Metodología para la generación y evaluación automática de hardware específico

Cecilia González[1], Daniel Jiménez-González[2], Xavier Martorell[3], Carlos Álvarez[4] and Georgi Gaydadjiev[5]

Resumen— **En el área de la bioinformática podemos encontrar aplicaciones que suponen un reto para el diseño de nuevas arquitecturas de procesadores en términos de rendimiento, ya que sus características difieren de las de las aplicaciones de propósito general. Por ello proponemos una nueva arquitectura con unidades funcionales reconfigurables para un dominio específico de aplicaciones. Así, el primer paso para definir la nueva arquitectura será la creación de la nueva ISA del procesador, que se compondrá de extensiones de la ISA original. Para conseguir dicho objetivo, presentamos una metodología para identificar automáticamente patrones de instrucciones y generar prototipos de las unidades funcionales que las ejecutan. Hemos implementado la metodología de manera experimental con el soporte de la infraestructura Trimaran para la identificación de extensiones de la ISA, la herramienta DWARV para la generación de código VHDL, y la plataforma MOLEN para la evaluación de los prototipos hardware específicos generados automáticamente. En las evaluaciones iniciales de los prototipos generados para una aplicación de estudio, ClustalW, se ha obtenido hasta un $8.54x$ de *speed-up* para un único acelerador, mientras que el *speed-up* de toda la aplicación está por encima de $2x$.**

Palabras clave— **Especialización hardware, hardware reconfigurable, bioinformática.**

I. Introducción

En los últimos años, las aplicaciones bioinformáticas han mostrado un notable aumento en complejidad, tanto algorítmica como del volumen de datos tratados. Este hecho ha motivado la investigación de nuevas arquitecturas en el área de Computación de Altas Prestaciones, ya que plantea interesantes retos relacionados con el aumento del rendimiento en la ejecución de estas aplicaciones. Aunque este tipo de planteamientos han sido vistos de manera similar en otras áreas de computación científica, existen ciertas particularidades en la bioinformática que inducen la necesidad de una arquitectura con un hardware especializado. Por otra parte, si tenemos en cuenta que las características de las aplicaciones del dominio bioinformático muestran una destacable variabilidad, tendremos numerosas opciones de especialización de la arquitectura. Por tanto, nuestro principal objetivo es proponer una arquitectura que sea lo suficientemente flexible para poder adaptarse al mayor número de aplicaciones dentro de un mismo dominio, priorizando el rendimiento y la eficiencia energética. Esta flexibilidad se consigue con tecnologías de reconfigurables que permiten la construcción de hardware específico.

A pesar de que la creación de hardware específico se aplica principalmente con sistemas empotrados, nosotros proponemos su aplicación en el contexto de la Computación de Altas Prestaciones para aplicaciones científicas. La especialización puede ser llevada a cabo manualmente, lo cual implica un esfuerzo considerable por parte del diseñador de hardware. Sin embargo, aunque aplicando este método pueden obtenerse importantes cifras de rendimiento, simpre requiere un esfuerzo considerable en el tiempo de diseño. Además, si la especialización se realiza no sólo para una aplicación, sino dentro de un dominio, la complejidad del proceso lo haría inviable. Por tanto, nosotros adoptamos como opción la especialización automática de hardware, sin intervención humana.

El proceso seguido para la generación automática de hardware específico identifica las secciones de las aplicaciones más críticas en tiempo de ejecución, de manera que sean migradas a hardware especializado, mientras que el resto de secciones se ejecutan en un procesador de propósito general (General Purpose Processor: GPP). En este contexto, el hardware especializado consistirá en unidades funcionales específicas que extienden las funcionalidades del GPP. Por consiguiente, la arquitectura del conjunto de instrucciones (Instruction Set Architecture: ISA) del procesador debe ser extendido para codificar las instrucciones que se ejecutan en las nuevas unidades funcionales. La extensión de la ISA de un procesador con nuevas instrucciones es un método considerablemente extendido para la especialización de un procesador dentro de un dominio de aplicaciones.

La mayoría de trabajos existentes sobre extensiones de la ISA prueban sus propuestas en entornos de simulación, debido a restricciones técnicas que se dan al usar hardware real. Esto es así por la ausencia de un mecanismo automático para probar experimentalmente las extensiones de la ISA en un entorno con hardware real. Nuestro trabajo explora el uso de una plataforma hardware reconfigurable, complementando un *toolchain* que identifica de manera automática aquellas partes de código más idóneas para ser propuestas como extensión de la ISA.

El presente artículo se organiza de la siguiente

[1]Barcelona Supercomputing Center, e-mail: cecilia.gonzalez@bsc.es.
[2]Dept. of Computer Architecture, Universitat Politècnica de Catalunya, e-mail: djimenez@ac.upc.edu.
[3]Barcelona Supercomputing Center - Universitat Politècnica de Catalunya, e-mail: xavier.martorell@bsc.es.
[4]Dept. of Computer Architecture, Universitat Politècnica de Catalunya, e-mail: calvarez@ac.upc.edu.
[5]Computer Engineering, Technical University of Delft, e-mail: g.n.gaydadjiev@tudelft.nl.

manera: en la sección II planteamos el problema de las extensiones de la ISA, así como trabajos relacionados. En la sección III explicamos tanto la arquitectura como la metodología de generación y evaluación de hardware específico. Seguidamente, en la sección IV podemos encontrar la implementación de dicha metodología, la cual se prueba experimentalmente y se explica en las secciones V y VI.

II. Trabajo relacionado

La generación automática de extensiones de la ISA se encuentra normalmente dividida en dos objetivos diferentes: (1) la identificación y (2) la selección de nuevas instrucciones [1]. La identificación de candidatos incluye la generación de patrones formados por las instrucciones de la ISA original. Por otra parte, la selección del conjunto final de instrucciones puede variar, dependiendo de las métricas que son evaluadas durante el proceso.

El enfoque más extendido en la exploración de las aplicaciones, está basado en el análisis de grafos de una representación intermedia del código. Para poder trabajar con la representación intermedia se suelen utilizar infraestructuras de compilación y análisis, como por ejemplo SUIF [2] (tal y como se encuentra en [3], [4]) y Trimaran [5] (puede verse en los trabajos: [6], [7], [8], [9]).

La principal diferencia que se encuentra entre los distintos trabajos sobre extensiones de la ISA es la estrategia que siguen para resolver los problemas de identificación y selección.

La identificación puede ser resuelta con una búsqueda exhaustiva a través del grafo que representa la aplicación. Como la complejidad algorítmica de la búsqueda puede ser exponencial, algunos autores proponen diferentes técnicas de backtracking [4]. En nuestra implementación también acotamos el espacio de búsqueda restringiendo el número de entradas y salidas de los candidatos a instrucciones.

La manera más común de abordar la selección es el uso de heurísticos, guiados por una función de coste [6], [7], [8], [10], [11]. A pesar de que no se obtiene al solución óptima, experimentalmente se ha demostrado que en ciertos casos se acerca al óptimo, dependiendo de las métricas escogidas de la función de coste. Con tal de obtener soluciones óptimas, también se ha propuesto resolver el problema de la selección con métodos de programación lineal entera [9], [10]. Nosotros hemos optado por realizar la selección con heurísticas.

Entre las referencias consultadas, la experimentación se realiza con simulaciones o estimaciones de ciclos de ejecución. Además, a pesar de que algunos modelos se traducen a VHDL y se sintetizan [9], son únicamente utilizados como dato de entrada para la función de coste de la selección. Sin embargo, en el trabajo que presentamos, hemos usado la traducción VHDL tanto para la función de coste como para la plataforma de evaluación en un hardware real.

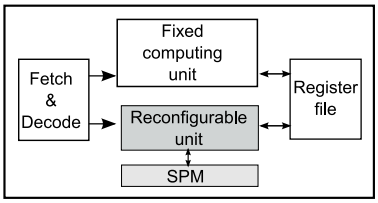

Fig. 1. Arquitectura propuesta para el dominio específico de aplicaciones.

III. Generación automática y evaluación de prototipos

En la Figura 1 se muestra un diagrama del concepto de arquitectura que proponemos para acelerar las aplicaciones dentro de un dominio.

La base de la nueva arquitectura es un GPP existente con una ISA y unidades funcionales fijas. La ISA del GPP se extiende con aquellas instrucciones especializadas que potencialmente mejoren el rendimiento de las aplicaciones, en referencia al GPP original. Por otra parte, el procesador base se amplía con unidades funcionales especializadas que ejecutan las extensiones de la ISA. Estas nuevas unidades son introducidas en el área reconfigurable del procesador. Además, puede existir una memoria de tipo Scratchpad (SPM), que se conecta directamente a la memoria principal del sistema mediante transferencias DMA.

El principal inconveniente de la experimentación práctica en esta arquitectura es que, dentro de lo que conocemos, no existe una plataforma rápida de generación de prototipos.

Nuestro trabajo actual está centrado en un *toolchain* automático para la generación rápida de prototipos de unidades funcionales específicas para un dominio de aplicaciones. El proceso de diseño de las nuevas unidades funcionales está confiado completamente al *toolchain*, sin intervención humana. En la Figura 2 mostramos un esquema de las partes más importantes que conforman el proceso que genera los prototipos. Así, en los pasos 1 (Profiling), 2 (Identificación de candidatos) y 3 (Selección de extensiones de la ISA) identificamos las nuevas extensiones de la ISA para nuestra propuesta de procesador de dominio específico. En los pasos 4 (Generación de descripción hardware) y 5 (Generación de código) producimos, respectivamente, el hardware necesario y el código binario para utilizar las nuevas extensiones con las aplicaciones del dominio elegido. Seguidamente exponemos en más detalle la descripción de los pasos mencionados.

Detección de extensiones de la ISA

Paso 1. Profiling

Debido a que uno de los objetivos de la arquitectura especializada es la aceleración de la ejecución de las aplicaciones, debemos identificar aquellos bloques de código que más se ejecuten, ya que basándonos en la ley de Amdahl, son éstos en los que podemos obtener mejoras de velocidad significativas. Por lo

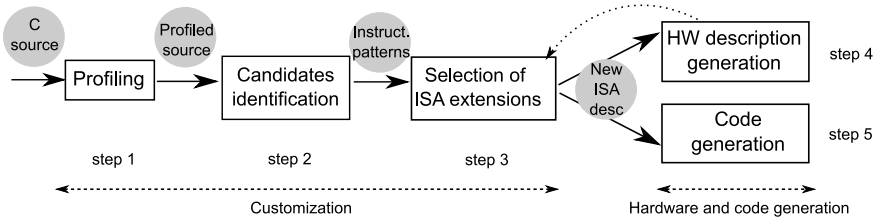

Fig. 2. Esquema del proceso automático de generación de prototipos.

tanto, una de las métricas de selección de las nuevas instrucciones es la frecuencia de ejecución de cada sección de código. Esta información se extrae del análisis del rendimiento (profile) del código de la aplicación.

Paso 2. Identificación de candidatos

Cada aplicación se representa con el grafo de flujo (Data Flow Graph: DFG) de la secuencia de instrucciones. El DFG es examinado a nivel de bloque básico para obtener subgrafos de instrucciones básicas que cumplan las restricciones de la arquitectura, por ejemplo, el número de entradas y salidas o el tipo de instrucción. Cada uno de los subgrafos identificados es un candidato para convertirse en nueva instrucción especializada.

Paso 3. Selección de extensiones de la ISA

La selección final se realiza mediante un algoritmo voraz. La búsqueda está guiada por una función de coste que utiliza la información extraída durante la fase de profiling. También puede utilizar información sobre el área que ocupará la nueva unidad que ejecuta la instrucción, obtenida en una fase posterior. La función de coste intenta maximizar la ganancia en rendimiento gracias a la nueva instrucción propuesta, dependiendo de la métrica que ha sido escogida. El número de nuevas instrucciones que como máximo son seleccionadas está limitado por el área total disponible en el hardware reconfigurable.

Generación de código y hardware

Paso 4. Generación de la descripción hardware

Para cada nueva instrucción seleccionada, generamos la descripción hardware de la unidad funcional que la ejecuta. Esta nueva unidad funcional es ubicada en el área reconfigurable del procesador, tal y como se muestra en la Figura 1.

Paso 5. Generación de código

Las nuevas instrucciones deben ser incorporadas al código de las aplicaciones del dominio escogidas. Para ello, el compilador de la nueva arquitectura se parametriza para detectar los patrones de código que se identifican con las nuevas instrucciones. De esta manera, el compilador será capaz de generar el código que utiliza las nuevas extensiones de la ISA.

Evaluación

Las pruebas también han sido pensadas para realizarse de manera automática, con el objetivo de obtener de forma rápida una plataforma de evaluación para cada aplicación considerada.

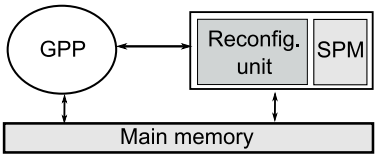

Fig. 3. Modelo de coprocesador utilizado en la implementación actual.

Así, aprovechamos la descripción hardware de las nuevas unidades funcionales para programar la parte reconfigurable de la arquitectura (Figura 1). De esta forma, obtenemos una plataforma para la evaluación del nuevo hardware especializado. Así mismo, esta plataforma dispondrá de un GPP en el que se ejecutará la parte del código que no contiene ninguna extensión de la ISA.

IV. Implementación actual

La plataforma de generación y evaluación de los prototipos en la implementación actual es MOLEN [12]. MOLEN es un procesador polimórfico compuesto de dos componentes principales: un núcleo de proceso, que realiza las tareas como un GPP, y un procesador reconfigurable con una unidad específica, llamada Custom Computing Unit (CCU), la cual se ejecuta como un coprocesador. En el paradigma de programación de MOLEN, las aplicaciones se ejecutan principalmente en el GPP. No obstante, algunas partes son implementadas en el hardware reconfigurable de la CCU para mejorar el tiempo completo de ejecución de la aplicación. Debido a que hemos tenido que adaptar nuestra arquitectura objetivo (Figura 1) a MOLEN, en la implementación de prueba las nuevas unidades funcionales se ejecutan como un coprocesador, tal y como se muestra en la Figura 3.

En la Figura 4 podemos ver un esquema de la implementación actual de la metodología propuesta. En dicha figura se especifica la correspondencia con los pasos de la metodología mostrados en la Figura 2. Seguidamente explicamos los pasos de la implementación de la Figura 4, exceptuando la Evaluación, que se expone con detalle en la sección VI.

Detección de extensiones de la ISA

Pasos 1 y 2. Profiling e Identificación de candidatos

Las fases de profiling y de identificación de candidatos han sido implementadas dentro de Trimaran

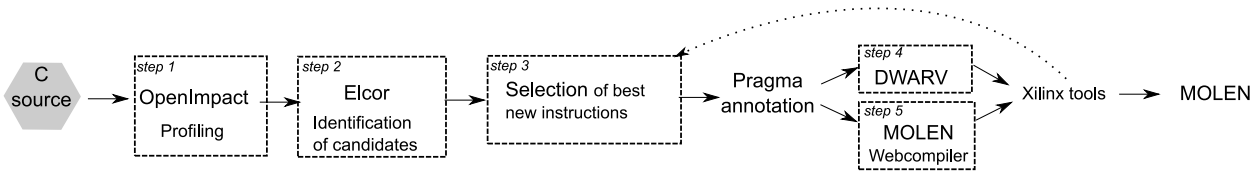

Fig. 4. Esquema de la implementación actual del *toolchain*.

[5]. Trimaran es una infraestructura de compilación compuesta por un frontend (OpenImpact), un backend (Elcor) y un simulador de arquitectura VLIW. Los pasos 1 y 2 se implementan en OpenImpact y Elcor, respectivamente. Además, hemos usado un módulo de Elcor denominado Instruction Set Extension, para parte del paso 2. Este módulo adicional ha sido desarrollado en la NU Singapore y contiene algoritmos y estructuras básicas para la enumeración de patrones de candidatos a instrucción.

Paso 3. Selección de extensiones de la ISA

La selección de las extensiones de la ISA es un programa aparte en el *toolchain*. En él se implementa un algoritmo voraz que se puede configurar con diferentes funciones heurísticas para la selección de nuevas instrucciones. Estas funciones intentan maximizar:

- La frecuencia de ejecución del candidato a nueva instrucción.
- El número de operaciones del candidato.
- El número de operaciones ajustado a la frecuencia de ejecución.
- El número de operaciones ajustado a la frecuencia, dividido por el área ocupada.

Generación de código y hardware

Paso 4. Generación de la descripción hardware

La generación de la descripción hardware se realiza mediante el *toolset* DWARV [13], que es un traductor de código C a código VHDL específico para MOLEN. Esta herramienta traduce las funciones precedidas por la directiva *to_dfg* al VHDL específico de las CCUs, de forma que puede ser fácilmente integrado en la plataforma MOLEN. Así, para cada nueva instrucción seleccionada, nuestro *toolchain* genera una función equivalente con las líneas de código fuente que incluyen la nueva instrucción, las cuales son anotadas con la directiva *to_dfg* para ser transformadas en la descripción hardware de la CCU. Por ejemplo, si queremos especializar MOLEN para que acelere el código que se muestra en la Figura 5.a, el *toolchain* generará automáticamente el código que se muestra en la Figura 5.b.

Esas líneas de código de la función pueden incluir algunas instrucciones que no han sido identificadas como parte de las nuevas instrucciones. Esto ocurre porque en la implementación actual la selección de nuevas instrucciones se ha establecido en un nivel de la representación intermedia, mientras que los pragmas se encuentran en alto nivel. Con esta estrategia automática, siendo menos estrictos con la

```
a) original_source_code_line
   /* Including the new instruction */

b) #pragma to_dfg
   int funct_example(int param_example)
   {  original_source_code_line }
   ...
   funct_example(param);

c) #pragma call_fpga CCU_funct_example
   int CCU_funct_example(int param_example)
   {  original_source_code_line }
   ...
   funct_example(param);
```

Fig. 5. (a) Código original. (b) Código para DWARV. (c) Código para el compilador de MOLEN.

precisión de los resultados, podemos generar y evaluar de forma rápida y con hardware real, las nuevas unidades funcionales.

Paso 5. Generación de código

La generación de código se realiza mediante el webcompiler de MOLEN. Ese compilador identifica las secciones de código que se ejecutarán en el área reconfigurable con la directiva *call_fpga*, y genera las instrucciones necesarias para iniciar la ejecución de la CCU. La entrada del compilador de MOLEN es un código C que nuestra implementación ha modificado automáticamente con pragmas y llamadas a las CCUs equivalentes a las nuevas instrucciones (véase la Figura 5.c).

V. Entorno de desarrollo

La identificación de candidatos a instrucciones se ha desarrollado con el soporte de la versión 4.0 de Trimaran.

El diseño de MOLEN se ha configurado con 64 KB de datos y 64 KB de instrucciones de memoria BRAM. Además, la frecuencia de la parte reconfigurable y el acceso a memoria se ha establecido a 100 MHz. Los procesos relacionados con la generación del bitstream de MOLEN se han realizado con la suite de diseño de Xilinx ISE 8.2. La placa configurada con este bitstream es de tipo XUP [14], formada por una FPGA Virtex II Pro y dos procesadores empotrados PowerPC 405.

Hemos probado el *toolchain* con una aplicación bioinformática, ClustalW, la cual realiza alineamiento de secuencias. Hemos limitado la evaluación solamente a la parte de ClustalW que más tiempo consume: el alineamiento por pares. Los datos de entrada de la evaluación se han extraído de Bioperf [15].

Las gráficas de la siguiente sección muestran cada

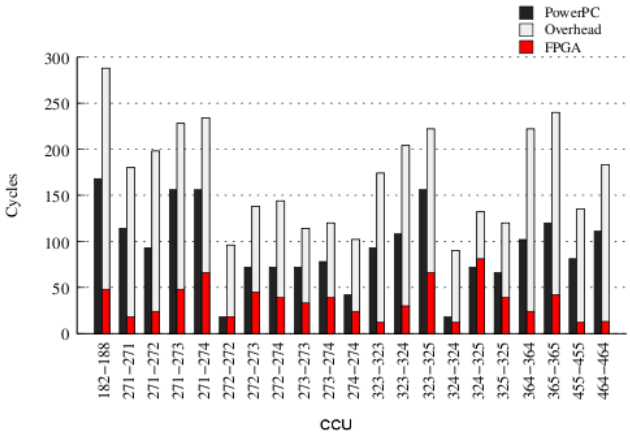

Fig. 6. Total de ciclos de ejecución para cada CCU y su equivalente software en el PowerPC.

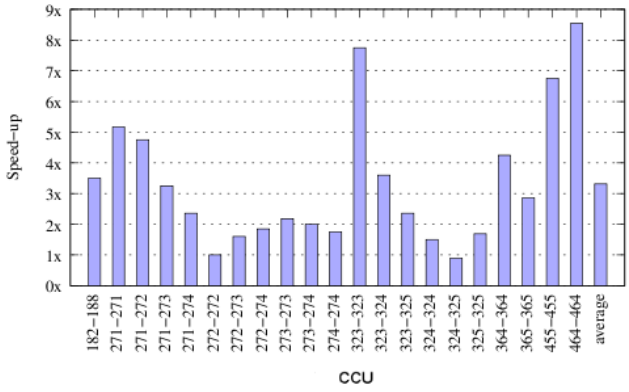

Fig. 7. Speed-up para cada CCU, en relación con la versión software.

nueva CCU identificada por las líneas reemplazadas en el código de alto nivel original. Por otra parte, la medición de ciclos se refiere a ciclos en el PowerPC.

VI. Evaluación

En la versión actual del *toolchain* no está permitido que una CCU tenga más de una variable de salida, ya que está limitado por las restricciones del traductor de VHDL, DWARV. Para obtener unidades funcionales más complejas, hemos unido líneas consecutivas correspondientes a diferentes CCUs, para formar una sola CCU.

La Figura 6 muestra el total de ciclos de ejecución (eje y) para cada CCU (eje x). Para cada CCU mostramos: en las barras negras, los ciclos de ejecución del código equivalente en el PowerPC; las barras mixtas indican, en gris, los ciclos de ejecución en la FPGA y en blanco el overhead de uso de las CCU, que se debe al tiempo de preparación de la ejecución en el coprocesador. En el modelo propuesto en este trabajo las unidades reconfigurables se encuentran dentro del procesador (Figura 1), por consiguiente se elimina el overhead de transferencia de datos.

La Figura 7 muestra el *speed-up* que cada CCU detectada puede obtener, en comparación con su equivalente software que se ejecuta en uno de los PowerPC de la FPGA. Tal y como puede verse, existe una considerable variación de *speed-ups* entre las diferentes CCUs, desde $1x$ hasta $8.54x$. Aquellas CCUs con los mejores *speed-ups* son originalmente código compuesto de diferentes operaciones aritméticas, o bien accesos a vectores que se encuentran en la memoria BRAM.

La Figura 8 muestra el *speed-up* (eje y) al utilizar una única CCU (eje x) para acelerar toda la aplicación, en relación a la aplicación software que se ejecuta en el PowerPC de la FPGA. Como podemos ver, una única CCU puede obtener hasta $1.19x$ de *speed-up* para toda la aplicación. En este caso, la frecuencia de ejecución de cada parte del código afecta los resultados del *speed-up*.

Finalmente, en la Figura 9 podemos ver el *speed-up* de toda la aplicación (eje x) que se obtiene para

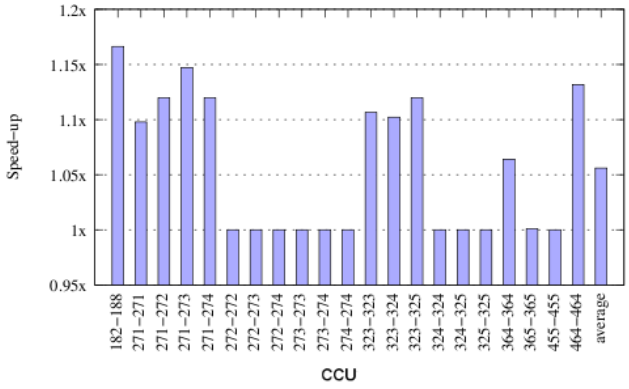

Fig. 8. Speed-up de toda la aplicación cuando una sola CCU es ejecutada.

algunas de las funciones heurísticas (eje y). Estas funciones maximizan: el número de operaciones (*Nodes* en la figura); la relación entre número de operaciones y frecuencia (*Freq-node*), y la relación entre número de operaciones, frecuencia y área (*Ratio*). Así mismo mostramos el caso ideal (*Ideal* en la figura) y la base del cálculo del *speed-up*, la ejecución software (*Software*). Entre las heurísticas, vemos que el mejor *speed-up* es de $2.15x$, mientras que el *speed-up* del caso ideal se establece en $2.25x$. Además, la influencia de la métrica del área para la selección no mejora, para este caso, el *speed-up*.

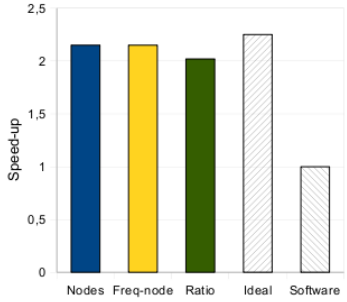

Fig. 9. Speed-up de toda la aplicación para cada heurística de selección.

VII. Conclusiones y trabajo futuro

En este artículo hemos propuesto una metodología de generación automática y evaluación de prototipos. La arquitectura de los prototipos está formada por una ISA base que se ejecuta en un GPP, y una ISA extendida que se ejecuta en unidades funcionales reconfigurables que se añaden al GPP base.

Con esta metodología podemos generar automáticamente hardware especializado para aplicaciones en un cierto dominio y evaluarlo sobre una plataforma real.

También hemos presentado resultados para una implementación preliminar de la metodología expuesta. Nuestra implementación ha utilizado Trimaran, y se ha basado en la plataforma reconfigurable MOLEN, junto con su webcompiler y la herramienta DWARV para la parte de evaluación en un hardware real.

Nuestros primeros resultados de las evaluaciones realizadas muestran que con el *toolchain* podemos obtener un *speed-up* por encima de 2x, para una aplicación bioinformática.

Para evitar las limitaciones de la implementación actual, se está trabajando en la migración de la plataforma de prototipos. La nueva plataforma estará basada en el procesador OpenSPARC T1, que cuenta con coprocesadores hardware y soporte de la ISA. Como la arquitectura que soporta Trimaran no incluye a SPARC, también se están portando los algoritmos de detección de extensiones de la ISA a un nuevo entorno. En este caso, la infraestructura elegida es el compilador LLVM, el cual genera objetos SPARC.

Agradecimientos

Nos gustaría agradecer a Yana Yankova de TU Delft por el soporte para usar su herramienta DWARV y a Pan Yu de NU Singapore por su ayuda con las extensiones de Trimaran.

Los investigadores de los centros BSC-UPC han sido financiados por el Ministerio de Ciencia e Innovación (no. TIN2007-60625, CSD2007-00050 y TIN2006-27664-E), el proyecto SARC de la Unión Europea (no. 27648), la red HiPEAC (no. IST-004408), el Xilinx University Program, y el proyecto MareIncognito bajo el acuerdo de colaboración BSC-IBM.

Referencias

[1] Carlo Galuzzi and Koen Bertels, "A framework for the automatic generation of instruction-set extensions for reconfigurable architectures," in *ARC '08: Proceedings of the 4th international workshop on Reconfigurable Computing*, Berlin, Heidelberg, 2008, pp. 280–286, Springer-Verlag.

[2] Gerald Aigner, Amer Diwan, David L. Heine, Monica S. Lam, David L. Moore, Brian R. Murphy, and Constantine Sapuntzakis, "The basic suif programming guide," Computer Systems Laboratory, Stanford University Stanford, CA., 2000.

[3] Cesare Alippi, William Fornaciari, Laura Pozzi, and Mariagiovanna Sami, "A dag-based design approach for reconfigurable vliw processors," in *DATE '99: Proceedings of the conference on Design, automation and test in Europe*, New York, NY, USA, 1999, p. 57, ACM.

[4] Kubilay Atasu, Laura Pozzi, and Paolo Ienne, "Automatic application-specific instruction-set extensions under microarchitectural constraints," *Int. J. Parallel Program.*, vol. 31, no. 6, pp. 411–428, 2003.

[5] "Trimaran: An infrastructure for research in backend compilation and architecture exploration," http://www.trimaran.org, August 2008.

[6] Nathan T. Clark and Hongtao Zhong, "Automated custom instruction generation for domain-specific processor acceleration," *IEEE Trans. Comput.*, vol. 54, no. 10, pp. 1258–1270, 2005, Member-Scott A. Mahlke.

[7] Bhuvan Middha, Varun Raj, Anup Gangwar, Anshul Kumar, M. Balakrishnan, and Paolo Ienne, "A Trimaran Based Framework for Exploring the Design Space of VLIW ASIPs with Coarse Grain Functional Units," in *Proceedings of the 15th International Symposium on System Synthesis*, 2002, pp. 2–7.

[8] Diviya Jain, Anshul Kumar, Laura Pozzi, and Paolo Ienne, "Automatically Customising VLIW Architectures with Coarse Grained Application-specific Functional Units," in *Proceedings of the 8th International Workshop on Software and Compilers for Embedded Systems*, 2004.

[9] Kubilay Atasu, Can Ozturan, Gunhan Dundar, Oskar Mencer, and Wayne Luk, "CHIPS: Custom Hardware Instruction Processor Synthesis," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 27, no. 3, pp. 528–541, March 2008.

[10] Pan Yu and Tulika Mitra, "Characterizing embedded applications for instruction-set extensible processors," in *DAC '04: Proceedings of the 41st annual conference on Design automation*, New York, NY, USA, 2004, pp. 723–728, ACM.

[11] Pan Yu and Tulika Mitra, "Scalable custom instructions identification for instruction-set extensible processors," in *CASES '04: Proceedings of the 2004 international conference on Compilers, architecture, and synthesis for embedded systems*, New York, NY, USA, 2004, pp. 69–78, ACM.

[12] S. Vassiliadis, S. Wong, G. N. Gaydadjiev, K.L.M. Bertels, G.K. Kuzmanov, and E. Moscu Panainte, "The molen polymorphic processor," *IEEE Transactions on Computers*, pp. 1363– 1375, November 2004.

[13] Y. D. Yankova, G.K. Kuzmanov, K.L.M. Bertels, G. N. Gaydadjiev, Y. Lu, and S. Vassiliadis, "Dwarv: Delftworkbench automated reconfigurable vhdl generator," in *In Proceedings of the 17th International Conference on Field Programmable Logic and Applications (FPL07)*, August 2007, pp. 697–701.

[14] "Xilinx university program," http://www.xilinx.com/univ/.

[15] David A. Bader, "An open benchmark suite for evaluating computer architecture on bioinformatics and life science applications," in *in Proceedings of the SPEC Benchmark Workshop 2006*, 2006.