Ajuste de la compilación a través de un sistema paralelo multi-objetivo. Aplicación a la mejora del servidor web Apache.

Jorge Calvo Zaragoza, Luis Alberto Vivas Tejuelo, Sergio Cuenca-Asensi, Antonio Martínez-Álvarez, [1] Andrés Ortiz García. [2]

Resumen— **Los compiladores modernos presentan un gran número de opciones de compilación que pueden alterar las características de los programas compilados. La selección de las opciones más ventajosas para mejorar un cierto objetivo (p.e. tiempo de ejecución, tamaño de código, uso de memoria caché, etc) requiere un conocimiento profundo de la arquitectura subyacente y del funcionamiento del compilador. En este trabajo se propone un algoritmo para la generación automática de los parámetros óptimos de compilación basado en algoritmos genéticos y el Modelo de Islas bajo una implementación paralela basada en MPI. El algoritmo es independiente del compilador, fácilmente escalable sobre un sistema multicomputador y soporta la optimización simultánea de varios objetivos contrapuestos gracias al uso del algoritmo NSGA-II. Como caso de estudio se utilizó el servidor web Apache, obteniéndose mejoras en el rendimiento de hasta un 10 %.**

Palabras clave— **Ajuste de la compilación, optimización multi-objetivo, NSGA II, algoritmo genético, Apache.**

I. Introducción

Actualmente, los compiladores ofrecen una gran cantidad de características de optimización. Sin embargo, esta capacidad nunca se explota completamente porque se necesita un conocimiento profundo de la arquitectura subyacente, la aplicación objetivo y el funcionamiento del compilador. La selección de las opciones más convenientes para mejorar un objetivo específico (p. ej. tiempo de ejecución, tamaño de código, fallos de caché, etc.) representa una tarea de enorme complejidad puesto que existen dependencias entre dichos objetivos que no pueden ser previstas por adelantado. Lo compiladores modernos más usados como GCC [1], CLang [2] o ICC [3], proporcionan un gran conjunto de opciones de compilación que pueden modificar las características del programa compilado. Muchas de estas opciones, además, soportan diferentes modificadores, lo que eleva el número de posibilidades a la hora de compilar una aplicación y, en consecuencia, complica el proceso de selección de opciones. Además, ciertas combinaciones de opciones pueden disminuir el rendimiento o incluso cambiar los resultados de ejecución (por ejemplo, vectorizar código podría relajar la precisión de las operaciones en punto flotante). Con miles de posibilidades a considerar, la obtención de una combinación óptima de opciones mediante una aproximación por fuerza bruta es inviable. Se hace por ello necesario utilizar una alternativa que, si bien no llegue a obtener la mejor solución, permita generar soluciones suficientemente buenas en un período de tiempo asumible. En este trabajo se utiliza una estrategia de optimización multi-objetivo, basada en un Algoritmo Genético, que permite una exploración eficiente del espacio de soluciones.

Nuestra solución considera el compilador como un caja negra en la que no se sabe nada acerca del funcionamiento interno. Este enfoque ya fue estudiado con anterioridad por Pinkers et al. [4], que basaron su trabajo en vectores ortogonales, tratando de inferir los efectos de cada opción de compilación en el rendimiento de la aplicación. ACOVEA [5] usa un Algoritmo Genético mono-objetivo para encontrar las mejores opciones para compilar programas utilizando GCC. Guy Bashkansky y Yaakov Yaari [6] propusieron un entorno (ESTO) que obtenía compilaciones subóptimas utilizando también un Algoritmo Genético. Todos los sistemas mencionados consideraban tan sólo el tiempo de ejecución como objetivo. Dentro de este contexto, la principal contribución de nuestra estrategia es el uso de un esquema de optimización multi-objetivo en el motor de búsqueda.

La mejora en el rendimiento de una aplicación puede caracterizarse a través de diversos criterios técnicos y restricciones de diseño que deben ser satisfechas simultáneamente y optimizadas en la medida de lo posible (p. ej. consumo de energía y rendimiento en sistemas empotrados). Ocasionalmente, estos criterios pueden ser contradictorios y resultar en un empeoramiento mutuo. Los problemas de optimización multi-objetivo (MOO) consisten en buscar un conjunto de soluciones óptimas para una serie de criterios contrapuestos. Aunque la combinación de esquemas de optimización multi-objetivo con Algoritmos Genéticos ha sido ampliamente estudiada con anterioridad [7] [8], todavía no se ha aplicado al problema aquí presentado. En nuestra propuesta, el algoritmo NSGA II [9] ha sido el utilizado para manejar la optimización multi-objetivo.

Como se verá más adelante, este algoritmo puede llegar a ser muy costoso computacionalmente por lo que también se propone el uso de una arquitectura paralela basada en el modelo de islas [10] para aprovechar el rendimiento que puede ofrecer un sistema multi-computador en este tipo de algoritmos. Este modelo también puede reducir el efecto de los mínimos locales en la búsqueda de proceso [11] [12].

[1] Departamento de Tecnología Informática y Computación. Universidad de Alicante. {sergio,amartinez}@dtic.ua.es

[2] Departamento de Ingeniería de Comunicaciones. Universidad de Málaga. aortiz@ic.uma.es

El resto del artículo está estructurado como sigue. En la siguiente sección se expondrán algunas consideraciones acerca del problema que se está resolviendo. La implementación del sistema propuesto se detalla en la sección 3. En la sección 4 se describe el caso de estudio experimental. Por último, en la sección 5 se muestran las conclusiones obtenidas.

II. Consideraciones sobre optimización a través de la compilación

El rendimiento de los microprocesadores actuales, con sus complejos *pipelines* y caché de datos e instrucciones integradas, es altamente dependiente del compilador y su habilidad para estructurar el código de forma óptima. Obtener la estructura y el *scheduling* más apropiado es una tarea compleja y específica de cada arquitectura, lo que lleva a diferencias en rendimiento dependiendo de la técnica de optimización empleada. Esta tarea se lleva a cabo de forma extrema en arquitecturas VLIW (Very Long Instruction Word) y EPIC (Explicitly Parallel Instruction Computing) como *Itanium* o *Itanium 2*. Estos microprocesadores delegan la planificación de las instrucciones al compilador para reducir la complejidad y liberar espacio en el circuito. En estos casos, el compilador debe determinar de forma estadística la estructura y explotar la arquitectura paralela para optimizar el rendimiento [13].

Añadido a lo anterior se debe tener en cuenta que las opciones de optimización de los compiladores modernos no funcionan de forma atómica, es decir, su influencia varía en función de otros modificadores. Debido a este hecho, la tarea de ajustar la compilación para sacar el máximo partido al sistema tiene una enorme complejidad, incluso con un conocimiento total del proceso. Aunque algunos compiladores incluyen niveles de optimización predefinidos (p. ej. GCC con `-Ofast`, `-O2`, etc.), en la mayoría de las aplicaciones éstos están lejos del óptimo, especialmente si se trabaja en diferentes contextos (computación de escritorio, servidores, sistemas empotrados, etc.).

Además, el uso de algunas técnicas de optimización puede producir efectos adversos. Por ejemplo, el *inlining* de funciones puede hacer que un programa se ejecute más rápido evitando el coste temporal de la llamada a rutinas. Sin embargo, el sobreuso de este efecto puede producir un programa muy largo, lo que afectaría directamente a los fallos de caché de instrucciones y, por tanto, al rendimiento final [14]. Debido a todas estas circunstancias, se propone la combinación de un algoritmo multi-objetivo, que permita optimizar la búsqueda de objetivos contrapuestos, con un algoritmo genético, que agilice la exploración del espacio de búsqueda.

III. Estrategia paralela de optimización multi-objetivo para el ajuste de la compilación

Como se ha explicado anteriormente, el problema no puede ser resuelto explorando todo el espacio de soluciones. Por ello, en este trabajo se propone la utilización de un Algoritmo Genético como motor de búsqueda. Los Algoritmos Genéticos (GA) son una técnica de búsqueda estocástica que se inspira en la teoría de la evolución y que pertenece al grupo de técnicas conocidas como Algoritmos Evolutivos. Estas técnicas se basan en imitar los procesos evolutivos como la selección natural, el cruce de genes y la mutación. Los GAs operan con un conjunto de individuos donde cada uno de ellos representa una posible solución al problema. Cada individuo está codificado por su cromosoma, compuesto por una serie de genes que representan partes de la solución del problema. Estos individuos se inicializan aleatoriamente y se van obteniendo mejores soluciones a través de los operadores de cruce y mutación. Posteriormente, los individuos son evaluados y selecciones de forma que sólo aquellos que representan las mejores soluciones pueden sobrevivir. Al final del proceso, se puede extraer un conjunto de soluciones de la población superviviente.

Para el problema aquí presentado, las mejores soluciones serán aquellas que más optimizan la aplicación objetivo a través del proceso de compilación. La optimización durante la compilación consiste en el ajuste de las opciones del compilador de forma que se puedan modificar y mejorar ciertas características del programa sin alterar los resultados, es decir, manteniendo la correctitud de la aplicación. Normalmente, el criterio más común es la reducción conjunta del tiempo de ejecución y el tamaño de código. No obstante, muchas de las opciones de compilación reducen el tiempo de ejecución incrementando el tamaño de código, y viceversa. Por tanto, encontrar el equilibrio adecuado entre las distintas posibilidades no es una tarea sencilla. Además, estos no son los únicos criterios conflictivos que podrían ser tenidos en cuenta, especialmente si el ámbito del problema no es la computación con máquinas de propósito general. Por ejemplo, en sistemas empotrados existen otros factores importantes como el consumo de energía, la seguridad o la tolerancia a fallos. En estas situaciones, cuando la evaluación de la calidad de una solución no está clara, deben adoptarse diferentes aproximaciones. En este trabajo, se propone adoptar una estrategia que utilice un esquema de optimización multi-objetivo.

En las siguientes subsecciones se describen los detalles de implementación de la estrategia propuesta.

A. Algoritmo Genético

En nuestra aproximación, cada individuo representa una posible compilación y cada gen codifica el valor de una única opción de compilación (Figura 1). La población se inicializa aleatoriamente, aunque también se permite especificar individuos iniciales para aprovechar conocimiento previo.

El proceso consiste en cinco etapas (Figura 2). En la etapa de inicialización, se crea una nueva población a través de la especificación de las opciones del compilador. En la etapa de selección, se controla el ta-

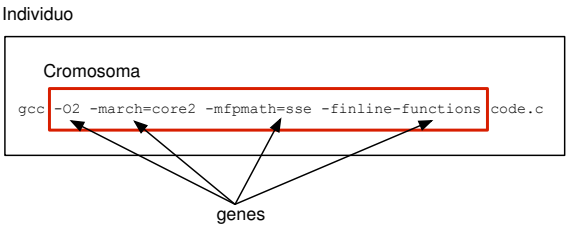

Fig. 1. Ejemplo de individuo para el compilador GCC

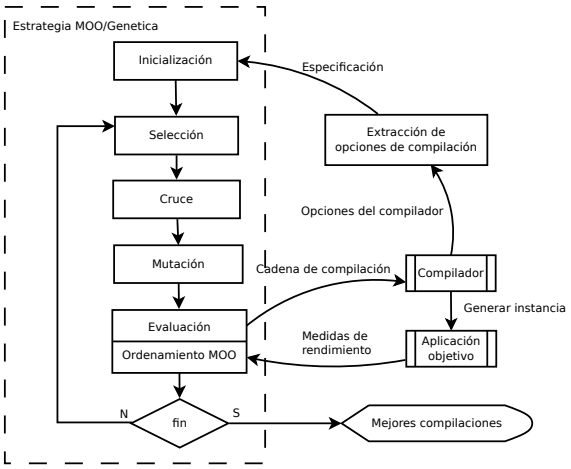

Fig. 2. Etapas de la estrategia propuesta

maño de la población eliminando los individuos adicionales. Posteriormente se crean nuevos individuos a través del operador de cruce. Este operador consiste en tomar dos individuos y crear uno nuevo escogiendo uniformemente genes de uno y de otro. En la etapa de mutación, sólo tres cuartos de la población son mutados para conservar las mejores soluciones. Para cada gen, este proceso decide si modificar el valor asociado utilizando una probabilidad baja. Por último, en la etapa de evaluación cada individuo compila la aplicación objetivo utilizando la codificación de los genes y evalúa el conjunto de criterios especificado. Al final de esta etapa se ordena la población utilizando el algoritmo MOO, que se explicará en la sección siguiente.

Estas últimas cuatro etapas se repiten hasta que el algoritmo alcanza una condición de parada (p. ej. número de iteraciones). Al final, la salida del algoritmo estará formada por el conjunto de compilaciones superviviente.

A.1 Esquema multi-objetivo

En los últimos años se han propuesto diferentes métodos para resolver problemas multi-objetivo. Estos métodos se pueden clasificar en dos grupos: los primero son algoritmos relativamente simples basados en el frente de Pareto (NSGA [15], NPGA [16], VEGA [17] and MOGA [18]), que ya han caído en desuso, y un segundo grupo formado por algoritmos elitistas que enfatizan la eficiencia computacional (SPEA [19], SPEA2 [20], NSGA II [9], MOGLS [21], PESA [22], PESA II [23]). En la actualidad, NSGA II (Nondominated Sorting Genetic Algorithm II) es reconocido como uno de los algoritmos más exitosos para este fin.

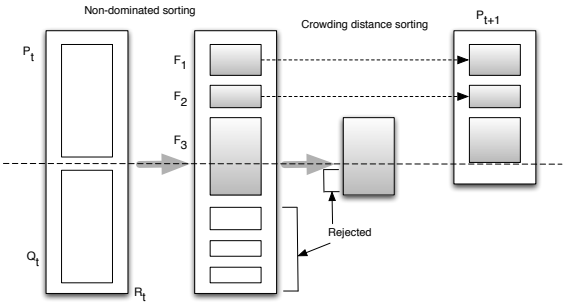

Fig. 3. Selección de población usando NSGA II

El esquema NSGA II consiste en clasificar los individuos en frentes de Pareto no-dominados. Los individuos clasificados en el primer frente (F_1) son aquellos que no están dominados. Los individuos que pertenecen al segundo frente (F_2) son aquellos que son no-dominados en ausencia de los del primer frente, y así, sucesivamente. Se dice que una solución X_i domina otra X_j si la primera es mejor o igual que la segunda en todos los criterios contemplados y estrictamente mejor en al menos uno de ellos. Un individuo no-dominado, por tanto, es aquel que no es dominado por ningún otro.

Una vez los frentes se han formado, los individuos deben ser ordenados de nuevo para mantener un buen abanico de soluciones. NSGA II utiliza la *distancia de crowding*, que estima el valor de diversidad de una solución con respecto a la población global. De esta forma, los individuos son evaluados en base a su diversidad en la dimensión de cada objetivo. La *distancia de crowding* global dependerá de la suma de los valores correspondientes para cada objetivo.

Tras la ejecución del algoritmo, la población estará primero ordenada por rangos de no-dominación y dentro de cada uno, por la *distancia de crowding* (Fig. 3).

B. *Paralelización de la estrategia de optimización*

En el sistema propuesto, la etapa de evaluación consume la mayor parte del tiempo debido a que requiere la ejecución o, al menos, la compilación, de la aplicación objetivo. Afortunadamente, ya que cada individuo puede ser evaluado de forma independiente, este problema presenta un gran nivel de paralelismo. Para poder aprovechar al máximo esta cualidad, se utiliza un esquema de paralelización basado en el modelo de islas. Este modelo considera diferentes poblaciones e itera simultáneamente sobre todas ellas. Después de un cierto número de generaciones, las poblaciones (o islas) intercambian sus mejores individuos en un proceso de migración. Por tanto, este modelo también permite mejorar el algoritmo de búsqueda ayudando a mantener la diversidad de soluciones. Esto permite evitar, en la medida de lo posible, la probabilidad de que las poblaciones se paren en un mínimo local.

El sistema propuesto implementa esta paralelización mediante el uso de la librería de paso de men-

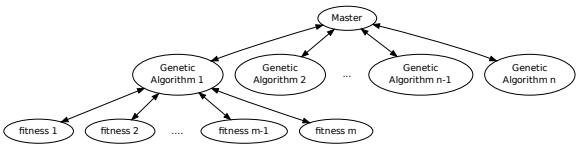

Fig. 4. Árbol de procesos creado para N poblaciones

sajes *Message Passing Library* (MPI) [24] y una estructura de programación maestro-esclavo. Además de los N procesos asociados a sus correspondientes islas, el proceso maestro se utiliza para coordinar la etapa de migración. Dada la sobrecarga de ejecución que introduce este proceso de migración, se escogerán sólo algunas de las islas en cada ejecución. Las islas se escogen a pares donde una actúa como emisora y la otra como receptora. El proceso maestro se encarga de hacer esta elección y notificar el rol (*emisor*, *receptor* or *no actúa*) a cada isla. El proceso emisor envía el 25 % de sus mejores individuos a la isla receptora, que los incluye en su propia población. Las siguientes etapas del algoritmo genético determinarán el valor de fitness de estos nuevos individuos. El proceso de migración se incluye como un nuevo paso al final del bucle principal del algoritmo genético, por lo que no modifica el funcionamiento interno explicado durante esta sección. Por otra parte, cada isla posee a su vez un conjunto de procesos de fitness (Figura 4). Estos procesos reciben la codificación de individuos y los evalúan para cada criterio especificado. De esta forma se permite que cada isla pueda evaluar varios individuos simultáneamente. Tras la evaluación, el proceso de fitness devuelve los resultados obtenidos.

La isoeficiencia de este esquema de paralelización se puede mantener fácilmente aumentando el número de individuos a evaluar, tal como se mostró en [25].

IV. Caso de estudio: optimización del servidor web Apache

En esta sección la estrategia propuesta se aplica para optimizar el servidor web Apache, basándose en las siguientes métricas de interés: número de cambios de contexto (extraído de `/proc/stat`), fallos de caché L2, *throughput* del servidor web y tiempo medio por petición.

Los servidor web modernos como Apache consumen la mayor parte del tiempo de ejecución en la pila de red (*networking stack*) del sistema operativo. Por consiguiente, la pila de red tiene una gran influencia en el rendimiento global. En estas aplicaciones, el número de fallos de caché suele ser alto debido al hecho de que la pila TCP/IP de Linux no cabe en las caché L2 de tamaño convencional. Además, si las páginas servidas contienen datos aleatorios -dificultando los predictores de los microprocesadores-, los fallos de caché L2 se incrementan aún más. Como la latencia de las DRAM suele ser superior a cientos de ciclos de reloj, estos fallos de caché reducen considerablemente el rendimiento del procesador. Por tanto, todos los criterios mencionados deben ser

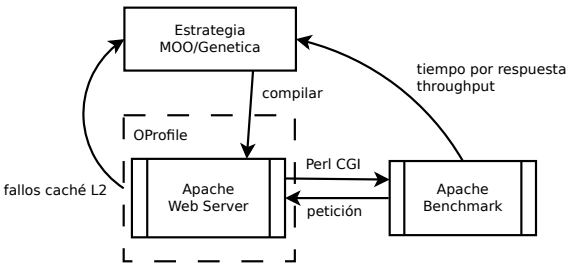

Fig. 5. Escenario del caso de estudio Apache

tenidos en cuenta a la hora de evaluar el rendimiento del servidor.

Para evaluar el rendimiento del servidor se ha utilizado la herramienta Apache Benchmark (AB) [26]. Esta herramienta permite realizar test de carga con diferente configuración y proporciona un amplio rango de indicadores de rendimiento como el número de peticiones procesadas por segundo o el *troughput* alcanzando. Por otra parte, se usó OProfile [27] para las mediciones de los fallos de caché. OProfile es una herramienta de grano fino de perfilado de código que consiste en un *driver* central, un *daemon* y varias herramientas de reporte. Esta herramienta recoge datos de los registros de conteo de rendimiento dentro de intervalos de tiempo definido. A diferencia de otras herramientas, OProfile puede perfilar el sistema al completo sin necesidad de modificar la aplicación objetivo.

Para tener en cuenta el peor caso, es obligatorio evitar los predictores de la CPU [28]. Por lo tanto, la página servida se ha establecido como un script CGI en Perl que genera números reales aleatorios. Este test también permite tener más precisión en las medidas ya que minimiza el uso de los puertos *E/S*. Para esta prueba se ha establecido el tamaño de página a 2KiB.

La configuración experimental está compuesto de máquinas Intel Core 2 Duo 2.3GHz con 4 GiB de RAM y un Realtek RTL8169 based gigabit network interface controller (NIC), corriendo un Debian Linux con 3.2.0-1-amd64 kernel, el compilador GCC 4.6.2 y el servidor web Apache en su versión 2.2.21. El MTU (Maximum transmission unit) se configuró al máximo permitido (7000) para optimizar el ancho de banda.

La prueba consistió en la computación de 145 iteraciones y duró 6 días completos.

A. Resultados experimentales

Los resultados obtenidos por los individuos evaluados se han expresado con una nube de puntos cercana al origen de coordenadas en ambos ejes. Los datos se muestran en diferentes gráficos para cada par de criterios. Se incluyen también los niveles de optimización predefinidos en GCC (`-O1, -O2, -O3`) para poder tener una mejor perspectiva de los resultados obtenidos. También se marcan aquellos individuos que pertenecen al frente de Pareto (individuos no-dominados).

Principalmente se puede observar que los resulta-

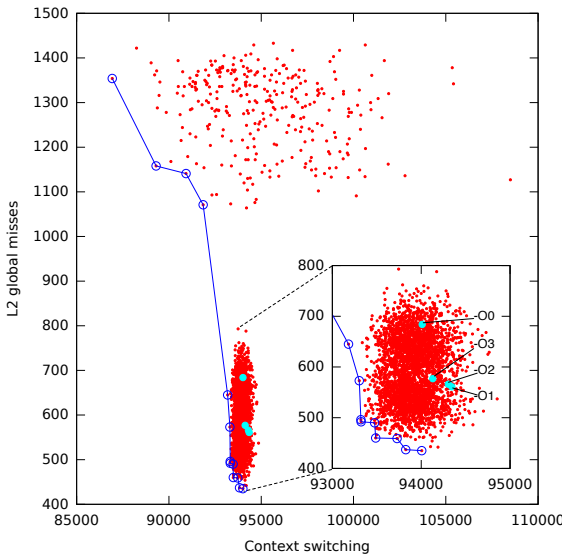

Fig. 6. Fallos de caché L2 contra cambios de contexto

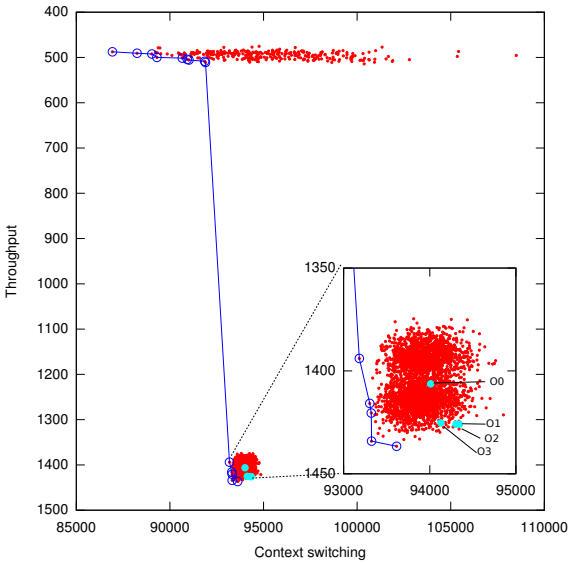

Fig. 7. Troughput contra cambios de contexto

dos se concentran en dos grupos a lo largo de todas las gráficas. En la Figura 6 se muestra la nube de puntos obtenida enfrentando el número de fallos de caché L2 y el número de cambios de contexto. El conjunto de soluciones que pertenecen al frente de Pareto cuenta tanto con individuos con un número pequeño de cambios de contexto y una gran cantidad de fallos de caché L2 como individuos más balanceados. También se puede apreciar una significativa reducción de los fallos de caché comparando con los niveles de optimización predefinidos.

La Figura 7 muestra las soluciones en términos de troughput (KiB/s) y cambios de contexto. Similar al gráfico anterior (Figura 6), existen dos grupos diferentes, todavía más compactos en este caso.

En la Figura 8, el eje de abscisas representa el número de fallos en caché L2 mientras que el eje de ordenadas muestra el troughput. Se puede observar la relación entre estos dos criterios, ya que los individuos con mayor número de fallos de caché tienen mucho menor troughput. No obstante, las opciones

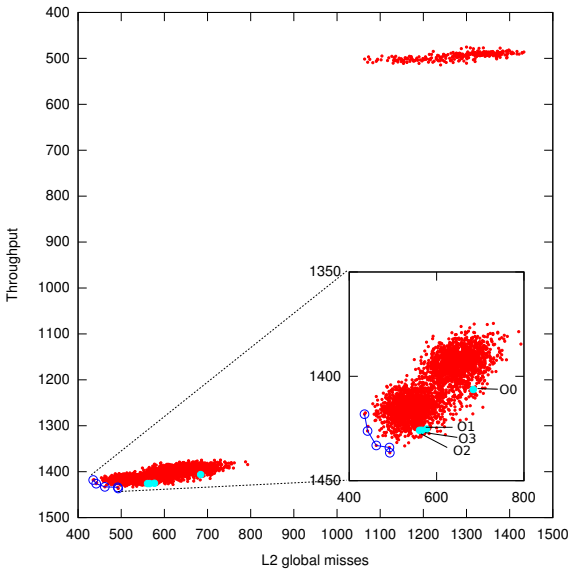

Fig. 8. Troughput contra fallos de caché L2

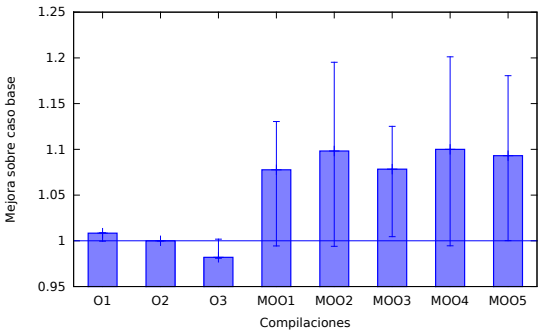

Fig. 9. Mejora obtenida respecto al caso base `-O2`

predefinidas de GCC muestran un buen troughput a pesar de no tener los mejores resultados en fallos de caché. Las soluciones que pertenecen al frente de Pareto tienen un aumento marginal del troughput aunque son significativamente mejores en términos de caché.

Dado que el objetivo del caso de estudio es optimizar de forma global el servidor web, en la Figura 9 se muestra la media de la mejora obtenida por cada criterio para los cinco mejores individuos. Esta media puede verse como una medida de mejora conjunta. La optimización predefinida *-O2* se ha establecido como linea base ya que es la que se activa por defecto en las distribuciones convencionales de Apache.

Viendo las nubes de puntos se puede observar que las opciones de optimización predefinidas no aportan una mejora del rendimiento significativa. Como se puede ver en las gráficas, las diferencias entre `-O0` y `-O2`/`-O3` son mínimas en el mejor de los casos. Los resultados obtenidos por nuestra estrategia tampoco ofrecen una mejora notable para cada uno de los criterios de forma aislada. No obstante, si se tiene en cuenta la ganancia conseguida teniendo en cuenta todos los criterios contemplados, el sistema es capaz de ofrecer compilaciones que muestran una mejora cercana al 10 %.

V. Conclusiones

En este trabajo se ha propuesto una estrategia paralelizable basada en un Algoritmo Genético para la obtención de las opciones óptimas de compilación. Además, esta estrategia se ha combinado con el uso del algoritmo multi-objetivo NSGA II para permitir una optimización de múltiples criterios al mismo tiempo. En el caso de estudio aplicado, enfocado a la optimización del servidor web Apache, se ha mostrado que existe un margen reducido de mejora con respecto a cada uno de los criterios contemplados por separado. No obstante, nuestra estrategia ha demostrado su habilidad para encontrar una mejora global que ha sido capaz de obtener hasta un 10 % de ganancia en la respuesta a peticiones de página de tamaño 2KiB.

Como próximo paso se pretende estudiar los efectos que produce el aumento o la reducción del tamaño de página en los resultados obtenidos. De cara a siguientes trabajos también se va a tratar de inferir la incidencia de las opciones de compilación para cada criterio contemplado. En este caso, sería interesante saber qué opción o combinación de opciones hace que los individuos pertenecen a la mejor o peor nube de puntos que se observan en los resultados. Otra cuestión de estudio interesante sería la de considerar otras aplicaciones que no tienen una alta dependencia de los recursos de red con el fin de comparar los resultados. Esto también nos permitirá hacer un estudio de la incidencia de las opciones de compilación en función del tipo de aplicación, cuyos resultados podrían ser utilizados en la estrategia como un conocimiento a priori, permitiendo acelerar la convergencia hacia soluciones óptimas.

Referencias

[1] GCC, "GNU Compiler Collection," `http://gcc.gnu.org`, 1987.

[2] CLang, "A C language family frontend for LLVM," `http://clang.llvm.org/`, 2011.

[3] ICC, "Intel C++ Compiler," `http://software.intel.com/en-us /intel-compilers/`, 2011.

[4] R.P.J. Pinkers, P.M.W. Knijnenburg, M. Haneda, and H.A.G. Wijshoff, "Statistical selection of compiler options," in *Modeling, Analysis, and Simulation of Computer and Telecommunications Systems, 2004. (MASCOTS 2004). Proceedings. The IEEE Computer Society's 12th Annual International Symposium on*, oct. 2004, pp. 494–501.

[5] Scott Robert Ladd, "Acovea," 2007, `http://kidsvid.altec.org` Last accessed: 30/03/2012.

[6] SMART '07, *Black box approach for selecting optimization options using budget-limited genetic algorithms.*, 2007.

[7] Eckart Zitzler and Lothar Thiele, "Multiobjective optimization using evolutionary algorithms - a comparative case study," 1998, pp. 292–301, Springer.

[8] Carlos M. Fonseca and Peter J. Fleming, "Genetic Algorithms for Multiobjective Optimization: Formulation, Discussion and Generalization," in *Genetic Algorithms: Proceedings of the Fifth International Conference*. 1993, pp. 416–423, Morgan Kaufmann.

[9] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: Nsga-ii," *Evolutionary Computation, IEEE Transactions on*, vol. 6, no. 2, pp. 182 –197, apr 2002.

[10] Cantú E. Paz, "A Survey of Parallel Genetic Algorithms," 1997.

[11] D.A. Van Veldhuizen, J.B. Zydallis, and G.B. Lamont, "Considerations in engineering parallel multiobjective evolutionary algorithms," *Evolutionary Computation, IEEE Transactions on*, vol. 7, no. 2, pp. 144 – 173, april 2003.

[12] F. de Toro, J. Ortega, J. Fernandez, and A. Diaz, "Psfga: a parallel genetic algorithm for multiobjective optimization," in *Parallel, Distributed and Network-based Processing, 2002. Proceedings. 10th Euromicro Workshop on*, 2002, pp. 384 –391.

[13] Carole Dulong, Rakesh Krishnaiyer, and Dattatraya Kulkarni, "An overview of the intel ia-64 compiler," 1999.

[14] Frederic P. Miller, Agnes F. Vandome, and John McBrewster, *Inline Expansion: Compiler Optimization, Called Party, Branch (computer science), Algorithmic Efficiency, Return Statement, CPU Cache, Constant (programming), Compiler, Copy and Paste Programming.*, Alpha Press, 2010.

[15] N. Srinivas and Kalyanmoy Deb, "Multiobjective optimization using nondominated sorting in genetic algorithms," *Evolutionary Computation*, vol. 2, pp. 221–248, 1994.

[16] Jeffrey Horn, Jeffrey Horn, Nicholas Nafpliotis, Nicholas Nafpliotis, David E. Goldberg, and David E. Goldberg, "A niched pareto genetic algorithm for multiobjective optimization," in *In Proceedings of the First IEEE Conference on Evolutionary Computation, IEEE World Congress on Computational Intelligence*, 1994, pp. 82–87.

[17] J. D. Schaffer, "Multiple objective optimization with vector evaluated genetic algorithms," in *Proc. of the International Conference on Genetic Algorithms and Their Applications*, Pittsburgh, PA, 1985, pp. 93–100.

[18] Carlos M. Fonseca and Peter J. Fleming, "Genetic algorithms for multiobjective optimization: Formulation, discussion and generalization," 1993.

[19] Eckart Zitzler and Lothar Thiele, "Multiobjective evolutionary algorithms: A comparative case study and the strength pareto approach," 1999.

[20] Eckart Zitzler, Marco Laumanns, and Lothar Thiele, "Spea2: Improving the strength pareto evolutionary algorithm," Tech. Rep., 2001.

[21] Andrzej Jaszkiewicz, "On the performance of multiple objective genetic local search on the 0/1 knapsack problem. a comparative experiment.," 2000.

[22] David W. Corne, Joshua D. Knowles, and Martin J. Oates, "The pareto envelope-based selection algorithm for multiobjective optimization," in *Proceedings of the Parallel Problem Solving from Nature VI Conference*. 2000, pp. 839–848, Springer.

[23] David W. Corne, Nick R. Jerram, Joshua D. Knowles, Martin J. Oates, and Martin J, "Pesa-ii: Region-based selection in evolutionary multiobjective optimization," in *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2001*. 2001, pp. 283–290, Morgan Kaufmann Publishers.

[24] MPI Forum, "Message Passing Interface (MPI) Forum Home Page," `http://www.mpi-forum.org/`, Dec. 2009.

[25] L. A. Vivas Tejuelo, J. Calvo Zaragoza, F. Restrepo Calle, S. Cuenca Asensi, A. Ortiz Garcia, and A. Martinez Alvarez, "Mejora del rendimiento de una aplicacion mediante la determinacion automatica de las opciones optimas de compilacion," in *Actas de las XXI Jornadas de Paralelismo*, sep. 2010, pp. 159–165.

[26] Apache HTTP server benchmarking tool, "The Apache Software Foundation," `http://httpd.apache.org/docs/2.0/programs/ab.html`, 2011.

[27] OProfile, "A System Profiler for Linux," `http://oprofile.sourceforge.net/`, 2011.

[28] Intel, *Intel 64 and IA-32 Architectures Optimization Reference Manual*, 2011.