3rd International Conference on Industrial Engineering and Industrial Management
XIII Congreso de Ingeniería de Organización
Barcelona-Terrassa, September 2nd-4th 2009

Algoritmos cooperativos multicore para el problema del taller de flujo de permutación*

Gema Escrivá, Eva Vallada

Grupo de Sistemas de Optimización Aplicada. Instituto Tecnológico de Informática. Universidad Politécnica de Valencia. Cno. de Vera s/n, 46022.Valencia. geesgas@doctor.upv.es , evallada@eio.upv.es

Keywords: taller de flujo, tardanza, algoritmos cooperativos

1. Introducción

En el problema del taller de flujo de permutación o Permutation Flowshop tenemos un conjunto de N trabajos, N = {1,…., n}, que son procesados en un conjunto de M máquinas, M = {1,…., m}. Cada trabajo debe procesarse en todas las máquinas, siendo la secuencia de proceso la misma para todos los trabajos. Además, el orden de los trabajos es el mismo en todas las máquinas por lo que tendremos un total de n! secuencias posibles. El objetivo de optimización más estudiado en la literatura es la minimización del tiempo máximo de finalización de las máquinas o Makespan. Sin embargo, cada vez está más extendido el estudio de otros objetivos más realistas, especialmente los relacionados con fechas de entrega de los trabajos. De entre todos ellos, destaca la minimización del retraso o tardanza total en el taller de flujo de permutación, denotado como $F/prmu/\sum T_j$, donde $T_j=\max(0,C_j-d_j)$, siendo d_j la fecha de entrega establecida para el trabajo j y C_j su tiempo de finalización.

En la literatura podemos encontrar métodos muy eficaces y eficientes para el problema definido (Hasija y Rajendran, 2004; Parthasarathy y Rajendran, 1998; Armentano y Ronconi, 1999; Rajendran y Ziegler, 2003; Vallada y Ruiz, 2009) entre otros. En Vallada, Ruiz y Minella (2008) podemos encontrar una completa revisión y evaluación de algoritmos para el problema descrito.

Sin embargo, una de las alternativas más prometedoras para mejorar la eficacia de un algoritmo es implementar su versión paralela o cooperativa. En los últimos tiempos la computación paralela vuelve a estar en auge debido a las nuevas tecnologías en redes de computadores, las cuales nos permiten conectar ordenadores de una manera barata, sencilla y rápida. Por otro lado, las nuevas generaciones de ordenadores personales ya incorporan como mínimo doble núcleo (dual core) en sus procesadores, es decir, dos procesadores o núcleos físicos dentro de un mismo chip, por lo que aprovechando estas características, es posible paralelizar un algoritmo utilizando un solo ordenador.

* Este trabajo está parcialmente subvencionado por el Ministerio de Ciencia e Innovación, bajo los proyectos "OACS - Optimización Avanzada de la Cadena de Suministro" y "SMPA - Secuenciación Multiobjetivo Paralela Avanzada: Avances Teóricos y Prácticos" con referencias IAP-020100-2008-11 y DPI2008-03511/DPI, respectivamente

En este trabajo, se propone un algoritmo genético cooperativo multicore para el problema descrito que aprovecha toda la potencia de las últimas tecnologías de los ordenadores descritas en el párrafo anterior. El trabajo está organizado de la siguiente manera: en la Sección 2 se explicarán los algoritmos cooperativos multicore propuestos para el problema descrito. La Sección 3 trata sobre los diferentes experimentos llevados a cabo. Por último, en la Sección 4 encontramos un resumen y conclusiones del trabajo.

2. Algoritmo genético cooperativo multicore

La idea básica detrás de un algoritmo paralelo es dividir una tarea en varias partes y resolverlas de manera simultánea en distintos ordenadores o procesadores. El objetivo general de la paralelización de algoritmos es disminuir el tiempo de ejecución del correspondiente algoritmo secuencial. Si el objetivo a alcanzar es la mejora de la eficacia del algoritmo, es decir, lo buenas que son las soluciones obtenidas y no de su eficiencia, como suele ser habitual en computación paralela, la paralelización consiste en el desarrollo de algoritmos cooperativos (Alba, 2005; Talbi, 2006).

Los algoritmos genéticos pueden verse como técnicas de búsqueda de soluciones basadas en la teoría de la evolución. Este tipo de algoritmos son paralelizables de una manera sencilla, ya que ciertas operaciones son independientes de otras. Los algoritmos genéticos cooperativos consisten en ejecutar en distintos ordenadores, pero al mismo tiempo, el mismo algoritmo pero de manera que entre ellos colaboren o cooperen, es decir, consiste en la ejecución del mismo método de manera distribuida entre distintos ordenadores, teniendo de esta manera varios algoritmos trabajando en espacios de búsqueda distintos y compartiendo la información entre ellos mediante el paso de mensajes (Wodecki y Bozejko, 2002; Bozejko y Wodecki, 2002; Bozejko y Wodecki, 2003; Alba, 2005; Bozejko y Wodecki, 2006).

Aprovechando las últimas tecnologías de los ordenadores, que ya incorporan varios núcleos o cores, los algoritmos genéticos cooperativos multicore se ejecutan en todos los cores de los ordenadores, de manera independiente y permitiendo la comunicación entre ellos. El algoritmo genético cooperativo multicore propuesto en este trabajo parte de los trabajos previos de Vallada y Ruiz (2009) y Vallada y Ruiz (2009b). Como características principales destacan la aplicación de búsqueda local acelerada, control de diversidad de la población y aplicación de la técnica Path Relinking.

2.1. Operador de migración

Como se ha comentado anteriormente, los algoritmos cooperativos se basan en la comunicación entre distintos algoritmos. Esta comunicación se realiza habitualmente mediante el paso de mensajes. Por tanto, es necesario definir el funcionamiento de las comunicaciones mediante el operador de migración. En el caso que nos ocupa, el operador de migración está basado en el modelo isla, cada una de ellas con su propia población, donde se ejecuta de forma paralela un algoritmo genético secuencial con intercambios regulares de individuos entre las distintas islas.

Las características más importantes del operador de migración son las siguientes:

- El primer envío se realiza cuando ha transcurrido la mitad del tiempo total de CPU disponible. De esta manera dejamos que cada algoritmo ejecutado evolucione de manera independiente.
- Solo hay comunicación cuando se encuentra una nueva mejor solución, evitando de esta forma inundar la red con mensajes.

– Los individuos recibidos se insertan en la población si son mejores que el peor individuo de la población y además no existe ya una copia idéntica en la misma, es decir, no se permiten clones en la población.

3. Experimentos computacionales y resultados

El algoritmo genético cooperativo multicore propuesto se ha comparado con su correspondiente versión serie y con el algoritmo voraz iterativo propuesto en Ruiz y Stützle (2007) que resulta ser un método muy eficaz para el objetivo de minimizar la tardanza total. Todos los métodos han sido implementados en Delphi 2007. En cuanto al criterio de parada, se ha optado por el tiempo máximo transcurrido de CPU, según la expresión:

$n \cdot (m/2) \cdot t$, donde t = 90 ms.

donde observamos que el tiempo disponible aumenta a medida que lo hace también el número de trabajos o máquinas.

El conjunto de instancias utilizado es el mismo que en Vallada y Ruiz (2009) y Vallada y Ruiz (2009b) y que podemos encontrar en http://soa.iti.es. En concreto, los tiempos de proceso de los trabajos se generan a partir de una distribución uniforme entre 1 y 99 como es habitual en la literatura. Las fechas de entrega también se generan a partir de una distribución uniforme, siguiendo la siguiente expresión:

$[P(1-T-R/2),P(1-T+R/2)]$

donde T y R son dos factores conocidos como Tardiness Factor (T) y Due Date Range (R), P es una cota inferior del makespan, y donde n varía entre 50 y 350 trabajos y m entre 10 y 50 máquinas. Se ha seleccionado un subconjunto del total de todas las instancias generadas, en concreto el que corresponde a las instancias más difíciles de resolver desde el punto de vista de los valores de los parámetros Tardiness Factor (T) y Due Date Range (R). Es decir, se ha escogido la combinación que proporcionaba fechas de entrega más difíciles de cumplir, la cual corresponde a la formada por valores altos de T junto con valores bajos de R. Por tanto las instancias seleccionadas para esta evaluación, en el caso del banco de datos que se utiliza en este trabajo, son las correspondientes a T=0.6 y R=0.2, donde n= 50, 150, 250 y 350 trabajos y m=10, 20 y 30 máquinas. En total, 60 instancias (5 réplicas por cada combinación de n x m). La razón de reducir el tamaño del banco de datos es debido al tiempo de computación necesario para poder evaluar las 540 instancias iniciales con los algoritmos paralelos. Por otro lado, el hecho de desarrollar algoritmos cooperativos es precisamente para poder atacar problemas más difíciles, ya que como hemos podido ver en secciones anteriores, ya existen diferentes algoritmos series que resultan ser muy eficaces.

Todos los algoritmos han sido ejecutados en 2, 4, 8 y 12 ordenadores con tecnología Intel Core 2 Duo, donde cada núcleo funciona a 2.4GHz con 2 GB de RAM, es decir, se han ejecutado en 4, 8, 16, 20 y 24 núcleos o cores.

A lo largo de los experimentos, se van a realizar comparaciones de los algoritmos ejecutándolos en ordenadores que utilizan un solo núcleo o core contra ejecuciones en los que se utilizarán los dos núcleos o cores, teniendo siempre en cuenta, que en ambos casos, se tratará de los mismos ordenadores. Por tanto creemos conveniente indicar que hablaremos de **islas**, cuando los algoritmos se ejecuten utilizando un solo núcleo o core de los dos posibles del ordenador, mientras que hablaremos de **cores** cuando se utilicen los dos núcleos o cores del ordenador (algoritmo multicore).

En cuanto a la medida de eficacia, utilizamos la Desviación Porcentual Relativa (RPD) sobre la mejor solución conocida siguiendo la siguiente expresión:

$$\text{RPD} = \frac{Met_{sol} - Best_{sol}}{Best_{sol}} * 100$$

Donde $Best_{sol}$ es la mejor solución conocida para el problema evaluado y Met_{sol} es la solución obtenida por el método a evaluar.

Se van a llevar a cabo varios experimentos computacionales que se detallan en las siguientes subsecciones.

3.1. Primer experimento computacional y resultados

El primer experimento realizado consiste en comparar el algoritmo cooperativo multicore con el correspondiente algoritmo serie y con otro algoritmo serie disponible en la literatura que ha mostrado ser muy eficaz (Ruiz y Stützle, 2007). El objetivo de este primer experimento consiste en comprobar el efecto que tienen las comunicaciones en el algoritmo propuesto. Para ello, se ejecutarán los algoritmos series en el mismo número de cores que el cooperativo multicore pero de una manera aislada, es decir, sin comunicaciones. De esta manera, se evalúan los métodos bajo las mismas condiciones.

Como ya se ha comentado, el experimento consiste en comparar los resultados obtenidos mediante la ejecución del algoritmo cooperativo multicore, al que denotamos como CMGA, con su correspondiente algoritmo serie (Vallada y Ruiz, 2009b) al que llamamos GA y con el algoritmo voraz iterativo de Ruiz y Stützle (2007), el cual hemos adaptado al objetivo de minimizar el retraso o tardanza total, que denotamos como IG. Como ya hemos adelantado, el objetivo es el de comprobar si las comunicaciones incorporadas en la versión cooperativa están contribuyendo a la mejora de la calidad de las soluciones.

En el caso de los algoritmos serie, GA (Vallada y Ruiz, 2009b) e IG (Ruiz y Stützle, 2007), los ejecutaremos en el mismo número de cores que el cooperativo, pero de manera aislada, es decir sin el operador de migración, por tanto sin envío/recibo de individuos. De este modo, cada algoritmo serie explora su espacio de búsqueda y al final se selecciona la mejor solución encontrada entre todos ellos. En cuanto al algoritmo cooperativo multicore, CMGA, lo ejecutaremos en los diferentes cores (procesadores) permitiendo la comunicación entre ellos.

Como ya hemos indicado, todos los algoritmos los ejecutamos en 2, 4, 6, 8 y 12 ordenadores, es decir en 4, 8, 12, 16, 20 y 24 núcleos o cores, permitiendo solo las comunicaciones en el caso del algoritmo cooperativo multicore, CMGA.

En la Tabla 1 podemos observar la desviación porcuentual relativa media obtenida para cada algoritmo ejecutado en 4, 8, 12, 16, 20 y 24 cores.

Tabla 1. Desviación Porcentual Relativa Media para los algoritmos evaluados

Algoritmo	4 cores	8 cores	12 cores	16 cores	20 cores	24 cores
CMGA	1,53	1,24	1,00	0,91	0,83	0,76
GA	1,87	1,65	1,51	1,44	1,39	1,35
IG	1,95	1,71	1,58	1,51	1,45	1,41

Podemos observar en la Tabla 1, que efectivamente las comunicaciones tienen un efecto muy importante en los resultados obtenidos, independientemente del número de cores utilizados en la ejecución. El algoritmo cooperativo multicore (CMGA) ejecutado en 24 cores obtiene unos resultados un 77% mejor que su correspondiente versión serie ejecutada en 24 cores de manera aislada. Para validar los resultados obtenidos, es interesante comprobar si las diferencias observadas en las desviaciones medias son estadísticamente significativas. Para ello aplicamos un análisis de la varianza (ANOVA) (Montgomery, 2000). En la Figura 1 podemos observar el gráfico de medias con intervalos de Tukey (95% de confianza) para los métodos evaluados. Podemos ver que efectivamente las diferencias son estadísticamente significativas por un amplio margen, especialmente entre el algoritmo cooperativo multicore (CMGA) y los otros dos métodos serie (GA e IG).

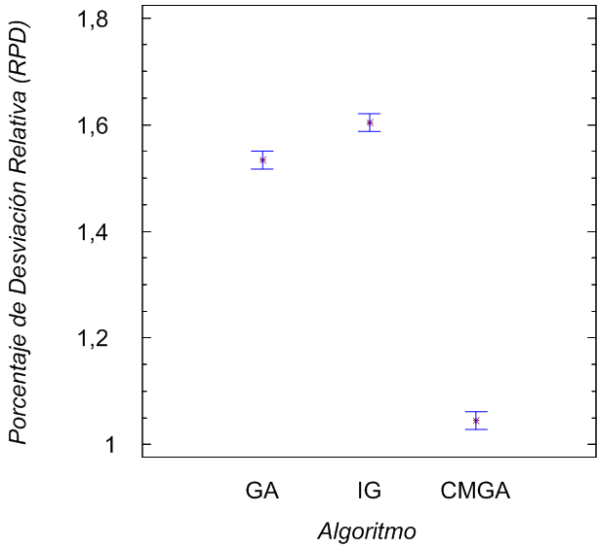

Figura 1. Gráfico de medias con intervalos HSD de Tukey a un nivel de confianza del 95% para el factor algoritmo para los algoritmos series GA y IG y para el algoritmo cooperativo CMGA

Por tanto, dicha mejora solo puede ser atribuida a la comunicación existente entre los algoritmos cooperativos. Otro factor muy importante a considerar en este tipo de algoritmos que se ejecutan en varios cores o núcleos de procesamiento es precisamente el número de cores utilizados en los experimentos, es decir, cómo afecta el factor número de cores a los resultados obtenidos. En la Figura 2 podemos observar el gráfico de medias con intervalos de Tukey (95% de confianza) para el factor número de cores. Podemos ver claramente como el RPD obtenido va disminuyendo a medida que lo hace el número de cores utilizados en los experimentos.

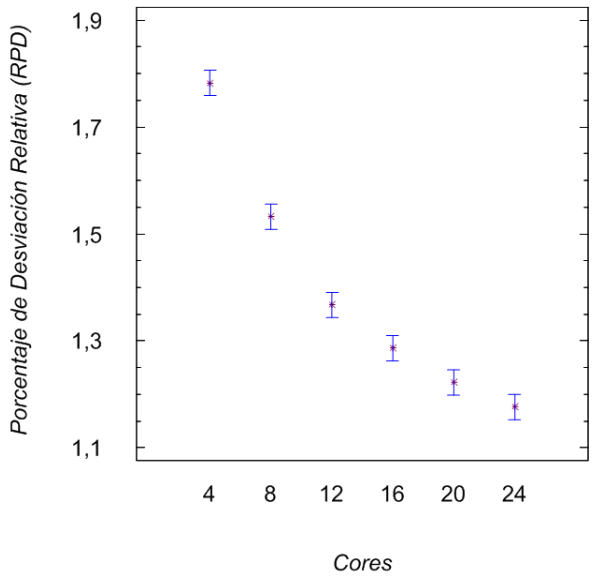

Figura 2. Gráfico de medias con intervalos HSD de Tukey a un nivel de confianza del 95% para el factor número de cores para todos los algoritmos.

Sin embargo, observamos que la tendencia de mejora va disminuyendo a medida que aumenta éste, llegando a intuirse que habrá un máximo de cores en el que la mejora ya no será estadísticamente significativa.

Por otro lado, cabe esperar que el factor número de cores no afecte de igual manera a los algoritmos serie que al algoritmo cooperativo multicore. En las Figuras 3 y 4 podemos observar el gráfico de medias para el factor número de cores en el caso del algoritmo cooperativo multicore (CMGA) y de su correspondiente versión serie ejecutada de manera aislada (GA) .Como se puede ver, para el caso del algoritmo serie GA, a partir de 16 cores la mejora obtenida al aumentar el número de cores ya no es estadísticamente significativa, es decir, a partir de 16 cores la eficacia del algoritmo serie ya no mejora al aumentar el número de cores. Sin embargo para el algoritmo CMGA, la mejora va disminuyendo a medida que aumentamos el número de cores, pero siendo siempre estadísticamente significativa.

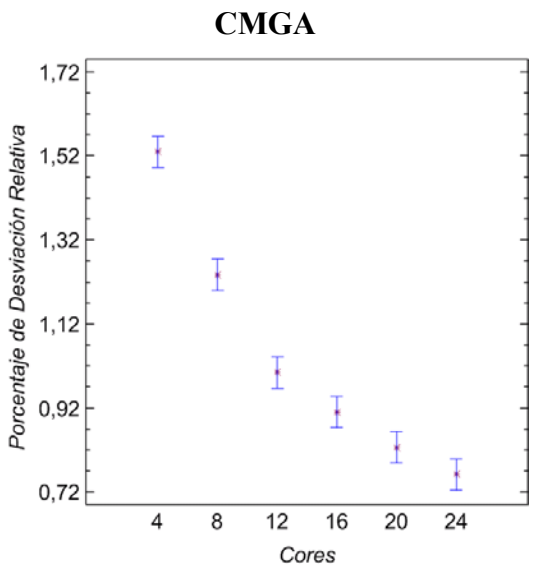

Figura 3. Gráfico de medias con intervalos HSD de Tukey a un nivel de confianza del 95% para el factor número de cores para el algoritmo CMGA

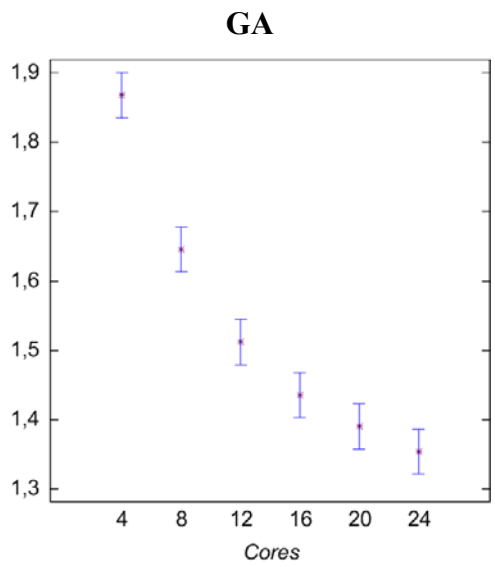

Figura 4. Gráfico de medias con intervalos HSD de Tukey a un nivel de confianza del 95% para el factor número de cores para el algoritmo GA

3.2. Segundo experimento computacional y resultados

El objetivo del segundo experimento realizado es comprobar el efecto de utilizar los dos cores disponibles en los ordenadores. En concreto, se tratará de responder a la siguientes pregunta: ¿es equivalente ejecutar un algoritmo en dos ordenadores que utilicen solo uno de sus cores (dos islas) que ejecutarlo en un solo ordenador que utilice sus dos cores (multicore)?. Por otro lado, realizaremos un último experimento con el objetivo de averiguar cuánto mejora la eficacia del algoritmo al utilizar toda la potencia del ordenador, es decir, al utilizar los dos cores. Recordemos que hablaremos de **islas** cuando el algoritmo se ejecute en ordenadores que utilizan solo uno de los cores disponibles y de **cores** cuando se ejecute el algoritmo en ordenadores que utilizan los dos cores. Hay que hacer hincapié también en que los ordenadores utilizados en ambos casos son los mismos, la diferencia es utilizar uno de los cores o los dos. En concreto, el primero de los experimentos descrito en esta subsección, consistirá en ejecutar los algoritmos series, GA y IG y el algoritmo cooperativo CMGA en 4, 8 y 12 islas (ordenadores en los que solo se utilizó un core) y realizar una evaluación de los resultados obtenidos. Posteriormente, se realizó el mismo experimento pero utilizando los dos cores de los ordenadores, por tanto, ejecutamos los mismos algoritmos en 4, 8 y 12 cores, es decir, en 2, 4 y 6 ordenadores utilizando los dos cores.

En la Tabla 2 podemos observar los resultados obtenidos, donde vemos, por ejemplo, que la desviación porcentual relativa (RPD) obtenida con 4 islas es muy similar a la obtenida con 4 cores, pero teniendo en cuenta que en el caso de la ejecución en 4 cores se han utilizado solo dos ordenadores, mientras que en el caso de la ejecución en 4 islas se han utilizado cuatro ordenadores.

Tabla2. Desviación Porcentual Relativa Media para los algoritmos evaluados

Algoritm	**4 islas**	**4 cores**	**8 islas**	**8 cores**	**12 islas**	**12 cores**
CMGA	1,57	1,53	1,2	1,24	1,02	1,00
GA	1,82	1,87	1,66	1,65	1,56	1,51
IG	1,90	1,95	1,72	1,71	1,60	1,58

Para confirmar estos resultados, aplicamos de nuevo un análisis de la varianza (ANOVA). En la Figura 5 podemos observar el gráfico de medias. Denotamos como GA1 al algoritmo GA ejecutado en islas, GA2 al algoritmo GA ejecutado en cores, IG1 al algoritmo IG ejecutado en islas, IG2 al algoritmo IG ejecutado en cores y por último CMGA1 al algoritmo CMGA ejecutado en islas, y CMGA2 al algoritmo CMGA2 ejecutado en cores. Podemos ver que efectivamente, no hay diferencias estadísticamente significativas entre ejecutar el algoritmo en un número x de islas y ejecutarlo en el mismo número x pero de cores, es decir en la mitad de ordenadores pero utilizando los dos cores del ordenador. Es decir, es lo mismo ejecutar el algoritmo CMGA en 4 islas (cuatro ordenadores físicos) que en 4 cores (dos ordenadores físicos que utilizan los dos cores).

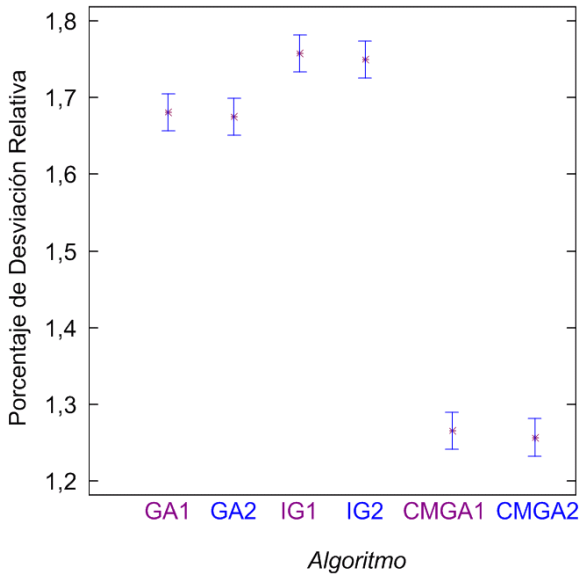

Figura 5: Gráfico de medias con intervalos HSD de Tukey a un nivel de confianza del 95% para el factor algoritmo para los algoritmos series GA y IG y para el algoritmo cooperativo CMGA

En vista de los resultados obtenidos en el experimento anterior, un último experimento consistirá precisamente en intentar calibrar o medir cuánto estamos mejorando la eficacia del algoritmo al utilizar toda la potencia disponible en el ordenador, es decir, los dos cores disponibles. Para ello hicimos la comparación de los resultados obtenidos para los dos algoritmos serie, GA y IG y para el algoritmo cooperativo CMGA, en 4, 8, y 12 islas (ordenadores en los que solo se utilizó un procesador) con los resultados obtenidos para los mismos algoritmos pero con 8, 16 y 24 cores, es decir, los mismos 4, 8 y 12 ordenadores utilizando sus dos cores o núcleos. En la Figura 6 podemos ver el gráfico de medias obtenido para este último experimento. Observamos que la mejora obtenida al utilizar la tecnología multicore, llega a ser de un 34%, es decir, que utilizando el mismo número de ordenadores (12 ordenadores o islas) pero con todos sus cores (24 cores), los resultados obtenidos son, en el caso del algoritmo cooperativo multicore (CMGA2), de un 34% mejor que el mismo algoritmo pero ejecutado en 12 islas (CMGA1).

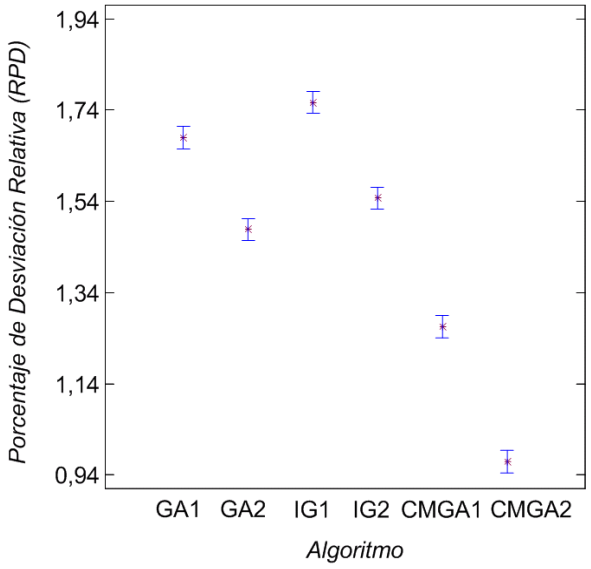

Figura 6: Gráfico de medias con intervalos HSD de Tukey a un nivel de confianza del 95% para el factor algoritmo para los algoritmos series GA y IG y para el algoritmo cooperativo CMGA

4. Conclusiones

En este trabajo se ha propuesto un algoritmo genético cooperativo multicore para el problema del taller de flujo de permutación con el objetivo de minimizar la tardanza total que aprovecha toda la potencia de las últimas tecnologías multicore. Para ello se ha comparado el algoritmo propuesto con otros algoritmos ya existentes, y se han ejecutado todos ellos utilizando uno y dos cores de los ordenadores.

De los resultados de la evaluación se concluye que el algoritmo propuesto (CMGA) obtiene mejores resultados utilizando menos ordenadores, que la utilización de las comunicaciones hace que obtenga hasta un 77% de mejoras en los resultados y que además, se consigue hasta un 34% de mejora utilizando el mismo número de ordenadores al aprovechar la tecnología de doble core. Por último se puede concluir que los resultados también muestran la escalabilidad del algoritmo, ya que mejoran su eficacia a medida que aumenta el número de cores.

Como líneas de trabajo futuro se podría considerar el diseñar y desarrollar algoritmos multicore para métodos heterogéneos, hacer hincapié en la mejora de las búsquedas locales así como en la utilización de ordenadores con mayor número de cores.

Referencias

Alba, E. (2005). Parallel Metaheuristics. A New Class of Algorithms. Wiley.

Armentano, V.A.; Ronconi, D.P. (1999). Tabu search for total tardiness minimization in flow-shop scheduling problems. Computers and Operations Research, Vol. 26, pp. 219-35.

Bozejko, W.; Wodecki, M. (2002). Solving the flow shop problem by tabu search. Proceedings of the International Conference on Parallel Computing in Electrical Engineering, pp. 1730-1737.

Bozejko, W.; Wodecki, M. (2003). Parallel genetic algorithm for the flow shop scheduling problem. Parallel Processing and Applied Mathematics, LNCS, Vol. 3019, pp. 566-571.

Bozejko, W.; Wodecki, M. (2006). A new inter-island genetic operator for optimization problems with block properties. Artificial Intelligence and Soft Computing, Lecture Notes in Artificial Intelligence, Vol. 4029, pp. 334-343.

Hasija, S.; Rajendran, C. (2004). Scheduling in flowshops to minimize total tardiness of jobs. International Journal of Production Research, Vol. 42, pp. 2289-2301.

Montgomery, D. (2000). Design and Analysis of Experiments. John Wiley & Sons, New York, quinta edición.

Parthasarathy, S.; Rajendran, C. (1997). A simulated annealing heuristic for scheduling to minimize mean weighted tardiness in a flowshop with sequence-dependent setup times of jobs-a case study. Production Planning and Control, Vol. 8, pp. 475-83.

Rajendran C.; Ziegler, H. (2003). Scheduling to minimize the sum of weighted flowtime and weighted tardiness of jobs in a flowshop with sequence-dependent setup times. European Journal of Operational Research, Vol. 149, pp. 513-22.

Ruiz, R; Stützle, T. (2007). A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem. European Journal of Operational Research, Vol. 177, pp. 2033-2049 .

Talbi, E. (2006). Parallel combinatorial optimization. Wiley.

Vallada, E.; Ruiz, R.; Minella, G. (2008). Minimising total tardiness in the m-machine flowshop problem: A review and evaluation of heuristics and metaheuristics. Computers & Operations Research, Vol. 35, pp. 1350-1373.

Vallada, E.; Ruiz, R. (2009). Genetic algorithms with path relinking for the minimum tardiness permutation flowshop problem. Aceptado para publicación en OMEGA, the International Journal of Management Science. Disponible on line en http://dx.doi.org/10.1016/j.omega.2009.04.002.

Vallada, E.; Ruiz, R; (2009b). Cooperative metaheuristics for the permutation flowshop scheduling problem. European Journal of Operational Research, Vol. 193, pp. 365-376.

Wodecki, M.: Bozejko, W. (2002). Solving the flow shop problem by parallel simulated annealing. Parallel Processing and Applied Mathematics, LNCS, Vol. 2328, pp. 236-244.