Modelado y autooptimización en esquemas paralelos de *backtracking*

Manuel Quesada y Domingo Giménez[1]

Resumen— **En este trabajo se estudia el modelado del tiempo de ejecución de esquemas paralelos de *backtracking*. Se estudian diferentes esquemas de programación identificando parámetros que influyen en el tiempo de ejecución. Se propone una metodología de optimización basada en la selección automática de parámetros para obtener ejecuciones con un tiempo reducido. Se estudian las propiedades que debería tener un esquema algorítmico secuencial de *backtracking* para poder realizar internamente la paralelización. De esta manera se pretende abstraer a los usuarios finales de las complejidades de la programación paralela.**

Palabras clave— **autooptimización, programación paralela, recorridos de árboles, esquema algorítmico de *backtracking*.**

I. Introducción

CADA vez es más habitual la aparición de máquinas multiprocesadores capaces de acelerar la resolución de problemas computacionales de alto coste. En los últimos dos o tres años, estas máquinas están adquiriendo mayor importancia con la aparición de los *multicores* en los ordenadores personales. Por este motivo, este campo ya no solo queda reservado a grupos reducidos de usuarios y se extiende a todos aquellos que utilizan un *PC*. Sin embargo, las rutinas paralelas traen consigo parámetros que deberemos seleccionar correctamente para minimizar el tiempo de ejecución: número de procesadores a utilizar, número de hilos, geometría de la malla de procesos, tamaños de los bloques de comunicación... [1] Seleccionar una configuración adecuada para estos parámetros no es una tarea sencilla. Incluso un usuario experto en paralelismo sería incapaz, a primera vista, de dar con la configuración óptima o simplemente adecuada para ejecutar las rutinas en máquinas totalmente distintas. Por este motivo, para que las rutinas puedan ser usadas de manera eficiente por los científicos que necesitan de paralelismo pero no son expertos en computación paralela, es necesario que el proceso de selección de parámetros sea automático y transparente al usuario, y si es posible que incluso la obtención de un programa paralelo se realice de forma automática a partir de código secuencial.

Desde hace algunos años se han desarrollado 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. Entre otros campos, las técnicas de autooptimización se han venido aplicando en problemas de álgebra lineal [2, 3, 4], trabajos sobre las transformadas de Fourier [5], sistemas de altas prestaciones [6]...

Una de las técnicas para el desarrollo de rutinas con capacidad de autooptimización se basa en la parametrización del modelo de tiempo de ejecución. Esta técnica se ha aplicado a rutinas de álgebra lineal con paso de mensajes [7], a la mejora de una jerarquía de librerías de álgebra lineal con autooptimización [3], al diseño de polilibrerías para acelerar la computación en álgebra linear [8] y se ha analizado la posibilidad de adecuarla a entornos heterogéneos [9, 10]. Además, 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 [11].

Más recientemente se ha empezado a trabajar en la aplicación de este tipo de técnicas a esquemas algorítmicos paralelos [1], por ejemplo en algoritmos de divide y vencerás [12], de programación dinámica [13] y en recorrido de árboles de soluciones [14]. También se trabaja en la aplicación de heurísticas en el mapeo de tareas sobre sistemas heterogéneos [13, 15, 16].

Sería interesante incorporar estás técnicas de autooptimización a esqueletos algorítmicos paralelos. De esta manera se simplificará al usuario la programación, abstrayéndole tanto del paralelismo como de la optimización. Los usuarios solo tendrían que programar en secuencial los fragmentos del esquema necesarios. Internamente se paralelizaría el problema eligiendo los mejores parámetros para su ejecución. Hay multitud de trabajos sobre diseño de esqueletos paralelos [17], algunos de ellos dedicados a técnicas de recorrido de árboles [18, 19].

En este documento se resume el trabajo [20] donde se estudian los mecanismos de autooptimización en esquemas paralelos de *backtracking*. En los esquemas de recorrido de árboles de soluciones se presenta la dificultad adicional de estimar el número de nodos generados, lo que depende no solo del algoritmo y el tamaño del problema, sino también de la entrada particular a resolver.

El contenido de este documento se estructura en una primera parte donde se estudia el modelado de los recorridos secuenciales, así como los resultados experimentales (Sección II). En la segunda parte nos centramos en esquemas paralelos (Sección III), su modelado (Sección IV) y los experimentos realizados (Sección V). Para finalizar se resumen las principales conclusiones y los trabajos futuros (Sección VI).

[1]Dpto. de Informática y Sistemas, Univ. Murcia, e-mail: manuel.quesada@alu.um.es y domingo@dif.um.es

II. Recorrido secuencial de árboles por medio de *backtracking*

A. Modelado de los recorridos secuenciales

Continuando con los trabajos sobre el modelado de rutinas realizados en [1], se utilizará un modelo del tiempo de ejecución que refleje las características de cómputo y de comunicaciones del sistema sobre el que se ejecutará (*hardware* y *software* básico instalado), y del algoritmo estudiado. Se parte de un modelo analítico del tiempo de la forma:

$$t(s) = f(s, AP, SP) \quad (1)$$

Donde s representa el tamaño y tipo de la entrada del problema. Al segundo parámetro lo hemos denominado *parámetros del sistema* (SPs) y serán específicos de cada uno de los sistemas computacionales donde deseamos ejecutar nuestra rutina, por lo que deberán ser calculados para cada nuevo sistema donde se instale. El tercer parámetro corresponde con los *parámetros del algoritmo* (APs) y son los que sería labor del usuario de nuestra rutina ajustar, para cada una de las máquinas donde se ejecute y cada una de las entradas a resolver.

Normalmente los valores de los parámetros del sistema estarán influenciados por los parámetros del algoritmo. Por esto, los parámetros del sistema se pueden expresar como una función de las propiedades del problema y de los parámetros del algoritmo:

$$t(s) = f(s, AP, g(s, AP)) \quad (2)$$

En este trabajo se utiliza esta metodología de parametrización para modelar el tiempo de ejecución de la técnica de *backtracking*. Se diseña una metodología independiente del tipo de recorrido que realicemos (búsqueda de una solución u optimización), la representación del árbol... [21] Aunque no todos los esquemas de *backtracking* realizan podas sobre el espacio de búsqueda, se modela este tipo de recorridos que es el más general. Al producirse podas en el espacio de búsqueda no se conoce el porcentaje exacto de nodos recorridos. El número de nodos podados es dependiente de la forma y recorrido del árbol, así como la propia naturaleza de las entradas. Para modelar este comportamiento se introduce en el modelo un parámetro que representa el porcentaje de nodos no podados para una determinada ejecución. A este porcentaje lo llamamos k, con $\{k \epsilon R \mid 0 \leq k \leq 1\}$. Teóricamente y para cada una de las entradas el valor de k es $1 - \frac{NP}{NT}$. Donde NP es el número total de nodos podados en el recorrido y NT es el número total de nodos del recorrido. El problema es que estos datos no son conocidos hasta que no se resuelve el problema.

Se modela el tiempo de ejecución de un recorrido secuencial utilizando la técnica de *backtracking* como:

$$t(n) = k \cdot NNG \cdot TCN \quad (3)$$

Donde *t(n)* representa el tiempo de ejecución para una entrada de tamaño n. La variable NNG equivale al número de nodos generados y dependerá del tipo de árbol que genere el problema al que nos enfrentamos. En este trabajo se van a considerar árboles de l niveles donde cada uno de los nodos generará h nodos hijos, siendo NNG totales del árbol $\frac{h^{l+1}-1}{h-1}$. TCN es el tiempo de cómputo de un nodo.

Se podrían utilizar otros modelos del tiempo de ejecución, pero presentarían la misma problemática. Lo que nos interesa es estudiar una metodología general para estimar los parámetros, y la que aquí se utiliza para este modelo particular podría extenderse a otros modelos.

B. Estrategias de estimación de los parámetros

Una vez se dispone del modelo del tiempo de ejecución, se deben programar estrategias para la estimación de los parámetros que lo componen. Esta estimación deberá realizarse antes de ejecutar nuestra rutina y sin que la decisión que debemos tomar nos lleve un tiempo excesivo en el momento de la ejecución, ya que supone un tiempo adicional al de resolver el problema.

Se propone una metodología de trabajo dividida en una fase de instalación y una fase de ejecución. Durante la fase de instalación se realizan experimentos para extraer información general del problema, para ello existe una rutina de instalación que será configurada por un *manager*. Esta configuración (número de experimentos, tamaño de las entradas, información a registrar, tiempo máximo de instalación...) dependerá de los conocimientos que tenga el *manager* sobre el tipo de entradas a resolver. A partir de esta configuración se irán generando y resolviendo entradas aleatorias del problema; para ello la metodología exige que se programe, además del programa que resuelve el problema, un generador de entradas aleatorias configurable. En la fase de ejecución se completará esta información general con la extraída de la entrada concreta a resolver.

Respecto a la estimación de TCN, en ocasiones será el mismo para todos los nodos del árbol, independientemente del nivel. Sin embargo, este tiempo puede ser más complejo de estimar: depender de la operación de generación de los nodos, de la operación de poda, del nivel actual... En nuestros experimentos se calcula como el cociente entre lo que tarda en ejecutarse la rutina y el número de nodos generados.

En cuanto a la estimación de k, se ha experimentado con tres posibles estrategias para estimarla a partir de los datos obtenidos en la fase de instalación:

- *EstMedia:* se realizan pruebas durante la instalación para distintos tamaños y se obtiene una media del índice de poda de todas las ejecuciones. Será el mismo valor para todas las entradas.
- *EstRecta:* se considera que el valor del índice de poda dependerá de los tamaños de la entrada. Almacenaremos valores de los índices de poda agrupados para los distintos tamaños en tiempo de instalación, y extrapolaremos por medio de una recta para nuevos tamaños que recibamos

en tiempo de ejecución.
- *EstFuncion:* se aproximan los datos obtenidos en la instalación a una función por medio de un ajuste por mínimos cuadrados. Por defecto se aproxima a la función a/x pero el *manager* puede definir nuevas.

El principal problema de todas estas aproximaciones es que no incluimos la información de las entradas que deseamos resolver. Esta información es interesante de incorporar cuando las entradas marcan de forma significativa el recorrido, y esto es lo que ocurre en recorridos de *backtracking*. Para cubrir estos casos se completa la información procedente de la instalación con otra extraída directamente en tiempo de ejecución, y se pondera cada una de ellas con un cierto factor: $k_{media} = f_1 \cdot k_{Ins} + f_2 \cdot k_{Ejec}$. Los coeficientes de peso pueden ser ajustados por el *manager* según las características del problema. En nuestros experimentos los hemos considerado $f_1 = 0.5$ y $f_2 = 0.5$.

Se proponen dos alternativas para calcular k_{Ejec}: agrupar valores contiguos de las entradas o seleccionar valores aleatoriamente procedentes de la entrada original. La filosofía de ambas es conseguir a partir de la entrada concreta, entradas más pequeñas donde el tiempo que empleamos a resolverlas es despreciable frente al tiempo total de ejecución. Por ejemplo, una entrada de tamaño 100 podríamos reducirla a tamaño 20 de modo que se ejecute en menos tiempo, y conserve las propiedades de la original.

En la Tabla I se muestran los valores de k estimados en los distintos casos. Se implementa una solución que resuelve el problema de la Mochila 0/1 ordenando los objetos de mayor a menor beneficio/peso, intentando incluir primero objetos en la mochila, y eliminando nodos cuyo peso exceda el de la mochila o que sumando al beneficio el beneficio obtenido con un avance rápido no se pueda mejorar el beneficio actual. Se experimenta con tres entradas aleatorias de tamaños 30, 35 y 40. Estos tamaños de entrada son mayores que los que hemos configurado en la fase de instalación para estudiar como extrapola nuestra técnica (generamos 50 pruebas para cada tamaño en el intervalo [5,25] incrementando de 5 en 5). Para cada entrada se obtiene el valor de k real (k_{Real}), el valor de k estimado por el método *EstMed* (k_M), el valor de k estimado extrapolando valores de instalación por medio de una recta (k_R) y utilizando este último método pero añadiendo información de la ejecución agrupando la entrada original (k_{RE}). Como última columna se añade el valor de k utilizando únicamente información de la ejecución (k_E).

Se puede concluir de la Tabla I, que para este tipo de técnicas donde existe alta dependencia de las entradas, merece la pena incluir algún tipo de información de la ejecución que modifique la que extraemos durante el proceso de instalación. En la Tabla II estudiamos la media del error de los cuatro métodos propuestos. La media de error la calcularemos como $|k_{Real} - k_{Estimada}|/k_{Real}$.

TABLA I
COMPARATIVA DE LOS VALORES DE k REAL Y LOS ESTIMADOS CON LOS DIFERENTES MÉTODOS.

Tam	k_{Real}	k_M	k_R	k_{RE}	k_E
30	0.999974	0.94	0.9877	0.9921	**0.9966**
	0.999970	—	—	0.9911	**0.9946**
	0.999999	—	—	0.9934	**0.9992**
35	0.999999	0.94	0.9880	0.9939	**0.9997**
	0.999999	—	—	0.9926	**0.9971**
	0.999999	—	—	0.9936	**0.9991**
40	0.999976	0.94	0.9884	**0.9866**	0.9849
	0.999999	—	—	0.9941	**0.9998**
	0.997213	—	—	**0.9786**	0.9688

TABLA II
TABLA COMPARATIVA DE PORCENTAJE DE ERROR CON CADA UNO DE LOS MÉTODOS.

	k_M	k_R	k_{RE}	k_E
30	5.4%	1.2%	0.7%	0.3%
35	5.4%	1.1%	0.6%	0.1%
40	5.3%	1%	1.3%	1.5%

C. Selección entre diferentes versiones que resuelven un mismo problema

Con el modelo de tiempo de ejecución propuesto, podemos utilizar nuestra metodología para elegir entre diferentes implementaciones que resuelven un mismo problema. Se implementan cinco versiones diferentes para resolver la Mochila 0/1 aunque en este documento solo se incluye el estudio de tres de ellas. Las versiones se diferencian en la forma de generar los nodos y realizar las podas, en todas se realiza algún tipo de poda. No se persigue obtener un algoritmo que resuelva el problema de forma óptima, sino a partir de un algoritmo diseñado por el usuario optimizar su tiempo de ejecución.

El problema que se plantea es que no se conoce a priori que versión tardará menos tiempo en ejecutarse para cada entrada. Por ejemplo, tenemos las versiones *Ver1*, *Ver2* y *Ver3*, y no se sabe cual será más rápida. Sin utilizar el modelo se elegiría una versión fija para todas las entradas, sin embargo, si utilizamos el modelo se estimará, para cada entrada, la versión que la resuelva más rápidamente.

En la Tabla III disponemos del cociente de error medio (frente al tiempo óptimo) al utilizar el esquema propuesto por el modelo o eligiendo fija una de las tres versiones. El esquema que comente menor error es la *Ver3*, sin embargo para usuarios inexpertos sería complicado detectar que este ofrece mejores prestaciones. Eligiendo el modelo, la ganancia obtenida respecto a una mala elección es mayor que el error cometido frente a la *Ver3*. Utilizando esta metodología somos capaces de detectar la tendencia de comportamiento de cada versión.

III. RECORRIDOS PARALELOS DE *backtracking*

Existen diversos esquemas paralelos aplicables a la técnica de *backtracking*. Este trabajo se centra en esquemas de tipo maestro esclavo (M/E). La idea ge-

TABLA III
TABLA DE VALORES MEDIOS DE COEFICIENTES ENTRE LOS TIEMPOS REALES Y LOS OBTENIDOS CON EL MODELO. COMPARATIVA ENTRE LOS DIFERENTES ESQUEMAS.

$\frac{TR_{Modelo}}{TR_{Opt}}$	$\frac{TR_{Ver1}}{TR_{Opt}}$	$\frac{TR_{Ver2}}{TR_{Opt}}$	$\frac{TR_{Ver3}}{TR_{Opt}}$
1.21	1.80	1.61	1.09

neral será distribuir el espacio de búsqueda entre los distintos procesadores disponibles, de forma que cada uno busque la solución del problema en un subespacio de soluciones distinto. El proceso maestro está encargado de coordinar el trabajo de los esclavos: distribución de los subproblemas, gestión de las comunicaciones... [22] Si se realizan podas durante la ejecución, los subespacios de búsqueda no tienen por qué ser iguales y existirán dependencias entre ellos. Estas dependencias hacen que la distribución del espacio de búsqueda sea un factor importante a la hora de diseñar las soluciones paralelas.

En un esquema *M/E* aplicado a *backtracking*, el proceso maestro recorre en primer lugar el árbol de búsqueda hasta nivel *l*. Una vez alcanzado este nivel, se guarda la información de los nodos encontrados a esta profundidad. Posteriormente se van haciendo nuevos recorridos desde los nodos almacenados hasta el último nivel. Cada uno de estos recorridos se puede considerar un *backtracking* secuencial independiente. El número de nodos que se encuentran a nivel l corresponderá con el total de trabajos que puede asignar a los procesos esclavos (*backtracking* secundarios). Son los *backtrackings* secundarios los que realmente se realizan en paralelo, y no podrán empezar hasta que el proceso maestro no haya terminado el primer recorrido. Vamos a estudiar dos alternativas en la asignación de los procesos por parte del proceso maestro: asignación estática y asignación dinámica.

Si seguimos un esquema de asignación estática, a cada procesador se le asignan un número fijo de subtrabajos. Cada procesador únicamente debe dedicarse a resolver aquellos problemas que le han sido asignados inicialmente. Denominaremos estos esquemas como *EMEAE*. Se pueden adoptar diferentes criterios para realizar la asignación de tareas: distribución por bloques donde se asignan bloques contiguos de subtrabajos ($AsigB$), distribución cíclica ($AsigC$), distribución aleatoria ($AsigA$)... Si seguimos un esquema de asignación dinámica, esquema *EMEAD*, los subtrabajos los sirve el proceso maestro a los esclavos bajo demanda. Esta opción favorece una distribución autobalanceada.

Cuando se realizan podas, compartir información local entre los distintos procesadores aumenta significativamente el índice de poda, y disminuye el tiempo de ejecución. Compartir esta información plantea problemas en arquitecturas donde la comunicación entre los procesadores tiene un coste asociado. Establecer el intervalo de intercambio de información (e) adecuado para cada máquina y problema es una tarea difícil, porque será dependiente de la máquina, del tipo problema y de las entradas. En esquemas de asignación dinámica ocurre algo similar, el proceso maestro no distribuirá los trabajos de uno en uno para no consumir excesivo tiempo en las comunicaciones. El número de subtrabajos a enviar (t) será también configurable. Otros parámetros ajustables que intervienen en el tiempo de ejecución son: el valor del parámetro l y el número de procesadores a utilizar p (no siempre elegir mayor valor para p nos llevará a menores tiempo de ejecución). Si se utiliza asignación estática también podremos seleccionar el tipo de asignación a utilizar (A).

IV. MODELADO DE LOS RECORRIDOS PARALELOS

Estas soluciones paralelas traen consigo un aumento del número de *APs* respecto a los esquemas secuenciales, y es más complicado elegir una buena configuración. Una primera aproximación del tiempo de ejecución de un esquema *M/E* estará compuesto por una parte secuencial y una parte paralela:

$$t(n,p,s) = t_{Sec}(l) + t_{Par}(n-l,p,s) \qquad (4)$$

Vamos a centrarnos en un esquema de asignación estática sin intercambio de información. El tiempo empleado en la parte paralela de este algoritmo depende de la duración de los subproblemas y del tipo de asignación utilizada. Al producirse un desbalanceo en la distribución de los subtrabajos, el tiempo paralelo a considerar en el modelo será el de aquel procesador que tarde más tiempo en resolver los problemas que le han sido asignados.

Se supone una asignación de subtrabajos a los procesadores disponibles (A). Esta asignación se define como una estructura de "Número de subtrabajos" (NS) elementos donde cada uno de sus componentes adquirirá un valor en el intervalo de enteros $[0..p-1]$ (donde p es el número de procesadores disponibles). Se define esta asignación como:

$$A\,(i), \forall i = 0..NS-1 \quad A(i)\epsilon[0..p-1] \qquad (5)$$

Por tanto se puede identificar el tiempo que se emplea en la parte paralela del esquema t_{Par} como:

$$\max_{j=0}^{p-1} \left\{ \sum_{i=0, A(i)=j}^{NS-1} t_{Secuencial}(n-l,s) \right\} \qquad (6)$$

Se ha diseñado una estrategia de estimación que permite aproximar la configuración de parámetros que lleva a los tiempos de ejecución mínimos. Entre los diferentes parámetros identificados en el modelo, k y TCN no son parámetros configurables. Por contra, p, l y A sí deben ser seleccionados por el usuario (son *APs*).

Utilizando la metodología de estimación de los recorridos secuenciales añadiendo información de la ejecución, podemos estimar cuanto tiempo vamos a

dedicar a cada uno de los subtrabajos. Para minimizar el tiempo de ejecución de la rutina, se debe minimizar la ecuación:

$$\min_{(l,p,A,s)} \{t_{Sec}(n,l) + t_{Par}(n-l,p,A,s)\} \quad (7)$$

Y el valor de esta ecuación depende directamente de la asignación de subtrabajos que hagamos a los procesadores. Una vez estimada la duración de los subtrabajos, debemos encontrar la asignación de tareas a procesadores que minimiza la ecuación 7. Lo ideal sería encontrar la solución óptima pero nos llevaría demasiado tiempo. Utilizaremos un algoritmo voraz (*AsigVoraz*) para obtener una solución aproximada: se recorren todos los subtrabajos y se asigna cada uno de ellos al procesador que menos incremente, según la distribución parcial hasta ese momento, el tiempo de ejecución del modelo. También se pueden utilizar distintas técnicas metaheurísticas [16].

V. Resultados experimentales

En este trabajo hemos implementado un esquema *EMEAE* sin intercambio de información en memoria compartida por medio de *OpenMP*. Se implementan las asignaciones ya mencionadas: asignaciones básicas (bloques, cíclica y aleatoria) y la asignación voraz de los subtrabajos (*AsigVoraz*).

Para modelar la ganancia que podría suponer utilizar esta técnica para un usuario de la rutina, definimos tres perfiles de usuarios. El *UsOpt* es capaz de dar siempre con la combinación de parámetros que nos lleva a la ejecución óptima, para cualquiera de las entradas. El *UsMed* elegirá como parámetros valores que empíricamente se consideran adecuados. Por último, definimos el *UsLow* que carece de conocimientos de paralelismo y elegirá valores al azar, por lo que sus decisiones pueden no ser acertadas.

En la Tabla IV se muestran cinco filas correspondientes a la resolución de cinco entradas diferentes de tamaño 40. Se utiliza la misma configuración para la fase de instalación que en el experimento secuencial y se estima k por el método k_{RE}. Las primeras cuatro columnas representan los parámetros de configuración seleccionados por el modelo (l, p, A) y el tiempo de ejecución con esta configuración (TR). A continuación se muestran los tiempos de ejecución con los parámetros seleccionados por los distintos perfiles de usuarios. La configuración del *UsMed* es (l, p, A)=(*5*, *4*, *AsigC*) y la del *UsLow* es (*10*, *2*, *AsigB*). En negrita se representan las configuraciones que nos llevan a los mejores tiempos.

Cabe destacar la gran diferencia en el tiempo de ejecución para entradas del mismo tamaño. Esto se debe a la gran dependencia de las entradas en este tipo de recorridos, lo que hace especialmente difícil la estimación del parámetro k.

De la Tabla IV se concluye que para entradas con recorridos más pesados (cuarta y quinta fila) utilizar el algoritmo voraz para realizar la asignación

TABLA IV
Tiempos de ejecución (en segundos) con los parámetros seleccionados por el modelo, comparados con los tiempos que conseguirían tres perfiles de usuarios.

l	***p***	***A***	***TR***	***UsOpt***	***UsMed***	***UsLow***
2	4	*c*	21.26	**13.18**	21.26	27.27
2	4	*c*	0.095	**0.07057**	0.0949	0.1116
1	1	*c*	0.00031	**0.00029**	0.00043	0.00037
18	4	*v*	**49.13**	67.37	90.78	128.326
4	4	*v*	**117.56**	121.54	150.93	228.02

de tareas obtiene resultados mejores que los que obtendríamos utilizando asignaciones básicas (cíclicas y bloques). Sin embargo, para entradas cuya naturaleza nos lleva a dedicar un tiempo relativamente pequeño, los parámetros elegidos por el modelo no suponen ninguna ventaja. En general, el tiempo medio ganado al utilizar el modelo para entradas pesadas supera al tiempo que perdemos para entradas ligeras.

Por último se ha implementado un esquema *M/E* con asignación estática e intercambio de información. Los parámetros a configurar serán (l, p, e), y fijamos A a *AsigC* para simplificar el estudio. Vamos a implementar esta solución para un entorno de memoria distribuida utilizando el estándar de paso de mensajes *MPI*. Con este experimento se pretende estudiar lo que pasaría en entornos donde hay un alto coste de las comunicaciones. En este esquema, con la utilización del modelo del tiempo de ejecución no se han obtenido buenos resultados. La metodología seguida consiste en, a partir de los datos obtenidos durante la fase de instalación (variando los parámetros ajustables), tomar como *configuración básica* de los parámetros, aquella que nos haya llevado a mejores tiempo de ejecución para los tamaños de entrada resueltos. En tiempo de ejecución, una vez conocida la entrada se realizará una búsqueda local alrededor de la *configuración básica*, con una entrada reducida de la original. A medida que se vayan mejorando los resultados iremos modificando la configuración. Una vez se empiecen a obtener peores resultados se detendrá la búsqueda.

El experimento se ha realizado en un cluster de 16 núcleos. Se comparan los resultados del modelo con los obtenidos por los tres perfiles de usuarios utilizados en la prueba anterior, a excepción del usuario medio que es sustituido por un otro que utiliza como parámetros los de la *configuración básica* (*UsInst*). El *manager* configurará la rutina de instalación para generar una batería de 50 entradas aleatorias para cada tamaño entre [5, 40] con incrementos de 5 en 5. En este experimento, el *manager* describirá la máquina configurando: p entre 1 y 16, l entre 1 y 25, y e tomará los valores 100, 500, 1000, 1500, 5000, 10000, 1000000. A partir de los experimentos de la fase de instalación la *configuración básica* obtenida es (l,p,e)=(13,12,10000). El *UserLow* elegirá valores al azar configurando (l,p,e)=(10,5,100). En la Tabla V se muestra los resultados al aplicar el modelo a cuatro entradas de tamaño 40.

TABLA V
TIEMPOS DE EJECUCIÓN (EN SEGUNDOS) CON LOS PARÁMETROS SELECCIONADOS POR EL MODELO, COMPARADOS CON LOS TIEMPOS QUE CONSEGUIRÍAN TRES PERFILES DE USUARIOS.

(l, p, e)	***TR***	***UsOpt***	***UsInst***	***UsLow***
(12, 14, 1000)	29.40	7.37	11.64	170.57
(13, 10, 1000)	0.135	0.030	0.033	0.4804
(13, 12, 1000)	0.00031	0.0002	0.0002	0.000234
(16, 16, 1000)	90.67	24.65	54.37	1233.31

De los resultados de la Tabla V se concluye que elegir los parámetros obtenidos en tiempo de instalación nos acerca al óptimo respecto a elecciones aleatorias de los usuarios. Las búsquedas locales se comportan de manera inadecuada al basarse en problemas de tamaño reducido a partir del original, y esto desvirtua el valor de e; se deberían extrapolar los valores de la búsqueda local al tamaño real. En la paralelización de los recorridos de *backtraking* una mala configuración de los parámetros podría llevarnos incluso a tiempos mayores que la solución secuencial, y esto lo evitamos al utilizar los parámetros propuestos tras la instalación.

VI. Conclusiones y trabajos futuros

El modelado del tiempo de ejecución en esquemas de *backtracking*, plantea problemas al existir alta dependencia de las entradas, por lo que es necesario incluir información de la entrada para actualizar la información obtenida en la instalación. En los esquemas paralelos los problemas de configuración de los parámetros se agudizan por aumentar en número y por las dependencias entre ellos. La metodología propuesta obtiene resultados parcialmente favorables. Aunque no obtenemos configuraciones que nos lleven a los tiempos de ejecución óptimos, se abstrae a los usuarios de la decisión de configuración proponiendo valores que permiten reducir los tiempo secuenciales.

Se identifican como trabajos futuros: extender el estudio de los esquemas con intercambio de información a otro tipo de plataformas, proponer nuevas técnicas para afinar la estimación de los nodos generados y acercarnos más al tiempo de ejecución real, implementar un esquema secuencial con paralelización interna y autoconfiguración, aumentar el banco de pruebas con tamaños más grandes de las entradas, aplicar la metodología a un problema real y estudiar otras técnicas de recorrido de árboles (Branch and bound).

References

[1] F. Almeida, J. M. Beltrán, M. Boratto, D. Giménez, J. Cuenca, L. P. García, J. P. Martínez and A. M. Vidal, *Parametrización de esquemas algorítmicos paralelos para autooptimización*, XVII Jornadas de Paralelismo, Septiembre 2006, pp 443-448.

[2] 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.

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

[4] T. Katagiri, K. Kise and H. Honda *Eect 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.

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

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

[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, 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.

[10] J. Cuenca, D. Giménez, J. González, J. Dongarra and K. Roche, *Automatic optimisation of parallel linear algebra routines in systems with variable load*, In Proceedings of the Euromicro Workshop on Parallel and Distributed Processing (EUROMICRO-PDP 2003), pages 401-408, 2003.

[11] J. Demmel, J. Dongarra, B. N. Parlett, W. Kahan, M. Gu, D. Bindel, Y. Hida, Xiaoye S. Li, O. Marques, E. Jason Riedy, C. Vömel, J. Langou, P. Luszczek, J. Kurzak, A. Buttari, J. Langou, and S. Tomov, *Prospectus for the next LAPACK and ScaLAPACK libraries*, In PARA, pages 11–23, 2006.

[12] M. Boratto, D. Giménez and A. M. Vidal, *Automatic parametrization on divide-and-conquer algorithms*, International Congress of Mathematicians, Madrid, 2006, pp 405-496.

[13] 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.

[14] J. M. Beltrán, *Autooptimización en esquemas paralelos de recorrido de árboles de soluciones: Esquema del problema de la mochila 0/1*, Trabajo de iniciación a la investigación, Universidad de Murcia, Septiembre 2006.

[15] 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., Umea, Sweden, June 18-21, 2006.

[16] F. Almeida, J. Cuenca, D. Giménez, A. Llanes and J. P. Martínez, *A Framework for the Application of Metaheuristics to Tasks-to-Processors Assignation Problems*, Admitido en Journal of Supercomputing.

[17] F. A. Rabhi and S. Gorlatch (Eds), *Patterns and Skeletons for Parallel and Distributed Computing*, Springer, 2003.

[18] M. I. Dorta, *Esquemas paralelos para la técnica de ramificación y acotación*, PhD thesis, Universidad de La Laguna, 2004.

[19] R. Ciegis and M. Baravyka, *Implementation of a Black-Box Global Optimization Algorithm with a Parallel Branch and Bound Template*, PARA 2006, pp 1115-1125.

[20] M. Quesada and D. Giménez *Técnicas de autooptimización en recorridos de árboles por medio de backtracking*, Proyecto fin de carrera, Ingeniería Informática, Universidad de Murcia, Febrero 2009.

[21] G. García, J. Cervera, N. Marín and D. Giménez, *Algoritmos y Estructuras de Datos, Volumen II: Algoritmos*, Diego Marín, 2003.

[22] F. Almeida, D. Giménez, J. M. Mantas and A. M. Vidal, *Introducción a la programación paralela*, Paraninfo, 2008.