Una experiencia de combinación de heurísticas y programación paralela en un curso guiado por problemas [*]

Domingo Giménez
Departamento de Informática y Sistemas
Universidad de Murcia
Campus de Espinardo
30071, Murcia
domingo@um.es

Resumen

Este trabajo explica una experiencia en la que se estudian conjuntamente métodos heurísticos y programación paralela. Se ha llevado a cabo durante dos cursos. El curso se guía por problemas de asignación de procesos a procesadores en sistemas heterogéneos. El problema se plantea en la primera sesión del curso y los temas se desarrollan con referencia a él. Cada alumno elige una metaheurística o un método de búsqueda exhaustiva usando heurísticas para abordarlo. Se estudian algoritmos secuenciales y su paralelización, desarrollando programas para memoria compartida (OpenMP) y por paso de mensajes (MPI). Los objetivos del curso son: motivar a los alumnos y favorecer el aprendizaje combinando dos temas de estudio tradicionalmente independientes, y guiar el trabajo centrando el curso en un problema que combina los dos campos. Los resultados de una encuesta pasada a los alumnos confirman que su percepción es que los objetivos se alcanzan satisfactoriamente.

1. Introducción

El curso "Algoritmos y Programación Paralela"[8] se guía por problemas de asignación de tareas o procesos a procesadores en un entorno heterogéneo. Se trata de obtener la asignación que proporciona el menor tiempo de ejecución teórico. El problema de asignación es NP [11], y se puede aproximar la solución usando heurísticas. Los problemas se proponen a los alumnos al principio del curso, y el desarrollo de cada tema se centra en el problema propuesto. Se muestra la experiencia en dos cursos consecutivos. En cada curso se propone un problema distinto.

En el primer problema, el sistema presenta heterogeneidad en la velocidad de los procesadores y la red de interconexión, y los procesos que se asignan forman un conjunto homogéneo (modelo HoHe, Homogeneous processes in Heterogeneous system, [10]).

En el segundo problema, un procesador maestro genera una serie de tareas independientes y las envía a unos procesadores esclavos, que resuelven las tareas y le devuelven las soluciones. Cada tarea tiene un coste computacional y unas necesidades de memoria, y los procesadores tienen distintas velocidades y no se consideran comunicaciones. Cada procesador dispone de una cierta cantidad de memoria, lo que impone restricciones en las tareas que puede resolver.

Cada alumno propone la resolución del problema con algún método metaheurístico o de búsqueda exhaustiva que incluya heurísticas. De esta manera se estudian de manera conjunta métodos heurísticos y computación en entornos heterogéneos.

El trabajo se organiza de la siguiente forma: la sección 2 explica la asignatura; la 3 presenta los problemas propuestos; en la 4 se resume el estu-

[*] El trabajo ha sido financiado en parte por la Consejería de Educación de la Comunidad de Murcia, Fundación Séneca, 02973/PI/05.

dio de la aplicación de algunas heurísticas y metaheurísticas; la encuesta pasada a los alumnos se muestra en la sección 5; la sección 6 resume las conclusiones.

2. Organización del curso

La asignatura es de quinto de la Ingeniería Informática, en la Universidad de Murcia. En cursos anteriores se estudian Algoritmos y Estructuras de Datos, Arquitectura de Computadores, Programación Concurrente e Inteligencia Artificial, entre otras asignaturas. La asignatura es optativa, por lo que los alumnos tienen un nivel elevado y están interesados en el tema, y la elige un reducido número de estudiantes, con lo que es posible una docencia personalizada y centrada en el trabajo de los alumnos. Cada alumno desarrolla algoritmos secuenciales y paralelos para resolver un problema computacional complejo. El problema propuesto es de asignación de procesos a procesadores en un entorno heterogéneo. Así, se desarrollan conjuntamente las dos partes del temario (métodos secuenciales aproximados y programación paralela). Los alumnos realizan como trabajos y prácticas: una presentación sobre alguna técnica algorítmica, tanto secuencial como paralela; solución y estudio teórico y experimental de un algoritmo secuencial para el problema propuesto; desarrollo y estudio de algoritmos paralelos para memoria compartida (OpenMP) y paso de mensajes (MPI) a partir de alguno de los algoritmos secuenciales desarrollados.

Ya que la asignatura es cuatrimestral y contiene métodos secuenciales y programación paralela, el paralelismo se estudia aproximadamente en dos meses. Esto junto con la dificultad de una aproximación inicial al paralelismo hace que el objetivo sea introducir a los alumnos en las técnicas de la programación paralela, pero no que desarrollen nuevos algoritmos paralelos y lleven a cabo estudios detallados, sino que estudien y programen algoritmos ya disponibles, los adecúen al problema propuesto y diseñen experimentos con que obtener conclusiones significativas.

Los temas del curso son: Introducción a la complejidad de problemas, Métodos de recorrido de árboles de soluciones, Algoritmos probabilistas, Metaheurísticas, Algoritmos matriciales, Modelos de programación paralela, Análisis de algoritmos paralelos, Esquemas algorítmicos paralelos.

En la primera sesión se analizan problemas de asignación de procesos a procesadores, y se explica detalladamente el problema que se propone. En las demás sesiones se estudian las ideas de cada tema y se analiza su utilización en la resolución del problema planteado. De esta manera se propicia la participación y el intercambio de ideas entre los alumnos, al trabajar en el problema desde el principio y estar cada alumno trabajando con un método distinto.

Inicialmente se analiza la dificultad de resolver algunos problemas en un tiempo reducido, y se estudia la complejidad de algunos problemas de asignación. A continuación se repasan métodos de recorrido exhaustivo de árboles de soluciones y se analizan técnicas heurísticas para guiar la búsqueda y eliminar nodos, además de la aplicación de estas técnicas al problema planteado. En las siguientes sesiones se repasan los modelos de sistemas paralelos y se estudian paradigmas de programación paralela. Se usan los estándares OpenMP [2] y MPI [13]. Después se estudian algoritmos probabilistas y metaheurísticas, y se proporciona una lista de métodos que podrán utilizarse en la resolución del problema: Backtracking o Branch and Bound con eliminación de nodos basada en heurísticas posiblemente eliminando nodos que podrían llevar a la solución óptima, Backtracking con recorrido guiado por heurísticas, Algoritmos probabilistas, Ascensión de colinas, Búsqueda tabú, Búsqueda dispersa, Algoritmos genéticos, Colonia de hormigas, Temple simulado, GRASP... Los alumnos consultan algunos de los libros disponibles sobre algoritmos [1, 3] y metaheurísticas [7, 9], así como información en la web, para decidir el método a utilizar. En el siguiente tema se estudian esquemas algorítmicos matriciales paralelos, y se analiza su utilización en algunas de las partes de la solución del problema. Finalmente, se estudia el análisis teórico y experimental de algoritmos paralelos y algunos esquemas algorítmicos paralelos.

Cada estudiante realiza dos presentaciones: una sobre las ideas generales de la técnica que utilizará y su adecuación al problema de asignación, y otra sobre la paralelización de alguno de los algoritmos secuenciales propuestos. La primera presentación se realiza después de la sesión sobre algo-

ritmos numéricos, y la segunda tras el estudio de los esquemas algorítmicos paralelos. Antes de cada presentación se realiza una tutoría individual con cada alumno, para asesorarlo sobre el enfoque a seguir en la presentación y en la solución del problema. Las presentaciones son previas al trabajo práctico, de manera que los alumnos pueden intercambiar ideas sobre la forma de abordar el problema. Se deben desarrollar y estudiar teórica y experimentalmente algoritmos secuenciales y paralelos. El estudio debe incluir el análisis de cómo el uso del paralelismo contribuye a reducir el tiempo de ejecución o a obtener una mejor solución en el problema de optimización. La comparación experimental de los resultados obtenidos por distintos alumnos se valora positivamente.

3. Los problemas de asignación

3.1. Distribución de un conjunto homogéneo de procesos en un sistema heterogéneo

Se plantea una versión simplificada de un problema de asignación propuesto en [6]. Se utiliza el modelo de tiempo de ejecución de un programa paralelo homogéneo (todos los procesos trabajan con el mismo volumen de datos y tienen el mismo coste computacional) para obtener el número de procesos a asignar a cada procesador de un sistema heterogéneo para obtener el menor tiempo de ejecución teórico. El tiempo de ejecución de un algoritmo paralelo se modela en función de parámetros algorítmicos y del sistema:

$$t(s) = f(s, AP, SP) \tag{1}$$

donde s representa el tamaño del problema, los parámetros del sistema (SP) representan las características del sistema (el coste de una operación aritmética básica, el tiempo de inicio de comunicaciones, t_s, y de envío de un dato, t_w). Los parámetros algorítmicos (AP) se pueden modificar para obtener tiempos de ejecución distintos. El modelo del tiempo de ejecución tiene la forma:

$$t(s,p) = t_c t_{comp}(s,p) + t_s t_{start}(s,p) + t_w t_{word}(s,p) \tag{2}$$

donde p representa el número de procesos, t_c el coste de una operación aritmética básica, t_{comp} el número de operaciones aritméticas básicas, t_{start} el número de comunicaciones y t_{word} el número de datos que se comunican.

En un sistema homogéneo los valores de t_c, t_s y t_w son los mismos en los distintos procesadores, y se selecciona el número de procesos y su topología lógica para minimizar el tiempo de ejecución.

En un sistema heterogéneo se selecciona el número de procesos a usar (p) y cuántos se asignan a cada procesador, $d = (d_1, d_2, \ldots, d_P)$, con P el número de procesadores del sistema. Los costes de las operaciones aritméticas básicas se almacenan en un array t_c con P componentes, donde t_{c_i} es el coste en el procesador i. Los costes de t_s y t_w entre cada par de procesadores se almacenan en dos arrays de tamaños $P \times P$, donde $t_{s_{ij}}$ y $t_{w_{ij}}$ representan el *start-up* y el *word-sending time* de un proceso en el procesador i a otro en el procesador j. El modelo del tiempo de ejecución es el de la ecuación 2, pero con los valores de t_c, t_s y t_w:

$$\begin{aligned} t_c &= \max_{\{i=1,\ldots,P\}} \{d_i t_{c_i}\} \\ t_{s_{ij}} &= \max_{\{i,j=1,\ldots,P,\ d_i \neq 0, d_j \neq 0\}} \{t_{s_{ij}}\} \\ t_{w_{ij}} &= \max_{\{i,j=1,\ldots,P,\ d_i \neq 0, d_j \neq 0\}} \{t_{w_{ij}}\} \end{aligned} \tag{3}$$

donde el coste de una operación básica en un procesador es proporcional al número de procesos en el procesador, y no se consideran interferencias en las comunicaciones por procesos en el mismo procesador.

Obtener la distribución óptima de procesos a procesadores se convierte en un problema de recorrido de un árbol de asignaciones. La figura 1 muestra un árbol con $P = 3$. Cada nivel representa los procesadores a que se puede asignar un proceso. La altura del árbol no está limitada. Como los procesos son idénticos el árbol es combinatorio, pero con repeticiones pues se puede asignar más de un proceso al mismo procesador. Los alumnos deben decidir la forma del árbol lógico, y la representación del árbol o el conjunto con que van a trabajar. Debe decidirse la representación de las soluciones. Cada nodo puede representarse al menos de dos maneras. El nodo gris de la figura se puede representar

por $(1, 2, 2, \ldots)$, o se puede almacenar el número de procesos que se asigna a cada procesador, con lo que se representaría con $(1, 2, 0)$.

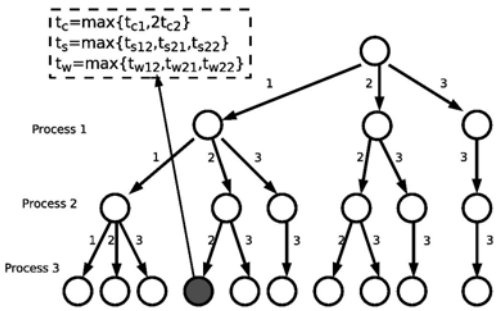

Figura 1: Árbol de asignación de procesos idénticos en un sistema con tres procesadores.

Para poder comparar los resultados, los alumnos deben experimentar con al menos dos funciones:

$$t_c \frac{n^2}{5p} + t_s \frac{p(p-1)}{2} + t_w \frac{n(p-1)}{2} \qquad (4)$$

que corresponde a un esquema paralelo de programación dinámica, y:

$$t_c \left(\frac{2}{3} \frac{n^3}{p} + \frac{n^2}{\sqrt{p}} \right) + t_s 2n\sqrt{p} + t_w \frac{2n^2}{\sqrt{}}$$

$$p \qquad (5)$$

que corresponde a una descomposición LU paralela. Además, los experimentos deben realizarse con valores en los rangos: $1 < t_c < 5$, $4 < t_w < 40$ y $20 < t_s < 100$. Los valores bajos de t_s y t_w se utilizan para simular sistemas de memoria compartida, los medios para multicomputadores de memoria distribuida, y los altos para sistemas distribuidos.

3.2. Asignación de tareas independientes en un sistema heterogéneo con restricciones de memoria

Un procesador maestro genera una serie de tareas independientes y las asigna a otros procesadores esclavos, que las resuelven y devuelven las soluciones. Las tareas tienen costes computacionales y requerimientos de memoria distintos. En la versión con que trabajan los alumnos no se tienen en cuenta las comunicaciones.

El sistema computacional consta de P procesadores. El array t_c almacena el coste de las operaciones aritméticas. Un array $m = (m_1, m_2, \ldots, m_P)$ representa la memoria de cada procesador.

Cada una de las T tareas requiere un volumen de datos dado por $i = (i_1, i_2, \ldots, i_T)$, y un coste aritmético representado por $c = (c_1, c_2, \ldots, c_T)$.

El objetivo es obtener la asignación $d = (d_1, d_2, \ldots, d_T)$ con restricciones $i_k \leq m_{d_k}$ con la que se obtiene el tiempo de ejecución mínimo:

$$\min_{\substack{d/\, i_k \leq m_{d_k} \\ \forall k = 1, \ldots, T}} \max_{j=1,\ldots,P} \left\{ t_{c_j} \sum_{\substack{l=1 \\ d_l = j}}^{T} c_l \right\} \qquad (6)$$

El árbol de posibles asignaciones tiene nivel T, y el nivel i corresponde a la tarea i, que se puede asignar a P procesadores. Cada nodo no terminal tiene P posibles hijos, y cada nodo terminal corresponde a una asignación. Hay un máximo de P^T asignaciones (se reduce con las restricciones de memoria), y no es posible resolver el problema en un tiempo razonable generando todas las asignaciones. Se puede obtener una solución aproximada por medio de alguna heurística.

Para unificar el estudio experimental, los alumnos deben realizar los experimentos al menos con los datos considerados en [5].

4. Aplicación de metaheurísticas a problemas de asignación

Se resume el trabajo realizado con tres métodos de distintas características. Dos son métodos metaheurísticos (algoritmo genético y búsqueda tabú) y el otro es un backtracking con heurísticas de poda de nodos que pueden eliminar ramas que llevan a la solución óptima. Se pretende obtener una asignación con tiempo de ejecución modelado cercano al óptimo, pero con tiempo de asignación reducido, ya que éste se sumaría al de ejecución.

En los algoritmos secuenciales el trabajo se centra en la representación algorítmica de alto nivel, que permite obtener diferentes versiones variando algunas rutinas o parámetros en el esquema. Para las técnicas metaheurísticas se puede usar un esquema común como el del algoritmo 1 [14].

```
Inicializar(S);
Mientras no CondicióndeFin(S) hacer
     SS = ObtenerSubconjunto(S);
     Si |SS| > 1
          SS1 = Combinar(SS);
     Si no
          SS1 = SS;
     Fin
     SS2 = Mejorar(SS1);
     S = IncluirSoluciones(SS2);
Fin
```

Algoritmo 1: Esquema de una metaheurística.

El trabajo en la parte secuencial ha consistido en:

- Con backtracking:

  Entender el problema de asignación y el incremento en el tiempo del backtracking para sistemas grandes, lo que lo hace impracticable.

  Desarrollar un esquema de backtracking para el problema. El esquema debe contener una rutina de poda de nodos fácilmente sustituible para experimentar con distintas técnicas.

  Identificar diferentes técnicas de poda, que pueden eliminar nodos que conducen a la solución óptima. Llevar a cabo experimentos para comparar los resultados con las distintas podas. Inicialmente se llevan a cabo experimentos con sistemas pequeños. Los mejores resultados se obtienen con el siguiente método. Se almacena el menor valor (GLV) de los tiempos de ejecución de los nodos generados. A cada nodo se le asocia un "valor mínimo"(NMV), y si $NMV > GLV$ se poda. NMV se calcula en un nodo sustituyendo en la fórmula el número de procesos por la máxima ganancia de velocidad que se puede alcanzar. Este método se usa en los experimentos sucesivos.

  Para sistemas pequeños el método se puede utilizar obteniendo tiempos modelados satisfactorios con un tiempo de asignación reducido. El paralelismo puede contribuir a reducir el tiempo de asignación, haciendo así el backtracking aplicable a un rango más amplio de sistemas.

- Con algoritmos genéticos:

  Comprender el problema de asignación e identificar posibles representaciones de individuos y de la población.

  Identificar la forma de las rutinas en el algoritmo 1 para un algoritmo genético.

  Desarrollar un esquema genético de alto nivel, en el que se puedan variar fácilmente algunos parámetros y rutinas.

  Ajustar experimentalmente el valor de los parámetros y las rutinas al problema. Para obtener tiempos de asignación reducidos es necesario reducir el número de individuos y de iteraciones, pero esto puede llevar a soluciones lejos de la óptima. Se obtuvieron resultados satisfactorios con 10 individuos y 10 iteraciones sin mejorar la solución. Los algoritmos genéticos no parecen la metaheurística más apropiada debido a que normalmente necesitan de poblaciones grandes, que conllevan grandes tiempos de ejecución.

- Con búsqueda tabú:

  Comprender el problema e identificar posibles representaciones de los elementos y el conjunto de elementos.

  Identificar la forma de las rutinas en el algoritmo 1 para la búsqueda tabú.

  Desarrollar un algoritmo de búsqueda tabú de alto nivel que permita variaciones en algunos parámetros y rutinas.

  Ajustar experimentalmente los valores de los parámetros y rutinas. Se obtienen resultados satisfactorios cuando: el número de iteraciones que un movimiento es tabú es la mitad de los procesadores; el estado inicial se obtiene asignando P procesos a los procesadores más rápidos; el número de iteraciones antes

de la diversificación es tres cuartos el máximo número de iteraciones.

En la parte paralela el trabajo ha consistido en:

- Con backtracking:

  Se consideran diferentes esquemas para obtener versiones paralelas, y finalmente se usa un esquema maestro-esclavo.

  Se obtiene una versión OpenMP donde el maestro genera nodos hasta un cierto nivel, y los nodos de ese nivel se asignan cíclicamente al conjunto de esclavos.

  La versión MPI trabaja del mismo modo, pero el maestro envía a los esclavos los nodos, y los esclavos realizan el backtracking a partir de los nodos que reciben y con información local y devuelven los resultados al maestro, que obtiene la solución global.

  Comparando las versiones secuencial y paralelas no se obtienen diferencias significativas en el tiempo modelado, y la reducción en el tiempo de ejecución por usar programación paralela está lejos de lo alcanzable, lo que puede ser debido a que se llevan a cabo backtracking independientes en los distintos procesos, con lo que se eliminan menos nodos en las versiones paralelas.

- Con algoritmos genéticos:

  La versión OpenMP sólo paraleliza la combinación de la población.

  Se analizan diferentes esquemas paralelos [12], y se usa el esquema de islas para la versión de paso de mensajes. Se incluyen como parámetros a ajustar el número de generaciones entre migraciones y el número de individuos que migran.

  Se comparan las versiones secuencial y paralela. Con OpenMP se obtienen las mismas asignaciones con una reducción importante del tiempo de ejecución. En MPI el tiempo no se reduce sustancialmente, pero normalmente se obtienen mejores asignaciones.

- Con búsqueda tabú:

  En la versión OpenMP, en cada paso cada *thread* explora a partir de un nodo.

  Para la versión MPI se analizan diferentes técnicas tabú [4]. Se utiliza la técnica pC/RS/MPDS: cada proceso realiza su propia búsqueda; no se comparte conocimiento entre procesos; se parte de varias soluciones iniciales; se usan estrategias de búsqueda distintas. Para diversificar la búsqueda algunos procesos comienzan con soluciones generadas de manera heurística, y otros las generan aleatoriamente, y el número de iteraciones que un movimiento es tabú depende del número de proceso.

  Se comparan las versiones secuencial y paralela. En OpenMP el *speedup* es satisfactorio, y en algunos casos superlineal. En MPI el tiempo no se reduce sustancialmente y se obtiene una pequeña mejora en la solución. Se puede conseguir una mayor reducción del tiempo de ejecución obteniendo asignaciones de valor similar reduciendo el número de iteraciones.

5. Evaluación de la docencia

Para comprobar si se han alcanzado los objetivos docentes se pasa una encuesta a los alumnos. Como el número de alumnos es reducido (26 entre los dos cursos, de los que unos 18 han seguido asiduamente la asignatura) y no todos han respondido la encuesta (10 alumnos), las conclusiones no se pueden considerar definitivas.

La encuesta contiene siete afirmaciones. En algunas un valor alto es positivo, pero en otras los valores positivos son los más bajos. Se pretende obligar a que la encuesta se realice leyendo cuidadosamente las cuestiones. Cada afirmación se valora de 1 a 5 (1 desacuerdo total y 5 acuerdo total). Se analizan dos aspectos de la docencia: si es apropiado combinar el estudio de métodos secuenciales aproximados con programación paralela (H-P), y si la utilización de un problema para guiar el curso tiene aspectos positivos (G-P).

Las afirmaciones de la encuesta son:

1. Combinar el estudio de algoritmos secuenciales de aproximación de las soluciones con el

de la programación paralela es adecuado pues propicia estudiar juntos dos métodos de resolución de problemas de alto coste computacional.

2. Combinar en una sola asignatura el estudio de métodos secuenciales con programación paralela hace que la asignatura tenga una carga excesiva, por lo que sería preferible que cada tema se estudiara en una asignatura distinta.

3. El planteamiento de un problema de asignación de programación paralela y abordar este problema con métodos heurísiticos ha servido para aclarar la ideas y profundizar en el estudio de métodos vistos en cursos anteriores (por ejemplo en Inteligencia Artificial o Algoritmos y Estructuras de Datos).

4. El problema de asignación planteado ha dificultado el estudio de la programación paralela, al plantearse un problema en un sistema heterogéneo (con heterogeneidad en la computación, la red y la memoria), mientras que las prácticas de programación se han realizado en sistemas homogéneos.

5. El planteamiento del problema de asignación desde el principio del curso ha servido para motivar y centrar desde el principio el estudio de la programación paralela.

6. Combinar y alternar la docencia de métodos secuenciales con temas de programación paralela ha dificultado seguir la asignatura al no profundizarse en un tema antes de pasar al siguiente.

7. El uso de un problema para guiar el curso es adecuado pues motiva el estudio de cada uno de los temas.

El objetivo H-P tiene una valoración positiva en las afirmaciones 1 y 3, y negativa en las 2, 4 y 6, y el G-P tiene valoración positiva en las 3, 5 y 7, y negativa en la 4. Para obtener la media para esos dos apartados, en los ítems negativos se intercambian los valores positivos y negativos (1 con 5 y 2 con 4). Se muestra el valor medio de las respuestas de los alumnos para cada afirmación:

1	2	3	4	5	6	7
3.78	2.89	4.33	2.56	3.78	3.11	4.11

La media para H-P es 3.51 y para G-P 3.92, y la conclusión es que según la opinión de los alumnos los dos objetivos docentes se alcanzan satisfactoriamente.

6. Conclusiones

Se presenta una experiencia que combina el estudio de métodos heurísticos con la programación paralela. Con esta combinación los alumnos comprenden mejor la importancia de los métodos aproximados y heurísticos para problemas de gran complejidad, y la dificultad e importancia del problema de asignación se entiende mejor al trabajar sobre él con métodos heurísticos. La programación paralela se introduce usando los mismos métodos heurísticos y metaheurísticos que se usan para tratar el problema de asignación, y se analizan posibilidades de paralelización a distintos niveles y en memoria compartida y por paso de mensajes. Como todos los alumnos trabajan con el mismo problema, colaboran en algunos puntos del trabajo, identificando representaciones comunes, rutinas que se pueden adaptar a distintos métodos, esquemas de paralelización comunes, diseñando conjuntamente baterías de experimentos, comparando los resultados experimentales...

Se ha pasado una encuesta a los alumnos para comprobar si se han alcanzado los objetivos pedagógicos. Los resultados son satisfactorios, por lo que se pretende continuar con la experiencia en cursos sucesivos utilizando otros poblemas del campo de la computación paralela: inclusión de comunicaciones en el problema de asignación con restricciones de memoria, búsqueda de servicios distribuidos...

Referencias

[1] G. Brassard and P. Bratley. *Fundamentals of Algorithms*. Prentice-Hall, 1996.

[2] Rohit Chandra, Ramesh Menon, Leo Dagum, David Kohr, Dror Maydan, and Jeff McDonald. *Parallel Programming in OpenMP*. Morgan Kauffman, 2001.

[3] Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest. *Introduction to Algorithms*. The MIT Press, 1990.

[4] T. G. Crainic, M. Gendreau, and J. Y. Potvin. Parallel tabu search. *Parallel Metaheuristics, E. Alba (ed.)*, 2005.

[5] J. Cuenca, D. Giménez, J. J. López-Espín, and J. P. Martínez-Gallar. A proposal of metaheuristics to schedule independent tasks in heterogeneous memory-constrained systems. In *Proc. IEEE Int. Conf. on Cluster Computing*. IEEE Computer Society, September 2007.

[6] J. Cuenca, D. Giménez, and J. P. Martínez-Gallar. Heuristics for work distribution of a homogeneous parallel dynamic programming scheme on heterogeneous systems. *Parallel Computing*, 31:717–735, 2005.

[7] J. Dréo, A. Pétrowski, P. Siarry, and E. Taillard. *Metaheuristics for Hard Optimization*. Springer, 2005.

[8] Domingo Giménez. Web page of the Algorithms and Parallel Programming course at the University of Murcia, `http://dis.um.es/~domingo/app.html`.

[9] Juraj Hromkovič. *Algorithmics for Hard Problems*. Springer, second edition, 2003.

[10] A. Kalinov and A. Lastovetsky. Heterogeneous distribution of computations while solving linear algebra problems on network of heterogeneous computers. *Journal of Parallel and Distributed Computing*, 61(44):520–535, 2001.

[11] H. Lennerstad and L. Lundberg. Optimal scheduling results for parallel computing. *SIAM News*, pages 16–18, 1994.

[12] G. Luque, E. Alba, and B. Dorronsoro. Parallel genetic algorithms. *Parallel Metaheuristics, E. Alba (ed.)*, 2005.

[13] Message Passing Interface Forum. *MPI: A Message Passing Interface Standard*. Univ. of Tennessee, Knoxville, Tennessee, 1995.

[14] G. R. Raidl. A unified view on hybrid metaheuristics. In *Hybrid Metaheuristics, Third International Workshop, LNCS*, volume 4030, pages 1–12, October 2006.