ELABORACIÓN DE UNA HERRAMIENTA EN LENGUAJE C QUE PERMITA IDENTIFICAR EL DESEMPEÑO DE UN ALGORITMO SOBRE UN FPGA A PARTIR DE SUS SENTENCIAS DE CONTROL

ANDRÉS FERNANDO PLATARRUEDA ACUÑA

PABLO ERNESTO VERBEL GONZÁLEZ

UNIVERSIDAD INDUSTRIAL DE SANTANDER

FACULTAD DE INGENIERÍAS FISICOMECÁNICAS

ESCUELA DE INGENIERÍAS ELÉCTRICA, ELECTRÓNICA Y DE TELECOMUNICACIONES

BUCARAMANGA

2012

ELABORACIÓN DE UNA HERRAMIENTA EN LENGUAJE C QUE PERMITA IDENTIFICAR EL DESEMPEÑO DE UN ALGORITMO SOBRE UN FPGA A PARTIR DE SUS SENTENCIAS DE CONTROL

Presentado por:

ANDRÉS FERNANDO PLATARRUEDA ACUÑA

PABLO ERNESTO VERBEL GONZÁLEZ

Dirigido por:

Director: MIE (c). SERGIO A. ABREO CARRILO

Director: MIE (c). CARLOS A. FAJARDO ARIZA

UNIVERSIDAD INDUSTRIAL DE SANTANDER

FACULTAD DE INGENIERÍAS FISICOMECÁNICAS

ESCUELA DE INGENIERÍAS ELÉCTRICA, ELECTRÓNICA Y DE TELECOMUNICACIONES

BUCARAMANGA

2012

TABLA DE CONTENIDO

LISTA DE FIGURAS

LISTA DE TABLAS

LISTA DE ANEXOS

RESUMEN

TITULO:

ELABORACIÓN DE UNA HERRAMIENTA EN LENGUAJE C QUE PERMITA IDENTIFICAR EL DESEMPEÑO DE UN ALGORITMO SOBRE UN FPGA A PARTIR DE SUS SENTENCIAS DE CONTROL[1].

AUTOR:

ANDRÉS FERNANDO PLATARRUEDA ACUÑA.
PABLO ERNESTO VERBEL GONZÁLEZ[2].

PALABRAS CLAVES:

FPGA, Lenguaje C, GPP.

DESCRIPCIÓN:

Este articulo muestra el desarrollo de una herramienta computacional que permite hacer una aproximación del desempeño de un algoritmo escrito en lenguaje C al ser implementado sobre un FPGA y evaluar la viabilidad de dicha implementación. El saber de manera previa si vale la pena implementar un algoritmo sobre un FPGA es muy valioso, ya que generalmente estos desarrollos implican bastante esfuerzo y tiempo. La herramienta fundamenta su análisis en la extracción de ciertos parámetros clave dentro del algoritmo, tales como el formato de los datos, la dependencia de los datos, numero de trasferencias de registros, número de operaciones aritméticas, las sentencias de control y su anidamiento. Estas características se extraen haciendo un barrido por las cadenas de caracteres del algoritmo y haciendo una análisis léxico y sintáctico de ellas. Se generó una figura de mérito que le asigna un porcentaje a cada característica extraída dependiendo de cuan favorable sea cada parámetro para la implementación y de esta forma obtener una aproximación de la viabilidad de que el proceso tenga buen rendimiento al ser implementado en un FPGA. La herramienta cuenta con una interfaz gráfica de usuario que permite introducir un algoritmo de entrada escrito en lenguaje C y obtener a la salida algunos parámetros de rendimiento.

[1] Trabajo de grado.
[2] Facultad de Ingenierías Físicomecánicas. Escuela Eléctrica, Electrónica y Telecomunicaciones. Director: MIE(c) Sergio A. Abreo Codirector: MIE(c) Carlos A. Fajardo.

ABSTRACT

TITLE:

DEVELOPMENT OF A COMPUTATIONAL TOOL IN C LANGUAGE THAT ALLOWS TO RECOGNIZE THE PERFORMANCE OF AN ALGORITHM ON AN FPGA FROM ITS CONTROL STATEMENTS [3].

AUTHOR:

ANDRÉS FERNANDO PLATARRUEDA ACUÑA.
PABLO ERNESTO VERBEL GONZÁLEZ[4].

KEY WORDS:

FPGA, C Language, GPP.

DESCRIPTION:

This paper shows the development of a computational tool, which gives an approximation of the application performance of an algorithm written in C language implemented on an FPGA and evaluate the feasibility of such implementation. Know whether it is worth make development of an algorithm in FPGA before implementing is very valuable, because generally these implementations demand considerable amount of time and development effort. The computational tool bases its analysis on the extraction of key parameters in the algorithm such as data format, data dependencies, data transfer, number of arithmetic operations, control statements and nested conditionals. These key parameters are related to intrinsic characteristics of the application. The intrinsic characteristics of the application are extracted by scanning characters of the algorithm and making a lexical and syntactic analysis of them. The paper shows the development of lexical and syntactic analyzers for the analysis of algorithm written in C language. This paper gives a figure of merit, assigning a percentage to each extracted intrinsic characteristic depending on how much it affects performance. Therefore, the computational tool provides a parameter that gives us the viability of acceleration of a process to be implemented in an FPGA. Finally, we developed a graphical user interface that allows anyone to easily interact with the tool obtaining at the output performance parameters of the algorithm in C language.

[3] Work Degree.
[4] Faculty of Physical-Mechanical Engineering. School of Engineerings Electrical, Electronic and Telecommunications. Directress: MIE(c) Sergio A. Abreo Codirectress: MIE(c) Carlos A. Fajardo.

Elaboración de una herramienta en lenguaje C que permita identificar el desempeño de un algoritmo sobre un FPGA a partir de sus sentencias de control

Andrés F. Platarrueda, Pablo E. Verbel, Sergio A. Abreo Carrillo y Carlos A. Fajardo

***Resumen*—Este articulo muestra el desarrollo de una herramienta computacional que permite hacer una aproximación del desempeño de un algoritmo escrito en lenguaje C al ser implementado sobre un FPGA y evaluar la viabilidad de dicha implementación. El saber de manera previa si vale la pena implementar un algoritmo sobre un FPGA es muy valioso, ya que generalmente estos desarrollos implica bastante esfuerzo y tiempo. La herramienta fundamenta su análisis en la extracción de ciertos parámetros clave dentro del algoritmo, tales como el formato de los datos, la dependencia de los datos, número de trasferencias de registros, número de operaciones aritméticas, las sentencias de control y su anidamiento. Estas características se extraen haciendo un barrido por las cadenas de caracteres del algoritmo y haciendo una análisis léxico y sintáctico de ellas. Se generó una figura de mérito que le asigna un porcentaje a cada característica extraída dependiendo de cuan favorable sea cada parámetro para la implementación y de esta forma obtener una aproximación de la viabilidad de que el proceso tenga buen rendimiento al ser implementado en un FPGA. La herramienta cuenta con una interfaz gráfica de usuario que permite introducir un algoritmo de entrada escrito en lenguaje C y obtener a la salida algunos parámetros de rendimiento.**

***Palabras Claves*— Computación de Alto Rendimiento, FPGA (Field Programmable Gate Array), Análisis de algoritmos en lenguaje C, GPP (General Purpose Processor).**

I. INTRODUCCIÓN

PARA la comunidad científica e industrial mundial uno de los grandes retos actuales es la computación de alto rendimiento, cuyo objetivo principal es agilizar el procesamiento de grandes volúmenes de información y que requieran algoritmos de gran complejidad para su tratamiento. Una de las formas de lograr buenos tiempos de cómputo es implementar paralelismo en el procesamiento de la información, el cual consiste en realizar varios cálculos computacionales simultáneamente. Para lograr este procesamiento paralelo una de las técnicas más llamativas actualmente es el uso de los FPGA como núcleos de procesamiento. [1]

Los FPGA tienen un gran potencial para aprovechar al máximo el paralelismo y con esto acelerar la velocidad de cómputo de algunos procesos, por lo que muchos investigadores se han dado a la tarea de migrar procesos a FPGA con la expectativa de que estos se aceleren. Uno de los inconvenientes a la hora de dar el paso de los GPP a los FPGA es que la implementación de un algoritmo sobre un FPGA es un proceso que demanda bastante tiempo y esfuerzo. Mientras que en un GPP podemos escribir un algoritmo en cuestión de días, esta misma implementación sobre un FPGA puede demorar meses e incluso más de un año, ya que esto implica hacer una migración del código en lenguaje C hacia lenguajes de descripción de hardware (HDL), como se puede ver en la Figura 1.

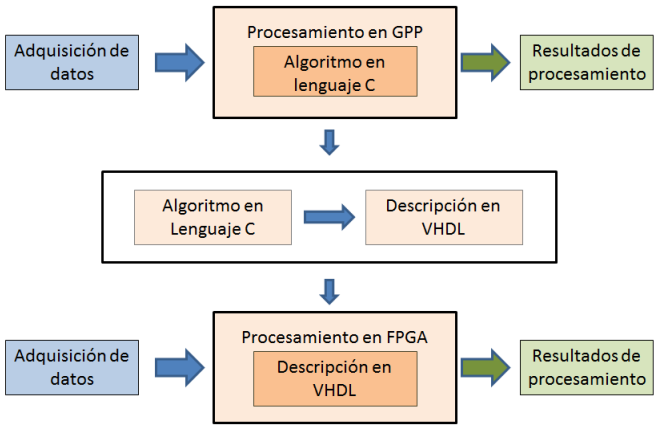

Figura 1: Diagrama de Migración de GPP a FPGA.

Por lo tanto si se pudiera predecir de cierta forma si la implementación sobre el FPGA va a acelerar el procesamiento sería de valiosa ayuda a la hora de hacer implementaciones sobre FPGA. Es por esto, que este trabajo de investigación pretende diseñar una herramienta que brinde un indicador de si el proceso va a acelerarse cuando se implementa sobre el FPGA, para tener un criterio de evaluación antes de hacer cualquier implementación.

Algunos de los trabajos publicados sobre procesadores

Andrés F. Platarrueda pertenece al grupo de investigación en Conectividad y Procesamiento de Señales, Universidad Industrial de Santander, Bucaramanga, Colombia. (e-mail: andresplatarrueda@gmail.com)

Pablo E. Verbel pertenece al grupo de investigación en Conectividad y Procesamiento de Señales, Universidad Industrial de Santander, Bucaramanga, Colombia. (e-mail:pablo_verbel@gmail.com)

Sergio A. Abreo es el director de proyecto. Ingeniero electrónico, Universidad Distrital Francisco José de Caldas, Colombia. MIE en ingeniería electrónica, Universidad Industrial de Santander, Colombia. (e-mail: abreosergio@gmail.com)

Carlos A. Fajardo es el codirector de proyecto. Ingeniero electrónico, Universidad Industrial de Santander, Colombia. MIE en ingeniería electrónica, Universidad Industrial de Santander, Colombia. (e-mail: cafajar@gmail.com)

específicos, implementados en un FPGA, en los que se han obtenido índices de aceleración superiores, comparados con los procesadores de propósito general son: análisis de secuencias genéticas (Puttegowda et ál., 2003; Bogdán et ál., 2008; Hoang y Lopresti, 1993), filtrado digital (Tessier y Burleson, 2001; Yamada y Nishihara 2001; Wang y Shen, 2008), criptografía (Patterson, 2000; Anghelescu et ál., 2008a; Anghelescu et ál., 2008), filtrado de paquetes de red (Sinnappan y Hazelhurst, 2001; Cho y Mangione-Smith, 2008, 2005), reconocimiento automático de objetos (Jean et ál., 1999; Villasenor et ál., 1996), identificación de patrones (Baker y Prasanna, 2004), entre otros. [2, pág. 2]

II. ALGORITMOS Y LENGUAJES DE PROGRAMACIÓN

Un algoritmo es un conjunto de instrucciones que se ejecutan de manera ordenada y en un lapso de tiempo finito, con el objeto de resolver un problema específico, generalmente a partir de datos de partida. Los algoritmos requieren de un lenguaje para ser descritos. Un lenguaje es un conjunto de símbolos y reglas que sirven para comunicarnos. Es importante conocer los símbolos, el orden y el significado que estos adquieren, tanto como para el que dice algo, como para el que debe interpretarlo. Dos aspectos que definen un lenguaje son la sintaxis (como se deben decir las cosas) y la semántica (que es lo que se dice). [3, pág. 8]

Los algoritmos pueden ser escritos en diversos lenguajes. Los lenguajes con los que se escriben algoritmos en la computadora o cualquier otro dispositivo programable pueden ser: lenguajes de máquina, lenguajes de bajo nivel y lenguajes de alto nivel. [4, pág. 1]

Los lenguajes de máquina son aquellos lenguajes interpretados directamente por el procesador, son propios de cada dispositivo, ya que están relacionados con el diseño del hardware de la máquina. Las instrucciones en lenguaje de máquina se expresan en términos binarios.

Los lenguajes de bajo nivel, también llamados lenguajes ensambladores, permiten al programador escribir instrucciones de un programa usando abreviaturas, también llamadas palabras nemotécnicas.

Los lenguajes de alto nivel son aquellos en los que las instrucciones son escritas con palabras similares a los lenguajes humanos, lo que facilita la escritura y comprensión del programa. Estos lenguajes son más pequeños, simples, precisos y tratan de eliminar las ambigüedades, para facilitar la creación de algoritmos.

Uno de los lenguajes de programación más usado y reconocido es el lenguaje C, el cual fue creado en 1972 por Dennis M. Ritchie en los Laboratorios Bell como evolución del lenguaje B. El lenguaje C es un lenguaje de propósito general y se puede considerar como un lenguaje de mediano nivel, ya que posee algunas cualidades propias de los lenguajes de alto nivel y otras de los de bajo nivel. Con el fin de garantizar la portabilidad de las aplicaciones escritas en C y eliminar las ambigüedades del lenguaje, ANSI publicó un estándar para C y se recomienda a los desarrolladores de software en C que acojan a los requisitos descritos en dicho estándar. [5]

III. FPGA

El concepto de arreglo de compuerta programable en campo (FPGA), nace en el año 1984 a manos de Ross Freeman y Bernard Vonderschmitt fundadores de Xilinx, una de las más grandes empresas que en la actualidad trabajan con FPGA. Los FPGA surgen como la evolución de los PLD (Programmable Logic Device), estos combinan la alta densidad y versatilidad de los Arreglos de Compuertas (Gate Arrays) con la popularidad y ventajas que ofrecían los PLD. [6, pág. 14]

Los FPGA no son más que un dispositivo semiconductor, compuesto en su interior por CLB (Cofigurable Logic Block) rodeados por bloques de entrada/salida y canales de enrutamiento. Los FPGA utilizan los lenguajes HDL (Hardware Description Lenguage) para hacer la descripción de los circuitos digitales que son implementados en ellos. VHDL y Verilog son los lenguajes HDL más ampliamente utilizados.

En los últimos años, los fabricantes de dispositivos lógicos programables han sabido aprovechar la progresión exponencial en la capacidad de los circuitos integrados. En la actualidad, un FPGA puede implementar sin problemas uno o más procesadores, una parte de la memoria del sistema a partir de bloques RAM internos, varios periféricos y aun así quedar espacio para incluir coprocesadores que aumenten la velocidad de las tareas de software. [7]

Actualmente una de las áreas de la electrónica digital que está en crecimiento es la migración del tratamiento de información de procesadores de propósito general a un FPGA con el fin de acelerar los tiempos de cómputo aprovechando el procesamiento paralelo de los FPGA e implementando algoritmos que aprovechen esta característica, logrando índices de aceleración anteriormente conseguidos únicamente en las supercomputadoras.

IV. CARACTERÍSTICAS INTRÍNSECAS DE LOS ALGORITMOS

Las características intrínsecas de la aplicación ponen un límite en el rendimiento que se puede obtener al implementar un algoritmo en un FPGA, ya que estas determinan la cantidad de paralelismo que existe, que tanto se puede explotar dicho paralelismo y la velocidad de reloj. Las características que tienen mayor impacto sobre el rendimiento de una aplicación son: la dependencia de datos, el tamaño de los datos, la complejidad aritmética de las operaciones y las sentencias de control. [8, pág. 442]

IV-A. DEPENDENCIA DE DATOS

La dependencia de datos se presenta cuando las instrucciones de un programa utilizan datos los cuales son resultado de otras instrucciones anteriores que aún no han sido completadas. Hay tres tipos principales de dependencia de datos, que son:

- RAW (Read After Write - leer después de escribir).
- WAR (Write After Read - escribir después de leer).
- WAW (Write After Write - escribir después de escribir). [11]

Algoritmos con grandes conjuntos de datos que tienen poca o ninguna dependencia de datos son ideales para ser implementados en un FPGA ya que permiten un alto rendimiento debido a la posibilidad de realizar muchos cálculos simultáneamente, y a la posibilidad de reprogramar las operaciones. [8, pág. 441]

IV-B. FORMATO DE LOS DATOS Y COMPLEJIDAD ARITMÉTICA

El formato de los datos y la complejidad aritmética son importantes porque influyen fuertemente en el tamaño del circuito y en la velocidad de procesamiento. El número de operaciones que se pueden desarrollar al mismo tiempo en un FPGA determina el límite de paralelismo que se puede alcanzar, por lo tanto, un formato de datos con muchos bits y operaciones aritméticas complejas se traduce en poco paralelismo ya que ocupan recursos significativos del FPGA como por ejemplo bloques de lógica y multiplicadores. La representación de los datos con el menor número de bits y la realización de cálculos con los operadores más simples por lo general conducen a un buen rendimiento. [8, pág. 442]

IV-C. SENTENCIAS DE CONTROL

Al implementar algoritmos en FPGA se puede conseguir mejor rendimiento si el algoritmo tiene pocas rutas de decisión definidas por sus sentencias de control. En pocas palabras, se necesita tiempo para tomar decisiones y los circuitos de toma de decisiones son a menudo la parte crítica de la implementación. [8, pág. 442]

V. DESCRIPCIÓN GENERAL DE LA HERRAMIENTA

La herramienta que analiza las características intrínsecas del algoritmo se desarrolló con software libre utilizando el sistema operativo Ubuntu. La siguiente es la lista del software que se utilizó:

- Flex: Generador de analizadores léxicos. [16]
- Bison: Generador de analizadores sintácticos. [17]
- Gambas: Un entorno de desarrollo que facilita la programación y diseño de interfaces gráficas. [18]
- GCC: Compilador C.

En adelante cuando se hable de la ejecución de comandos se hace referencia al uso de consola de Ubuntu.

La herramienta toma un algoritmo de entrada escrito en lenguaje C, y dos parámetros que el usuario debe proporcionar que son: el numero de iteraciones del ciclo while y el numero de transferencias de datos.

Se realiza un análisis para identificar parámetros dentro del algoritmo como son: identificadores, formato de datos, sentencias de control, operaciones y transferencias de registros o asignaciones. Este análisis se lleva a cabo por un algoritmo principal que está contenido dentro de un archivo ejecutable.

La herramienta cuenta con una interfaz gráfica con la que el usuario interactúa ingresando un algoritmo a analizar y posteriormente esta nos proporciona los resultados del análisis. Estos resultados pueden ser contrastados con la figura de mérito para tener una idea del rendimiento que se tendría al implementar la aplicación en un FPGA. En la Figura 2 se puede ver la descripción general de la herramienta.

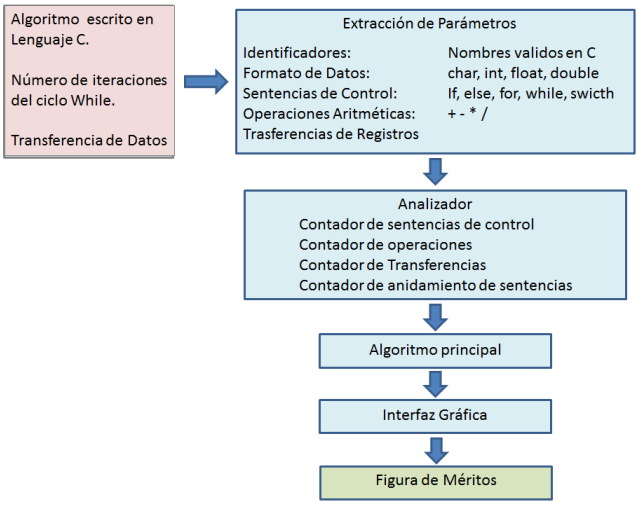

Figura 2: Descripción general de la Herramienta.

Las condiciones que debe tener el algoritmo de entrada se detallan en el ANEXO A.

VI. ELABORACIÓN DE LA HERRAMIENTA

La herramienta permite identificar parámetros de la aplicación para obtener información sobre la probabilidad de que al ser implementada en un FPGA tenga un buen rendimiento. Para lograr esto la herramienta analiza algoritmos escritos en lenguaje C extrayendo las siguientes características intrínsecas de la aplicación: la dependencia de datos, el formato de los datos, la cantidad de operaciones aritméticas y las sentencias de control dentro del algoritmo.

La herramienta tiene como entrada un algoritmo escrito en lenguaje C del cual obtiene información sobre sus características intrínsecas, estas características son contrastadas con una figura de mérito y posteriormente se

obtienen como salida indicadores que ayudan a determinar el rendimiento, al implementar este algoritmo sobre un FPGA.

La herramienta cuenta con un analizador léxico y un analizador sintáctico que facilitan la extracción de las características intrínsecas de interés dentro de un algoritmo de entrada al hacer un barrido de caracteres y analizar las frases gramaticales.

El analizador léxico se conoce también como analizador lineal debido a que se trata de un barrido secuencial del código fuente. La función de un analizador léxico es la de identificar los elementos básicos del lenguaje los cuales son indivisibles y se les denomina *token*.

El analizador sintáctico toma los *token* que le envía el analizador léxico y comprueba si con ellos se puede formar alguna patrón válido del lenguaje C. Los patrones se componen de símbolos. Aquellos que son construidos agrupando construcciones más pequeñas de acuerdo a reglas gramaticales se denominan símbolos no terminales y a los que no pueden subdividirse se les denomina símbolos terminales o *token*.

Un *token* es un símbolo que el analizador léxico puede leer directamente como números, símbolos especiales, palabras claves e identificadores. Por otra parte, un símbolo no terminal es una composición de símbolos terminales y no terminales como por ejemplo una operación o una expresión condicional.

Para la realización de los analizadores léxico y sintáctico se utilizaron las herramientas Flex y Bison. Flex es un una herramienta que permite generar analizadores léxicos a partir de un archivo de entrada con extensión '*.l' y Bison es un generador de analizadores sintácticos de propósito general a partir de un archivo de entrada de extensión '*.y'.

VI-A. REALIZACIÓN DEL ANALIZADOR LÉXICO

Flex es un una herramienta que genera un analizador léxico a partir de un fichero de entrada llamado "flexfile.l" el cual contiene la descripción del analizador que se va a generar. La descripción se encuentra en forma de parejas de expresiones regulares y código C, denominadas reglas. Flex genera como salida un fichero fuente en C llamado "lex.yy.c" en el que se define una función "yylex()" que es la encargada de reconocer los *token* en el código de entrada. Este proceso se puede ver en la Figura 3.

Figura 3: Proceso de construcción del Analizador Léxico.

Tabla I: Palabras reservadas del lenguaje C.

auto	break	case	char
const	continue	default	do
enum	extern	float	for
goto	if	int	long
else	return	short	signed
sizeof	static	struct	double
register	switch	typedef	union
unsigned	void	volatile	while

El fichero de entrada "flexfile.l" para generar el analizador léxico de la herramienta creada está compuesto de las siguientes secciones:

VI-A1. SECCIÓN DE DEFINICIONES: La herramienta construida utiliza Flex y Bison juntos, por lo tanto las definiciones de los *token* están contenidas en el archivo "bisonfile.tab.h" el cual contiene la definición de los *token* y es generado por la herramienta Bison.

VI-A2. SECCIÓN DE REGLAS: Los patrones o *token* que debe reconocer la herramienta para realizar el análisis son palabras reservadas del lenguaje C, identificadores, números y símbolos especiales tales como el punto, punto y coma, paréntesis, corchetes, entre otros.

Los patrones en la entrada se escriben utilizando un conjunto extendido de expresiones regulares y usando como alfabeto cualquier carácter ASCII. Algunos de los patrones de Flex y su significado son:

x	carácter "x'.
.	cualquier carácter excepto una línea nueva.
[abj-oZ]	conjunto de caracteres con un rango; empareja una "a", una "b", cualquier letra desde la "j" hasta la "o", o una "Z".
r*	cero o más r's, donde es cualquier expresión regular.
r+	una o más r's.
r?	cero o una "r".
r|s	una "r" o una "s".

Palabras reservadas:

Una palabra reservada tiene un significado concreto en el lenguaje de programación. La herramienta reconoce las palabras reservadas del lenguaje C que se pueden ver la Tabla I. Al encontrar alguno de estos patrones las acciones a ejecutar son guardar el valor del *token* reconocido en la variable yyval y retornar el *token* al analizador sintáctico. Un ejemplo de lo anterior es la palabra reservada *auto* que es

reconocida por el analizador mediante la siguiente regla:

auto {yylval.tipoc = yytext; return AUTO;}

Al reconocer la palabra “auto” dentro del algoritmo se guarda el valor del *token* de tipo puntero a *char*, en la variable global yyval y retorna el *token* AUTO al analizador sintáctico.

Para declarar un tipo de dato combinado se escribe una expresión regular para definir el patrón. Por ejemplo:

(“unsigned” | “signed”)?int

Esta expresión nos dice que se reconoce la palabra “unsigned” o la palabra “signed” o ninguna de las dos seguida de la palabra “int”.

Además de las palabras reservadas del lenguaje C la herramienta ha reservado las palabras “itwhile” y “transferdata” para obtener los parámetros de entrada relacionados con el número de iteraciones del ciclo while y la transferencia de datos. Estos parámetros son introducidos por el usuario a través de la interfaz gráfica.

Identificadores en C:

El nombre de las variables o identificadores en lenguaje C tienen las siguientes características.

- El primer símbolo del identificador será un carácter alfabético (a,..., z, A,..., Z, ’_ ’). Después de este primer carácter es posible poner caracteres alfanuméricos (a,..., z) y (0, 1,..., 9) y ’_ ’.
- El ’_ ’ se interpreta como una letra más.
- Los identificadores no pueden llevar el nombre de las palabras reservadas.

Para reconocer un identificador se definió el patrón que cumple con las características antes mencionadas, mediante la siguiente expresión regular:

[a-zA-Z_]?[a-zA-Z0-9_]+ {yylval.tipoc = yytext; return VARIABLE;}

La acción a ejecutar es guardar el valor del *token* reconocido en la variable yyval y retornar el *token* VARIABLE al analizador sintáctico.

Números:

La herramienta reconoce números enteros y decimales. Los números decimales se componen de un número entero cualquiera, llamado parte entera, separado por un punto de la parte fraccionaria. Por lo tanto es posible unir el reconocimiento de números enteros y de números decimales en un solo patrón el cual reconoce uno o más dígitos del sistema decimal, seguido de cero o un punto y cero o más dígitos del sistema decimal.

[0-9]+“.”?[0-9]* {yylval.tipod = atof(yytext); return NUMBER;}

La acción al encontrar este patrón es guardar el valor en la variable “yyval”, pero esta vez guarda el número utilizando el tipo de dato tipod. En este patrón se utiliza la función “atof” para convertir el contenido de “yytext” a formato *double*. Otra acción que se realiza al encontrar este patrón es retornar el *token* NUMBER al analizador sintáctico.

Símbolos especiales:

Los símbolos especiales incluyen operadores aritméticos, operadores relacionales, operadores binarios, operadores boléanos y signos de puntuación. Para estos patrones la acción es guardar el contenido de “yytext” en “yyval” y retornar un *token* al analizador sintáctico. Por ejemplo:

“.” {yylval.tipoc = yytext; return PUNTO;}

La regla anterior nos dice que el analizador al encontrar un punto dentro del algoritmo guarda el punto en “yyval” y retorna el *token* PUNTO.

Hay dos símbolos que tienen importancia en el analizador léxico que son “\n” (carácter de nueva línea) y “\t” (carácter tabulador). Al encontrar una nueva línea en el algoritmo de entrada se retorna el *token* EOL, mientras que si se encuentran tabuladores no se hace nada.

VI-B. REALIZACIÓN DEL ANALIZADOR SINTÁCTICO

Bison es un generador de analizadores sintácticos para gramáticas libre de contexto. En las gramáticas de libre contexto las reglas sintácticas son de la forma:

V: w

Donde “V” es un símbolo no terminal y “w” es una agrupación de símbolos terminales y/o no terminales. El término libre de contexto se refiere al hecho de que el no terminal V puede ser sustituido por w sin tener en cuenta el contexto en el que ocurra. Por ejemplo:

simbolo_no_terminal: TOKEN1 TOKEN2

La linea anterior nos dice que el símbolo no terminal llamado “simbolo_no_terminal” se forma agrupando “TOKEN1” con “TOKEN2”.

Se suministra a Bison el fichero “bisonfile.y” que contiene la descripción del analizador sintáctico para generar los archivos

"bisonfile.tab.c" y "bisonfile.tab.h" como se muestra en la Figura 4.

Figura 4: Proceso de construcción del Analizador Sintáctico.

El fichero de entrada "bisonfile.y" para generar el analizador léxico de la herramienta está compuesto de las siguientes secciones:

VI-B1. SECCIÓN DE DECLARACIONES EN C: En esta sección se incluyen los archivos de cabecera necesarios para la compilación de la herramienta, los cuales son:

1. stdio.h
2. stdlib.h
3. string.h

Se declaran las variables utilizadas para guardar información del algoritmo de entrada como por ejemplo un contador de líneas del código de entrada, una variable para guardar la información del tipo de dato, vectores que guardan las variables que contiene el código de entrada, contador de operaciones máximo y mínimo, contadores de transferencias entre registros, contadores de cada una de las sentencias teniendo en cuenta el grado de anidamiento, entre otras.

Al final del análisis realizado por la herramienta algunas de estas variables contienen información sobre las características intrínsecas del algoritmo, con lo cual se puede dar un indicador sobre el rendimiento que se podría obtener al implementar el código de entrada en un FPGA.

VI-B2. SECCIÓN DE DECLARACIONES DE BISON: En esta sección se encuentra la definición de los tipos de datos utilizados en los analizadores léxico y sintáctico mediante la declaración *%union* como se muestra a continuación:

```
%union{
int tipoi;
double tipod;
char *tipoc;
}
```

La declaración *%union* permite tener varios tipos de variable para analizar los *token* y símbolos no terminales. Los tipos de datos que se definieron son: "tipoi" que hace referencia a variables de tipo entero, "tipod" que hace referencia a variables de tipo real y "tipoc" que hace referencia a cadenas de caracteres.

Como la herramienta analiza principalmente las cadenas de caracteres de un código de entrada se utiliza principalmente el $<*\text{tipoc}>$ tanto para los *token* como para los símbolos no terminales.

Se definen los símbolos terminales o *token* con la declaración de Bison *%token* especificando el tipo de dato. Por ejemplo:

%token <tipoc> AUTO

La línea anterior define el *token* AUTO de tipo char*.

Se definen los símbolos no terminales con la declaración de Bison *%type* especificando el tipo de dato. Por ejemplo:

%type <tipoc> global

La línea anterior define el símbolo no terminal "global" de tipo char*.

VI-B3. SECCIÓN DE REGLAS GRAMATICALES: Esta sección es de gran importancia ya que en esta se definen las reglas gramaticales del lenguaje C que la herramienta debe reconocer en el código de entrada. Al reconocer una regla gramatical el analizador ejecuta acciones asociadas a dicha regla y puede realizar un llamado a una función que se haya definido en la sección de código C adicional.

A continuación se mencionan los símbolos no terminales que describen las reglas gramaticales más relevantes para el análisis del algoritmo de entrada. Cada símbolo no terminal puede tener una o más reglas asociadas y en cada regla se puede definir una acción diferente.

global

La primera regla define el símbolo inicial, el cual se denomina global. Este símbolo puede contener una cadena vacía o puede contener otros símbolos. Por ejemplo:

```
global: { }
| global asignación {fun_asignación()}
;
```

La regla anterior nos dice que el símbolo "global" acepta la cadena vacía o el símbolo no terminal "asignación". Se puede observar que al reconocer la regla "global asignación" la acción a ejecutar es llamar a la función fun_asignacion(), la cual debe estar definida en la sección de código C adicional.

declaracion

Este símbolo no terminal tiene asociada una regla gramatical que es utilizada para encontrar las palabras reservadas asociadas al tipo de dato. La acción asociada es guardar el tipo de dato en la variable tipo_dato. La regla tiene asociado el símbolo no terminal "identificador".

identificador

Este símbolo no terminal tiene asociada una regla gramatical que reúne principalmente el componente terminal VARIABLE, el cual es el *token* que define un identificador en lenguaje C. La regla está definida para detectar cada identificador contenido en el código de entrada a analizar. Las acciones que se ejecutan en esta regla son: guardar el valor del *token* VARIABLE en la variable "cadena" y ejecutar la función fun_declaracion() definida en la sección de código C adicional.

variables

Este símbolo no terminal tiene asociada una regla gramatical que reúne principalmente el componente terminal VARIABLE. Esta regla está orientada a identificar el nombre de las variables utilizadas en una operación. La acción principal que se ejecuta al encontrar esta regla es llamar a la función fun_dependato(), la cual ayuda a encontrar la dependencia de los datos entre las operaciones dentro de una sentencia de control en el algoritmo de entrada.

iteracioneswhile

Este símbolo no terminal tiene asociada una regla gramatical que detecta las iteraciones de los ciclos while introducidas por el usuario de la herramienta.

sentencia_if, sentencia_else, sentencia_for, sentencia_switch, sentencia_while

Cada uno de estos símbolos no terminales tiene asociada una regla gramatical que detecta la definición de una sentencia de control específica dentro del algoritmo. Las sentencias de control que reconoce la herramienta son: if, else, for, switch y while.

inicio_if, inicio_else, inicio_for, inicio_switch, inicio_while, inicio_case

Cada uno de estos símbolos no terminales tiene asociada una regla gramatical que detecta el inicio de una sentencia de control específica dentro del algoritmo. Cada regla hace un llamado a las funciones if_inicio(), else_inicio(), for_inicio(), switch_inicio(), while_inicio() y case_inicio() respectivamente.

final_case

Este símbolo no terminal tiene asociada una regla gramatical que detecta el final de la sentencia case y hace el llamado a la función case_final().

senten_fin

Este símbolo no terminal tiene asociada una regla gramatical que detecta el final de una sentencia de control (if, else, for, switch) y realiza el llamado a la función sentencia_fin().

condicion

Este símbolo no terminal tiene asociada una regla gramatical que detecta la condición de una sentencia if.

asignacion

Este símbolo no terminal tiene asociada una regla gramatical que detecta la asignación de una variable, es decir una operación o una transferencia entre registros. Hace el llamado de la función fun_asignacion() a través de la regla definida con el símbolo terminal global. Este símbolo no terminal está relacionado con el símbolo no terminal "exp", el cual tiene asociadas varias reglas para detectar expresiones matemáticas dentro del algoritmo

VI-B4. SECCIÓN DE CÓDIGO C ADICIONAL: Esta sección contiene la función main, las funciones propias del analizador generado y las funciones definidas para realizar el análisis del algoritmo de entrada.

Función main

En la función main se hace un llamado a la función yyparse() que es necesaria para comenzar el proceso de análisis.

yywrap(char *s)

Esta función se ejecuta al finalizar el análisis del código de entrada. Contiene los resultados finales del análisis como por ejemplo número de operaciones máximo y mínimo dentro del algoritmo, número de sentencias de control, número de dependencia de datos, entre otros. Estos valores son útiles para calcular la probabilidad de que al implementar el algoritmo en el FPGA se obtenga un rendimiento deseable.

fun_dependato()

Función que detecta posible dependencia de datos entre operaciones realizando una comparación de las variables utilizadas dentro de una sentencia de control. En este caso de encontrar una dependencia de dato ya ses de tipo RAW, WAR o WAW, se aumenta el contador de dependencias de dato respectivo.

fun_declaracion ()

Esta función se ejecuta al encontrar una asignación de variable dentro del algoritmo. Se definen contadores de variables de cada tipo de dato.

if_inicio(), else_inicio(), for_inicio(), switch_inicio(), while_inicio() y case_inicio()

Estas funciones realizan acciones similares que consisten

básicamente en aumentar los contadores de sentencias teniendo en cuenta el nivel de anidamiento.

case_final()

Se ejecuta al detectar el final de un case dentro una sentencia switch. Al terminar el case se toma la decisión de si se debe modificar el contador máximo y mínimo de operaciones con relación a los otros *case* dentro de la sentencia. Se debe realizar la misma comparación para el número de transferencias entre registros.

sentencia_fin()

Esta función que se ejecuta al detectar el final de una sentencia (if, else, for o switch). Al terminar alguna de estas sentencias de control se toma la decisión de modificar o no los contadores de operaciones y transferencias máximas y mínimas entre registros, teniendo en cuenta el nivel de anidamiento y la sentencia de control específica. Factores como el tipo de sentencia y el grado de anidamiento son importantes a la hora de tomar la decisión de modificar los contadores de operaciones. Por ejemplo si se está finalizando una sentencia "else" el analizador realiza una comparación con el número de operaciones de la sentencia "if" anterior del mismo grado de anidamiento para determinar cual contiene más operaciones y cual contiene más transferencias de registro. Además una sentencia de control que contenga sentencias de control anidadas debe tener la información de dichas sentencias anidadas.

fun_asignacion()

Función que se ejecuta al detectar una asignación en una variable, ya sea operación o transferencia entre registros. Se tiene en cuenta el número de iteraciones del ciclo de control donde está contenida la asignación de la variable a la hora de realizar el conteo de operaciones y transferencias.

VII. COMPILACIÓN DEL ANALIZADOR

Se suministra a Flex el archivo de entrada "flexfile.l", que contiene la especificación del analizador léxico y el archivo de entrada de Bison "bisonfile.y" que contiene la descripción del analizador sintáctico. Al utilizar las herramientas Flex y Bison obtenemos los archivos "lex.yy.c" y "bisonfile.tab.c" los cuales se compilan para producir un archivo ejecutable, el cual es capaz de analizar un código de entrada. Este proceso se puede observar en la Figura 5.

La herramienta en C es un analizador de códigos que se produce al juntar los analizadores léxico y sintáctico. Para generar el analizador de algoritmos se realizan los pasos para usar conjuntamente Flex y Bison que son:

1. bison -d bisonfile.y
2. flex flexfile.l

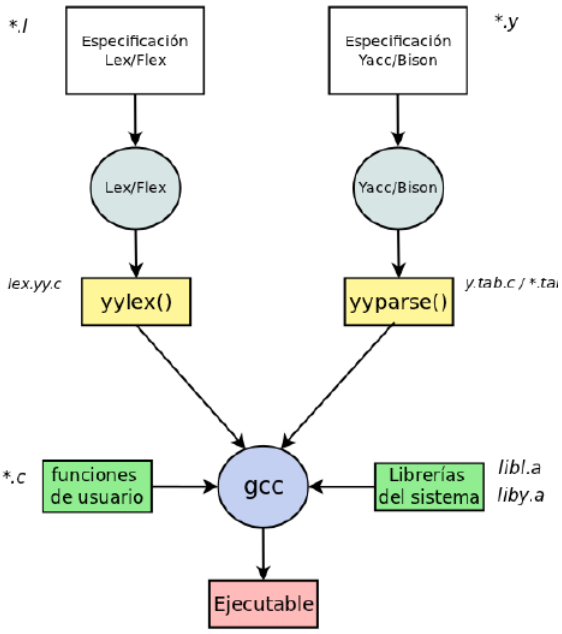

Figura 5: Proceso de construcción del Ejecutable.

3. gcc -o ejecutable bisonfile.tab.c lex.yy.c –lfl

Estos pasos generan un archivo ejecutable llamado "ejecutable" el cual es el analizador de algoritmos. Estas líneas están reunidas en un archivo makefile, el cual automatiza los comandos necesarios para la compilación del ejecutable, así como la limpieza de archivos.El comando *make ejecutable* realiza la compilación del archivo ejecutable y el comando *make clean* realiza la limpieza de archivos.

VIII. FUNCIONAMIENTO DEL ANALIZADOR

El analizador generado utiliza las funciones "ylex" y "yyparse" para leer los caracteres del algoritmo de entrada. El flujo de control de las funciones "yylex()" e "yyparse()" en el archivo ejecutable es el que se observa en la Figura 6.

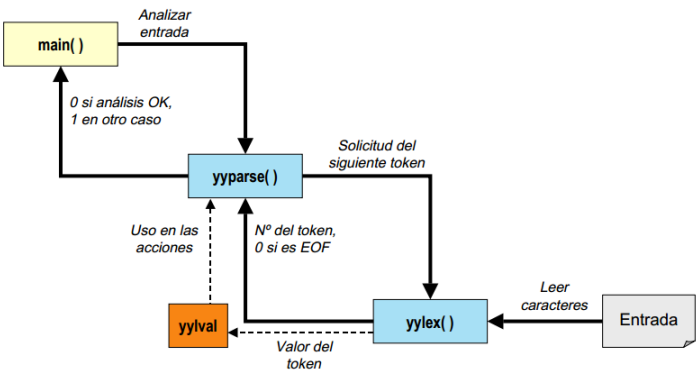

Figura 6: Funcionamiento del analizador generado.

Para realizar el análisis desde consola el usuario puede utilizar el siguiente comando:c ./ejecutable <“codigoc_de_entrada”>“archivo_salida”

Donde “codigoc_de_entrada” es el archivo que contiene el código en lenguaje C que será analizado y “archivo_salida” es un archivo donde se guardarán los resultados del análisis.

IX. DISEÑO DE LA INTERFAZ GRÁFICA

Con el fin de facilitar la interacción de los usuarios con la herramienta, se creó una interfaz gráfica que permite a cualquier persona usar la herramienta de manera amigable, sin la necesidad de que esta sea un usuario informático avanzado. La interfaz se desarrolló en Gambas, que es un lenguaje orientado a objetos con un gran número de capacidades y un entorno de desarrollo basado en un intérprete de BASIC. Gambas es un entorno gráfico con el objetivo centrado en la creación rápida y ágil de interfaces gráficas para aplicaciones. Se encuentra publicado bajo licencia GNU (General Public Licence).

La interfaz gráfica está provista de cinco botones que permiten cargar los archivos y parámetros que la herramienta necesita para el análisis, que son: La carpeta de trabajo, el algoritmo de entrada, el analizador, el número de iteraciones WHILE y la transferencia de datos; dos botones adicionales que me permiten ejecutar el analizador y otro detenerlo; un cuadro de texto que muestra en todo momento el estado del análisis; una pestaña en la que podemos seleccionar entre dos formas de visualizar la información de salida y finalmente dos botones adicionales, uno que me permite limpiar los cuadros de texto para hacer otro análisis y un botón de salir.[La información detallada de estos parámetros puede verse en el de manual de ayuda que ofrece la herramienta].En la Figura 7 se muestra una captura de la interfaz de usuario de la herramienta.

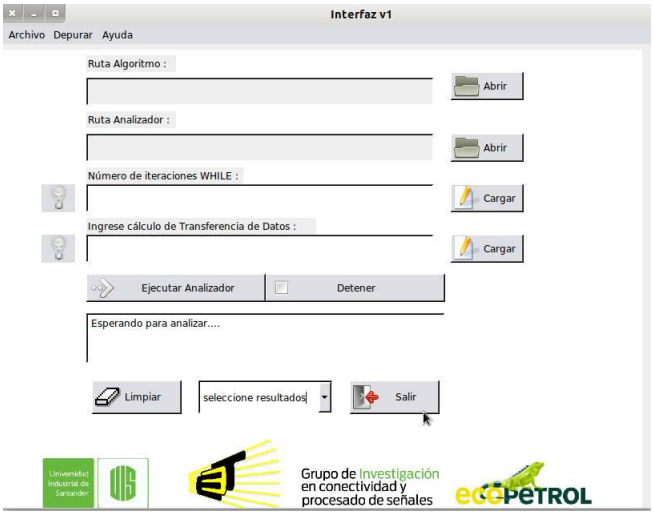

Figura 7: Interfaz gráfica.

IX-A. SECUENCIA DE USO DE LA INTERFAZ

La interfaz gráfica funciona secuencialmente, es decir se siguen una serie de pasos para llegar al análisis. El primer paso es, pulsar sobre el botón “abrir carpeta de trabajo” que permite seleccionar una carpeta en el ordenador en donde queremos que se alojen todos los archivos del análisis.

Posteriormente debemos pulsar sobre el botón “abrir algoritmo” que posibilita buscar la ubicación en el ordenador en donde se encuentra el algoritmo a analizar.

Después debemos pulsar sobre el botón “abrir analizador” que nos ayuda a buscar la ubicación en el ordenador en donde se encuentra el ejecutable del analizador.

Luego debemos ingresar dos parámetros importantes para el análisis de la herramienta que son el número de iteraciones WHILE y el cálculo de la transferencia de datos cada uno en su campo correspondiente.

Finalmente la herramienta está lista para realizar el análisis el cual inicia al presionar el botón “ejecutar analizador”. La herramienta procesa los datos de entrada y genera los documentos con la información de salida.

Una vez ha terminado el análisis podemos seleccionar en la pestaña de resultados entre una versión general del informe de análisis y una versión más detallada. En la Figura 8 se muestra una captura de los resultados en versión general de un análisis .

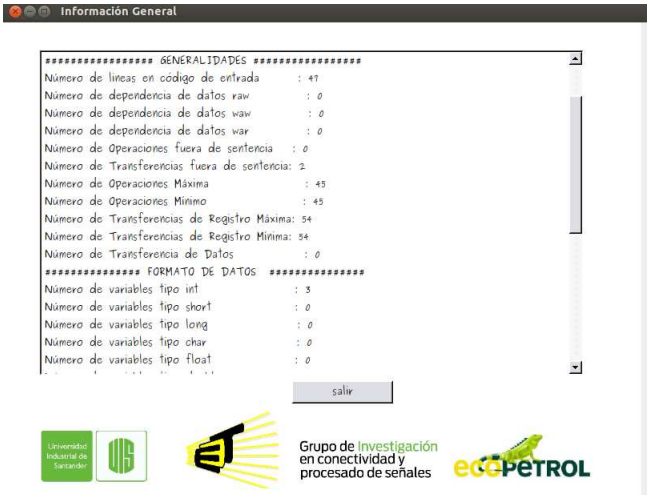

Figura 8: Resultados en versión general.

X. CREACIÓN DE LA FIGURA DE MÉRITO

Se creó una figura de mérito teniendo en cuenta los parámetros tratados en esta investigación, la cual me permita evaluar cada una de las características y tener una apreciación cualitativa del desempeño del algoritmo analizado. Se tienen en cuenta

las características que influyen en el rendimiento del proceso al ser implementado sobre un FPGA.

X-A. PROCESO LIMITADO POR TRANSFERENCIA DE DATOS Ó POR CÓMPUTO

Esta característica es bastante importante para evaluar la viabilidad de la implementación de un algoritmo en un FPGA, pero no es el fuerte de este trabajo de investigación ya que no es directamente extraíble a partir de una lectura del algoritmo. Se constituye en un primer filtro del proceso a analizar pero no puntúa sobre la figura de mérito creada.

Se debe analizar si el proceso esta limitado por transferencia de datos o por cómputo, ya que si el proceso esta limitado por transferencia de datos, es poco probable que se obtenga una mejora en el proceso solo con un buen rendimiento computacional. Por lo tanto si el proceso está limitado por transferencia de datos no vale la pena analizar las características intrínsecas de la aplicación. Pero si la aplicación está limitada por cómputo se procede a extraer las características intrínsecas para contrastarlas con la figura de mérito.

X-B. PORCENTAJE DE CADA PARÁMETRO Y RECOMENDACIONES PARA OBTENER PONDERACIÓN

X-B1. DEPENDENCIA DE DATOS: Teniendo en cuenta la revisión del estado del arte la dependencia de datos es muy importante para obtener alto rendimiento debido a la posibilidad de realizar muchos cálculos simultáneamente. Por lo tanto se asigna un porcentaje alto a este parámetro.

Podemos observar este parámetro contrastando el número de ocurrencias de cualquiera de los tres tipos de dependencia mencionados con el numero de asignaciones dentro del algoritmo.

X-B2. FORMATO DE LOS DATOS Y COMPLEJIDAD ARITMÉTICA: El formato de los datos y la complejidad aritmética son importantes porque influyen fuertemente en el tamaño del circuito y en la velocidad de procesamiento. El número de operaciones que se pueden desarrollar al mismo tiempo en un FPGA determina el límite de paralelismo que se puede alcanzar. Por lo tanto se asigna un porcentaje alto a este parámetro.

Para analizar el formato de los datos se observa el tipo de datos dentro del algoritmo y entre menos número de bits tenga las variables, hay más factibilidad de un buen desempeño. Para determinar la complejidad aritmética se realiza el conteo de cada tipo de operador dentro del algoritmo y se mira el grado de complejidad de las operaciones que estos representan.

X-B3. SENTENCIAS DE CONTROL: Al implementar algoritmos en FPGA se puede obtener buen rendimiento si todas las operaciones están estáticamente programadas tanto como sea posible, en otras palabras se puede tener buen rendimiento cuando el algoritmo tiene pocas rutas de decisión definidas por sus sentencias de control. Se puede observar la cantidad de sentencias de control y sus grados de anidamiento para estimar la cantidad de rutas de decisión.

En en la Tabla II se tiene cada parámetro con su respectivo porcentaje que indica que tanto afecta al rendimiento de un proceso.

Tabla II: Figura de mérito.

PARÁMETRO DE RENDIMIENTO	PORCENTAJE ASIGNADO
DEPENDENCIA DE DATOS	40
FORMATO DE LOS DATOS Y COMPLEJIDAD ARITMÉTICA	40
SENTENCIAS DE CONTROL	20

XI. PRUEBAS DE LA HERRAMIENTA

Para la validación de la herramienta se analizaron los algoritmos Producto Punto y Multiplicación de Matrices (DGEMM). Estos algoritmos son ampliamente utilizados para para medir el rendimiento de un sistema utilizando la técnica llamada ***benchmark***. Más formalmente puede entenderse que un *benchmark* es el resultado de la ejecución de un programa informático o un conjunto de programas en una máquina, con el objetivo de estimar el rendimiento de un elemento concreto, y poder comparar los resultados con máquinas similares. Dentro las fuentes libres para realizar *benchmark* se encuentra la librería *Lapack* (Linear Algebra PACKage) que son un conjunto de rutinas para realizar operaciones básicas de álgebra lineal.

También se analizó el algoritmo de la serie de Fibonacci. Se pueden ver los resultados del análisis de los tres algoritmos en el ANEXO B.

Producto punto

De acuerdo a los resultados que arroja la herramienta, este algoritmo no presenta dependencia de datos a la hora de realizar los cálculos y no tiene anidamiento de sus sentencias de control. Se puede observar también que el algoritmo contiene variables de tipo float, lo cual es un punto en contra para implementar el código en un FPGA, pero también se aprecia que no presenta complejidad aritmética. Al contrastar toda la información con la figura de mérito, se determina que este algoritmo tiene buenas probabilidades de aceleración.

Haciendo una revisión del estado del arte, se puede ver que se ha obtenido buen rendimiento al implementar este algoritmo en un FPGA realizando los cálculos en precisión

de punto fijo.[9]

Multiplicación de Matrices (DGEMM).

De acuerdo a los resultados que arroja la herramienta, este algoritmo no presenta dependencia de datos a la hora de realizar los cálculos y las sentencias de control presentan solo dos grados de anidamiento. Se puede observar que el algoritmo contiene variables de tipo *double*.

El formato de datos es el punto débil de este algoritmo ya que la lógica de coma flotante es compleja. Sin embargo los FPGA han mostrado grandes ganancias en rendimiento de operaciones en coma flotante logrando rendimientos de 144 GFLOPS en el algoritmo de DGEMM. [10] **Serie de Fibonacci.**

Se analizó el algoritmo de la serie de Fibonacci el cual tiene buena características computacionales, ya que utiliza formato de datos tipo entero, no presenta dependencia de datos y contiene sentencias de control sencillas y sin grado de anidamiento. Por lo tanto este algoritmo es buen candidato para ser implementado sobre un FPGA.

XII. OBSERVACIONES

- Se desarrolló una subrutina que permite extraer el formato de los datos usados en el algoritmo que está siendo analizado. Los tipos de dato que reconoce la herramienta son: caracteres (char), enteros (int), números en coma flotante (32 bits), números en coma flotante de doble precisión (64 bits). Se recomienda que el algoritmo de entrada tenga las declaraciones de las variables en la parte inicial del código. Sin embargo la herramienta puede detectar las declaraciones de variables en cualquier lugar del código.

- La herramienta cuenta con una subrutina que permite determinar y contabilizar las sentencias de control que contiene el algoritmo de entrada. Las sentencias de control que reconoce la herramienta son: If, for, while y switch. También se contabiliza los grados de anidamientos de dichas sentencias, ya que el grado de anidamiento debe ser tomado en cuenta a la hora de realizar un conteo adecuado de las operaciones dentro de los bucles de control.

- Se realizó una subrutina que determina cuantas veces son actualizadas las variables dentro del algoritmo, teniendo en cuenta un número máximo y mínimo de transferencias de registros. También cuenta el número de operaciones aritméticas que se realizan, teniendo en cuenta un rango mínimo y máximo de operaciones. Este número mínimo y máximo de operaciones depende de las sentencias de control y sus grados de anidamiento. En teoría este conteo permite determinar si el proceso es limitado por transferencia de datos o si el proceso está limitado por cómputo.

- El usuario de la herramienta debe hacer un conteo manual de las transferencias de entrada y salidas para saber realmente si el proceso está limitado por transferencia de datos o si el proceso está limitado por cómputo.

- Se desarrolló una subrutina que permite determinar cuando una variable es utilizada para realizar el cálculo de otra variable, es decir que el analizador detecta dependencia de datos dentro de una sentencia de control. Esta primera aproximación para detectar dependencia de datos puede ser útil para determinar el rendimiento de un algoritmo. Sin embargo este detector de dependencia de datos es muy básico.

- La herramienta cuenta con una interfaz gráfica capaz de interactuar con el usuario de manera sencilla; proporcionándole todas las funciones necesarias para hacer un análisis correcto de un algoritmo de entrada. Además cuenta con un material de ayuda que permite la aclaración de cualquier inquietud de manejo.

- Se realizó un cuadro de mérito en el cual se tiene en cuenta la dependencia de datos, el formato de los datos, la complejidad de las operaciones aritméticas y las sentencias de control. A cada parámetro se le asignó un peso el cual se determinó teniendo en cuenta una revisión del estado del arte y la experiencia en este ámbito del grupo de investigación CPS.

XIII. CONCLUSIONES

- La herramienta elaborada ayuda a extraer características que permiten determinar la viabilidad de la migración de procesos que se realizan en procesadores de propósito general a FPGA y conseguir buenos índices de aceleración.

- La detección del formato de los datos dentro de un algoritmo se realiza mediante la definición de reglas semánticas sencillas mientras que la extracción de otros parámetros como el grado de anidamiento de las sentencias de control requieren reglas mas elaboradas.

- Generalmente no se puede determinar el número de iteraciones de una sentencia “while” en un algoritmo solo analizando su código en C, ya que la variable de control de esta sentencia puede definirse durante la ejecución del algoritmo.

- La extracción de las limitantes por transferencia de datos de un proceso, no se puede realizar fácilmente analizando el algoritmo del proceso ya que depende de factores propios de la plataforma de desarrollo con la que se esté trabajando y de la aplicación.

- No se puede obtener un conteo adecuado de transferencias al hacer una lectura del algoritmo, ya que no todas las transferencias conllevan una transferencia de entrada y salida. Las transferencias que se llevan a cabo en registros locales no afectan de manera significativa el rendimiento ya que se pueden realizar en un ciclo de reloj.

- Elaborar un algoritmo que permita analizar la sintaxis de otros algoritmos es una tarea compleja. Herramientas de software libre como Flex y Bison facilitan el desarrollo de este tipo de aplicaciones.

- El uso de una interfaz gráfica, permite a usuarios con conocimientos informáticos básicos, usar de manera sencilla herramientas computacionales con un trasfondo informático avanzado.

XIV. TRABAJO FUTURO

- Profundizar en el estudio de la transferencia de datos, ya que este es un parámetro determinante en el rendimiento de un algoritmo al ser implementado en un FPGA.

- Ahondar en la determinación de la dependencia de los datos, la cual es una característica vital para determinar la cantidad de paralelismo aplicable en una implementación basada en un FPGA.

- Creación de más reglas gramaticales del lenguaje C para complementar la herramienta.

REFERENCIAS

[1] Barron Stone, *Cómo los Instrumentos Basados en FPGA Pueden Acelerar sus Mediciones.* Instrumentation Newsletter, Marzo 2012

[2] Sergio A. Abreo Carrillo y Ana B. Ramírez Silva, *Reconfigurable Computing the Theory and Practice of FPGA-Based Computation*, 1rd ed. The Morgan Kaufmann Publishers, Burlington, MA , 2008.

[3] Joan Vancells i Flotats, *Algoritmos y programas*, 1rd ed. Universitat Oberta de Catalunya, 2002.

[4] Lisandro Peralta Murúa, *Análisis de Lenguaje*, INSTEC-UDA.

[5] Dennis M. Ritchie, *The Development of the C Language*, Copyright Association for Computing Machinery Inc, 1993.

[6] José A. Vera Repullo, Manuel J. Buendía, A. José González Valdé, *FPGAs. Circuitos de lógica programable*, 1rd ed. Universidad de Murcia, 1995

[7] Ian Kuon,Russell Tessier, Jonathan Rose, *FPGA Architecture*, now Publishers Inc, Hanover USA, 2008.

[8] Scott Hauck and Andre DeHon, *Reconfigurable Computing the Theory and Practice of FPGA-Based Computation*, 1rd ed. The Morgan Kaufmann Publishers, Burlington, MA , 2008.

[9] Antonio Roldao Lopes and George A. Constantinides, *A Fused Hybrid Floating-Point and Fixed-Point Dot-product for FPGAs.* Electrical and Electronic Engineering, Imperial College London, London SW7 2BT, England.

[10] http://paralelizados.com/?p=786. *Las Últimas FPGAs Mostraron Grandes Ganancias En Rendimiento De Coma Flotante.* by Rodo on Abril 20, 2012.

[11] *Procesadores segmentados.* Departamento de informática Universidad de Valladolid.

[12] Rubén Béjar Hernández. Introducción a Flex y Bison. Dpto. Informática e Ingeniería de Sistemas Universidad de Zaragoza.

[13] José Antonio Camarena Ibarrola. . Notas para la Materia de Compiladores. Marzo de 2008.

[14] M Cándida Luengo Diez. . Análisis sintáctico en procesadores de lenguaje apuntes de procesadores de lenguajes. Oviedo Enero 2005.

[15] Charles Donnelly y Richard Stallman, . Bison, El Generador de Analizadores Sintácticos compatible con YAC. Bison Version 1.27. 1999.

[16] Manual oficial de Flex en www.sourceforge.net . Lexical Analysis With Flex. http://flex.sourceforge.net/manual/ Copyright © 2008 The Flex Project.

[17] Manual oficial de Bison en http://www.gnu.org/ . Bison 2.5 . http://www.gnu.org/software/bison/manual/bison.html Copyright © 2011 Free Software Foundation, Inc.

[18] Sitio web oficial de Gambas http://gambas.sourceforge.net/en/main.html . Copyright (C) 1989, 1991 Free Software Foundation, Inc.

[19] A. Dandalis, V. K. Prasanna. *Fast parallel implementation of DFT using configurable devices. Field-programmable logic: Smart applications, new paradigms, and compilers.* Proceedings 6th International Workshop on Field-Programmable Logic and Applications, Springer-Verlag, 1997.

[20] G. Panneerselvam, P. J. W. Graumann, L. E. Turner. *Implementation of fast Fourier transforms and discrete cosine transforms in FPGAs.* Fifth International Workshop on Field-Programmable Logic and Applications, September 1995.

[21] J. E. Vuillemin. *On computing power. Programming languages and system architectures.*, Lecture Notes in Computer Science, vol. 781. Springer-Verlag, 1994.

[22] B. W. Kernighan, D. M. Ritchie. *El Lenguaje de Programación C*, 2rd ed. 1991.

[23] NI Developer Zone! . *Introducción a la Tecnología FPGA: Los Cinco Beneficios Principales*, http://zone.ni.com/devzone/cda/tut/p/id/8259 , Junio 6 de 2011.

[24] V. John, H. Brad, *The flexibility of Configurable Computing.*, IEEE Signal Processing Magazine, Septiembre 1998.

[25] K. Compton, S. Hauck, *Reconfigurable Computing: A Survey of Systems and Software In: ACM Computing Surveys Vol. 34 No. 2*, , 2002.

[26] A. B Ignacio, C. P. René, Coordinación de Ciencias Computacionales INAOE. *Arquitectura Reconfigurable Para La Implementación De Algoritmos Estándares De Criptografía Aplicados A Comunicaciones* , 2004.

[27] B. L. Hutchings, B. E. Nelson. *Implement Applications With FPGAs.* Department of Electrical and Computer Engineering. Brigham Young University.

Andrés Fernando Platarrueda Acuña Actualmente es estudiante de Ingeniería Electrónica de la Universidad Industrial de Santander. Su perfil de Investigación esta enfocado al análisis computacional avanzado y los FPGA.

Pablo Ernesto Verbel González Actualmente es estudiante de Ingeniería Electrónica de la Universidad Industrial de Santander. Su perfil de Investigación esta enfocado al análisis computacional avanzado y los FPGA.

Sergio Alberto Abreo Carrillo Es docente línea de sistemas digitales en la Universidad Industrial de Santander y pertenece a los grupos de investigación en Conectividad y Procesamiento de Señales (CPS) y Petrosísmica. Se graduó como Ingeniero en Control Electrónico de la Universidad Distrital Francisco José de Caldas en Abril del 2008. Sus intereses de investigación incluyen diseño e implementación de procesadores específicos usando FPGA, procesamiento digital de señales y computación de alto rendimiento.

Carlos A. Fajardo Ariza Ingeniero Electrónico y Especialista en Docencia de la Universidad Industrial de Santander, en la cual es catedrático desde el año 2002. Actualmente se encuentra adscrito al grupo de investigación Conectividad y procesamiento de Señales (CPS). Sus campos de interés son los FPGA, los Sistemas Embebidos y la aceleración de algoritmos por medio de FPGA.

ANEXO A. Manual de Uso de la Herramienta.

1 Instalación de la Herramienta

Para poder hacer uso de la herramienta, el ordenador debe disponer de un sistema operativo Linux.
A continuación se expondrá la forma en que debe instalarse la herramienta para su correcto funcionamiento:

1. Instalar el paquete "interfazv4_0.0-1_all.deb", contenido dentro de la carpeta de archivos necesarios (Hacer doble clic sobre el mismo y el asistente de instalación guiará paso a paso el proceso de instalación).

2. Para completar el proceso de instalación, copiar la carpeta "Ayuda" contenida dentro de la carpeta de archivos necesarios a la carpeta personal, es decir al directorio ./Home/Nombre Usuario.

2 Antes de Empezar

- Crear una carpeta con permisos de Lectura y Escritura. Esta carpeta es por la que se preguntará posteriormente en la interfaz de la herramienta como: "carpeta de trabajo".

- El archivo llamado "ejecutable" contenido dentro de la carpeta de archivos necesarios, es el analizador en su versión inicial y es por el que se preguntará posteriormente en la interfaz de la herramienta como: "analizador".

- El archivo de entrada debe tener una linea vacía al final de código.

3 Características Generales del Uso de la Herramienta

3.1 Datos Suministrados por el usuario:

- Para la herramienta es imposible determinar el número de veces que va iterar una sentencia WHILE debido a que este parámetro generalmente se define con la ejecución del algoritmo. Por esta razón, se pide al usuario que especifique un número de iteraciones por sentencia WHILE dentro del

Algoritmo y de esta forma simular, el comportamiento de esta sentencia. Si el usuario no especifica este parámetro la herramienta tomara como base para la simulación una iteración por cada ciclo WHILE.
La forma de hacerlo se explicará con el fragmento de código siguiente:

```
WHILE(i<a){
  i=i+1;
}
WHILE(j<b){
  j=j+1;
}
WHILE(k<c){
  k=k+2;
}
```

Si el usuario ingresa en la casilla de iteraciones WHILE un 5. Se asumirá que el primer WHILE itera 5 veces, el segundo 1 vez y el tercero 1 vez. Es decir que la herramienta detectaría 7 operaciones.

Si el usuario ingresa en la casilla de iteraciones WHILE un 5 y un 3 (espaciados). Se asumirá que el primer while itera 5 veces, el segundo 3 veces y el tercero 1 vez. Es decir que la herramienta detectaría 9 operaciones.

Si el usuario deja vacía la casilla de iteraciones WHILE. Se asumirá que el primer while itera 1 vez, el segundo 1 vez y el tercero 1 vez. Es decir que la herramienta detectaría 3 operaciones.

La estrategia que se plantea para determinar el número de iteraciones de las sentencias While es agregar un contador dentro cada sentencia While en el algoritmo de entrada. Luego compilar el código y ejecutarlo. De esta manera los contadores proporcionarán el número de iteraciones de cada sentencia While, durante la solución del problema. Posteriormente suministrar estos datos a la herramienta.

- Las limitantes por transferencia de datos, no son directamente extraíbles a partir de parámetros descritos en el algoritmo de entrada, por lo que no se puede determinar solo analizando la gramática del código. Las limitantes en la transferencia de datos depende de factores propios de la plataforma de desarrollo y de la aplicación con la que se este trabajando.

  Una estrategia para determinar si el proceso esta limitado por transferencia de datos, es observar cuantas veces son actualizadas las variables de entrada y salida dentro del algoritmo y también contar el número de operaciones aritméticas que se realicen. si el número de operaciones es menor o igual al número de asignaciones de variables de entrada y salida, el proceso es limitado por transferencia de datos, mientras que si el

número de operaciones realizadas es mayor que el número de asignaciones de variables de entrada y salida, el proceso será limitado por cómputo. La herramienta desarrollada puede determinar las limitantes por transferencia de datos de esta forma. Para esto el usuario debe determinar cuantas veces se reciben y se envían datos hacia periféricos externos, durante la ejecución del algoritmo. Este dato se suministra a la herramienta y esta se encarga de contrastar esta información con el número de operaciones aritméticas que se realicen durante la ejecución del algoritmo y de informarnos si según este criterio, la aplicación esta limitada por transferencia de datos.

3.2 Algoritmo de Entrada

La herramienta en esta, su primera versión, analiza algoritmos planos y sin llamadas a subfunciones ni librerías; Para analizar este tipo de algoritmos se debe analizar el algoritmo principal y posteriormente los códigos individuales de cada subfunción. Se le asigna una ponderación a cada análisis y se saca un veredicto.

Para la correcta detección de las sentencias, el algoritmo debe estar escrito según las recomendaciones de la regla ANSI para el Lenguaje C. A continuación se mostrarán las estructuras de código que soporta la herramienta y la forma en que estas deben ser escritas:

3.2.1 FORMATO DE LOS DATOS

La herramienta soporta los siguientes formatos de datos:

Enteros (int):

No inicializados:

```
1 int var1, var2, var3, ...., varn;
```

Incializados:

```
1 int var1=valor, var2=valor, ...., varn=valor;
```

Combinación de inicializados y no inicializados:

```
1 int var1, var2=valor, var3, ...., var n= alor;
```

Vectores:

```
int var1[100], var2[2]=[valor, valor], ...., varn[10];
```

Matrices:

```
int var1[2][2]=[val, val; val, val],....,varn[10][2];
```

Combinación de todo lo anterior:

```
int var1, var2[2]=[val, val], var3[2][4], ...., varn=0;
```

Al igual que los enteros funcionan los demás formatos admitidos que son:

unsigned int, *short* , *long*, *float*, *double*, *long double* y *char*.

El formato *char* admite solo vectores ya que la definición de char$[x][x]$ es un vector.

No se admiten estructuras, ni punteros.

3.2.2 SENTENCIAS DE CONTROL

Sentencia IF:

Sentencia IF simple:

```
if(var comparador var){
  instrucciones;
}
```

Sentencia IF - ELSE:

```
if(var comparador var){
  instrucciones;
}else{
  instrucciones;
}
```

Sentencias IF Anidadas:

Ejemplo IF:

```
if(var comparador var){
```

```
    if(var comparador var){
      instrucciones;
    }
  }
```

Ejemplo IF - ELSE:

```
if(var comparador var){
  instrucciones;
}else{
  if(var comparador var){
    instrucciones;
  }else{
    if(var comparador var){
      instrucciones;
    }else{
    instrucciones;
    }
  }
}
```

Sentencias IF Concatenadas:

```
if(var comparador var){
  instrucciones;
} else if(var comparador var){
  instrucciones;
} else if(var comparador var){
  instrucciones;
} else {
  instrucciones;
}
```

Recomendaciones sentencia IF:

1. El fragmento de código afectado por la condición del *if* debe sangrarse para que visualmente se interprete correctamente el ámbito de la sentencia *if*.
2. Olvidar los paréntesis al poner la condición del if es un error sintáctico (los paréntesis son necesarios)
3. Confundir el operador de comparación == con el operador de asignación = puede producir errores
4. Los operadores de comparación ==, ! =, <= y >= se escriben sin espacios intermedios
5. => y =< no son operadores válidos en el Lenguaje C

Sentencias SWITCH

```
switch(expresion)
case constante 1:
  instrucciones 1;
break;
case constante 2:
  instrucciones 2;
break;
...
case constante n:
  instrucciones n;
break;
default:
 instrucciones por defecto;
}
```

Recomendaciones sentencia SWITCH:

1. La expresión del *switch* debe ser de tipo entero.
2. Los valores de los casos del *switch* deben ser constantes.
3. La etiqueta *default* marca el bloque de código que se ejecuta por defecto.

Sentencias WHILE

```
while (condicion) {
  instrucciones;
}
```

Sentencias FOR

```
for (expresion 1; comparacion ; expresion 2) {
  instrucciones;
}
```

Observaciones para la sentencia FOR:

1. La expresión 1, solo acepta expresiones del tipo: $variable = numero$, por ejemplo:
   La expresión: $i = 0$, será reconocida por la herramienta y se determinará el número de iteraciones del ciclo.

La expresión: $i = x$, donde 'x' es otra variable, será reconocida por la herramienta, pero no se podría calcular el número de iteraciones, ya que el valor de la variable 'x' no esta determinado.

2. La comparación, solo acepta comparaciones del tipo: $variable < numero$, $variable > numero$, $variable <= numero$, $variable >= numero$.

3. La expresión 2, solo acepta expresiones del tipo: $variable{+}{+}$, $variable{-}{-}$, $variable{+} = numero$

NOTA: Se deben respetar las formas y posiciones en donde se escriben las '{', '}', ';', ':' y las mismas sentencias.

ANEXO B. Validación de la Herramienta.

1 Resultados Multiplicación de Matrices(dgemm)

A continuación se muestran los resultados del análisis en la herramienta, para el algoritmo *dgemm*:

```
################## GENERALIDADES ########################
Numero de lineas en codigo de entrada      : 47
Numero de dependencia de datos raw         : 0
Numero de dependencia de datos waw         : 0
Numero de dependencia de datos war         : 0
Numero de Operaciones fuera de sentencia   : 0
Numero de Transferencias fuera de sentencia: 2
Numero de Operaciones Maxima               : 1002000000
Numero de Operaciones Minimo               : 1002000000
Numero de Transferencias de Registro Maxima: 1003000000
Numero de Transferencias de Registro Minima: 1003000000
Numero de Transferencia de Datos           : 0
################ FORMATO DE DATOS  ######################
Numero de variables tipo int               : 3
Numero de variables tipo short             : 0
Numero de variables tipo long              : 0
Numero de variables tipo char              : 0
Numero de variables tipo float             : 0
Numero de variables tipo double            : 3000003
Numero de variables tipo long double       : 0
Numero total de variables declaradas       : 3000006
############ COMPLEJIDAD ARITMETICA ####################
Numero de sumas                            : 0
Numero de restas                           : 0
Numero de multiplicaciones                 : 1000000000
Numero de divisiones                       : 2000000
Numero de operaciones mod                  : 0
Numero total de operaciones                : 1002000000
############## SENTENCIAS DE CONTROL ####################
Numero de sentencias if                : 0
Numero de sentencias if nivel 0        : 0
Numero de sentencias if nivel 1        : 0
Numero de sentencias if nivel 2        : 0
Numero de sentencias if nivel 3        : 0
```

```
Numero de sentencias if nivel 4        : 0
Numero de sentencias if nivel 5        : 0
Numero de sentencias else              : 0
Numero de sentencias else nivel 0      : 0
Numero de sentencias else nivel 1      : 0
Numero de sentencias esle nivel 2      : 0
Numero de sentencias else nivel 3      : 0
Numero de sentencias else nivel 4      : 0
Numero de sentencias else nivel 5      : 0
Numero de sentencias for               : 8
Numero de sentencias for nivel 0       : 3
Numero de sentencias for nivel 1       : 4
Numero de sentencias for nivel 2       : 1
Numero de sentencias for nivel 3       : 0
Numero de sentencias for nivel 4       : 0
Numero de sentencias for nivel 5       : 0
Numero de sentencias switch            : 0
Numero de sentencias switch nivel 0    : 0
Numero de sentencias switch nivel 1    : 0
Numero de sentencias switch nivel 2    : 0
Numero de sentencias switch nivel 3    : 0
Numero de sentencias switch nivel 4    : 0
Numero de sentencias switch nivel 5    : 0
Numero de sentencias while             : 0
Numero de sentencias while nivel 0     : 0
Numero de sentencias while nivel 1     : 0
Numero de sentencias while nivel 2     : 0
Numero de sentencias while nivel 3     : 0
Numero de sentencias while nivel 4     : 0
Numero de sentencias while nivel 5     : 0
############## LISTA DE VARIABLES ##############
Variables Tipo int:
i
j
k
det

Variables Tipo double:
A
B
C
start
finish
maxr
det
```

2 Resultados Producto Punto

A continuación se muestran los resultados del análisis en la herramienta, para el algoritmo del producto punto:

```
################## GENERALIDADES ##################
Numero de lineas en codigo de entrada      : 25
Numero de dependencia de datos raw         : 0
Numero de dependencia de datos waw         : 0
Numero de dependencia de datos war         : 0
Numero de Operaciones fuera de sentencia   : 0
Numero de Transferencias fuera de sentencia: 0
Numero de Operaciones Maxima               : 6
Numero de Operaciones Minima               : 6
Numero de Transferencias de Registro Maxima: 3
Numero de Transferencias de Registro Minima: 3
Numero de Transferencia de Datos           : 0
################ FORMATO DE DATOS  ################
Numero de variables tipo int               : 1
Numero de variables tipo short             : 0
Numero de variables tipo long              : 0
Numero de variables tipo char              : 0
Numero de variables tipo float             : 7
Numero de variables tipo double            : 0
Numero de variables tipo long double       : 0
Numero total de variables declaradas       : 8
############ COMPLEJIDAD ARITMETICA ###############
Numero de sumas                            : 3
Numero de restas                           : 0
Numero de multiplicaciones                 : 3
Numero de divisiones                       : 0
Numero de operaciones mod                  : 0
Numero total de operaciones                : 6
############## SENTENCIAS DE CONTROL ##############
Numero de sentencias if                 : 0
Numero de sentencias if nivel 0         : 0
Numero de sentencias if nivel 1         : 0
Numero de sentencias if nivel 2         : 0
Numero de sentencias if nivel 3         : 0
Numero de sentencias if nivel 4         : 0
Numero de sentencias if nivel 5         : 0
Numero de sentencias else               : 0
Numero de sentencias else nivel 0       : 0
Numero de sentencias else nivel 1       : 0
Numero de sentencias esle nivel 2       : 0
Numero de sentencias else nivel 3       : 0
Numero de sentencias else nivel 4       : 0
Numero de sentencias else nivel 5       : 0
Numero de sentencias for                : 1
```

```
Numero de sentencias for nivel 0       : 1
Numero de sentencias for nivel 1       : 0
Numero de sentencias for nivel 2       : 0
Numero de sentencias for nivel 3       : 0
Numero de sentencias for nivel 4       : 0
Numero de sentencias for nivel 5       : 0
Numero de sentencias switch            : 0
Numero de sentencias switch nivel 0    : 0
Numero de sentencias switch nivel 1    : 0
Numero de sentencias switch nivel 2    : 0
Numero de sentencias switch nivel 3    : 0
Numero de sentencias switch nivel 4    : 0
Numero de sentencias switch nivel 5    : 0
Numero de sentencias while             : 0
Numero de sentencias while nivel 0     : 0
Numero de sentencias while nivel 1     : 0
Numero de sentencias while nivel 2     : 0
Numero de sentencias while nivel 3     : 0
Numero de sentencias while nivel 4     : 0
Numero de sentencias while nivel 5     : 0
############## LISTA DE VARIABLES ##############
Variables Tipo int:
i
det

Variables Tipo float:
vector1
vector2
proesc
det
```

3 Resultados Fibonacci

A continuación se muestran los resultados del análisis en la herramienta, para el algoritmo de las Series de Fibonacci:

```
################## GENERALIDADES ################
Numero de lineas en codigo de entrada      : 16
Numero de dependencia de datos raw         : 0
Numero de dependencia de datos waw         : 0
Numero de dependencia de datos war         : 0
Numero de Operaciones fuera de sentencia   : 0
Numero de Transferencias fuera de sentencia: 0
Numero de Operaciones Maxima               : 60
Numero de Operaciones Minimo               : 60
Numero de Transferencias de Registro Maxima: 90
Numero de Transferencias de Registro Minima: 90
```

```
Numero de Transferencia de Datos           : 0
################ FORMATO DE DATOS  ############
Numero de variables tipo int               : 5
Numero de variables tipo short             : 0
Numero de variables tipo long              : 0
Numero de variables tipo char              : 0
Numero de variables tipo float             : 0
Numero de variables tipo double            : 0
Numero de variables tipo long double       : 0
Numero total de variables declaradas       : 5
############ COMPLEJIDAD ARITMETICA ############
Numero de sumas                            : 60
Numero de restas                           : 0
Numero de multiplicaciones                  : 0
Numero de divisiones                       : 0
Numero de operaciones mod                  : 0
Numero total de operaciones                : 60
############## SENTENCIAS DE CONTROL ######
Numero de sentencias if                 : 0
Numero de sentencias if nivel 0         : 0
Numero de sentencias if nivel 1         : 0
Numero de sentencias if nivel 2         : 0
Numero de sentencias if nivel 3         : 0
Numero de sentencias if nivel 4         : 0
Numero de sentencias if nivel 5         : 0
Numero de sentencias else               : 0
Numero de sentencias else nivel 0       : 0
Numero de sentencias else nivel 1       : 0
Numero de sentencias esle nivel 2       : 0
Numero de sentencias else nivel 3       : 0
Numero de sentencias else nivel 4       : 0
Numero de sentencias else nivel 5       : 0
Numero de sentencias for                : 1
Numero de sentencias for nivel 0        : 1
Numero de sentencias for nivel 1        : 0
Numero de sentencias for nivel 2        : 0
Numero de sentencias for nivel 3        : 0
Numero de sentencias for nivel 4        : 0
Numero de sentencias for nivel 5        : 0
Numero de sentencias switch             : 0
Numero de sentencias switch nivel 0     : 0
Numero de sentencias switch nivel 1     : 0
Numero de sentencias switch nivel 2     : 0
Numero de sentencias switch nivel 3     : 0
Numero de sentencias switch nivel 4     : 0
Numero de sentencias switch nivel 5     : 0
Numero de sentencias while              : 0
Numero de sentencias while nivel 0      : 0
Numero de sentencias while nivel 1      : 0
Numero de sentencias while nivel 2      : 0
```

```
62  Numero de sentencias while nivel 3     : 0
63  Numero de sentencias while nivel 4     : 0
64  Numero de sentencias while nivel 5     : 0
65  ############## LISTA DE VARIABLES ##############
66  Variables Tipo int:
67  n
68  cont
69  a
70  b
71  c
72  det
```