Actas XVIII JENUI 2012, Ciudad Real, 10-13 de julio 2012
I.S.B.N. 10: 84-615-7157-6 | I.S.B.N. 13:978-84-615-7157-4
Páginas 215-222

Experiencia de organización del Primer Concurso Español de Programación Paralela

Francisco Almeida, Vicente Blanco Pérez
Departamento de Estadística, I.O y Computación, Universidad de La Laguna
{falmeida,vicente.blanco}@ull.es
Javier Cuenca, Ricardo Fernández-Pascual
Departamento de Ingeniería y Tecnología de Computadores, Universidad de Murcia
jcuenca@um.es, r.fernandez@ditec.um.es
Ginés García-Mateos, Domingo Giménez
Departamento de Informática y Sistemas, Universidad de Murcia
{ginesgm,domingo}@um.es
José Guillén, Juan Alejandro Palomino Benito, María-Eugenia Requena
Centro de Supercomputación, Fundación Parque Científico, Murcia
{jguillen,jpalomino,mrequena}@parquecientificomurcia.es
José Ranilla
Departamento de Informática, Universidad de Oviedo, Spain
ranilla@uniovi.es

Resumen

El primer Concurso Español de Programación Paralela se organizó en septiembre de 2011 en las Jornadas de Paralelismo en La Laguna. La finalidad del concurso es difundir la programación paralela entre estudiantes de informática, generando además material docente. El concurso es similar a otros concursos de programación secuencial o paralela, con equipos que resuelven problemas en un tiempo limitado, pero tiene algunas características diferenciadoras: se utiliza una herramienta automática (Mooshak) para validación, y se ha modificado para enviar las soluciones a un cluster y obtener la clasificación en función del *speed-up*; se puede participar de forma presencial u *online*; en la página del concurso se mantienen los records para cada problema, con explicaciones y códigos de las soluciones, de forma que el material se puede utilizar para docencia. El artículo resume la experiencia y perspectivas del concurso.

Summary

The first Spanish Parallel Programming Contest was organized in September 2011 within the *Jornadas de Paralelismo* in La Laguna. The aim of the contest is to disseminate parallelism among 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 it has been modified to send the solutions to a cluster and to obtain the classification based on the speed-ups; candidates can participate both in situ and online; a classification with the records for each problem is maintained on the 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.

Palabras clave

Concursos de programación, Juez *online*, Programación paralela.

1. Introducción

En la actualidad los sistemas computacionales básicos (portátiles y equipos de sobremesa duales o quad, con *multithreading* y tarjetas gráficas programables) son sistemas paralelos, al igual que los clusters y supercomputadores, formados por nodos multinúcleo. Esta situación ha motivado la aparición de cursos de computación paralela [16] y algunas iniciativas del IEEE Technical Committee on Parallel Processing [10] tal como una propuesta de curriculum con tópicos de paralelismo para su posible

introducción en estudios de informática.

En este contexto se organiza el Primer Concurso Español de Programación Paralela [20] en septiembre de 2011 dentro de las Jornadas de Paralelismo [12]. La finalidad del concurso es fomentar la utilización del paralelismo, al mismo tiempo que el material generado se puede usar en cursos de programación paralela.

Si observamos la historia reciente, encontramos distintos tipos de concursos de computación, que pueden comprender campos distintos y con distintas finalidades [7, 9]. Los concursos son útiles para aumentar el interés de los estudiantes por la programación y mejorar sus habilidades en este campo. Así, hay varios concursos de programación, siendo los principales la Olimpiada Internacional de Informática [11] y el Concurso de Programación de ACM [2]. Además, en algunos cursos se usan concursos con resultados satisfactorios [3, 14].

Hay diferentes herramientas para la organización de concursos de programación [13]. Algunas mantienen un conjunto de problemas con los que se puede practicar [22] y otras se utilizan para la organización de concursos [1].

Por otro lado, al mismo tiempo que ha aumentado la popularidad del paralelismo han surgido competiciones dedicadas a este tema, por ejemplo la Student Cluster Competition [21] y el Marathon of Parallel Programming [15]. El Concurso Español de Programación Paralela se organiza en una forma similar, pero tiene algunas particularidades:

- Se puede participar de forma presencial y *online*, con clasificaciones diferentes, lo que permite participar sin necesidad de asistir a las Jornadas.
- Se ha modificado la herramienta Mooshak [1] para mandar las soluciones de los participantes a un cluster de cuatro nodos, cada uno con ocho núcleos, y obtener la clasificación en función del *speed-up*.
- Se mantiene una clasificación con los records de los problemas en la página del concurso [20], junto a explicaciones y códigos de las mejores soluciones. De esta manera, esta página se puede usar en cursos de programación paralela.

El artículo se organiza de la siguiente forma. En la Sección 2 se detalla la organización general del concurso. Los problemas de la primera edición se comentan en la Sección 3, y el desarrollo del concurso en la 4. Finalmente, en la Sección 5 se discuten algunas perspectivas.

2. Organización general

El concurso se organiza de forma similar a otros concursos de programación, con equipos de tres estudiantes (está orientado a estudiantes de últimos cursos de carrera, de máster o doctorado) y un profesor que actúa como entrenador. Se trata de resolver varios problemas en un tiempo limitado. En nuestro caso se proporciona una solución secuencial para cada uno de los problemas, y el objetivo de los concursantes es diseñar soluciones paralelas con las que reducir el tiempo de ejecución obtenido con la solución secuencial.

Los programas se desarrollan en C, y se utilizan OpenMP [17] y MPI [19] para desarrollar versiones de memoria compartida y paso de mensajes, y, además, se puede combinar OpenMP y MPI para obtener paralelismo híbrido.

Se utiliza el cluster Arabí del Centro de Supercomputación de la Fundación Parque Científico de Murcia [6]. El cluster consta de 102 nodos, cada uno con ocho núcleos, de los que en el concurso se usan cuatro nodos. Se facilita el uso ocasional de este subcluster para docencia, y en el concurso se usa para preparar y evaluar los problemas, para una sesión de calentamiento y para la realización del concurso. Para planificar las ejecuciones se utiliza una cola de trabajos de cara a asegurar que no se ejecuta más de un programa simultáneamente, situación que es necesaria para el cálculo del *speed-up* y la obtención de la clasificación.

El juez *online* Mooshak [1] se ha instalado en una máquina virtual desde la que se mandan a la cola las soluciones aportadas por los concursantes. Ha sido necesario incluir algunas modificaciones en esta herramienta para adaptarla a las características del concurso: se ha adaptado para trabajar con un subcluster de Arabí, y se le ha añadido una nueva forma de obtener la clasificación basada en el *speed-up* (tiempo de ejecución del programa secuencial dividido por el del paralelo, $S_p = t_s/t_p$ [8]).

Los equipos envían sus soluciones al Mooshak, que se conecta a un *host*, donde se compilan los programas y se mandan a la cola, desde la que se

mandan los trabajos a la parte del subcluster especificada en el trabajo enviado. La solución se valida en el *host* comparándola con la obtenida con el programa secuencial proporcionado por la organización. Finalmente, el *host* devuelve a Mooshak un reconocimiento de la validez de la solución. En caso de error se proporciona información adicional (error de compilación o de ejecución, demasiados recursos solicitados, etc).

La clasificación se obtiene en base al *speed-up*. En nuestro caso, t_s es el tiempo obtenido para la entrada de test con la solución secuencial proporcionada por la organización, y t_p el tiempo de ejecución para la misma entrada con el programa enviado por los concursantes. El *speed-up* sería igual a 1 para soluciones que no mejoran el programa secuencial, por lo que la puntuación asignada a una solución correcta es $max\{S_p - 1, 0\}$, de modo que las soluciones que no mejoren la secuencial tienen una puntuación de cero. Cada equipo puede mandar para un problema un máximo de diez soluciones sin penalización, y a partir de la décima cada envío tiene una penalización de uno, y la puntuación es $max\{max\{S_p\} - 1 - max\{s - 10, 0\}, 0\}$, donde s representa el número de envíos para ese problema y $max\{S_p\}$ el máximo *speed-up* obtenido con los s envíos. Para los problemas para los que algún equipo obtiene una puntuación mayor de 15 (*speed-up* de 16, que es la mitad del número total de cores) se modifican linealmente las puntuaciones de todos los equipos, de manera que la puntuación máxima para el problema sea 15. De esta forma se evitan puntuaciones muy altas (que se pueden obtener con uso eficiente del sistema paralelo en combinación con una mejora de la solución secuencial) y el peso excesivo de algunos problemas en la puntuación final (problemas con complejidad algorítmica distinta pueden tener dificultad de paralelización distinta). La puntuación final de cada equipo se obtiene sumando las puntuaciones en cada uno de los problemas para los que han proporcionado soluciones válidas.

De cada problema se proporciona su descripción, un ejemplo de entrada, un esquema de ejecución y la solución secuencial. Esta entrada de ejemplo es similar en número y forma de los problemas a resolver, y en tiempo de ejecución a la entrada que se usa para la validación y puntuación. De esta forma los concursantes pueden usar esta entrada para evaluar la solución en sus portátiles o en el sistema computacional al que les da acceso la organización. El esquema de ejecución es un programa en C (fichero `esquema.c`) que no se puede modificar. La entrada y salida se realiza a través de este esquema, que además tiene limitado el tiempo de ejecución y genera la solución en un fichero que se compara con la salida del programa secuencial para validar automáticamente la solución generada. El esquema se compila y se enlaza con el fichero que contiene la solución secuencial (`sec.c`), y el ejecutable resultante se ejecuta con un único proceso MPI y un único hilo OpenMP, de manera que esta se considera la versión secuencial con la que se comparan las soluciones enviadas por los concursantes. El fichero `sec.c` tiene una cabecera de la forma:

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

Con `CPP_NUM_CORES` se establece el número de núcleos a usar (máximo 32), `CPP_PROCESSES_PER_NODE` indica el número de procesos MPI a lanzar en cada nodo, y `CPP_PROBLEM` el nombre del problema. El ejemplo corresponde a la cabecera de un programa secuencial, en el que el número de núcleos y el de procesos por nodo es uno. Los equipos modifican la función secuencial para obtener la correspondiente solución paralela, y envían el fichero con la nueva función y con la cabecera indicando el número de núcleos y procesos. El número de nodos reservados es $NUM\_NODES = \lfloor (CPP\_NUM\_CORES - 1)/8 \rfloor + 1$, y el de procesos MPI es $NUM\_PRO = NUM\_NODES * CPP\_PROCESSES\_PER\_NODE$. En cada proceso MPI el número de hilos OpenMP (NUM_THR) se establece con la función *set_omp_num_threads*. Así, un programa paralelo usa un máximo de 32 núcleos, y es posible usar paso de mensajes ($CPP\_NUM\_CORES > 1$ y/o $CPP\_PROCESSES\_PER\_NODE > 1$, y $NUM\_THR = 1$), paralelismo de memoria compartida ($CPP\_NUM\_CORES \leq 8$ y $NUM\_THR > 1$) o paralelismo híbrido. Los concursantes pueden experimentar con distintas combinaciones de los valores de estos parámetros para obtener el máximo *speed-up*.

3. Selección de los problemas a resolver

Cuando se organiza un concurso de programación es necesario seleccionar cuidadosamente los problemas con los que trabajar. Hay alguna bibliografía dedicada a la selección de tareas en competiciones informáticas [5, 23, 9], pero para un concurso de programación paralela la selección de los problemas tiene algunas peculiaridades. En esta sección se comentan los problemas usados en la primera edición del Concurso, y se discuten los criterios seguidos para la selección de los problemas, comparándolos con los recomendados en otros artículos.

Se propusieron cinco problemas a resolver (generar soluciones paralelas y adaptarlas al cluster) en un tiempo máximo de cuatro horas. Los enunciados se encuentran en la página del concurso [20]. A continuación discutimos brevemente cada uno de los problemas:

A **Multiplicación de matrices con huecos rectangulares**: Se trata de multiplicar dos matrices cuadradas de números reales, con rectángulos de ceros y los rectángulos pueden solaparse. La solución secuencial que se proporciona usa la estructura de ceros para reducir el coste computacional. Los concursantes pueden paralelizar esta versión o desarrollar su versión paralela a partir de otra solución secuencial. Por ejemplo, podrían usar una versión de la multiplicación de matrices densas o dispersas, pero de esta manera los resultados pueden ser peores al no ser las matrices con las que se trabaja de esos tipos. Además, la versión secuencial que se proporciona no optimiza el acceso a memoria, por lo que el coste de la multiplicación AB puede reducirse trasponiendo la matriz B para acceder por filas.

B **Juego de la vida con vecindad variable**: Es el juego de la vida en el que el valor en cada posición depende de valores en posiciones vecinas en la generación anterior, pero la vecindad varía según la generación en que nos encontremos, y está formada por las casillas a una determinada distancia de Manhattan. El esquema secuencial sigue un esquema iterativo, y sólo se puede paralelizar dentro de cada iteración. El coste computacional y el de acceso a memoria tienen en cada iteración orden $\theta\left(n^2\right)$, lo que hace difícil obtener soluciones eficientes.

C **Obtener los valores en unas posiciones dadas de una lista de números una vez ordenados**: Dada una secuencia de números enteros y un conjunto de posiciones, se trata de obtener los números que quedarían en esas posiciones si la secuencia estuviera ordenada. La solución secuencial ordena las posiciones y realiza un particionado de forma recursiva. Se utiliza la estrategia de pivotamiento del *quicksort* para obtener el valor en la posición intermedia, y después se aplica el mismo método a las zonas izquierda y derecha de los números y con las posiciones a la izquierda y a la derecha de la intermedia. Es posible obtener la solución ordenando la secuencia, pero se realizaría más trabajo del necesario y el *speed-up* obtenido sería bajo.

D **Multiplicación de cuatro matrices cuadradas densas**: Es el problema más simple del concurso. Se realizan tres multiplicaciones de matrices típicas de tres bucles. Se puede optimizar el acceso a memoria como se indicó para el problema A, y paralelizando la multiplicación de matrices se obtienen *speed-ups* satisfactorios, pues el coste computacional es $\theta\left(n^3\right)$ y el de acceso a memoria $\theta\left(n^2\right)$. Dos de las multiplicaciones se pueden realizar en paralelo, lo que puede permitir un mejor uso del cluster y mayor *speed-up*.

E **Problema de la mochila con afinidades**: Se dispone de varias mochilas con cierta capacidad cada una de ellas, y de un conjunto de objetos con unos ciertos pesos y con afinidades entre ellos. El objetivo es obtener la asignación de objetos a las mochilas que proporciona mayor afinidad y cumpliendo las restricciones de peso. La afinidad total es la suma de las afinidades entre objetos asignados a la misma mochila. La solución secuencial sigue un esquema de *backtracking*. Se pueden obtener mejores soluciones secuenciales mejorando el *backtracking* o con un algoritmo tipo *branch and bound*, pero cuál es la mejor solución depende de la entrada a resolver. Además, hay que tener en cuenta que los problemas no pueden ser grandes pues producirían tiempos de ejecución muy altos. Por tanto, posiblemente la mejor solución consista en paralelizar el algoritmo secuencial generando un conjunto de subproblemas con todas las

posibles asignaciones de un número reducido de objetos, y asignar varios subproblemas a cada uno de los procesos o hilos en un programa paralelo.

Los cinco problemas siguen esquemas algorítmicos clásicos, y se pueden paralelizar usando esquemas paralelos que aparecen en libros de paralelismo [4, 8, 18]. Los concursantes deben conocer los esquemas secuenciales y los algoritmos paralelos básicos, de forma que la paralelización no sería un gran problema si no tuvieran un límite de cuatro horas. Además, se permitió acceso a internet por varios motivos: la posibilidad de participar *online*, el uso de Mooshak a través de internet, la necesidad de acceder a un sistema paralelo remoto, y porque en la actualidad la mayoría de la bibliografía que se consulta está en la web. Así, los problemas no deben ser problemas típicos, o en el caso de ser problemas conocidos, que sea necesario modificar su solución para alcanzar las máximas prestaciones en el sistema donde se trabaja. Las soluciones deben ser correctas (se validan automáticamente) pero el objetivo último es alcanzar *speed-ups* altos, y para esto es necesario resolver el problema en paralelo, pero también se puede optimizar el programa secuencial. Además hay que adaptar la solución paralela al sistema computacional si se quiere alcanzar buenas cotas de *speed-up*. Aunque todos los programas secuenciales siguen esquemas algorítmicos bien conocidos, las soluciones para los problemas A, B y D usan un esquema muy regular (varios bucles), mientras que las de los problemas C y E tienen una estructura más compleja, y puede ser más difícil obtener programas paralelos para ellos.

Podemos estimar la dificultad de paralelización de cada problema multiplicando los *speed-ups* esperados por optimización secuencial, por uso de varios nodos con paso de mensajes, y por uso de varios núcleos en un nodo. Los *speed-ups* máximos estimados se muestran en la tabla 1. Los valores están basados en estimaciones empíricas del comité organizador. Se comentan a continuación las posibilidades de mejora de los diferentes problemas:

- El *speed-up* secuencial corresponde a posibles optimizaciones en el programa secuencial. Como se ha mencionado, las multiplicaciones de matrices (A y D) se pueden mejorar cambiando el acceso a los datos, trasponiendo una de las matrices o usando un algoritmo por bloques. En el problema A las matrices no son densas, y la mejora esperada es menor. En el C se puede modificar el nivel máximo de recursión, con lo que se puede reducir ligeramente el tiempo de ejecución. Para el problema E es más difícil hacer una estimación de la mejora, ya que depende de la entrada, pero se pueden realizar algunos cambios en el *backtracking* o se puede utilizar algún otro método, por lo que se asigna un valor de 2 al posible *speed-up* secuencial.
- Como el cluster consta de cuatro nodos que trabajan juntos con paso de mensajes, el máximo *speed-up* por paso de mensajes se establece en 4. Se puede utilizar el sistema completo con procesos MPI, pero el *speed-up* alcanzable con el uso de varios núcleos en un mismo nodo se considera dentro del *speed-up* por memoria compartida. Los problemas D y E tienen el mayor coste computacional, por lo que se les asigna el mayor *speed-up*. El valor es menor que 4 debido al coste de las comunicaciones. El problema A es una multiplicación de matrices, que es fácilmente paralelizable, pero la estructura de las matrices reduce el coste de computación y por tanto el *speed-up* alcanzable. Los problemas B y C tienen menor coste, y será más difícil obtener *speed-ups* altos.
- El número de núcleos por nodo es 8, por lo que este es el máximo *speed-up* en memoria compartida. Es relativamente fácil obtener *speed-ups* altos en memoria compartida, especialmente para multiplicaciones densas de matrices (problema D). El menor *speed-up* se ha asignado al problema C por su bajo coste computacional.

	A	B	C	D	E
secuencial	3	1	1.2	4	2
paso de mensajes	3	1.5	1.5	3.5	3.5
memoria compartida	6	6	4	7	6
speed-up máximo	54	9	7.2	98	63

Tabla 1: *Speed-up* para los cinco problemas: total máximo estimado, y con optimización secuencial, con paso de mensajes y en memoria compartida.

Comentamos algunas de las recomendaciones de generación de tareas sugeridas en [5], e indicamos

cómo se aplican a los problemas del concurso:

- Los enunciados de los problemas son cortos y fáciles de entender, con lo que los participantes pueden concentrarse en la paralelización de los problemas. Algunos enunciados son muy cortos, como por ejemplo el del problema D: "Multiplicación de cuatro matrices cuadradas densas".
- Los problemas son modificaciones de problemas clásicos, y se proporcionan soluciones con algoritmos secuenciales típicos, pero para obtener soluciones paralelas eficientes es necesario diseñar las versiones paralelas teniendo en cuenta el sistema en el que se trabaja.
- Para cada problema hay varias posibilidades de paralelización, con dificultad y eficiencia diferente, y para algunos problemas se pueden obtener fácilmente versiones paralelas, programándolas o adaptando alguna solución encontrada en internet (esto es especialmente fácil para el problema D). Así, será fácil obtener puntuación en algunos problemas.
- No hay soluciones "oficiales" al margen de las secuenciales que se proporcionan.
- Los problemas propuestos se basan en problemas típicos, y también los esquemas con los que se resuelven, pero no está claro cuál es la mejor solución, por lo que es necesario modificar los esquemas básicos y adaptarlos al sistema computacional.

4. Desarrollo del concurso

El concurso se celebró en las Jornadas de Paralelismo, en septiembre de 2011 en La Laguna. Las Jornadas son un evento anual en el que participan la mayoría de los investigadores españoles en paralelismo, por lo que es el marco adecuado para un concurso de este tipo orientado a estudiantes de últimos cursos de la Ingeniería Informática, de Máster y de Doctorado, algunos de los cuales participan en las Jornadas presentando sus primeros trabajos de investigación.

Para facilitar la participación se puede participar in situ u *online*, con dos clasificaciones: una para los equipos que participan en las Jornadas y otra para todos los participantes. Participaron ocho equipos de seis universidades (A Coruña, Almería, Castilla-La Mancha, Autónoma de Madrid, La Laguna y Murcia), cuatro in situ y cuatro *online*. La participación fue reducida, pero consideramos que es satisfactoria para la primera edición, teniendo en cuenta la distancia geográfica de La Laguna a la mayoría de las universidades españolas. Nuestra finalidad principal es la difusión de la programación paralela, y con esta primera edición esperamos haber puesto las bases para una mayor participación en ediciones sucesivas.

En julio se realizó una sesión de calentamiento para evaluar las modificaciones incluidas en Mooshak y la respuesta del sistema computacional, y para que los participantes se familiarizaran con la mecánica del concurso, el Mooshak y el cluster. La sesión estuvo abierta cuatro días para facilitar la realización de experimentos, y consistió en dos problemas simples: ordenación por mezcla y multiplicación de matrices. Para la multiplicación de matrices, además de la solución secuencial, se proporcionaron soluciones con OpenMP, MPI y paralelismo híbrido MPI+OpenMP, de manera que los equipos pudieran experimentar con diferentes tipos de paralelismo en el sistema.

Mooshak valida los envíos y calcula el *speed-up* y la clasificación en tiempo real. Además, permite entrar al concurso como invitado, con lo que se puede seguir su desarrollo. Aproximadamente 25 invitados siguieron los últimos momentos del concurso, lo que da una idea del interés despertado. La clasificación se muestra en la Figura 1, donde se ve la clasificación final (sólo se incluyen los equipos que resolvieron algún problema). Para cada equipo y problema se muestra la puntuación y entre paréntesis el menor tiempo de todos los envíos (0 si no se obtiene ninguna solución correcta), el máximo *speed-up* alcanzado y el número de envíos. Las dos últimas columnas muestran el número de problemas para los que el equipo ha enviado soluciones correctas y la puntuación total. Los problemas C y E no han sido resueltos por ningún equipo en un tiempo menor que el del programa secuencial, lo que concuerda con la mayor dificultad de paralelización estimada para estos problemas. El mayor *speed-up* se obtuvo para el problema D, seguido por el problema A y finalmente el B, lo que coincide con las estimaciones recogidas en la Tabla 1. Para los problemas A y D el máximo *speed-up* se obtuvo con optimización secuencial (trasponiendo una matriz) combinado con paralelización por memoria compartida o paso de men-

sajes. Ningún equipo combinó los dos tipos de paralelismo, por lo que los *speed-ups* obtenidos están lejos de los máximos estimados.

		Problems					
Team	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

Figura 1: Clasificación final del primer Concurso Español de Programación Paralela.

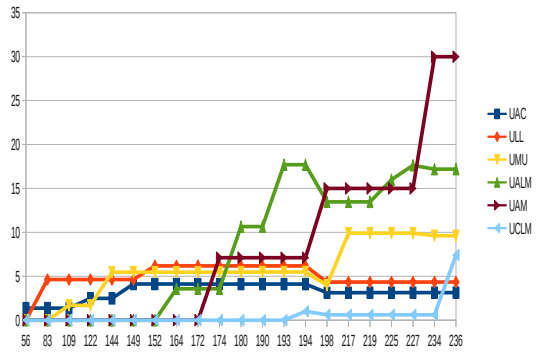

Figura 2: Evolución de la clasificación durante el primer Concurso Español de Programación Paralela.

La Figura 2 muestra la evolución de la clasificación. Se muestran los minutos en que hubo modificaciones. En algunos puntos la puntuación de algunos equipos disminuye (minutos 198 y 234), lo que ocurre cuando un equipo obtiene un *speed-up* mayor de 16, pues se recalculan con interpolación las puntuaciones para ese problema. La mayoría de los participantes estuvieron en algún momento en primera posición, y el mayor número de envíos se realizó en la última hora, con cuatro cambios en la cabeza. En la evolución de la clasificación se observa que los grupos que tardaron más en empezar a enviar soluciones (el rojo y el verde) obtuvieron los mejores resultados, probablemente porque meditaron un rato antes de ponerse a programar.

5. Conclusiones y perspectivas

En este trabajo se ha mostrado la experiencia de la organización del primer Concurso Español de Programación Paralela. A la vista de la experiencia, podemos considerar que la primera edición del concurso ha sido satisfactoria. Ha habido un número reducido de participantes, lo que es normal para la primera edición teniendo en cuenta la especialización del paralelismo. La experiencia fue positiva para los equipos participantes y los recursos generados en el concurso se han utilizado en cursos de paralelismo en el primer cuatrimestre del curso 2011/2012 en las universidades de La Laguna y Murcia. Tanto los organizadores del concurso como los de las Jornadas de Paralelismo han considerado interesante realizar la segunda edición dentro de las Jornadas de Paralelismo a celebrar en Elche en septiembre de 2012.

Algunas perspectivas futuras serían:

- En la segunda edición se organizará una sesión para seguir los últimos momentos del concurso y discutir las soluciones aportadas por los concursantes y otras posibles soluciones.
- La implementación de la página del concurso y su organización en inglés facilitará la participación *online* de estudiantes no españoles, y hará que el uso de la página web como recurso educativo se extienda.
- Es posible incluir en Mooshak algunas modificaciones adicionales para adaptarlo mejor al concurso. Por ejemplo, podría ser interesante incluir una visualización de la clasificación en el formato de la Figura 2.
- Está previsto realizar una prueba de programación en CUDA. Esto supone un trabajo adicional de organización, porque el paradigma de programación cambia y es necesario generar problemas que puedan resolverse con el

paradigma SIMD y estimar el *speed-up* esperable para estos problemas con este paradigma.

Agradecimientos

Subvencionado por la Fundación Séneca de la Región de Murcia, 08763/PI/08, y el Ministerio de Educación, TIN2008-06570-C04 y TIN2011-24598. Los organizadores del concurso agradecen al Centro de Supercomputación de la Fundación Parque Científico de Murcia el uso de sus recursos y su apoyo en la realización del concurso, a HP su patrocinio del concurso en 2011 y a NVidia en 2012.

Referencias

[1] Mooshak, system for managing programming contests on the web. `http://mooshak.dcc.fc.up.pt/~zp/mooshak/`.

[2] ACM International Collegiate Programming Contest. `http://cm.baylor.edu/welcome.icpc`.

[3] José Luis Fernández Alemán. Automated assessment in a programming tools course. *IEEE Trans. Education*, 54(4):576–581, 2011.

[4] Francisco Almeida, Domingo Giménez, José Miguel Mantas, and Antonio M. Vidal. *Introducción a la programación paralela*. Paraninfo Cengage Learning, 2008.

[5] Benjamin A. Burton and Mathias Hiron. Creating Informatics Olympiad Tasks: Exploring the Black Art. *Olympiads in Informatics*, 2:16–36, 2008.

[6] Centro de Supercomputación de Murcia. `http://www.cesmu.es/inicio/`.

[7] Valentina Dagienė. Sustaining informatics education by contests. In Juraj Hromkovic, Richard Královic, and Jan Vahrenhold, editors, *Teaching Fundamentals Concepts of Informatics*, volume 5941 of *Lecture Notes in Computer Science*, pages 1–12, 2010.

[8] A. Grama, A. Gupta, G. Karypis, and V. Kumar. *Introduction to Parallel Computing*. Addison-Wesley, second edition, 2003.

[9] Lasse Hakulinen. Survey on informatics competitions: Developing tasks. *Olympiads in Informatics*, 5:12–25, 2011.

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

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

[12] Jornadas de Paralelismo, 2011. `http://jp2011.pcg.ull.es/`.

[13] Rob Kolstad. Infrastructure for contest task development. *Olympiads in Informatics*, 3:38–59, 2009.

[14] Ramon Lawrence. Teaching data structures using competitive games. *IEEE Transactions on Education*, 47(11):753–759, September 2004.

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

[16] David J. Meder, Victor Pankratius, and Walter F. Tichy. `http://www.multicore-systems.org/separs/downloads/GI-WG-SurveyParallelismCurricula.pdf`.

[17] OpenMP web page. `http://openmp.org/wp/`.

[18] Michael J. Quinn. *Parallel Programming in C with MPI and OpenMP*. McGraw Hill, 2004.

[19] Marc Snir and William Gropp. *MPI. The Complete Reference. 2nd edition*. The MIT Press, 1998.

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

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

[22] UVa Online Judge. `http://online-judge.uva.es/problemset`.

[23] Troy Vasiga, Gordon Cormack, and Graeme Kemkes. What Do Olympiad Tasks Measure? *Olympiads in Informatics*, 2:181–191, 2008.