X Congreso de Ingeniería de Organización
Valencia, 7 y 8 de septiembre de 2006

Influencia de la granularidad en las prestaciones de algoritmos paralelos

José Miguel León Blanco[1], José Manuel Framiñán Torres[1], Pedro Luis González Rodríguez[1], Paz Pérez González[1]

[1] Dpto. de Organización Industrial y Gestión de Empresas. Universidad de Sevilla. Camino de los Descubrimientos s/n, 41092 Sevilla. miguel@esi.us.es, jose@esi.us.es, pedroluis@esi.us.es,

Resumen

En el ámbito del procesamiento paralelo en computación, se conoce como granularidad la relación entre el tiempo de computación y el tiempo de comunicación en un algoritmo paralelo. En el presente trabajo se presentan las ventajas que aporta el incremento de la granularidad en algoritmos de búsqueda local aplicados en este caso al problema de la secuenciación de trabajos en flujo uniforme sin restricciones de capacidad.

Palabras clave: Procesamiento paralelo, granularidad, metaheurística

1. Introducción

Las ventajas del proceso paralelo en la implementación de algoritmos de búsqueda local se han destacado en la literatura desde hace más de una década (Verhoeven y Aarts (1995)). Por un lado, se busca una implementación que consiga mejorar los tiempos de búsqueda en grandes espacios de soluciones. Una vez conseguidas soluciones de calidad en tiempos razonablemente cortos, se observa que la exploración más exhaustiva del espacio de soluciones que consigue el algoritmo paralelo, conlleva mejoras en la calidad de las soluciones. En este trabajo se han comprobado, mediante la experimentación con un sistema paralelo, las variaciones que se producen, tanto en la calidad de soluciones como en el tiempo de ejecución, cuando se cambia la granularidad en los algoritmos paralelos.[*]

2. Granularidad en algoritmos paralelos

La granularidad de un algoritmo se ha venido definiendo, ver por ejemplo, Cung et al (2001), como la proporción de tiempo que dedica dicho algoritmo a la comunicación entre procesos y el tiempo dedicado a computación. En el caso de metaheurísticas, su importancia reside en el hecho de que el aumento en el esfuerzo de comunicación entre procesos mejora el conocimiento que adquieren estos procesos sobre la exploración del espacio de soluciones que está realizando el resto de procesos. Esta mejora de la exploración es la base del aumento de prestaciones de los algoritmos paralelos frente a la simple ejecución repetitiva del algoritmo original.

3. Aplicación a metaheurísticas

[*] Este trabajo se deriva de la participación de sus autores en el proyecto de investigación financiado por CICYT con referencia DPI-2004-02902 titulado “Sistemas Avanzados de Secuenciación y Control”.

Es interesante encontrar el balance adecuado entre tiempo de comunicación y tiempo de exploración del espacio de soluciones. Un aumento en el esfuerzo de computación, acerca el comportamiento del algoritmo paralelo al del algoritmo secuencial. En este escenario, la mejora en calidad de las soluciones se debe, en mayor medida, a la búsqueda que ejecuta cada proceso por separado. El aumento del esfuerzo de comunicación, conlleva un mayor conocimiento de las soluciones encontradas en otros procesos, pero también implica mayores tiempos de espera hasta completar la comunicación. En este caso, la diversificación que se produce por la ejecución en paralelo, incrementa la del propio algoritmo y suele evitar el estancamiento en mínimos locales.

En este trabajo, se ha experimentado con la versión paralela del algoritmo CLM (Complete Local Search with Memory) de Ghosh y Sierksma (2002) aplicada a la secuenciación de trabajos en flujo uniforme propuesta en León et al (2005). Se trata de una versión maestro-esclavo en la que se simula una memoria central mediante una lista de soluciones que se mantiene en el proceso master. Este proceso genera soluciones de partida seleccionándolas de entre las existentes en la lista o *pool* central para que los procesos esclavos realicen una búsqueda local. Se ha variado el algoritmo CLM original, de forma que los procesos esclavos mantienen una lista adicional a las ya empleadas, LIVE, DEAD y NEWGEN, de soluciones exploradas por el resto, que se construye con una parte de las soluciones de la lista central mantenida en el proceso master.

Se presenta a continuación un resumen del seudocódigo del algoritmo:

Proceso master:

1 Envío inicial: Para cada proceso esclavo
  - 1.1 Se genera una solución aleatoria y distinta para cada proceso
  - 1.2 Se añade al pool central
  - 1.3 Se envía la solución al proceso esclavo
2 Mientras el número de iteraciones sea menor al fijado
  - 2.1 Se reciben las b mejores soluciones de un proceso esclavo
  - 2.2 Se almacenan en el pool central las que sean distintas de las existentes
  - 2.3 Se genera una nueva solución de partida para el proceso esclavo
  - 2.4 Se genera una nueva política de exploración para ese proceso
  - 2.5 Se envía la solución de partida junto con la política de exploración
3 Se envía una señal de parada a los procesos esclavos

Proceso esclavo:

1. Recibe la solución inicial del proceso master junto con la política de exploración
2. Se incluye en la lista LIVE
3. Mientras no se reciba la señal de parada
   - 3.1. Mientras no se cumpla la condición de parada y la lista LIVE no esté vacía
     - 3.1.1. Se toman k soluciones de la lista LIVE
     - 3.1.2. Se pasan a la lista DEAD
     - 3.1.3. Se actualiza el umbral de aceptación de soluciones
     - 3.1.4. Se generan soluciones vecinas de cada una de las anteriores tales que estén dentro del umbral de aceptación
     - 3.1.5. Si no están en ninguna de las listas, se incluyen en la lista NEWGEN
     - 3.1.6. Si alguna mejora la mejor solución hasta el momento, se actualiza el valor del mínimo

3.1.7. Si se supera el tamaño máximo de memoria (condición de parada) se pasa a postproceso
3.1.8. Se pasan todas las soluciones de NEWGEN a LIVE
3.1.9. Si el número de iteraciones sin mejora supera el tope, condición de parada.
3.2. Postproceso: Se pasan todas las soluciones que queden en la lista LIVE a DEAD después de una búsqueda local
3.3. Se envían las B mejores soluciones de la lista DEAD al proceso master
3.4. Se vacían las listas DEAD y POOL
3.5. Recibe la siguiente solución de partida junto con la política de exploración y la señal de continuar o parar del master
3.6. Si no recibe la señal de parada, recibe también las soluciones de la lista central y las almacena en la lista POOL.

3.1. Variación de los parámetros de búsqueda

El balance entre comunicación y computación se logra, en un primer momento, variando el valor de los parámetros de búsqueda. Un aumento del tamaño de las listas de soluciones que manejan los procesos esclavos implica mayor tiempo de proceso y por tanto mayor granularidad del algoritmo paralelo. Se trata de un algoritmo tipo *coarse-grained* o de grano grueso.

El tiempo de ejecución del algoritmo de búsqueda local empleado es muy sensible ante variaciones en el tamaño de las listas de soluciones, la memoria del algoritmo. Esto también es debido a la propia estructura del espacio de soluciones, en la que el paisaje, más que contener grandes valles (ver, para una descripción más detallada, el trabajo de Reeves y Eremeev (2004)) que puedan conducir tanto al óptimo global como a óptimos locales, parece más bien estar formado por grandes llanuras, en las que se pueden encontrar decenas de soluciones con el mismo valor de la función objetivo.

Para reducir el tiempo de exploración, se puede disminuir el tamaño de las listas de soluciones antes mencionadas. Para compensar la pérdida de calidad de las soluciones, debido a que la exploración es menos exhaustiva en cada iteración de cada proceso esclavo, se puede aumentar el número de iteraciones globales, con lo que se consigue un mayor intercambio de información entre los procesos. De esta forma, se mejora la diversificación del algoritmo de búsqueda. Con este aumento del intercambio de información, se observa (ver tabla 1), incluso con un reducido número de procesos, dos en este caso, una mejora en la calidad de las soluciones y una enorme disminución en el tiempo de computación.

Tabla 1. Experimentos con problemas grandes de la batería de Taillard(1993)

Dimensión	**200x10**		**500x20**	
Parámetros	ARPD (%)	Tiempo (s)	ARPD (%)	Tiempo (s)
mem:150, it:500, glob_it: 10	8,6	1452	12,5	47902
mem:10, it:50, glob_it: 20	6,5	164	11.2	5279.4

3.2. Experimentación en un sistema paralelo

Se han realizado experimentos adicionales en un sistema Linux debian de 12 ordenadores Dell Poweredge 750, con un procesador P IV a 3,2 GHz y 1 GB de memoria cada uno, conectados mediante un concentrador Gigabit ethernet. Para codificar el algoritmo se ha

empleado lenguaje C junto con la librería LAM-MPI para la comunicación entre los procesos. En la tabla 2 se muestran los parámetros empleados en el algoritmo, siendo n la dimensión del problema (número de trabajos). Para una descripción más detallada de los parámetros del algoritmo CLM puede consultarse Ghosh y Sierksma (2002):

Los experimentos referidos en la tabla 3 y siguientes, se refieren a la ejecución mediante 12 procesos, con objeto de obtener tiempos de ejecución reducidos. No se ha variado el número máximo de iteraciones sin mejora. En ella se observan las mismas tendencias observadas en la ejecución con dos procesos.

Tabla 2. Parámetros empleados en la ejecución del algoritmo

Parámetros globales	Tamaño del pool central (número de soluciones)	0,1*(n-1)*n/2
	Número de Iteraciones globales	Variable según el número de procesos
Parámetros de la búsqueda CLM	α	0,1
	β	0,1
	K	2
	Número máximo de iteraciones sin mejora	200
	Memoria máxima	Variable

Tabla 3. Experimentos para 12 procesos y 55 iteraciones globales (5 por proceso esclavo)

Dimensión	**100x5**		**200x10**		**200x20**		**500x20**	
Memoria máxima (*n*(n-1)/2)	ARPD (%)	Tiempo (s)	ARPD (%)	Tiempo (s)	ARPD (%)	Tiempo (s)	ARPD (%)	Tiempo (s)
0,1	2,42	0,1774	7,44	2,3563	14,60	3,5514	13,34	169,5086
0,2	2,13	0,4282	6,69	6,8932	14,68	9,5443	-	-
0,3	2,03	0,8785	5,88	15,8209	15,08	19,8690	13,65	1818,6598

A continuación se aumentó el número de iteraciones globales a 7 por proceso. Los resultados se muestran en la tabla 4:

Tabla 4. Experimentos para 12 procesos y 77 iteraciones globales (7 por proceso esclavo)

Dimensión	**100x5**		**200x10**		**200x20**		**500x20**	
Memoria máxima (*n*(n-1)/2)	ARPD (%)	Tiempo (s)	ARPD (%)	Tiempo (s)	ARPD (%)	Tiempo (s)	ARPD (%)	Tiempo (s)
0,1	2,16	0,2192	5,85	3,0160	14,06	4,6935	12,88	231,6118

Para comprobar la eficiencia del algoritmo, en la tabla 5 se presentan los resultados de repetir el experimento con 5 procesos, un master y cuatro esclavos:

Tabla 5. Experimentos adicionales para 5 procesos y 80 iteraciones globales (20 por proceso esclavo)

Dimensión	**100x5**		**200x10**		**200x20**		**500x20**	
Memoria máxima (*n*(n-1)/2)	ARPD (%)	Tiempo (s)	ARPD (%)	Tiempo (s)	ARPD (%)	Tiempo (s)	ARPD (%)	Tiempo (s)
0,1	0,55	0,4821	3,47	7,9024	10,93	13,4543	10,98	888,9193

En resumen, se observan las mismas tendencias que con un número reducido de procesos. El aumento en la diversificación propiciado por la mayor granularidad del algoritmo, tiene mejor

influencia en la mejora de las soluciones que el tiempo dedicado a la exploración más exhaustiva del espacio de soluciones. Esto es particularmente interesante en el caso de los problemas de mayor dimensión, que son los que necesitan de un mayor esfuerzo de computación para lograr resultados aceptables de la función objetivo. La implementación en paralelo de algoritmos de búsqueda local se ve muy beneficiada con la consecución de un balance eficiente entre computación y comunicación.

4. Líneas de investigación

Para lograr un algoritmo de grano fino, se han observado, dentro de la estructura del algoritmo de búsqueda, los instantes en los que aparecen soluciones nuevas para realizar ahí los intercambios de soluciones entre los procesos.

En los procesos que realizan la búsqueda local, se seleccionan soluciones de una lista para así ir generando una nueva lista de soluciones vecinas de aquellas en cada iteración. El momento en el que la lista inicial se amplía con las vecinas recién generadas es el momento más apropiado. Aparece en el seudocódigo anterior con el punto 3.1.8. Antes de pasar las soluciones recién generadas a la lista LIVE, se toman las b mejores, para comunicarlas al resto de procesos. Se incrementa así el intercambio de información antes de pasarla al proceso master, influyendo así antes en el desarrollo de la búsqueda del resto de procesos.

Para llevarla a cabo, se emplea la comunicación colectiva que permite la librería MPI. Así, cada proceso envía sus B mejores soluciones recién encontradas a todos los demás procesos. Por otro lado, si el total de procesos es n, recibirá (n-1)·B soluciones de los procesos esclavos, ya que se incluyen las enviadas por el mismo proceso. La función empleada, MPI_Allgather, está orientada a la comunicación colectiva, de forma que se acelera el proceso de comunicación frente a un envío secuencial de la información de cada proceso a cada uno de los demás.

En la tabla 6 se muestran los resultados de implementar el algoritmo con 12 procesos, en las mismas condiciones que el apartado anterior.

Tabla 6. Experimentos con 12 procesos y 33 iteraciones globales (3 por cada proceso esclavo)

Dimensión	**100x5**		**200x10**		**200x20**		**500x20**	
Memoria máxima (*n*(n-1)/2)	ARPD (%)	Tiempo (s)	ARPD (%)	Tiempo (s)	ARPD (%)	Tiempo (s)	ARPD (%)	Tiempo (s)
0,1	1,98	0,1245	5,75	1,0234	13,29	1,5346	11,44	23,6456

Los resultados en tiempo de ejecución son enormemente prometedores, en cuanto a tiempo de ejecución y calidad de las soluciones, si se comparan con los obtenidos mediante la versión anterior del algoritmo paralelo que se muestra en las tablas 3 y 4.

De todas formas, siempre que se incrementa la comunicación entre procesos y a pesar de contar con una red de comunicaciones rápida entre los nodos que componen el cluster donde se han realizado los experimentos, siempre tiene lugar una pérdida de eficiencia del algoritmo paralelo. Recordamos que la eficiencia mide el aprovechamiento de los nuevos procesadores añadidos al sistema paralelo según el tiempo que se tarda en alcanzar soluciones de una calidad dada. Suele emplearse la expresión (1)

$$eficiencia = \frac{tiempo_{1\,proceso}}{tiempo_{nprocesos} \cdot n} \quad (1)$$

La siguiente de las dificultades que habrá de superarse es relativa a la comparación de las prestaciones del algoritmo así modificado con el original. En este caso, la comunicación entre los procesos esclavos se lleva a cabo de forma síncrona, con lo que las condiciones de parada empleadas en el algoritmo original, tamaño total de las listas y número de iteraciones sin mejora, deben sustituirse por un número de iteraciones prefijado, 3 en el caso de estos experimentos, de manera que el intercambio de información se produzca de forma simultánea. Esto conlleva mayores tiempos de espera y por tanto también mayores pérdidas de eficiencia.

En general, mediante la adecuada experimentación, se llega a un compromiso en los valores de los parámetros entre la calidad de las soluciones y el tiempo de ejecución del algoritmo, aun a costa de perder la eficiencia de la implementación paralela.

Referencias

Cung, V.D.; Martins, S.; Ribeiro, C.; Roucariol, C. (2001) Strategies for the parallel implementation of Metaheuristics. *Essays and Surveys in Metaheuristics*. C.C. Ribeiro and P. Hansen, editors, Kluwer Academic Publishers.

Ghosh, D.; Sierksma, G.; (2002). Complete Local Search with Memory. *Journal of Heuristics*, No. 8, pp. 571-584. Kluwer.

León, J.M.; González, P.L.; Ruiz Usano, R. (2005). Algoritmo paralelo basado en agentes inteligentes aplicado al problema de secuenciación de trabajos en flujo uniforme. *Actas del IX Congreso de Ingeniería de Organización*. pp. 173-174.

Reeves, C.R; Eremeev, A.V. (2004). Statistical Analysis of Local Search Landscapes. *Journal of the Operational Research Society*, No. 55, pp. 687-693.

Taillard, E. (1993). Benchmarks for basic scheduling problems. *European Journal of Operational Research*, Vol. 64, pp. 278-285.

Verhoeven, M.G.A.; Aarts, E.H.L., (1995). Parallel Local Search. *Journal of heuristics*, 1: 43-65. Kluwer Academic Publishers.