Departamento de Informática
Universidad de Valladolid
Valladolid - España

XMLCetus y Sirius: análisis y traducción de código C utilizando herramientas XML

Sergio Aldea López, Diego R. Llanos, Arturo González-Escribano

Departamento de Informática, Universidad de Valladolid, España.
sergio@infor.uva.es, diego@infor.uva.es, arturo@infor.uva.es

Resumen Cetus es un eficiente compilador que permite realizar transformaciones de código C a código C. Se ha realizado una modificación de Cetus, llamada XMLCetus, que crea un árbol DOM en XML a partir del árbol de compilación generado por Cetus. Esto permite la utilización de una gran variedad de herramientas disponibles para la inspección y manipulación de árboles DOM en XML.
Además de XMLCetus, se ha desarrollado otra herramienta, denominada Sirius, que realiza la conversión desde el árbol DOM de un fichero XML que representa un programa en C el árbol DOM a un fichero fuente en código C, de funcionalidad equivalente al original.
Entre otras aplicaciones, estas herramientas nos permitirán utilizar la representación XML de un programa en C para detectar nichos de paralelización especulativa con herramientas de manipulación de árboles DOM, así como la reescritura de dichos árboles incorporando las funciones de invocación a nuestro motor de ejecución especulativa.

Informe Técnico IT-DI-2010-001

1. Introducción

Cetus [3,4,9] es un eficiente compilador de código C que permite realizar transformaciones fuente a fuente. Partiendo del software Cetus, se ha construído una versión modificada denominada XMLCetus, que crea un árbol DOM en XML a partir del árbol de compilación generado por Cetus. Esto permite la utilización de todas las herramientas disponibles para la manipulación e inspección de árboles DOM en XML, y que serán de utilidad en la futura inserción del motor de paralelización especulativa en Cetus. El objetivo consiste en insertar las funciones de paralelización especulativa de manera que modificando el fichero XML obtenido por XMLCetus se obtenga otro fichero XML con toda la nueva información y directivas de la paralelización especulativa.

Posteriormente, se desarrolla la herramienta Sirius que realiza la conversión desde el árbol DOM a un fichero fuente en código C, de funcionalidad equivalente al original.

La arquitectura del sistema propuesto sería la siguiente:

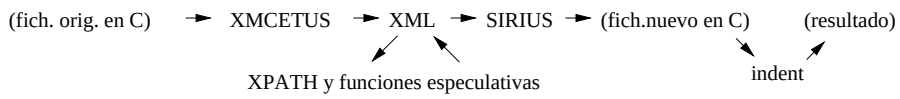

Figura1. IR de Cetus.

El fichero original en C sirve de entrada para la herramienta XMLCetus, que genera un fichero XML con toda la información del código original. En este punto, mediante las herramientas de manipulación de árboles DOM en XML se generaría un fichero XML modificado, con las directivas para la paralelización especulativa del código. Este fichero XML sirve de entrada para la herramienta Sirius, que construye un fichero en C de funcionalidad equivalente al original. Por último, y para facilitar la lectura de este fichero generado, se pasa el código por alguna herramienta de formato como *indent*.

La idea que subyace a todo el proyecto es poder utilizar estas herramientas para modificar automáticamente código escrito en C, con el fin de utilizar nuestro motor de paralelización especulativa [7,8,10], denominado SpecEngine, para paralelizar automáticamente programas en C, en concreto las pertenecientes al benchmark de aplicaciones de cálculo intensivo SPEC CPU2006.

Dado que Cetus (y, por extensión, XMLCetus) sólo funciona correctamente si el programa en C a procesar está compuesto de un único fichero, como paso previo a la construcción de XMLCetus y Sirius se han fusionado los programas en C del SPEC CPU2006 en un solo fichero, al objeto de comprobar que Cetus es capaz de analizarlos correctamente.

La estructura de este documento es la siguiente. En primer lugar se describe el proceso general de fusión de los benchmarks. El objetivo final es que los bench-

marks, una vez extraídos del entorno del SPEC CPU2006 y fusionados cada uno en un fichero único, tengan el mismo funcionamiento que tenían bajo la suite. En segundo lugar se describe la herramienta XMLCetus, junto con una descripción del software Cetus en la que se explica brevemente su funcionamiento. A continuación se describe la herramienta Sirius. Por último se incluyen dos apéndices. En el apéndice A se encuentran recogidos los diferentes problemas que surgieron durante la fusión de los ficheros fuente de los benchmarks, así como la solución tomada para resolver el problema. En el apéndice B se recogen los códigos fuente de las diferentes ficheros que fueron utilizados en las pruebas de regresión de las herramientas.

2. Tratamiento Previo de los Benchmarks a utilizar

2.1. Motivación

El motor de paralelización especulativa desarrollado funciona solamente con programas que se compilan a partir de un único fichero. Además el funcionamiento de Cetus se ve comprometido cuando se le pasan múltiples ficheros, por las relaciones y dependencias que se producen entre ellos. Por tanto, para poder utilizar este motor sobre las aplicaciones C del SPEC CPU2006 previamente es necesaria su fusión en ficheros únicos.

2.2. Proceso general de fusión

Los benchmarks que utilizan C como lenguaje fuente son los siguientes:

- 400.perlbench
- 401.bzip2
- 403.gcc
- 429.mcf
- 433.milc
- 456.hmmer
- 458.sjeng
- 462.libquantum
- 464.h264ref
- 470.lbm
- 482.sphinx3

Tres de los 11 benchmarks 429.mcf, 458.sjeng y 470.lbm) se encontraban ya fusionados previamente, como resultado del trabajo realizado en [1,2]. Además su fusión no ocasionó ningún problema, ya que no fue necesario utilizar nada más que el comando UNIX `cat`.

Para unir los restantes benchmarks, se los ordenó en función de los siguientes criterios:

1. En primer lugar se comenzó por aquellas aplicaciones cuyo funcionamiento fuera del SPEC estaba comprobado, trabajo realizado anteriormente. Estos benchmarks son:
   - 401.bzip2
   - 433.milc
   - 462.libquantum
2. En segundo lugar se ordenaron los benchmarks siguientes en función de su tamaño en número de líneas.
   - 464.h264ref
   - 456.hmmer
   - 482.sphinx3
3. Por último, los dos benchmarks más complejos, tanto en número de líneas, como en número de ficheros y complejidad estructural:
   - 400.perlbench
   - 403.gcc

El proceso general de fusión se realizó de forma progresiva, añadiendo en cada etapa un nuevo fichero con código fuente y realizando un pase completo de compilación. De este modo fueron detectándose y resolviéndose los problemas generados por la última fusión, principalmente debido a dependencias en las definiciones de macros e inclusiones de ficheros de cabecera. El apéndice A describe los problemas concretos encontrados en la fusión de cada benchmark.

3. XMLCetus: Conversión de la Representación Intermedia de Cetus a XML

3.1. Cetus

El proyecto Cetus [3,4,9] está desarrollado por la Universidad de Purdue (Indiana,USA), escrito en lenguaje Java y se encuentro bajo *Licencia Artística Modificada* [6], que permite la distribución de las modificaciones de su software, habiendo de ser públicas y conocidas por la Universidad de Purdue.

Cetus es un traductor que permite la transformación de código fuente C a código fuente C (no genera ficheros ejecutables), y que, entre otras funciones, permite la paralelización automática de bucles en tiempo de compilación mediante la introducción de directivas OpenMP [5], detectando variables privadas y compartidas.

La herramienta que utiliza Cetus como reconocedor e intérprete de la gramática C es ANTLR [11], cuyos orígenes se sitúan también en la Universidad de Purdue.

Cetus trabaja con una Representación Intermedia (en adelante IR) que refleja la estructura de bloques de un programa en C. En Cetus, el concepto de sentencias y expresiones es muy cercano a la sintaxis del lenguaje C, facilitando la traducción fuente-a-fuente. Sin embargo, esto aumenta la complejidad para los escritores (en términos de sintaxis de C) y limita la extensibilidad de Cetus

a otros lenguajes. Este problema es mitigado en parte mediante la provisión de múltiples clases abstractas, que representan construcciones genéricas de control.

La IR de Cetus (Figura 2) se encuentra construida a partir de un conjunto de clases, las cuáles se encuadran dentro de una jerarquía de clases.

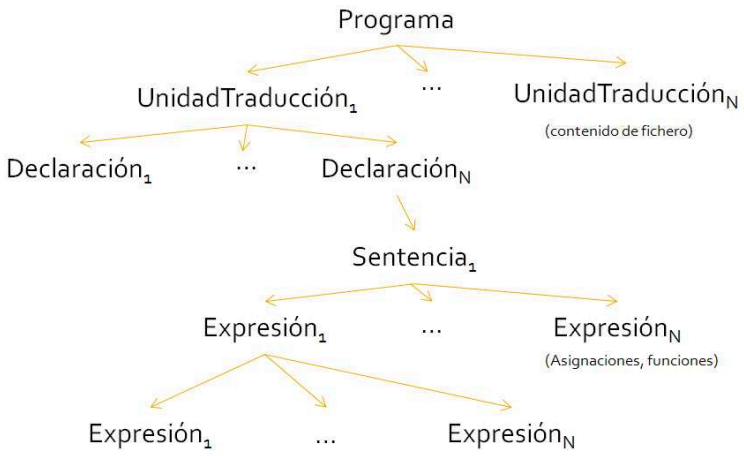

Figura2. IR de Cetus.

La estructura de la IR es independiente de la jerarquía de clases. Por ejemplo, en la IR siempre aparecerá el nodo *Program* como nodo padre de uno o varios nodos *TranslationUnit*, pero sin embargo *Program* y *TranslationUnit* no tienen un nodo padre en la jerarquía de clases.

En la Figura 3 se puede ver parte de la IR de un código fuente C.

3.2. XMLCetus

Partiendo de Cetus se trata de incorporar el motor de paralelización especulativa al mismo, que permite la extracción de paralelismo de bucles no analizables en tiempo de compilación. Para ello se ha optado en una primera fase de desarrollo por crear un árbol DOM en XML a partir del árbol de compilación generado por Cetus, ya que la extracción eficiente de este árbol permite la utilización de todas las herramientas disponibles para la manipulación e inspección de árboles DOM en XML.

Posteriormente, se desarrolla la herramienta Sirius que realiza la conversión desde el árbol DOM al fichero fuente original, una vez transformado por Cetus.

La creación del árbol DOM se realiza sobre el mismo software de Cetus, introduciendo una nueva clase Java de nombre *Xml* que realiza estas funciones. Esta modificación del código original de Cetus se denomina XMLCetus.

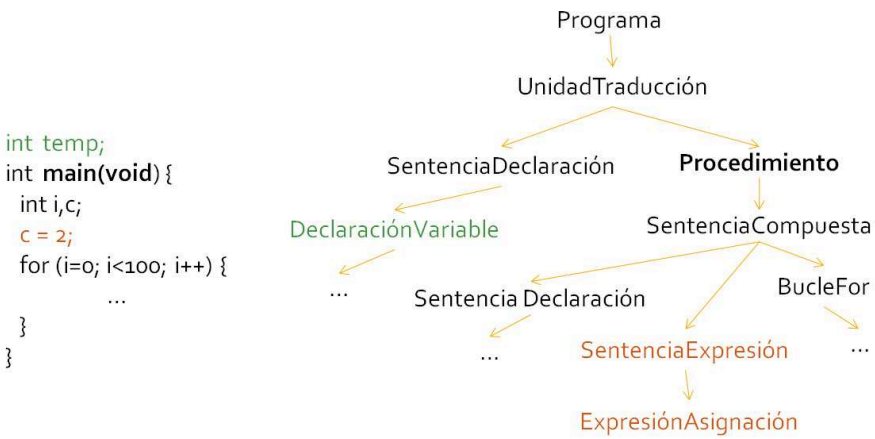

Figura3. Ejemplo de IR.

Desarrollo

En un primer momento se pensó en crear un árbol por cada uno de los ficheros que se le pasara a Cetus, obteniendo por tanto un fichero de salida por cada uno. Sin embargo, se prefirió mantener la estructura original de la representación intermedia de Cetus, en el que cada fichero, representado como un nodo *TranslationUnit*, descienda de un único nodo padre: *Program*, y por consiguiente perteneciendo al mismo árbol.

Por otro lado, aunque lo más directo era modificar alguna de las clases Java ya existentes de Cetus, se optó por desarrollar una clase Java con la nueva funcionalidad e importarla en las clases necesarias como un paquete independiente. En este caso, la clase que tuvo que modificarse para soportar la nueva funcionalidad fue la clase *Driver*, que supone el punto de entrada a Cetus. El punto de modificación se sitúa exactamente después del momento en el que Cetus ha terminado el reconocimiento de código C y ha generado el árbol de Representación Intermedia. En ese punto, se llama a la función *createDomTree()* de la clase creada, pasándole la referencia al primer nodo del árbol: el nodo *Program*. Esta función crea un documento DOM con el árbol XML, de manera que con una llamada a la función *getDocument* es posible recuperar este documento DOM, y mediante la función *printDomTree()* se consigue imprimir el documento DOM sobre un fichero de salida, de nombre `xml.out`.

En la representación intermedia de Cetus, están presentes nodos de todas las categorías. Por ejemplo: Un nodo *FloatLiteral* y un nodo *CharLiteral* son siempre descendientes de un nodo *Literal*. Esto plantea la posibilidad de almacenar en el árbol DOM únicamente los nodos de jerarquía más baja. Sin embargo, y a fin de

respetar la representación original de Cetus y pensando en futuras posibilidades, se decidió crear un nodo por cada uno de los presentes en el árbol de Cetus.

En relación con la cuestión anterior surgió un problema: determinados nodos podían ser instancia de una superclase pero no ser instancia de ninguna de sus subclases (según herencia Java). Por ejemplo: un literal entero es instancia de *Literal* y de su subclase *IntegerLiteral* y sin embargo, el valor inicializador de una variable es instancia de *Initializer* pero no de alguna de sus subclases: *ConstructorInitializer*, *ListInitializer* y *ValueInitializer*. A la hora de desarrollar la herramienta y el algoritmo que recorre el árbol es necesario tener en cuenta esta consideración.

Según está implementado por Cetus, nodos padres e hijos almacenan información redundante. Por ejemplo un nodo padre tiene como atributo la cadena `i=0`, mientras que sus nodos hijos representan esa información a través de su estructura y categoría. Por tanto, a la hora de representar la información en el árbol DOM se recoge únicamente la mínima necesaria.

El procedimiento de transformación de la representación intermedia de Cetus a un árbol DOM en XML es el siguiente:

1. Comenzando por el primer nodo (nodo *Program*, del que descienden tantos nodos *TranslationUnit* como ficheros se pasen a Cetus, en este caso sólo uno) se van obteniendo los descendientes de cada uno de los nodos [1] mediante un método recursivo. La forma de recorrer el árbol es *Primero en profundidad en preorden*.
   Para obtener los nodos hijo de cada nodo se utiliza la función *getChildren()*, que devuelve una lista con esos nodos. Mediante una operación de casting, esa lista pasa a ser una lista de objetos de tipo *Transversable*, definido por Cetus como la clase genérica para cualquier tipo de nodo.
2. A continuación se comprueba la categoría de los nodos que se van obteniendo. Esto se realiza mediante un bucle que recorre cada uno de los nodos hijo obtenidos en el punto anterior. Para cada nodo se verifica, mediante sucesivas comprobaciones, de qué clase es instancia (operador *instanceof* de Java). Un nodo puede ser instancia de una clase y de la clase padre de ésta (se crean elementos DOM que representan ambos nodos). Pero también, un nodo puede ser instancia de una superclase y no serlo de ninguna de sus subclases. Esto se tiene en cuenta a la hora de crear los nodos DOM y establecer sus relaciones.
3. Cuando la comprobación de la instancia del nodo es positiva con determinada clase se crea un objeto instancia de la misma clase, a partir de dicho nodo y mediante la función *get()* de la lista y una operación de casting.
   En función de su categoría se crea un elemento DOM (función *createElement()*) de mismo nombre que el nodo de la IR de Cetus, asignándole los atributos (función *setAttribute()*) necesarios y correspondientes según la categoría del nodo original.

[1] Determinados nodos del árbol Cetus no se han tenido en cuenta al tratarse exclusivamente de nodos para representar construcciones de C++.

4. Por último, después de la creación de cada elemento, éste se engancha a su correspondiente elemento padre en el árbol DOM mediante la función *appendChild()* del objeto que representa al elemento padre.

Después de todo este proceso se genera un fichero en XML que representa este árbol DOM. Utilizando este fichero como entrada de la herramienta Sirius es posible reconstruir el código fuente original transformado por Cetus.

3.3. Problemas y errores derivados de Cetus y XMLCetus

Una vez fusionados los ficheros de los benchmarks en uno sólo, estos debían formar parte del conjunto de pruebas por el que se pasaba a XMLCetus. Como consecuencia de estas pruebas se produjeron diversos errores, que tuvieron que ser solucionados modificando en determinados aspectos los ficheros fuente fusionados. Alguno de estos errores son provocados por la inclusión del código de las cabeceras del fichero fuente original en el fichero fuente generado por Sirius, ya que reconstruye el código a partir del árbol generado por Cetus y éste incluye nodos correspondientes a las cabeceras.

- El primero de los problemas encontrados es que Cetus, al transformar el código, no acepta como entrada la introducción de definiciones para el compilador (introducidas en el comando de ejecución con la opción `-D`), necesarias en la mayoría de los benchmarks analizados. Esto provocaba que la transformación no fuera llevada a cabo con éxito. Para evitar este problema fue necesario crear un fichero de cabecera (`compiler.h`) en que el se encontraran estas definiciones en su forma `#define`.
- Otro de los errores que introduce Cetus es el renombramiento de las constantes o variables con palabras que empiezan por el nombre del fichero. Si al fusionar todos los ficheros en uno sólo, éste recibe un nombre que comienza por un número (por ejemplo `458sjeng.c`), el renombramiento tomará como base el nombre de fichero y provocando un error posterior, puesto que el nombre de variables, constantes, etc. no puede comenzar por un número.

El apéndice B contiene las pruebas de regresión utilizadas para la detección de errores en el funcionamiento de XMLCetus y Sirius.

4. Sirius: Conversión de XML a C

4.1. Descripción

Una vez obtenido el fichero de salida XML (`xml.out`) con el árbol DOM de XMLCetus, éste puede transformarse en otro formato. La mejor forma de realizar este proceso es mediante una transformación XSL o XSLT. Por tanto, se construye una hoja de estilo XSLT con tal propósito, definiendo reglas de plantilla (o patrones de transformación) a fin de transformar las apariciones de

nodos DOM en código fuente compilable y equivalente al código fuente original tras la transformación de Cetus.

Para procesar esta hoja de estilo (`xslC.xslt`) se eligió *Saxon* [12] por su carácter *opensource* e implementar la especificación 2.0 de XSLT y XPath, útil para trabajos posteriores.

Hay dos formas de ejecutar Sirius:

1. **Ejecutando la transformación desde la línea de comandos**. Directamente a partir del fichero XML de salida y la hoja de estilo XSLT desarrollada se realiza la transformación. Esta forma no genera un fichero fuente de salida; únicamente muestra por pantalla el resultado de la transformación.

   ```
   $java net.sf.saxon.Transform xml.out xslC.xslt
   ```

2. **Compilando la transformación, generando un ejecutable**. Se elabora un fuente Java que compilado ejecuta la transformación XSLT sobre el fichero `xml.out` y genera a partir del árbol un fichero fuente de nombre `prueba.c` si no se indica lo contrario pasando un argumento al ejecutable. El script que controla la ejecución de Sirius es el siguiente:

   ```
   #!/bin/bash
   if [ $# = 0 ]
   then
      javac XmlToC.java; java XmlToC prueba.c; indent prueba.c; more prueba.c
   else
      javac XmlToC.java; java XmlToC $1; indent $1; more $1
   fi
   ```

   El funcionamiento del código Java consiste en la creación de un documento a partir del fichero XML mediante una clase Factoría, y que posteriormente se transforma utilizando la hoja de estilo desarrollada.
   Dado que la hoja de estilo creaba un fichero fuente con un formato ilegible, es necesario pasar dicho fichero fuente de salida por un comando como `indent` que aplica un formato que permite la lectura del mismo.

Al igual que ocurría con la herramienta XMLCetus, a la hora de elaborar las reglas de plantilla podía podía plantearse la alternativa de trabajar con los atributos de los nodos padre o con la información y estructura de los nodos hijo. Dado que en el caso de XMLCetus se optó por la opción de mantener el aspecto de árbol y guardar la información en los nodos de más baja jerarquía, en Sirius se trabaja con estos nodos.

La estructura de la hoja de estilo desarrollada consiste en un conjunto de reglas de plantilla, una por cada uno de los elementos del árbol DOM que deben ser transformados a construcciones del lenguaje C. Las reglas de plantilla "escriben" el código C correspondiente al elemento DOM que identifican y especifican la aplicación de otra plantilla donde sea necesario. Por ejemplo, en el caso de un elemento *BinaryExpression*, la regla de plantilla correspondiente "escribe" el operador binario según el valor del atributo correspondiente y determina la aplicación de las reglas de plantilla correspondientes a las expresiones del lado izquierdo y derecho del operador binario.

4.2. Problemas y errores derivados de Sirius

Surgió un problema con el `indent` utilizado para dar formato a los ficheros fuente salida de Sirius. El comando `indent` al encontrarse con un identificador que comienza por un número, separaba la parte numérica del resto. Así sucede en el benchmark *462.libquantum* con la expresión `1.0IF`, separada a `1.0 IF`. Como caso excepcional se optó por modificar el código fuente salida de Sirius, volviendo a unir cada una de las 6 apariciones de dicho identificador en el código.

El apéndice B contiene las pruebas de regresión utilizadas para la detección de errores en el funcionamiento de XMLCetus y Sirius.

5. Conclusiones y trabajo futuro

El desarrollo de este trabajo supone una gran ventaja en el tratamiento de la Representación Intermedia de Cetus, ya que su transformación a un árbol XML permite utilizar XPath y XQuery, que no sólo proporcionan una gran potencia y permiten simplicidad en las búsquedas, sino que también constituyen un estándar, con una gran comunidad de usuarios detrás.

Otro de los rasgos de este trabajo es la demostración de la gran versatilidad que proporciona XSLT, en un principio diseñado para realizar transformaciones hacia lenguajes de marcas (HTML) y que sin embargo se muestra muy útil en otro tipo de lenguajes como C, con características ciertamente lejanas de los lenguajes de marcas.

La elaboración de una herramienta capaz de transformar la información almacenada en un fichero XML generando un fichero en código C, cuya funcionalidad es equivalente al fichero fuente original que originó el árbol DOM supone un paso imprescindible en la posterior inserción del motor de paralelización especulativa en Cetus y en la generación de códigos que contengan directivas de paralelización provenientes de la correcta modificación del árbol DOM.

El plan de trabajo futuro pasa por aprovechar las herramientas de manipulación e inspección de árboles DOM en XML (XPath y XQuery) para desarrollar una aplicación que sea capaz de extraer información del código fuente, a partir de su representación en XML. Información relevante sería la detección de bucles, identificación de variables privadas y compartidas en los mismos, así como cuáles de esos bucles son paralelizables y cuáles son susceptibles de ser paralelizados mediante técnicas especulativas. La introducción de directivas que indiquen esta información dentro del propio código fuente sería el siguiente paso, es decir, realizar anotaciones que contengan dicha información sobre los elementos del árbol DOM.

Por otra parte, es deseable que este funcionamiento añadido pueda ser insertado dentro de Cetus como una funcionalidad más, dando la opción al usuario de utilizarla.

Por último, el objetivo final consiste en insertar el motor de paralelización especulativa dentro de Cetus, creando un software mucho más potente, siendo

capaz no sólo de paralelizar automáticamente, como hace hasta ahora, sino también de aplicar las técnicas de paralelización especulativa que permite extraer un mayor rendimiento sobre determinadas aplicaciones.

A. Problemas y errores surgidos en la fusión

A.1. Problemas comunes a todos los benchmarks

A pesar de que cada uno de los benchmarks constituye una aplicación única e independiente de las demás, en el proceso de extracción de estos de la suite surge un problema común. Las aplicaciones se compilan a partir de múltiples ficheros, de modo que cada uno de ellos incluye los ficheros de cabecera que le sean necesarios. En algunas ocasiones, la presencia de cabeceras repetidas en el fichero fusionado provocaba errores de redefinición. Por ello, se quitaron las cabeceras repetidas de estos ficheros, dejando únicamente la primera aparición de la misma.

A.2. 401.bzip2

1. Se cambia la definición `#define Bool unsigned char` de `spec.c` por `#define Boole unsigned char` (adoptando este cambio en donde sea necesario), ya que entra en conflicto con el tipo definido en `bzlib_private.h`.
2. Se elimina la definición de la función `myfeof` del fichero `bzlib.c` porque ya está definida en el fichero `bzip2.c`.

A.3. 433.milc

1. Hay conflicto entre las definiciones de `MAX_LENGTH` y `MAX_NUM`, definidas en las cabeceras `gauge_action.h` y `quark_action.h` con valores diferentes. Por ello, se cambian las definiciones de la cabecera `gauge_action.h` a `MAX_LENGTH_GAUGE` y `MAX_NUM_GAUGE` respectivamente, así como en aquellos lugares en los se utiliza dicha versión de la definición.
2. La función `r_parallel` está definida como

   ```
   void r_parallel (gauge_file * gf)
   ```

   y sin embargo su prototipo se declara como

   ```
   void r_parallel (gauge_file *, field_offset)
   ```

   Por ello se cambia el prototipo a

   ```
   void r_parallel (gauge_file *)
   ```

A.4. 462.libquantum

1. Las cabeceras `quantum.h` y `matrix.h` tiene la misma definición:

   ```
   struct quantum_matrix_struct {
     int rows;
     int cols;
     COMPLEX_FLOAT *t;
   };
   ```

De manera que para evitar el error se elimina dicha definición de una de las cabeceras. En este caso se optó por `matrix.h`.

2. Para evitar conflictos en declaraciones, las siguientes declaraciones de `matrix.h` fueron movidas a `quantum.h`:

```
#define M(m,x,y) m.t[x+y*m.cols]

extern unsigned long quantum_memman(long change);
extern quantum_matrix quantum_new_matrix(int cols, int rows);
extern void quantum_delete_matrix(quantum_matrix *m);
extern void quantum_print_matrix(quantum_matrix m);
```

3. Puesto que definen varias cosas idénticas, y a partir de esos elementos se definen otros, el contenido de la cabecera `qureg.h` se copia en el fichero `quantum.h`, eliminando aquellas líneas comunes y dejando las declaraciones que se realizan a partir de ellas.

A.5. 464.h264ref

1. Se deshacen las macros siguientes del fichero `biariencode.h`:
   - Elow
   - Erange
   - Ebits_to_follow
   - Ebuffer
   - Ebits_to_go
   - Ecodestrm
   - Ecodestrm_len

   Es decir, se añade `eep->` delante de cada una de las construcciones anteriores.
2. Se define `CONTEXT_INI_C` en la línea anterior a la inclusión del fichero de cabecera `ctx_tables.h`, y `INCLUDED_BY_CONFIGFILE_C` antes de `configfile.h`. Esto hace que las constantes de la cabecera sean definidas. Este error ocurre al quitar cabeceras repetidas del fichero general, que hace que desaparezcan las definiciones de ciertas constantes.
3. Error del tipo *undefined reference to* se soluciona incluyendo los primeros en el fichero general los fuentes `macroblock.c` y `configfile.c`.
4. Existen dos funciones idénticas, pero una de ellas con un argumento más que no utilizaba. Al fusionar los ficheros se producía un error de redefinición y por tanto se sustituye las apariciones de la función de más argumentos por la de menos.
5. Definición de `FILE *f` eliminada porque ya se encuentra definida anteriormente como `STATIC`
6. La definición `#define P_X` del fichero `transform8x8.c` se cambia por `#define P_X2`, así como en el resto de las apariciones de la definición, ya que se encuentra ya definida en el fichero `block.c` con otro valor.

A.6. 456.hmmer

1. Al unir varios ficheros en uno solo se producen errores de redefinición, al utilizar variables con el mismo nombre. Por tanto se cambia el nombre de las variables en todo el entorno que sea afectado (originalmente el fichero de procedencia):
   - `banner` a `bannercalibrate` y `bannersearch`.
   - `usage` a `usagecalibrate` y `usagesearch`.
   - `experts` a `expertscalibrate` y `expertssearch`.
   - `OPTIONS` a `OPTIONScalibrate` y `OPTIONSsearch`.
   - `NOPTIONS` a `NOPTIONScalibrate` y `NOPTIONSsearch`.
   - `main_loop_serial` como `main_loop_serialcalibrate` y `main_loop_serialsearch`.
   - En `struct.h` se cambia `#define END MATCH` (MATCH es 0) a `#define END 0` para que coincida con la definición de `hmmer.c`.
   - `V20magic` de `ssi.c` a `V20magicssi`.
   - `V20swap` de `ssi.c` a `V20swapssi`.
2. Error del tipo *undefined reference to* se soluciona incluyendo en la parte inicial del fichero general el fuente `ucbqsort.c`.
3. Las definiciones de las funciones:
   - regnode:
     ```
     static char * regnode(cp, op)
        register struct comp *cp;
        char op; {}
     ```
   - regc:
     ```
     static void regc(cp, b)
        register struct comp *cp;
        char b; {}
     ```
   - reginsert:
     ```
     static void reginsert(cp, op, opnd)
        register struct comp *cp;
         char op;
         char *opnd; {}
     ```

   son sustituidas por las de sus prototipos ya modificados:
   - regnode:
     ```
     static char *regnode(register struct comp *cp, char op);
     ```
   - regc:
     ```
     static void regc(register struct comp *cp, char c);
     ```
   - reginsert:
     ```
     static void reginsert(register struct comp *cp, char op, char *opnd);
     ```

   Esto soluciona un error en el que los prototipos no se correspondían con las funciones.

A.7. 482.sphinx3

1. Este benchmark tiene varios ficheros de cabecera en otro directorio: `/libutil`. Primero se mueven estos ficheros al directorio general y después se modifica el resto de los ficheros que pueden estar afectados. Dado que todas las cabeceras del directorio eran incluidas solo por el fichero `libutil.h`, solo es necesario aplicar el siguiente comando:

   ```
   $sed -e 's_<libutil/libutil.h>_<libutil.h>_' -i *.h *.c
   ```

   Esto hace que todas las rutas en las que aparecía el directorio, este desaparezca.

2. La presencia duplicada de `#include` en el fuente general provoca *conflicto de tipos*. Se soluciona eliminado las cabeceras repetidas.

3. Se cambia `BYTE_ORDER_MAGIC` a minúsculas. Soluciona un error *undefined reference to*

4. Se elimina una definición de función repetida: `NO_UFLOW_ADD`.

5. La línea `i=(int32) gnode_int32 (lgn);` daba un error. Para solucionarlo se resolvió la macro de la siguiente manera:

   ```
   i=((int)lgn->data.uint32);
   ```

   cambiando `int32` por `uint32`. Si esto no se hace se produce un *Segmentation Fault* al ejecutar.

A.8. 400.perlbench

1. Presencia de un `#define PERL_IN_<fuente>_C` al principio de cada fichero. Al juntarlo es necesario hacer `#undef PERL_IN_<fuente>_C` en la última línea de cada fichero correspondiente.

2. Errores del tipo *undefined reference to* en las declaraciones estáticas de funciones que tienen el prefijo `S_`. Se soluciona el problema eliminando dicho prefijo.

3. Error *undefined reference to 'PERL_HASH_INTERNAL'*. Se incluye su definición en el fichero `perlbench.c`. Debería de no dar de error, puesto que esta definido en `hv.h`, que es incluido por `perl.h`. Su definición depende de una variable, pero esta es definida antes de incluir `perl.h`.

4. Error del tipo *número de argumentos inválido*. Se cambia `yyparse()` (del fichero `pp_ctl.c`) por su definición `Perl_yyparse(aTHX)` (`embed.h`).

5. La cabecera `reg_comp.h` está incluida en `reg_comp.c`, definiendo antes `REG_COMP_C`. Como ya está incluida antes `reg_comp.h`, esto provoca errores de redefinición. Se soluciona introduciendo directamente en el texto la parte afectada por esa variable.

B. Pruebas de regresión

A continuación se muestran los códigos fuente de los ficheros en C que sirvieron como pruebas de regresión de las herramientas XMLCetus y Sirius. La utilización conjunta de ambas herramientas permiten que el código C del que consta la prueba se convierta a la representación XML y de ésta nuevamente a su representación en C, por lo que la salida esperada para cada prueba ha de coincidir con su entrada.

1. **t001-procedure.c**

```
int main(void ){}
```

2. **t002-procedureArguments.c**

```
int main(int arg1,int arg2){}
```

3. **t003-variableDeclarations.c**

```
int temp[4][3][5],tempp[5];
int main (int arg1, int arg2){
   int temp1;
}
```

4. **t004b-escapeCharacters.c**

```
int main(int arg1,int arg2){
   char x;
   x = 'a';
   x = '\n';
   x = '\t';
   x = '\b';
   x = '\f';
   x = '\v';
   x = '\'';
   x = '\?';
   x = '\a';
   x = '\0';
   //x = '\"';
   //x = '\\';
   //x = '\e';
}
```

5. **t004-variableInitial.c**

```
int tem[4] = {1,2,3,4};
signed int temp[4][3][5],tempp[5];
char temppp2 = 'a';
char temppp3[] = "prueba";
```

```
float tempf = 1.0;
int hex = 19968;
int oct = 24;

int main(int arg1,int arg2){
   int temp1;
}
```

6. **t005-define-typedef.c**

```
typedef int bool;
#define FALSE 0;
bool f = FALSE;

int main (int arg1, int arg2){
   int temp1;
}
```

7. **t006-ifStatement.c**

```
int main(int arg1,int arg2){
   int temp1 = 0;

   if (temp1 == 0) {
      printf("hola mundo: %d",temp1);
      temp1 = 1;
   } else {
      temp1 = 2;
   }
   temp1 = 3;
}
```

8. **t007b-ifElseStatement.c**

```
int alto = 1;
int bajo = 0;
int temp1;

int main(int arg1,int arg2){

   if (alto && (!bajo))
      temp1 = 0;
   else if (bajo && (!alto))
      temp1 = 1;
   else
      temp1 = 2;
}
```

9. **t007-ifElseStatement.c**

```
int main(int arg1,int arg2){
   int temp1 = 1;

   if ((temp1 >= 1) && (temp1 != 6))
   {
      printf("hola mundo: %d",temp1);
      temp1 = 0;
   } else if (temp1 == 2)
      temp1 = 0;
   else temp1 = 4;
}
```

10. **t008-forLoop.c**

```
int main(int arg1,int arg2){

   int i;
   for (i=1;i<2;i++)
      printf("Holamundo");
}
```

11. **t009-functionCall-Sum.c**

```
int main(int arg1){

   printf("holamundo%d",1);

   arg1 = (arg1 + 1);
}
```

12. **t010-assignmentOperators.c**

```
int main(void ){
   int x = 1;
   int y = 2;
   x = y;
   x += y;
   x -= y;
   x *= y;
   x /= y;
   x %= y;
}
```

13. **t010b-bitwiseOperators.c**

```
int f;
int main(int arg1,int arg2){
   int x;
   int y;
```

```
   int z;
   x &= 3;
   x |= 2;
   y ^= 4;
   z >>= 3;
   x <<= 2;
   x & (y << 2);
   (5 + 2);
   (y != 2);
   (x << y);
   z = (x << y);
   z = x & y;
   z = x | y;
   z = x ^ y;
   z = (~x);
}
```

14. **t010c-arithmeticOperators.c**

```
int main(void ){
   int i;
   int j = 1;
   int k = 1;

   char *  s = "hello";
   printf((( * s)+1));

   i = (((j * k)/ 3)- (-2));
   j = (k % 2);
   (j++);
   k = (++j);
}
```

15. **t011-annotationStatement.c**

```
/* An Annotation Statement */
   /* An another annotation
    * :D
    * :D
    */

}
```

16. **t012-whileLoop-breakStatement.c**

```
int main(void) {

   int num = 1;
```

```
   while (num <= 10) {
      if (num == 5)
         break;
      num = num + 1;
   }
}
```

17. **t013-whileLoop-continueStatement.c**

```
int main(void) {

   int num = 1;
   while (num <= 10) {
      if (num == 5)
         continue;
      num = num + 1;
   }
}
```

18. **t014-case-switchStatement-default.c**

```
int main (void){

   int num = 1;
   switch (num)    {
      case 1 : {
         num = 1;
         num = num + 1;
         break;
        }
      case 2 : num = 1;
         num + 1;
         break;
      case 3 : num = 3;
         break;
      default : num = 0;
         break;
   }
}
```

19. **t015-gotoStatement-returnStatement.c**

```
int main (void){
   int i;
   if (i==1) {
      i = 1;
      goto etiqueta1;
   }
```

```
   else
      goto etiqueta2;

etiqueta1:
   i = i * 2;
   return i;
etiqueta2:
   i = i / 2;
   return i;

}
```

20. **t016-pointers.c**

```
int main (void){
   int mat[5][3], **p,*q;
   **p = mat;

}
```

21. **t017-arrayAccess-pointers.c**

```
int main() {

   char cad[] = "hello";
   char * p;

   int x = 10;         // entero
   int* iptr = &x;     // puntero a entero
   int** pptr = &iptr; // puntero a puntero a entero

   p = cad;            //Puntero 'p' apunta a 'cad'
   p = &cad[0];

}
```

22. **t018-multipleIndexArrayAccess.c**

```
int main() {
   int m[3][3];
   m[0][0]=2+5;
}
```

23. **t019-classDeclaration.c**

```
int main (void){
   struct persona {
      char nombre[31];
      unsigned long telefono;
```

```
   };

   struct persona persona2 = {"Mike Thomas",555458978};
}
```

24. **t020-doWhileLoop.c**

```
int main(void){
   int i=0;
   do {
      i = i + 1;
      i += 6;
   }
   while (i<3);
}
```

25. **t021-accessExpression.c**

```
int main (void){
   struct persona {
      char nombre[31];
      unsigned long telefono;
   } persona1;

   struct persona persona2 = {"Mike Thomas",555458978};

   persona1.telefono = 555695362;
}
```

26. **t022-accessExpression2.c**

```
int main (void){
   struct persona {
      unsigned long telefono;
      unsigned long dni;
   };

   struct persona persona1;

   struct persona *pt;
   pt = &persona1;

   pt->telefono = 555686868;
   (*pt).dni = 71568987;
}
```

27. **t023-enumeration.c**

```
int main (void) {

   enum cosas {tenedor, cuchillo};
   enum dias {lunes=1, martes, miercoles, jueves, viernes, sabado, domingo};
   enum days {monday=1, tuesday, wednesday, thrusday=7, friday, saturday, sunday};
   enum palo {oros, copas, espadas, bastos} carta1, carta2;

}
```

28. **t024-commaExpression.c**

```
int main (void){
   int i;
   for (i=0;i<=3;i++,printf("El valor de i",i),printf(" es %d\n",i));
}
```

29. **t025-conditionalExpression.c**

```
int main(void){
   int s,x;
   s = ( x < 0 ) ? -1 : (x * x, s + x);
}
```

30. **t026-infiniteForLoop.c**

```
int main(void) {
   int j=1;
   for(;;)
      printf("InfiniteLoop\n");
   for(;j<2;)
      printf("InfiniteLoop\n");
   for(;;j++)
      printf("InfiniteLoop\n");
}
```

31. **t027-offsetExpression.c**

```
int main (void) {
   int mynums[3]={12,5,27};
   int newnums[3];
   int i;

   for (i=0; i<3;i++){
      printf("%d\n", *(mynums+i) );
   }
   for (i=0; i<3; i++){
      *(newnums+i) = *(mynums + i);
   }
   return 0;
}
```

32. **t028-sizeofExpression.c**

```
int main (void) {
   long var_1;
   var_1 = sizeof(double);
   var_1 = sizeof(4*5);

}
```

33. **t029-statementExpression.c**

```
int main(void) {

#define maxint(a,b) ({int _a = (a), _b = (b); _a > _b ? _a : _b; })

   int i;
   i = maxint(2,5);
   printf("%d\n",i);
}
```

34. **t030-typecast.c**

```
int main(void) {
   int k;
   k = (int) 1.7 + (unsigned int) 0.6;
}
```

35. **t031-macros.c**

```
int main(void){
   int i;
#define WIN32
#ifdef WIN32
   i=0;
#else
   i=1;
#endif
}
```

36. **t032-procedureDeclarator.c**

```
unsigned int proc1(int ex1);
void proc2(int ex1,int ex2);

int main(void) {
   int i = 1, j=2;
   i = proc1(i);
   printf("%d\n",i);
   proc2(i,j);
```

```
}

unsigned int proc1(int ex1) {
   return ex1+2;
}

void proc2(int ex1,int ex2) {
   printf("%d\n",ex1*ex2);
}
```

37. **t033-procedureDeclarator2.c**

```
char* proc1();

int main(void) {
   printf("%s\n",proc1());
}

char* proc1() {
   char *p,*a[3] = {"uno","dos","tres"};
   p = a[0];
   return p;
}
```

38. **t034b-structInsideStruct.c**

```
int main(void) {

   struct parent{

      struct son{

         struct grandson{

            struct grandgrandson{
               char name[31];
               char surname[31];
            } grandgrandson1;

            char name[31];
            char surname[31];
         } grandson1,grandson2;

          char name[31];
          char surname[31];
       } son1,son2;
```

```
        char name[31];
        int dni;
        int birthdate[3];
    };

    struct parent Bryan = { {{{"Bryan Jr.Jr.Jr.","grandgrandson1"},
      "Bryan Jr.Jr.","grandson1"},"Bryan Jr.","son1"},
      {{{"Bryan Jr2.Jr.Jr.","grandgrandson2"},"Bryan Jr.2.Jr",
      "grandson2"},"Bryan Jr.2","son2"},"Bryan",
         36452489,{31,12,2009} };

    printf("%s\n",Bryan.son2.grandson1.grandgrandson1.name);
}
```

39. **t034-structInsideStruct.c**

```
int main(void) {

    struct parent{

       struct son{
          char name[31];
          char surname[31];
       } son1,son2;

       char name[31];
       int dni;
       int date[3];

    };

    struct parent Bryan = {{"Bryan Jr."},{"Bryan Jr2"},
                           "Bryan",555,{31,12,2009}};

    printf("%s\n",Bryan.son1.name);
}
```

40. **t035-structInsideStruct2.c**

```
int main(void){

    typedef struct{
       int day;
       int month;
       int year;
    }birthday;
```

```
   typedef struct{
      char name[50];
      struct birthday;
   }person;
}
```

41. **t037-includeShort.c**

```
int x;
#include <stab.h>
//#include <stdio.h>

int main(void){
   printf("Hello world");
}

//#include <utime.h>
```

42. **t038-includeLarge.c**

```
#include <stdio.h>

int main(void){
   printf("Hello world");
}
```

43. **t039-structOnly.c**

```
int main (void) {

   struct gconv;
   int i;

}
```

44. **t040-includeAfterInclude.c**

```
#include <stdio.h>
int main (void){
  int i;
#include <stab.h>
}

#include <xlocale.h>
```

45. **t041-ifWhileVariable.c**

```
int main (void) {
   static long condition = 1;
   int x;
```

```
   if (condition)
     x=1;

   while (condition)
     x=x+1;
}
```

46. **t042-nestedDeclarator.c**

```
int main (void) {
   struct aux1;
   typedef unsigned int size_t;
   typedef int (* nest) (struct aux1 *, const unsigned char *  * ,
                      const unsigned char * , unsigned char *  * ,
                         size_t * , int , int );
   void *(* bzalloc) (void *, int, int);
}
```

47. **t043-switchStatement-anidados.c**

```
int main (void){

   int num = 1,num2=1;
   switch (num)    {
      case 1 : switch(num2){
                  case 1: num=1;break;
                  case 2: num=2;break;
                  case 3: num=3;break;
               }
         num = num + 1;
         break;
      case 2 : num = 2;break;
      case 3 : num = 3;break;
      default : num = 0;
   }
   printf("num = %d\n",num);
}
```

48. **t044-switchStatement-complex2.c**

```
int main (void){

   int num = 4;
   switch (num) {
      {
         case 0:
         num = 77;
      }
```

```
      num = num + 1;
      printf("después de llave\n");
      break;

      {
         case 12 : num = 12;
      }
      break;

      {
         case 3 : num = 3;
            {
               case 4: num=4;
                  {
                     case 5: num=5;
                  }
            }
      }
      printf("después de anidamiento\n");
      break;
      {
         default : num = 0;
      }
      break;
   }

   printf("num = %d\n",num);
}
```

49. **t045-switchStatement-complex.c**

```
int main (void){

   int num = 4 ;
   switch (num) {
      {
         case 4: num = 4;
            {
               case 1 :
                  num = 77;
                  num = num + 1;
            }
            printf("después de llave\n");
            break;
            case 12 : num = 12;
      }
      break;
```

```
      case 3 : num = 3;
         break;
      default : num = 0;
   }

   printf("hola mundo %d\n",num);
}
```

50. **t046-vaArgExpression.c**

```
typedef __builtin_va_list __gnuc_va_list;
typedef __gnuc_va_list va_list;

int main(void ){
   int i;
   typedef __gnuc_va_list va_list;
   va_list args;
   i = __builtin_va_arg (args, int);

}
```

51. **t047-initializerInsideInitializer.c**

```
int main(void ){
   int vector[5] = {1,2,3,4,5};
   int matrix2D[2][3] = { { 1, 2, 3} , { 4, 5, 6} };

   int matrix3D[2][2][2] = { {{1,2},{1,2}},{{1,2},{1,2}} };
   int matrix4D[2][2][2][2] = { {{{1,2},{1,2}},{{1,2},{1,2}}} ,
                                {{{1,2},{1,2}},{{1,2},{1,2}}} };
}
```

52. **t048-operandBinary.c**

```
int main(void){

   struct named_milc_original_c_17610
   {
      unsigned long r0, r1, r2, r3, r4, r5, r6;
      unsigned long multiplier, addend, ic_state;
      double scale;
   };
   typedef struct named_milc_original_c_17610 double_prn;
   double_prn * prn_pt;
   register int t, s;

   prn_pt->scale*(t^((s>>8)&16777215));
}
```

53. **t049-warningCast.c**

```
/*warning: assignment makes pointer from integer without a cast*/
#include <stdio.h>;
int main(void){

   FILE * info_fp;
   char info_filename[256];
   int x=2;

   if (((info_fp=fopen(info_filename, "w"))==((void * )0)))
      printf("warning cast test");

   if (((x+=2))==(2+2))
      printf("another test");

}
```

54. **t050-nestedDeclaratorInitializer.c**

```
int main (void) {
   unsigned short (*get_pel) (int, int, int, int) = 3;
}
```

Referencias

1. Sergio Aldea. *Análisis de la capacidad de paralelización de compiladores comerciales y viabilidad del uso de técnicas especulativas sobre CPU SPEC 2006*. PFC. Universidad de Valladolid, Departamento de Informática, July 2008.
2. Sergio Aldea, Diego R. Llanos, and Arturo González-Escribano. Evaluación de compiladores comerciales usando SPEC CPU2006. In *Actas XIX Jornadas de Paralelismo, Castellón, Spain*, 17-19 September 2008.
3. Hansang Bae, Leonardo Bachega, Chirag Dave, Sang-Ik Lee, Seyong Lee, Seung-Jai Mind, Rudolf Eigenmann, and Samuel Midkiff. Automatic parallelization with cetus. Technical report, HPCLAB, ECE, Purdue University, 2008.
4. Hansang Bae, Leonardo Bachega, Chirag Dave, Sang-Ik Lee, Seyong Lee, Seung-Jai Mind, Rudolf Eigenmann, and Samuel Midkiff. Cetus: A Source-to-Source Compiler Infrastructure for Multicores. In *Proceedings of the 14th Int'l Workshop on Compilers for Parallel Computing, CPC*, 2009.
5. Gabriele Jost Barbara Chapman and Ruud van der Pas. *Using OpenMP. Portable Shared Memory Parallel Programming*. The MIT Press, October 2007.
6. Cetus Project. Modified Artistic License. http://cetus.ecn.purdue.edu/license_mod.html.
7. Marcelo Cintra and Diego R. Llanos. Toward efficient and robust software speculative parallelization on multiprocessors. In *PPoPP '03: Proc. of the 9th ACM SIGPLAN Symp. on Principles and Practice of Parallel Programming*, pages 13–24, San Diego, California, USA, June 2003. ACM Press. Publicado también en ACM SIGPLAN Notices, 38 (10), 13–24, octubre 2003, ISSN 0362-1340.
8. Marcelo Cintra and Diego R. Llanos. Design space exploration of a software speculative parallelization scheme. *IEEE Transactions on Parallel and Distributed Systems*, 16(6):562–576, June 2005. ISSN 1045-9219.
9. Chirag Dave, Hansang Bae, Seung-Jai Mind, Seyong Lee, Rudolf Eigenmann, and Samuel Midkiff. Cetus: A Source-to-Source Compiler Infrastructure for Multicores. *IEEE Computer*, 42(12):36–42, December 2009.
10. Arturo González-Escribano and Diego R. Llanos. Speculative parallelization. *Computer*, 39(12):126–128, December 2006. ISSN 0018-9162.
11. Terence Parr. *The Definitive ANTLR. Reference: Building Domain-Specific Languages*. The Pragmatic Bookshelf, May 2007.
12. Saxonica Limited. Saxonica. XSLT and XQUERY Processing. http://www.saxonica.com.