Parametrización de esquemas algorítmicos paralelos para autooptimización

Francisco Almeida
Departamento de Estadística, Investigación Operativa y Computación, Universidad de La Laguna, falmeida@ull.es
Juan Manuel Beltrán, Murilo Boratto, Domingo Giménez
Departamento de Informática y Sistemas, Universidad de Murcia, jbeltran@um.es, mdc23108@alu.um.es, domingo@dif.um.es
Javier Cuenca
Departamento de Ingeniería y Tecnología de Computadores, Universidad de Murcia, javiercm@ditec.um.es
Luis Pedro García
Servicio de Apoyo a la Investigación Tecnológica, Universidad Politécnica de Cartagena, luis.garcia@sait.upct.es
Juan Pedro Martínez
Departamento de Estadística, Matemáticas e Informática, Universidad Miguel Hernández, Alicante, jp.martinez@uhm.es
Antonio M. Vidal
Departamento de Sistemas Informáticos y Computación, Universidad Politécnica de Valencia, avidal@dsic.upv.es

Resumen— **En este artículo se estudia la técnica de autooptimización de rutinas paralelas por medio de la parametrización del modelo de tiempo de ejecución. Se analizan las peculiaridades de esta técnica cuando se aplica a rutinas de distintos esquemas algorítmicos paralelos: esquemas iterativos, divide y vencerás y backtracking.**

Palabras clave— **Esquemas algorítmicos, autooptimización, programación paralela.**

I. Introducción

DURANTE los últimos años se han venido aplicando técnicas de autooptimización de rutinas paralelas con el fin de conseguir rutinas que se adapten automáticamente a las características del sistema de cómputo, reduciendo el periodo de tiempo necesario para tener rutinas optimizadas para un nuevo sistema, y que sean capaces de ejecutarse de manera eficiente independientemente de los conocimientos del usuario sobre programación paralela, ya que los principales usuarios potenciales de sistemas paralelos de alto rendimiento son científicos e ingenieros que tienen problemas de alto coste computacional pero sin conocimientos profundos de paralelismo. Las técnicas de autooptimización se han venido aplicando en diferentes campos [1], [2], [3], y especialmente a rutinas de álgebra lineal [4], [5], [6], que constituyen el elemento básico de cómputo en la resolución de muchos problemas científicos.

Una de las técnicas utilizadas para el desarrollo de rutinas con capacidad de autooptimización es la técnica de parametrización del modelo del tiempo de ejecución. En anteriores trabajos hemos estudiado la aplicación de la técnica a rutinas de álgebra lineal [7], y su aplicación a la mejora de una jerarquía de librerías de álgebra lineal con autooptimización [5], [8]. Los trabajos realizados se han centrado principalmente en sistemas homogéneos, pero se ha analizado también la posibilidad de adecuar la técnica a entornos heterogéneos [9], [10], [11].

Por otro lado, es importante resaltar que en la propuesta de desarrollo futuro de ScaLAPACK se incluye la posibilidad de utilizar parametrización para obtener rutinas que sean más fáciles de usar y que se ejecuten de manera más eficiente [12].

Más recientemente hemos empezado a trabajar en la aplicación a esquemas algorítmicos [13], [11], [14], [15]. En este trabajo resumimos las investigaciones que estamos llevando a cabo sobre técnicas de parametrización aplicadas a esquemas algorítmicos paralelos.

En la sección 2 se muestran las ideas generales de la parametrización de modelos de tiempos de ejecución de rutinas paralelas. En las secciones 3, 4 y 5 se analizan las peculiaridades de esta técnica cuando se aplica a distintos esquemas algorítmicos paralelos: esquemas iterativos, divide y vencerás y backtracking. Finalmente, en la sección 6 se plantean las líneas futuras de investigación.

II. Parametrización de modelos de tiempo de ejecución

Se pretende construir un modelo analítico del tiempo de ejecución de una rutina, de manera que este modelo sea una herramienta útil para decidir los valores de unos parámetros con los que se obtenga una ejecución eficiente, y con ello minimizar su tiempo de ejecución. Para esto, el modelo debe reflejar las características de cómputo y de comunicaciones de los algoritmos y del sistema sobre el que se ejecutará (hardware y software básico instalado). De este modo, el modelo analítico del tiempo de ejecución tendrá la forma:

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

donde s representa el tamaño de la entrada, SP (*System Parameters*) representa los parámetros que reflejan las características del sistema, y AP (*Algorith-*

mic Parameters) los que intervienen en el algoritmo y cuyos valores deben ser seleccionados adecuadamente para obtener un tiempo de ejecución reducido.

Parámetros SP típicos son: el coste de una operación aritmética, o de operaciones de distintos tipos o niveles (BLAS 1, 2 y 3 en rutinas de álgebra lineal), y los tiempos de inicio de las comunicaciones (start-up time, t_s) y de envío de un dato (word-sending time, t_w) en operaciones de comunicación. Con la utilización de estos parámetros se reflejan las características del sistema de cómputo y de comunicaciones, tanto físico (hardware) como lógico (las librerías que se utilizan para llevar a cabo las comunicaciones y operaciones de cómputo básicas).

Algunos parámetros AP típicos son: el número de procesadores a utilizar de entre todos los disponibles, y qué procesadores utilizar si el sistema es heterogéneo, el número de procesos a poner en marcha y su mapeo en el sistema físico, parámetros que identifiquen la topología lógica de los procesos (como puede ser el número de filas y columnas de procesos en un algoritmo para malla lógica 2D), el tamaño de los bloques de comunicación o de particionado de los datos entre los procesos, o el tamaño de los bloques de computación en algoritmos de álgebra lineal que trabajan por bloques. El valor de todos estos parámetros debe ser seleccionado para obtener tiempos de ejecución reducidos, y no podemos pretender que usuarios no expertos tengan suficiente conocimiento (del problema, del algoritmo y del sistema) como para realizar una selección satisfactoria, por lo que creemos que es conveniente utilizar técnicas de autooptimización.

De cara a obtener un modelo más realista, se puede tener en cuenta que los valores SP están influenciados por los de los AP. Así por ejemplo, los tiempos de inicio de comunicación y de envío de un dato pueden depender del tamaño de los bloques de comunicación, o de la topología y el número de procesadores, mientras que en algoritmos de álgebra lineal por bloques el coste de las operaciones aritméticas depende del tamaño del bloque de computación. Por esto, los SP se pueden expresar como una función del tamaño de la entrada y de los parámetros AP, y la fórmula 1 queda de la forma:

$$t(s) = f(s, AP, g(s, AP)) \tag{2}$$

Los valores SP se obtendrán en el momento de instalar la rutina (o la librería que la contenga) en un nuevo sistema. Para esto, el diseñador de la rutina debe haber desarrollado el modelo del tiempo de ejecución, haber identificado los parámetros SP que intervienen en el modelo, y haber diseñado una estrategia de instalación, que incluye para cada SP los experimentos a realizar para su estimación y los parámetros AP y sus valores con los que hay que experimentar. Los valores obtenidos para los SP se incluyen junto con el modelo del tiempo de ejecución en la rutina que se está optimizando, que se instala de esta forma con información del sistema para el que se está optimizando.

En tiempo de ejecución, para un tamaño de la entrada concreto, la rutina obtiene de manera automática valores de los AP con los que se obtiene una ejecución óptima según el modelo de ejecución de la rutina y los valores de los SP obtenidos en la instalación. El tiempo de obtención de estos valores debe ser reducido debido a que incrementa el tiempo de ejecución de la rutina.

III. Esquema iterativo

El esquema paralelo iterativo que consideramos tiene la forma:

```
In Each Processor j; 1 ≤ j ≤ P:
   for i = 1 to number_of_iterations
      computation in each processor with
             the block it has assigned
      communication step
   endfor
endInEachProcessor
```

Se pueden utilizar esquemas iterativos en la solución de muchos problemas [16], [17]: programación dinámica, algoritmo de Dijkstra, algoritmos genéticos, multiplicación de matrices, relajación de Jacobi, resolución de sistemas de ecuaciones, etc. Se ha estudiado la aplicación de un esquema iterativo en la solución por programación dinámica del problema de la mochila y del de la devolución de monedas, pero la metodología podría aplicarse a otros problemas y esquemas iterativos. La solución del problema se obtiene completando una tabla bidimensional donde las columnas representan el tamaño de la entrada y las filas el número de decisiones. Cada entrada en la tabla contiene la solución óptima de un subproblema, con tamaño representado por la columna y número de decisiones indicado por la fila.

Consideramos el caso en que los datos se asignan a los procesos por bloques contiguos de igual tamaño. Cada iteración consiste de una parte computacional seguida de otra de comunicaciones entre pasos consecutivos. De esta forma, el tiempo de ejecución es la suma de los tiempos de ejecución de las diferentes iteraciones, y basta con obtener el tiempo de una única iteración, que tendrá la forma:

$$t(s, D) = t_c t_{comp}(s, D) + t_{comm}(s, D) \tag{3}$$

donde s representa el tamaño de la entrada, D el número de procesos usados en la solución del problema, t_c el coste de una operación aritmética básica, t_{comp} el número de operaciones aritméticas básicas, y t_{comm} el coste de las comunicaciones, que contendrá un término con t_s y otro con t_w (start-up y word-sending time).

Para analizar el comportamiento del método elegimos el problema de las monedas. Inicialmente, hay que determinar los parámetros que intervienen en el modelo. El tamaño del problema viene determinado por el número de monedas a devolver (C) y el número de valores diferentes de monedas (n). Las monedas serán de unos ciertos valores ($v_1, v_2, \ldots, v_n$). En

cuanto a la cantidad de monedas de cada valor, da lugar a distintas versiones del problema: se puede considerar un número ilimitado de monedas de cada tipo, una única moneda de cada tipo (este problema sería equivalente al de la mochila 0/1), o una cantidad determinada de monedas de cada tipo $(q_1, q_2, \ldots, q_n)$. Los vectores que indican los valores de las monedas y las cantidades de cada tipo también son parte del tamaño de la entrada.

Los parámetros del sistema serán t_c, t_s y t_w. El único parámetro algorítmico cuyo valor hay que determinar es el número de procesos a usar (D).

Una vez determinados los parámetros que influyen en el tiempo de ejecución, se obtiene un modelo analítico de éste. Con el esquema del problema de las monedas con una moneda de cada tipo quedaría:

$$\frac{n}{p}t_c + (p-1)\left(t_s + \frac{n}{p}t_w\right) \qquad (4)$$

Se puede ver que el coste de las comunicaciones y las computaciones tiene el mismo orden, y no se obtendrá ventaja del uso del paralelismo, pero estamos interesados en el estudio del esquema, y no del problema concreto, por lo que hemos estudiado su comportamiento variando la granularidad de la computación.

Los valores de los parámetros del sistema se obtienen en el momento de la instalación, ejecutando una versión reducida de la rutina o de la parte de la rutina en que aparecen los parámetros. t_s y t_w se pueden obtener utilizando un `ping-pong`, pero esto puede dar lugar a valores que no reflejan bien el comportamiento de la rutina, pues en ella se llevan a cabo comunicaciones simultáneas. Otra posibilidad es obtener el valor de estos parámetros midiéndolos cuando se llevan a cabo estas operaciones simultáneas. También es posible utilizar una operación colectiva (`scatter`) para las comunicaciones, y estimar el coste de esta rutina en nuestro sistema por medio de un ajuste por mínimos cuadrados.

t_c se puede estimar resolviendo en secuencial una versión reducida del problema.

Una vez obtenidos los valores de los parámetros del sistema, se incluyen junto con el modelo en la rutina que se está instalando, y se instala ésta, posiblemente compilándola con el código que se le ha añadido.

En tiempo de ejecución, dada una entrada, se decide el valor de los parámetros algorítmicos (en este caso sólo el número de procesadores) para resolver el problema con un tiempo de ejecución cercano al óptimo. Para esto, la rutina sustituye en el modelo teórico del tiempo posibles valores del número de procesadores, y se ejecuta con el valor con que se obtiene el menor tiempo teórico de ejecución.

En la tabla I se resumen algunos resultados variando la granularidad y el tamaño de la entrada. Se muestra el cociente entre el menor tiempo de ejecución obtenido variando el número de procesadores y el obtenido con el número de procesadores que decide nuestro método. Se muestran resultados estimando t_s y t_w con `ping-pong` (pp) y con una rutina de comunicaciones simultáneas (cs). Se comprueba que los tiempos que se obtienen con el método propuesto no están lejos de los mínimos, especialmente cuando se utiliza una rutina con comunicaciones simultáneas para estimar los parámetros de las comunicaciones. La versión del problema utilizada es la de un número de monedas determinado de cada tipo.

TABLA I
COCIENTE ENTRE EL TIEMPO DE EJECUCIÓN OBTENIDO CON LOS PARÁMETROS SELECCIONADOS CON EL MODELO Y EL MENOR TIEMPO DE EJECUCIÓN. CON EL ESQUEMA DEL PROBLEMA DE LA DEVOLUCIÓN DE MONEDAS, CON UN NÚMERO DE MONEDAS DETERMINADO DE CADA TIPO. SE HA VARIADO EL TAMAÑO DE LA ENTRADA Y DE LA COMPUTACIÓN.

tamaño:	100		500	
granularidad	pp	cs	pp	cs
10	1	1	1	1
50	1.04	1	1.25	1
100	1.23	1.6	1.20	1

IV. Esquema divide y vencerás

Con un esquema divide y vencerás se resuelve un problema por medio de la solución de problemas del mismo tipo pero de menor tamaño, y combinando las soluciones de estos subproblemas se obtiene la del problema original. Se puede utilizar un esquema recursivo:

```
DV(p:problema,n:tamaño):solución
   if n ≤ b
      devolver solución de p
             por método directo
   en otro caso
      dividir(p,p1,p2,...,pk)
      for  i = 1 to k
         si=DV(pi,ni)
      endfor
      devolver combinar(s1,s2,...,sk)
   endif
```

donde un problema p de tamaño n se divide en k subproblemas p_i de tamaños n_i. Cuando se llega a un cierto tamaño base (b) acaba la recursión resolviéndose el problema por un método directo. A la vuelta de la recursión se combinan los resultados de los subproblemas para obtener la solución del problema.

Si el tiempo de ejecución del método directo es $f(n)$ y el de dividir un problema en subproblemas y combinar los subproblemas es $g(n)$, y si cada problema se divide en k subproblemas del mismo tamaño $\frac{n}{k}$, el tiempo de ejecución del método divide y vencerás es $t(n) = \frac{n}{b}f(b) + \sum_{i=0}^{m-1} g\left(\frac{n}{k^i}\right)k^i$, con $\frac{n}{k^m} = b$. En este esquema secuencial tenemos como parámetros del sistema los valores que nos aparecen en $f(n)$ y $g(n)$, que se pueden obtener ejecutando estas rutinas en el momento de la insta-

lación. Los parámetros algorítmicos serán el tamaño del caso base de la recursión y, si se dispone de varios médodos directos para resolver los problemas de tamaño reducido, el método directo que se utiliza.

Por ejemplo, en un esquema de mergesort, podemos utilizar como métodos directos el método de la burbuja, de inserción, selección, etc. Para cada uno de ellos tendremos $f(n) = k_{met}n^2$, y el valor de k_{met} se obtiene al instalar la rutina, realizando ejecuciones para varios tamaños y haciendo un ajuste por mínimos cuadrados por una parábola. En la instalación también se obtienen las constantes de $g(n) = k_1 n + k_2$ correspondiente a la mezcla.

El modelo del tiempo, junto con los parámetros obtenidos para el sistema en el que se instala la rutina, se incorporan a ésta previamente a su instalación.

En tiempo de ejecución, la rutina obtendrá, a partir del modelo, el método directo y el tamaño del caso base con que se tiene menor tiempo de ejecución teórico, y se ejecutará con esos valores.

En la tabla II se compara el tiempo de ejecución con los parámetros seleccionados por el método explicado con el menor tiempo de ejecución variando el tamaño del caso base y el método directo (se ha usado burbuja, selección e inserción). Se comprueba que la diferencia entre los dos tiempos es mínima, con lo que la técnica de selección de los parámetros da resultados satisfactorios.

TABLA II

Comparación entre el tiempo de ejecución obtenido con los parámetros seleccionados con el modelo y el menor tiempo de ejecución, variando el volumen de datos, para la ordenación por mezcla.

tamaño	tie. min.	tie. mod.	desv. %
2 mill.	0.38	0.40	5
4 mill.	0.81	0.84	3
6 mill.	1.28	1.31	2
8 mill.	1.67	1.76	5
10 mill.	2.14	2.23	4

El método se usa también con éxito con otros algoritmos, como por ejemplo el de multiplicaciıón rápida de Strassen. En este caso cada problema se divide en 7 subproblemas de tamaño una cuarta parte del problema original, el método directo es una multiplicación de matrices tradicional, y la descomposición del problema en subproblemas y la combinación de los resultados se hace por medio de sumas y restas, de coste $O\left(n^2\right)$.

En la tabla III se muestra, para el caso secuencial y en un procesador de un sistema HPC160, el menor tiempo de ejecución obtenido variando el nivel de recursión y el obtenido con el nivel proporcionado por el modelo. También aparecen los niveles con los que se obtienen estos tiempos, y el tiempo obtenido con la rutina `dgemm` de BLAS [18]. Se comprueba que a partir del modelo se obtienen buenas decisiones, con lo que se alcanzan tiempos cercanos a los mínimos.

TABLA III

Comparación entre el tiempo de ejecución obtenido con los parámetros seleccionados con el modelo y el menor tiempo de ejecución, variando el tamaño del problema, para la multiplicación de Strassen secuencial.

	Mínimo		Modelo		dgemm
tamaño	tie.	niv.	tie.	niv.	
512	0.17	1	0.17	1	0.16
1024	1.21	1	1.21	1	1.24
2048	8.48	3	8.68	2	9.85
4096	59.08	4	60.64	3	83.28

Un método paralelo (se pueden considerar otros esquemas) podría trabajar con los datos en un procesador inicial P_0, y dividiendo el problema hasta llegar a obtener problemas para el número de procesadores que va a intervenir en la computación. Cada problema que no se sigue dividiendo en P_0 se envía a un procesador. En la fase de combinación, los procesadores que actúan como esclavos envían a P_0 la solución de sus subproblemas, y éste combina las subsoluciones.

En este caso intervienen también los parámetros del sistema correspondientes a las comunicaciones, y en los parámetros algorítmicos hay que decidir el número de procesadores que se utilizan, además del método directo y el tamaño del caso base.

En la tabla IV se muestra, para el caso paralelo usando cuatro procesadores de un sistema HPC160, el menor tiempo de ejecución variando el número de procesadores y el obtenido para nivel de recursión uno y con el número de procesos esclavo seleccionados por el modelo. También aparece el número de procesadores con que se obtienen estos tiempos.

TABLA IV

Comparación entre el tiempo de ejecución obtenido con los parámetros seleccionados con el modelo y el menor tiempo de ejecución, variando el tamaño del problema, para la multiplicación paralela de Strassen de nivel uno.

	Mínimo		Modelo	
tamaño	tie.	pro.	tie.	pro.
512	0.08	3	0.08	3
1024	0.53	4	0.55	3
2048	3.35	4	3.35	4
4096	23.21	4	23.21	4

V. Esquemas de exploración de árboles de soluciones

En la solución de problemas por recorrido de un árbol de soluciones se recorre un árbol lógico donde cada nodo representa una posible solución. El coste de ejecución depende del número de nodos recorridos y del coste del recorrido de cada nodo. El coste del recorrido puede ser calculado normalmente, pero

es difícil hacer una estimación del número de nodos que se recorren. El recorrido se puede hacer con técnicas como `branch and bound` o `backtracking` ([16], [17]). En la actualidad estamos estudiando esquemas de `backtracking`.

Para recorrer un árbol con n niveles y donde cada nodo no terminal tenga h nodos hijos, consideramos un esquema paralelo del tipo maestro-esclavo. El proceso maestro genera todos los nodos hasta un nivel l (no seguimos, por tanto, el mismo orden de recorrido que un backtracking secuencial), con lo que si no ha habido poda recorre $\frac{h^{l+1}-1}{h-1}$ y genera h^l problemas. Estos problemas serán asignados a los p procesos esclavos, con lo que cada uno de ellos recorrerá si no se realizan podas un total de $\frac{h^{n+1}-h^{l-1}}{p(h-1)}$ nodos. Si cada procesador realiza podas con la información correspondiente a los nodos que evalúa, el porcentaje de nodos podados puede ser pequeño en comparación con el que se obtendría con información global. Esto hace que pueda ser conveniente transferir información de un esclavo al maestro cada cierto número de nodos explorados (e), y el maestro la envía a cada esclavo cuando éste realiza sus envíos. De esta forma tendremos un porcentaje de nodos recorridos (k) que depende del problema y de los valores concretos de entrada, por lo que habrá que estimarlo en tiempo de ejecución.

El modelo del tiempo de ejecución será:

$$p\left(t_s + t_w m_1\right) + k\frac{h^{n+1} - h^{l+1}}{p(h-1)}\left(t_c + \frac{2t_s + t_w m_2}{e}\right) \tag{5}$$

donde m_1 y m_2 son los tamaños de los mensajes iniciales y de comunicación de información. Los parámetros del sistema son t_c, t_s y t_w, y se obtienen, como siempre, en tiempo de instalación resolviendo en secuencial un problema de pequeña dimensión y con un `ping-pong`. Hay tres parámetros algorítmicos (l, e y p) cuyo valor hay que determinar en tiempo de ejecución.

La principal dificultad es la estimación de k, que interviene en el modelo analítico del tiempo pero que no es parte del tamaño de la entrada, ni de los parámetros del sistema ni algorítmicos, sino que su valor depende de la entrada concreta del problema que se quiera resolver. Por ejemplo, si consideramos el problema de la mochila 0/1, podemos recorrer todo el árbol sin realizar podas, pero también se puede realizar podas utilizando los pesos de los objetos, o utilizando los pesos y los beneficios. Cuanto más elaborada sea la poda más nodos se eliminarán y el tiempo de ejecución será menor, pero será más difícil realizar una buena estimación de k en un tiempo de ejecución reducido, lo que es necesario al estimarse en tiempo de ejecución. Además, el valor de k depende de l, e y p ($k(l, e, p)$). Si aumentamos e hay menos compartición de la información, con lo que las podas que cada proceso realiza en la zona del árbol que explora serán menos efectivas. Al aumentar el valor de l cada proceso trabaja con una zona del árbol más pequeña, por lo que las podas serán menos efectivas. Al aumentar p, cada proceso recibe información de más zonas del árbol, con lo que puede que las podas sean más efectivas.

Normalmente el valor de l será mucho menor que el de n, con lo que para minimizar el tiempo dado por 5 basta con minimizar

$$k(l, e, p)\left(t_c + \frac{2t_s + t_w m_2}{e}\right) \tag{6}$$

Se está experimentando con distintos métodos de estimación de k. El más simple es asignarle un valor constante, con lo que las decisiones que se tomen no deberán ser muy buenas pero aún así pueden ser preferibles a dejar que el usuario seleccione los parámetros según su criterio. Otra posibilidad es resolver en tiempo de ejecución un problema reducido a partir del problema de entrada, y tomar como estimación de k para el problema de entrada el obtenido con el problema reducido. Con el ejemplo del problema de la mochila, se podría formar un subproblema con la mitad de los objetos tomando sólo los impares, y con capacidad de la mochila la mitad, o se podría considerar una mochila con la misma capacidad y cada objeto tomado con la suma de los pesos y los beneficios de dos objetos del problema inicial. Una tercera posibilidad consiste en generar aleatoriamente secuencias de decisiones, considerar el orden en que se recorrerían en el método de `backtracking`, calcular el porcentaje de nodos que se recorren de entre los que intervienen en esas soluciones (el nivel medio al que se llega), y tomar este valor como k en el problema original.

En la tabla V se muestra la diferencia relativa entre el valor de k real y estimado ($\frac{|k_e - k_r|}{k_r}$), en el problema de la mochila con poda por peso y cuando se utiliza el método de estimación de generación de un problema similar de menor tamaño (prob.) y cuando se genera un conjunto de soluciones (conj.). Se comprueba que con el método de generación de un subproblema se estima mejor el valor de k. Pero este método es dependiente del problema, mientras que el de generación de soluciones no.

TABLA V
ESTIMACIÓN DEL PARÁMETRO k EN LA SOLUCIÓN DEL PROBLEMA DE LA MOCHILA 0/1 POR `backtracking`, REALIZANDO PODAS POR PESO Y CUANDO SE GENERA UN PROBLEMA DE MENOR TAMAÑO (PROB.) Y UN CONJUNTO DE SOLUCIONES (CONJ.).

tamaño	prob.	conj.
20	0.18	0.51
25	0.08	0.41
30	0.08	0.29

En el caso de poda por peso y beneficio la estimación de k es mucho peor, pero la diferencia relativa entre el menor tiempo de ejecución variando los parámetros (mte) y el tiempo experimental obtenido con los parámetros proporcionados por el modelo (epm) podemos considerarla satisfactoria si tenemos

en cuenta que el no elegir los parámetros adecuadamente nos llega a dar ejecuciones con tiempos mucho mayores que el mínimo. En la tabla VI se muestra la diferencia relativa entre los dos tiempos ($\frac{|mte-epm|}{mte}$), cuando se realiza la poda por peso (peso) y por peso y beneficio (pe+be).

TABLA VI
COMPARACIÓN DEL MENOR TIEMPO EXPERIMENTAL DE EJECUCIÓN Y EL TIEMPO CON LOS PARÁMETROS PROPORCIONADOS POR EL MODELO, EN LA SOLUCIÓN DEL PROBLEMA DE LA MOCHILA 0/1 POR `backtracking`, REALIZANDO PODAS POR PESO (PESO) Y POR PESO Y BENEFICIO (PE+BE).

tamaño	peso	pe+be
20	0.38	0.03
25	0.23	0.20
30	0.00	0.03

Otra posibilidad es combinar información obtenida en tiempo de desarrollo de la rutina con otra obtenida en tiempo de ejecución. Durante el desarrollo se puede obtener el valor de k real (k_r) y estimado (k_e) para distintos tamaños de entrada. A partir de ellos se obtiene el cociente $\frac{k_r}{k_e}$ como una constante o como una función de n que podría ser ajustada por mínimos cuadrados. En el instante de la ejecución se estima k con alguno de los métodos considerados, que además será el mismo utilizado en la instalación. Finalmente se actualizará k multiplicándolo por el cociente $\frac{k_r}{k_e}$ calculado con experimentación con la rutina.

VI. Trabajos futuros

En este trabajo se resume la investigación que se está realizando sobre aplicación de técnicas de autooptimización a esquemas algorítmicos paralelos. En la actualidad se está trabajando con algunos esquemas (iterativos, divide y vencerás y backtracking) estudiando por medio de problemas básicos y típicos (de la mochila y de la devolución de monedas, ordenación por mezcla y multiplicación rápida de Strassen) las características específicas de la aplicación a los distintos esquemas de la técnica de autooptimización basada en parametrización del modelo de tiempo de ejecución. La idea es obtener conocimiento suficiente como para poder diseñar una metodología de autooptimización general para cada uno de los esquemas. Las técnicas de optimización se podrían combinar con el desarrollo de esqueletos paralelos ([19], [20]), de manera que el científico que necesite resolver un problema en paralelo pueda programar en secuencial funciones particulares del esqueleto, y la ejecución paralela sería transparente al usuario, al no serle necesario programar en paralelo y al no ser necesario que tenga conocimientos de paralelismo para poder obtener tiempos de ejecución reducidos.

Agradecimientos

Este trabajo ha sido subvencionado en parte por el proyecto MCYT y FEDER de código TIC2003-08238-C02-02.

Referencias

[1] E. A. Brewer, *Portable High-Performance Supercomputing: High-Level Platform-Dependent Optimization*, Ph. D. Thesis, MIT, 1994.

[2] M. Frigo, *FFTW: An Adaptive Software Architecture for the FFT*, in: Proc. ICASSP Conf., V 3, 1998, pp 1381.

[3] Ahmad Faraj and Xin Yuan, *Automatic Generation and Tuning of MPI Collective Communication Routines*, 19th ACM Int. Conf. on Supercomputing, Cambridge, Massachusetts, June 20-22, 2005.

[4] Z. Chen, J. Dongarra, P. Luszczek and K. Roche, *Self Adapting Software for Numerical Linear Algebra and LAPACK for Clusters*, Parallel Computing, 29 (11-12), 2003, pp 1723-1743.

[5] J. Cuenca, D. Giménez and J. González, *Architecture of an Automatic Tuned Linear Algebra Library*, Parallel Computing, 30 (2), 2004, pp 187-220.

[6] T. Katagiri, K. Kise and H. Honda, *Effect of Auto-tuning with User's Knowledge for Numerical Software*, in: S. Vassiliadis, J.L. Gaudiot and V. Piuri, eds., Proc. First Conf. on Computing Frontiers, Ischia, Italy, 2004, pp 12-25.

[7] J. Cuenca, D. Giménez and J. González, *Modeling the Behaviour of Linear Algebra Algorithms with Message-passing*, 9th EUROMICRO Workshop on Par. and Dist. Proc. PDP, February 7-9, 2001, Mantova, Italy, pp 282-289.

[8] P. Alberti, P. Alonso, A. M. Vidal, J. Cuenca and D. Giménez, *Designing polylibraries to speed up linear algebra computations*, Int. Jour. of High Performance Computing and Networking, V 1, N 1/2/3, 2004, pp 75-84.

[9] J. Cuenca, D. Giménez, J. González, J. Dongarra and K. Roche, *Automatic Optimisation of Parallel Linear Algebra Routines in Systems with Variable Load*, 11th EUROMICRO Workshop on Par. Dist. and Net. Proc. PDP, February 5-7, 2003, Genova, Italy, pp 409-416.

[10] J. Cuenca, L. P. García, D. Giménez and J. Dongarra, *Processes Distribution of Homogeneous Parallel Linear Algebra Routines on Heterogeneous Clusters*, in Proc. IEEE Int. Conf. on Cluster Computing, IEEE, HeteroPar05, 27-30 September 2005, Boston, Massachusetts.

[11] J. Cuenca, D. Giménez and J. P. Martínez, *Heuristics for work distribution of a homogeneous parallel dynamic programming scheme on heterogeneous systems*, Parallel Computing, 31, 2005, pp 771-735.

[12] J. Demmel and J. Dongarra, *LAPACK 2005 Prospectus: Reliable and Scalable Software for Linear Algebra Computations on High End Computers*, LAPACK Working Note 164, January, 2005.

[13] D. Giménez and J. P. Martínez, *Automatic Optimization in Parallel Dynamic Programming Schemes*, in VECPAR04, 28-30 June 2004, Valencia.

[14] J. P. Martínez, F. Almeida and D. Giménez, *Mapping in heterogeneous systems with heuristical methods*, Workshop on state-of-the-art in Sci. Par. Comp., Umeå, Sweden, June 18-21, 2006.

[15] M. Boratto, D. Giménez and A. M. Vidal, *Automatic Parametrization on Divide-and-Conquer Algorithms*, Int. Cong. Math., 22-30 August 2006, Madrid.

[16] T. H. Cormen, C. E. Leiserson, R. L. Rivest and C. Stein, *Introduction to Algorithms, 2nd edition*, The MIT Press, 2001.

[17] D. Giménez, J. Cercera, G. García y N. Marín, *Algoritmos y Estructuras de Datos, Vol II, Algoritmos*, Diego Marín, 2003.

[18] J. J. Dongarra, J. Du Croz, I. S. Duff and S.Hammarling, *A set of Level 3 Basic Linear Algebra Subprograms*, ACM Trans. Math. Soft., 16, 1990, pp 1-17.

[19] R. Čiegis and M. Baravykaite, *Implementation of Parallel Generalized Branch and Bound Template*, Workshop on state-of-the-art in Sci. Par. Comp., Umeå, Sweden, June 18-21, 2006.

[20] O. González, C. León and A. Rojas, *MPI and OpenMP skeletons for the Divide-and-Conquer Technique*, Workshop on state-of-the-art in Sci. Par. Comp., Umeå, Sweden, June 18-21, 2006.