check the modifications made to Mooshak and the correct response of the computational system, and to allow the participants to get familiar with the mechanics of the contest, the tool Mooshak and the cluster. That session lasted four days (to facilitate participation and experimentation), and it consisted of two very simple problems: mergesort and matrix multiplication. For the matrix multiplication, in addition to the sequential solution, OpenMP, MPI and hybrid MPI+OpenMP programs were provided, so that the teams could gain experience of the behavior of the system with different types of parallelism.

The contest runs on Mooshak, which validates the submissions and calculates the speed-ups and the classification in real time. Furthermore, it allows guests to be invited into the contest, and that way the evolution of the contest can be followed by non participants. The last moments of the contest were followed by about 25 guests, which gives an idea of the interest the contest aroused. Mooshak provides a classification in the form shown in Fig. 2, which shows the final classification. For each team and problem the mark obtained is shown, and between brackets the lowest execution time from all the submissions (0 if no correct solution is obtained), the maximum achieved speed-up and the number of submissions. The last two columns show the number of problems for which the team has sent a correct solution and the total points. Problems C and E were not

Team	Problems A	B	C	D	E	Total	Points
1 UAM	15.000000 (1103 15.792384 2)			15.000000 (626 25.880192 9)		2	30.000000
2 UALM	8.241741 (1914 8.677116 3)	0.000000 (0 0.000000 1)	0.000000 (0 0.000000 2)	8.963288 (1022 15.464775 4)	0.000000 (21954 0.000000 2)	3	17.205029
3 UMU	5.651547 (2665 5.950094 3)	1.681804 (6386 1.681804 2)		2.297346 (3390 3.963717 4)		3	9.630697
4 UCLM	0.000000 (0 0.000000 1)			7.421084 (1219 12.803938 4)		1	7.421084
5 ULL	0.000000 (0 0.000000 4)	1.553071 (6708 1.553071 2)		2.807980 (2879 4.844738 3)		2	4.361051
6 UAC	0.014791 (18238 0.015572 9)	1.634769 (6500 1.634769 6)	0.000000 (18477 0.000000 1)	1.517337 (4651 2.617932 5)	0.000000 (21222 0.000000 1)	5	3.166897

Fig. 2. Final classification of the First Spanish Parallel Programming Contest.

solved correctly by any team in a time lower than the sequential time, which is in concordance with the higher difficulty we considered for these two tasks. Furthermore, the highest speed-up has been obtained for problem D, followed by problem A and finally problem B. This coincides with the easiness estimated in Table 1. In problems A and D the maximum speed-ups were obtained by sequential optimization (optimization of the access to memory by transposition of the second matrix in the multiplications) combined with shared-memory or message-passing parallelization. No team combined both types of parallelism, which means the speed-ups obtained are far from the maximum estimated.

The evolution of the classification is shown in Fig. 3. The minutes at which modifications in the classification happen are represented. There are some points where the marks of some teams decrease (minutes 198 and 234). This happens when a speed-up higher

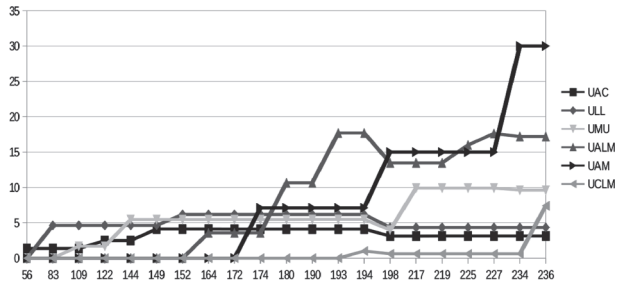

Fig. 3. Evolution of the classification throughout the First Spanish Parallel Programming Contest.

than 16 is obtained, because in this case the marks of all the teams for that problem are recalculated. All six universities were at some moment in the first position, and the last hour has the maximum number of submissions, with four changes in the first position.

5. Conclusion and Perspectives

The paper shows the experience of the first Spanish Parallel Programming Contest, held in September 2011. This event aims to disseminate parallelism among Scientific Computing students. The contest is similar to other programming competitions, and there are other parallel programming contests, but there are some differences: this one has two classifications, in situ and online; the tool Mooshak is used for an automated and real time evaluation of the submissions and for the classification, for which a new classification scheme has been implemented along with the system to connect Mooshak to the cluster where the contest is carried out; and a record table is maintained in the web page of the contest, with explanations and codes of the fastest solutions obtained in the contests or submitted outside them, so that the page can be used for preparing the participation in successive editions and for educational purposes, all of which are being used in various parallel programming courses in Spanish universities.

Additionally, the task generation process has been described. Recommendations of other authors for task generation have been considered, and the adaptation to a parallel programming contest are commented on.

We can consider the first edition of the contest has been successful, with a small number of participants, which is normal for the first edition, due to the specialization of parallelism (which is becoming more and more popular, but at present is not studied by all Computer Science students), and to the celebration of the contest at the University of La Laguna, which is far away from most of the Spanish universities.

The perspectives of the contest are:

- We are now working on the preparation of the next edition, in September 2012. We plan to organize an open session in the *Jornadas de Paralelismo* to follow the last minutes of the competition, and to discuss the solutions provided by the contestants and other possible solutions.
- The web page and the contest will also be in English, so that non Spanish students can participate online, and the use of the web page as an educational tool will be more visible.
- Some additional modifications can be included in Mooshak to better adapt it to the contest. For example, it could be interesting to include the generation of classifications with the format of Fig. 3, and not only in table form (Fig. 2).
- The inclusion of a CUDA competition is being considered. This supposes additional organizational work, because the programming paradigm changes, and problems which can be solved with a SIMD approach should be generated, and the expected speed-up estimated. Furthermore, some small modifications should be included in Mooshak to addapt it to the job queue system in a multicore+GPU environment.

Acknowledgements. This work has been funded in part by the Spanish MCYT under Grant TIN2008-06570-C04-02 and by the Fundación Séneca, Consejería de Educación de la Región de Murcia, 08763/PI/08. The authors gratefully acknowledge the computer resources and assistance provided by the Supercomputing Center of Fundación Parque Científico of Murcia.

References

Almeida, F., Giménez, D., Mantas, J.M., Vidal, A.M. (2008). *Introducción a la programación paralela*. Paraninfo Cengage Learning.

Burton, B.A., Hiron, M. (2008). Creating informatics olympiad tasks: exploring the black art. *Olympiads in Informatics*, 2, 16–36.

Chandra, R., Menon, R., Dagum, L., Kohr, D., Maydan, D., McDonald, J. (2001). *Parallel Programming in OpenMP*. Morgan Kauffman.

Dagienė, V. (2010). Sustaining informatics education by contests. In: Hromkovic, J., Krlovic, R., Vahrenhold, J. (Eds.) *Teaching Fundamentals Concepts of Informatics*. Springer, 1–12.

Fernández Alemán, J.L. (2011). Automated assessment in a programming tools course. *IEEE Trans. Education*, 54(4), 576–581.

García-Mateos, G., Fernández Alemán, J.L. (2009). A course on algorithms and data structures using on-line judging. In: *ITiCSE*, 45–49.

Grama, A., Gupta, A., Karypis, G., Kumar, V. (2003). *Introduction to Parallel Computing*. Addison-Wesley.

Hakulinen, L. (2011). Survey on informatics competitions: developing tasks. *Olympiads in Informatics*, 5, 12–25.

IEEE Technical Committee on Parallel Processing.
`http://www.cs.gsu.edu/~tcpp/curriculum/index.php`.

International Olympiad in Informatics.
`http://www.ioinformatics.org/index.shtml`.

Jornadas de Paralelismo (2011).
`http://jp2011.pcg.ull.es/`.

Kolstad, R. (2009). Infrastructure for contest task development. *Olympiads in Informatics*, 3, 38–59.

Lawrence, R. (2004). Teaching data structures using competitive games. *IEEE Transactions on Education*, 47(11), 753–759.

Leal, J.P., Silva, F.M.A. (2003). Mooshak: a web-based multi-site programming contest system. *Softw. Pract. Exper.*, 33(6), 567–581.

Marathon of Parallel Programming.
`http://regulus.pcs.usp.br/marathon/current/index.html`.

Meder, D.J., Pankratius, V., Tichy, W.F. (2008).
`http://www.multicore-systems.org/separs/downloads/GI-WG-SurveyParallelismCurricula.pdf`.

Quinn, M.J. (2004). *Parallel Programming in C with MPI and OpenMP*. McGraw Hill.

Snir, M., Gropp, W. (1998). *MPI. The Complete Reference*. The MIT Press.

Spanish Parallel Programming Contest
`http://cpp.fpcmur.es`.

Student Cluster Competition.
`http://sc10.supercomputing.org/?pg=studentcluster.html`.

Supercomputing Centre of Murcia.
`http://www.cesmu.es/inicio/`.

UVa Online Judge and Contest System Developed by the University of Valladolid (Spain).
`http://online-judge.uva.es/problemset`.

Vasiga, T., Cormack, G., Kemkes, G. (2008). What do olympiad tasks measure?. *Olympiads in Informatics*, 2, 181–191.

F. Almeida received his degree and the MSc in mathematics from the University of La Laguna in 1989 and 1992 respectively. He obtained his PhD in Computer Science in 1996. Currently he is professor in the Department of Statistics and Computer Science in the University of La Laguna. His research interests are primarily in the areas of parallel computing, parallel algorithms for optimization problems, parallel systems performance analysis and prediction, skeleton tools for parallel programming and web services for high performance computing and grid technology.

V. Blanco Pérez received his degree in physics and his MSc in physics from the University of Santiago de Compostela in 1992 and 1993 respectively. He obtained his PhD in physics in 2002. In October 2000 he became an assistant professor in the Department of Statistics and Computer Science in the University of La Laguna. Since 2009 he is associated professor in the same department. His research interests include performance analysis of parallel codes, parallel algorithms for dense and sparse algebra, Grid technology, and GPGPU tecnology.

J. Cuenca is an associate professor in the Computer Engineering Department at the University of Murcia, Spain. He received his BSc (engineering in computer science) from the University of Murcia in 1994, and his PhD in computer science from the University of Murcia in 2004. He was director of the Computer Engineering Department from 2008 until 2011. Since 1998 he has taught several subjects: "Fundamentals of Computer's', "Fundamentals of Operating Systems" in the computer science degree, and "Parallel Programming" in the computer science master. His research interests include issues related to parallel computing, linear algebra software and software autotuning techniques.

R. Fernández-Pascual received his MSc and PhD degrees in computer science from the University of Murcia, Spain, in 2004 and 2009, respectively. In 2004, he joined the Computer Engineering Department as a PhD student with a fellowship from the regional government. Since 2006, he has been an assistant professor at the University of Murcia. His research interests include general computer architecture, fault tolerance, chip multiprocessors and performance simulation.

G. Garcia-Mateos is a professor working at the Computer Science Faculty of the University of Murcia, Spain. He received his PhD degree in 2007, and is a member of the Computer Vision Research Group. Since 1998 he has been teaching algorithms and data structures, and has written two textbooks and several scientific papers on computer science education and programming contests. In 2002 he participated in the creation of the Programming Olympiad in Murcia for computer science students, and in 2008 the Informatics Olympiad in Murcia for high school students. Currently, he is the director of both olympiads. These contests are the Murcia

local stages of the ACM International Collegiate Programming Contest and the International Olympiad in Informatics, respectively.

D. Giménez is an associate professor in the Computer Science Department at the University of Murcia, Spain. He has been a faculty member of the university since 1988, where he teaches algorithms and parallel computing. He received his degree in mathematics from the University of Murcia in 1982, and his PhD in computer science from the Polytechnic University of Valencia in 1995. In 2002 he participated in the creation of the Programming Olympiad in Murcia for computer science students. His research interests include scientific applications of parallel computing, matrix computation, scheduling and software auto-tuning techniques.

J. Guillén is a telecommunications engineer from the Polytechnic University of Valencia and executive MBA in the Escuela de Organización Industrial. Since 2009 he has been project manager for the Supercomputing Center of Fundación Parque Científico de Murcia, managing R&D collaboration projects involving public research centres, universities and companies in different sectors such as industry, naval, biotechnology and engineering. Previously, for more than 4 years, was project manager for Ericsson, coordinating international projects for telecom operators from different countries such as Ireland, Hungary, Egypt and Nigeria, mainly related to network rollouts and systems integrations in all project phases. First job position since 2003 as analyst and team leader for the international IT and business consulting company Everis for the customer Telefónica Spain.

J.A. Palomino Benito (1983) holds a computer engineering degree from the University of Alicante. Currently he is doing his PhD thesis research since he presented his dissertation about parallel linear systems solvers in the Department of Science of the Computation and Artificial Intelligence at the University of Alicante. Since 2008, he has been working as application area manager in the Supercomputing Center of the Murcia Science Park in Spain. Hence his interests focus on HPC and algorithms in general, and he is a member of the Spanish Parallel Programming Contest organizing committee.

María E. Requena studied telecommunications engineering at the Polytechnic University of Valencia and did her PhD at the Polytechnic University of Cartagena (UPCT). She did her final project in the Repsol-YPF refinery in Cartagena. She began her career in Madrid in the SDB ALTRAN consulting involved in reengineering projects like power switching Moviline at Motorola and the design and implementation Imagenio platform at Telefonica R&D. She returned to Cartagena where she completed doctoral courses and worked in several R&D projects. While in the UPCT she published several books, international and national papers. Currently she is a head of the Supercomputing Center of the Science Park Foundation of Murcia a position she has held since March 2008.

J. Ranilla is a PhD in computer science and associate professor at the Faculty of Informatics, University of Oviedo (Spain). His research interests include information retrieval, knowledge management, parallel computing, and machine learning. Contact him at the Computer Science Department at the University of Oviedo, 33271, Campus de Viesques, Gijón, Spain.