Instituto Tecnológico de Costa Rica

Escuela de Ingeniería en Electrónica

Optimización del tiempo de ejecución de las pruebas aplicadas a los microprocesadores IA-32 e IA-64

Informe de Proyecto de Graduación para optar por el título de Ingeniero en Electrónica con el grado académico de Licenciatura

Marco Espinoza Murillo

Cartago, Junio de 2010

INSTITUTO TECNOLOGICO DE COSTA RICA

ESCUELA DE INGENIERIA ELECTRONICA

PROYECTO DE GRADUACIÓN

TRIBUNAL EVALUADOR

Proyecto de Graduación defendido ante el presente Tribunal Evaluador como requisito para optar por el título de Ingeniero en Electrónica con el grado académico de Licenciatura, del Instituto Tecnológico de Costa Rica.

Miembros del Tribunal

Ing. Carlos Badilla Corrales, M.Sc.

Profesor lector

Ing. Leonardo Rivas Arce

Profesor lector

Dr.-Ing. Alfonso Chacón Rodríguez

Profesor asesor

Los miembros de este Tribunal dan fe de que el presente trabajo de graduación ha sido aprobado y cumple con las normas establecidas por la Escuela de Ingeniería Electrónica

Cartago, 22 de junio de 2010

Declaración de autenticidad

Declaro que el presente Proyecto de Graduación ha sido realizado enteramente por mi persona, utilizando y aplicando literatura referente al tema e introduciendo conocimientos propios.

En los casos en que he utilizado bibliografía, he procedido a indicar las fuentes mediante las respectivas citas bibliográficas.

En consecuencia, asumo la responsabilidad total por el trabajo de graduación realizado y por el contenido del correspondiente informe final.

23 de junio de 2010

Marco Vinicio Espinoza Murillo

2-0623-0003

Resumen

El costo de manufactura de los productos de Intel depende de muchas variables tal como el tiempo total que un producto toma completando su programa de prueba (tiempo de prueba).

Este proyecto consiste en realizar una herramienta que se encargue de optimizar el tiempo de ejecución de las pruebas actuales sin impactar la cobertura de las mismas y genere ahorros significantes de producción.

La reducción del tiempo de las pruebas funcionales en IA-32 e IA-64 es posible removiendo instrucciones en la inicialización y ejecución de cada prueba. El estudio realizado en este proyecto demuestra que existen muchos casos de optimización que se pueden implementar. Para ello se desarrolló una herramienta que toma una lista de pruebas y optimiza y simula cada una usando un simulador de ensamblador. La herramienta analiza el código de ensamblador de cada prueba y determina el grupo de instrucciones que pueden ser removidas o modificadas para reducir el tiempo de ejecución.

Esto se logra debido al análisis de un grupo de scripts sobre las pruebas (cada script es un caso de optimización a analizar) responsable de encontrar y reducir las instrucciones. El diseño modular de la herramienta permite agregar sin ningún problema nuevas optimizaciones que se encuentren en el futuro sin alterar las que ya están disponibles.

Las optimizaciones actuales fueron tomadas de la inicialización de la prueba, la cual no genera cobertura adicional sobre el producto y en la etapa de ejecución de la misma cuidando que la cobertura no sea afectada.

Palabras Clave: IA-32, IA-64, tiempo de prueba, cobertura de errores, ensamblador.

Abstract

Intel’s product manufacturing cost depends on many variables; one of them is the total time that a product’s test program takes to complete, also called Test Time.

This project consists in develop a test optimization tool that focus on test time reduction without impacting the test coverage and generate significant production savings.

Reducing the functional test time in IA-32 and IA-64 architecture is possible by removing unnecessary instructions in the initializations and execution of each test. Our initial investigation shows that there are many different test cases that can be improved. We have developed a tool that reads a list of tests to optimize and simulates each test using an IA architecture simulator; the tool analyzes the test’s assembly code to determine the group of instructions that can be removed or modified to reduce test execution time.

The tool runs a set of analysis scripts (each script is an optimization case that can be analyzed) responsible of finding and fixing instructions in the test. The modular design of the tool allows for the future integration of new optimization cases, as more test time reduction cases are found.

These optimizations are implemented mostly during initialization stages, because initialization does not provide any additional fault coverage, and in the execution stages without impacting the fault coverage.

Keywords: IA-32, IA-64, Test Time, Fault Coverage, Assembly code.

Agradecimiento

A Dios porque gracias a él logré finalizar satisfactoriamente esta carrera, además por estar siempre conmigo tanto en los buenos momentos como en los malos.

A toda mi familia, ya que siempre estuvieron conmigo apoyándome a lo largo de mi carrera profesional, en especial a mis padres porque si no hubiera sido por ellos este logro no lo habría alcanzado.

A la familia Pizarro Morales, por toda la ayuda y apoyo que me brindaron en el final de mi carrera,

Al profesor asesor Alfonso Chacón, por la asesoría, comprensión y apoyo que me aportó en la realización de este proyecto de graduación.

A todas las personas que de una u otra forma intervinieron para que este logro fuera posible.

Marco Vinicio Espinoza Murillo

Junio 2010

Dedicatoria

Este proyecto se lo dedico a Dios porque gracias a Él he logrado las metas que me propuesto en mi vida.

También se lo dedico a toda mi familia, en especial a mis abuelos, hermanos y a mis padres Marco Antonio Espinoza Quesada y Milena Murillo López, porque ellos siempre confiaron en mí y estuvieron conmigo tanto en los buenos como en los malos momentos, dándome todo lo necesario para lograr este proyecto y mi carrera profesional. A ellos les debo todos mis logros.

Marco Vinicio Espinoza Murillo

Junio 2010

.

Índice general

Índice de Figuras

Índice de Tablas

Capitulo 1: Introducción

La etapa de prueba de los microprocesadores de Intel es una de las más importantes de todas, ya que de esta depende que el producto sea lanzado al mercado o no. Es por esta razón que cada microprocesador pasa por miles de pruebas (entre 3000 y 7000) para que sea validado correctamente.

Debido a la cantidad de pruebas que se utilizan por microprocesador, una reducción del tiempo de ejecución de cada una de ellas lograría un aumento en la producción de microprocesadores generando mayor ganancia para la empresa.

Es a partir de ahí que toma importancia encontrar una manera para reducir de manera automática las pruebas que se tienen actualmente y las que se generarán a futuro.

1.1 Definición del problema

1.1.1 Generalidades

Cada prueba se encarga de probar una parte específica del microprocesador, y el tiempo que dura cada prueba depende de cuántas instrucciones posea y qué lógica es la que se está ejecutando.

Las pruebas presentan una parte de inicialización en la cual se configuran los modos de operación que se van a utilizar, tales como activar interrupciones, activar el modo protegido o la memoria virtual entre otras; esto consume tiempo debido a que se tiene que ir verificando qué se debe y qué no se debe activar según los requerimientos de la prueba.

Por causa del tiempo que toma cada prueba realizando esta inicialización y a la cantidad de pruebas que se deben de utilizar para cada microprocesador, es útil buscar una forma de optimizarlas, determinando patrones comunes y redundantes para así reducir el tiempo que consumen estas pruebas ejecutándose.

1.1.2 Síntesis del problema

Los patrones de prueba de los microprocesadores Intel IA-32 e IA-64 presentan secuencias de inicialización, ejecución, configuración y programación redundantes, lo cual conlleva a un incremento innecesario del tiempo total de ejecución de las pruebas, haciendo este proceso ineficiente.

1.2 Solución seleccionada

Una de las opciones para poder optimizar los patrones de prueba que se utilizan para los microprocesadores Intel consiste en analizar los patrones de inicialización y ejecución de la prueba.

La etapa de inicialización es similar para todas las pruebas, y cada vez que una prueba arranca se empieza a verificar si se deben de configurar cada uno de los modos de operación, lo que consume tiempo debido a las comparaciones y configuraciones que se deben ir realizando. Es por ello que se deben estudiar los patrones de inicialización de cada una de las pruebas basándose en la arquitectura del microprocesador, e ir comparando la cantidad de pruebas que contienen los patrones encontrados, para así centrarse en los más importantes.

Para esto se debe diseñar y crear una herramienta de software encargada de tomar cada una de las pruebas y detecte qué partes del código, ya sea en la inicialización o durante la ejecución del mismo, pueden eliminarse de la prueba y, en caso afirmativo modificar la prueba original y sustituirla con un código optimizado. La nueva prueba no debe presentar errores de simulación, además debe mantener el mismo porcentaje de cobertura que la prueba anterior y reducir el tiempo de ejecución.

Para que esto sea factible se deben conocer las microinstrucciones del microprocesador, ya que es en base a estas que se va a hacer la optimización.

Capítulo 2: Metas y objetivos

En el diseño, validación y ejecución de un proyecto es sumamente importante definir sus límites y alcances, ya que esto es la base que ayudará a llevar al éxito el proyecto si se definieron correctamente desde el inicio.

Es por ello que desde un inicio se definieron los objetivos y la meta que se desea alcanzar en la realización de este proyecto. La idea fue realizar una herramienta capaz de optimizar las pruebas existentes para validar los microprocesadores Intel de la familia IA 32 e IA 64.

2.1 Meta

Este proyecto busca reducir el tiempo de ejecución de las pruebas aplicadas a los microprocesadores Intel de la arquitectura Sandy Bridge por medio de la reducción de su secuencia de inicialización y ejecución. Dependiendo del tiempo reducido así será el ahorro en dinero que se obtiene por cada millón de unidades probadas.

En este caso la meta es reducir dicho tiempo en al menos un 1%. Para ello se creará una herramienta que sea capaz de analizar, modificar y reducir el tiempo de ejecución de las pruebas manteniendo el porcentaje de cobertura de cada prueba.

2.2 Objetivo General

1. Generar un sistema de pruebas para microprocesadores de la arquitectura Sandy Bridge que permita acelerar el proceso de prueba, de modo tal que se reduzca en al menos un 1% el tiempo de ejecución de un conjunto especificado de pruebas, sin afectar el porcentaje de cobertura.

Indicador: Comprobar que se redujo en un 1% el tiempo de ejecución del conjunto de pruebas utilizadas manteniendo el porcentaje de cobertura de estas.

2.3 Objetivos Específicos

1. Definir un conjunto de patrones de inicialización que estén dentro de las pruebas del microprocesador bajo análisis, que permitan configurar el modo de operación del microprocesador y puedan sustituirse por nuevos patrones optimizados para disminuir el tiempo de ejecución de cada prueba.

Indicador: Un conjunto de nuevas pruebas que presenten patrones que reduzcan el tiempo de simulación respecto al que toman los patrones de las pruebas equivalentes anteriores.

2. Desarrollar una herramienta de software que analice un patrón de inicialización de la prueba de un microprocesador y lo reduzca de forma que:

   a. Disminuya el tiempo de ejecución de la prueba por ejecutar.

   b. Mantenga el porcentaje de cobertura de la prueba por ejecutar.

Indicador: La herramienta de software modifica y reduce las pruebas del microprocesador de forma automática. Además mantiene el porcentaje de cobertura de cada prueba y el total de pruebas reducen en al menos un 1% el tiempo total de ejecución.

Capítulo 3: Marco Teórico

En este capítulo se comentará sobre los principales conceptos relacionados con el problema y la solución, se explicará la descripción del proceso a mejorar.

3.1. Descripción del proceso a mejorar

Todos los microprocesadores producidos por Intel deben de someterse a un conjunto de pruebas para verificar su correcto funcionamiento. Estas son diseñadas para validar diferentes regiones del microprocesador y las personas encargadas de escribirlas son llamadas “test writers” que en español significa diseñadores de pruebas.

Debido a que el objetivo de las pruebas es aportar cobertura adicional al microprocesador existen programas que se encargan de escribirlas basado en el área que se desea probar. Sin embargo estas herramientas se enfocan solamente en la cobertura de la prueba y no en el tiempo de ejecución; por esta razón hacen las mismas configuraciones de inicialización para todas, sin que esto sea muchas veces necesario.

De lo anterior surge la idea de diseñar y generar una herramienta capaz de detectar y eliminar cualquier patrón de inicialización innecesario en una prueba de cobertura.

Para ello el proyecto consta de una parte inicial de investigación, donde se debe de hacer un estudio de las pruebas para encontrar los patrones que pueden ser modificados o removidos y otra parte de ejecución que consiste en diseñar y programar la herramienta para que automáticamente modifique las pruebas.

3.2 Descripción de los principales principios electrónicos relacionados con la solución del problema.

Para la implementación de este proyecto fue necesario estudiar diferentes conceptos y configuraciones que tienen que ver con los microprocesadores de Intel. Esto para entender las diferentes instrucciones ejecutadas por las pruebas y así poder determinar patrones innecesarios que pueden ser modificados o excluidos.

Seguidamente se da una explicación general de estos conceptos.

3.2.1. Arquitectura de los microprocesadores IA 32 e IA 64

Es importante entender el funcionamiento y la arquitectura de los microprocesadores Intel ya que para poder analizar las pruebas es necesario conocer las diferentes transacciones y los distintos modos de operación que pueden hacerse, para así poder determinar si se hacen configuraciones innecesarias que pueden eliminarse de una prueba cualquiera.

3.2.1.1. Modos de Operación

Para los procesadores IA32 existen tres modos y un cuasi modo de operación, en IA 64 se presenta un modo adicional denominado IA32e. Estos se citan a continuación:

I. Modo Real

El microprocesador siempre arranca en este modo de operación. Provee el ambiente del Intel 8086 con pequeñas extensiones (como habilitar el paso a modo protegido o al modo de manejo del sistema).

II. Modo Protegido

Este modo provee múltiples funcionalidades, tales como la protección de memoria, paginación, soporte de un manejo de memoria virtual así como potenciar la multitarea y mejorar la estabilidad del sistema.

III. Modo de manejo del sistema (SMM)

A este modo se accede por la activación del pin de interrupción (#SMI), el cual genera una interrupción de manejo de sistema (SMI). Aquí el procesador utiliza un espacio de memoria separado, donde guarda el contexto del programa o la tarea que se está ejecutando.

IV. Modo Virtual-8086

En el modo protegido, el microprocesador soporta un cuasi modo de operación conocido como virtual-8086. Este permite ejecutar software del 8086 en modo protegido y en un ambiente de multitarea.

V. Modo IA-32e

Está presente solo en microprocesadores IA 64. Este modo soporta dos submodos: modo de compatibilidad y modo de 64 bits. El modo de 64 bits provee un direccionamiento de memoria que soporta hasta 64 GBytes[1].

El modo de compatibilidad permite más control en aplicaciones ejecutadas en modo protegido.

La figura 3.1 presenta un diagrama de cómo el microprocesador accede a los diferentes modos de operación:

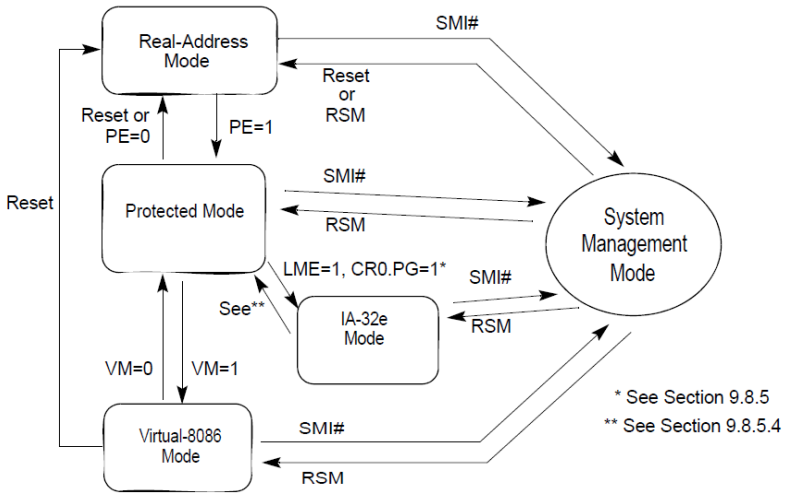

Figura 3.1. Transición entre los diferentes modos de operación del microprocesador[1].

El microprocesador inicia en modo real, para pasar a modo protegido debe de escribir el valor de uno en la variable PE, la cual es un bit del registro de control CR0.

En modo protegido se puede pasar a modo virtual modificando la variable VM, que se encuentra en el registro de banderas, o también se puede pasar al modo IA-32e escribiendo la variable LME y PG del registro CR0, tal y como se observa en la figura 3.1.

También sin importar el modo en que se encuentre el microprocesador, si se da la interrupción SMI, el mismo pasa al modo de manejo del sistema.

3.2.1.2. Registros de Control

Existen 5 registros de control (CR0, CR1, CR2, CR3 y CR4) que determinan el modo de operación del microprocesador y las características de la tarea en ejecución.

Registro de Control 0 (CR0)

Controla el modo de operación y el estado actual del microprocesador. La figura 3.2 muestra como están organizados los bits de este registro. [1].

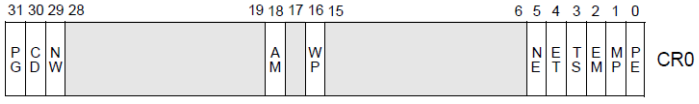

Figura 3.2. Bits de configuración del registro CR0 [1].

Respectivamente, cada bit tiene el siguiente significado:

- PG: Habilita la paginación de la memoria.
- CD: Habilita y deshabilita la cache.
- NW: Configura la cache.
- AM: Habilita el chequeo automático de alineamiento a 16 bits.
- PE: Habilita el modo protegido.

Registro de Control 1 (CR1)

Este registro es de uso reservado.

Registro de Control 2 Y 3 (CR2 Y CR3)

El registro CR2 contiene la dirección lineal que causa la paginación y el registro CR3 contiene la base de la dirección de la página. En la figura 3.3 se muestran ambos registros.

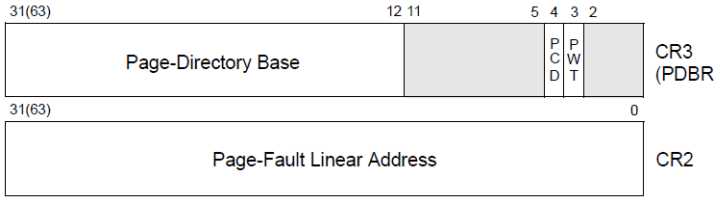

Figura 3.3. Distribución de bits de los registros CR2 y CR3 [1].

Registro de Control CR4

Habilita diversas extensiones de la arquitectura. En la figura 3.4 se muestra el registro y los bits que lo conforman.

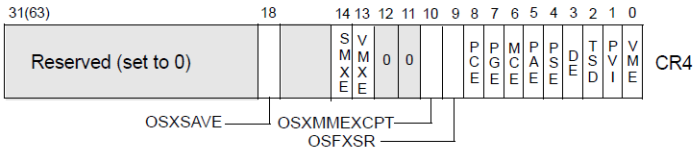

Figura 3.4. Bits de configuración del registro CR4 [1].

Los bits más importantes del registro CR4 son:

- VME: Habilita interrupciones y excepciones manejadas en el modo virtual 8086.
- PSE: Habilita páginas de tamaño de 2 o 4 MB.

- PVI: Habilita el hardware de soporte para las interrupciones virtuales en modo protegido.
- PAE: Habilita el mecanismo de paginación para tener direcciones físicas de 36 bits.

3.2.1.3. Registros para el manejo de memoria

El microprocesador posee cuatro registros para el manejo de memoria (GDTR, LDTR, IDTR y TR) que especifican la localización de las estructuras de datos que son manejados por segmentos de control. Se necesitan instrucciones especiales para escribir estos registros.

La figura 3.5 detalla los registros utilizados para el manejo de memoria.

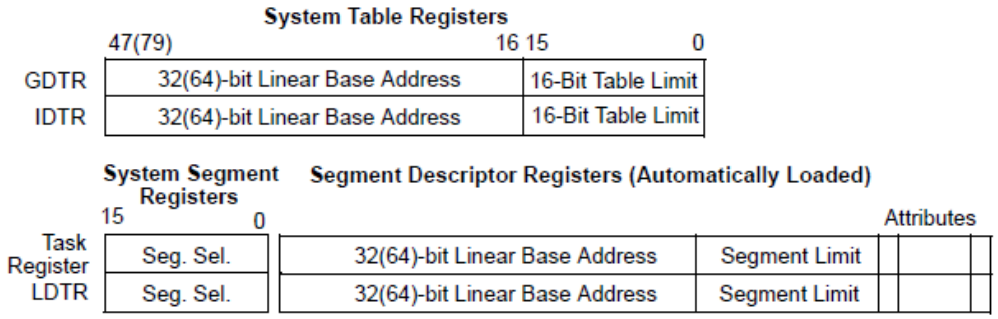

Figura 3.5. Registros utilizados para el manejo de la memoria [1].

Registro para la tabla de descriptores globales (GDTR)

El GDTR tiene la dirección de memoria lineal base (es de 32 bits en modo protegido o 64 bits en modo IA-32e) y el límite de la tabla del GDT (tabla de descriptores globales) que tiene un tamaño de 16 bits. La dirección base especifica la dirección lineal del byte 0 del GDT y el límite de la tabla especifica el numero de bytes en la tabla [1].

Las instrucciones que se utilizan para leer y escribir este registro son LGDT y SGDT respectivamente.

Registro para la tabla de descriptores locales (LDTR)

Este registro tiene un segmento selector de 16 bit, una dirección base (de 32 bits en modo protegido y 64 bits en modo IA-32e), un segmento límite y un descriptor de atributos para el LDT. Al igual que en el GDTR la dirección base especifica la dirección

lineal del byte 0 del LDT, el límite del segmento especifica el número de bytes en el segmento [1].

Las instrucciones para leer y escribir este registro son LLDT y SLDT respectivamente.

Registro para la tabla de descriptores de interrupciones (IDTR)

Al igual que los registros anteriores tiene una dirección base (32 bits en modo protegido y 64 bits en el modo IA-32e) y 16 bits para el límite de la tabla de IDT. La dirección base especifica la dirección lineal del byte 0 del IDT y el límite de la tabla especifica el número de bytes en la tabla [1].

Las instrucciones de lectura y escritura a este registro son LIDT y SIDT.

3.2.1.4. Tabla de descriptores globales y locales

Cuando se opera en modo protegido, todos los accesos de memoria pasan por la tabla de descriptores globales o opcionalmente por la tabla de descriptores locales. Estas tablas contienen entradas llamadas segmentos descriptores, el cual provee la base de la dirección de los segmentos a accesar y el tipo (si es GDT o LDT).

Cada segmento descriptor tiene asociado un segmento selector, para accesar un byte en un segmento, un segmento selector y un offset deben ser aplicados. Con el segmento selector se tiene el acceso al segmento descriptor y desde este el procesador obtiene la base de la dirección del segmento en la dirección de memoria lineal. El offset contiene la localización del byte relativo desde la base de la memoria[1].

Este mecanismo puede ser usado para accesar cualquier línea válida de código, datos o un segmento de pila.

La figura 3.6 muestra un ejemplo de esta tabla.

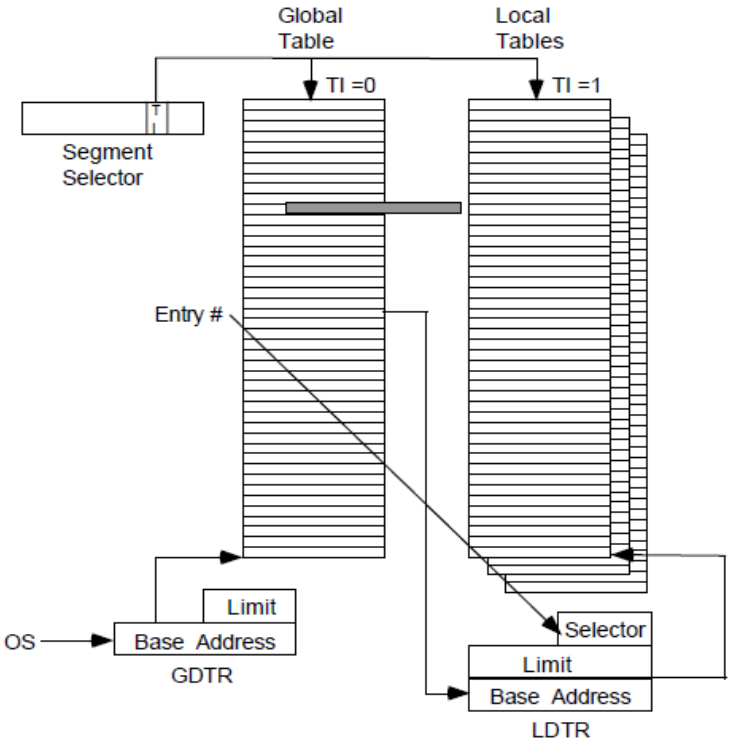

Figura 3.6. Tabla de descriptores globales y locales [1].

Del segmento selector se lee el bit TI, el cual indica si el acceso es al GDT o al LDT. Del registro GDTR se extrae la base de la dirección de memoria donde se encuentra la tabla a accesar y con el offset del segmento selector se determina el campo dentro de la tabla al que se desea acceder.

3.2.1.5. Manejo de interrupciones y excepciones

Las interrupciones y excepciones son eventos que indican que existe algo en el sistema, en cualquier programa o tarea en ejecución, que requieren atención del microprocesador. El programa o tarea en ejecución ejecuta una rutina especial que activa la interrupción. En respuesta, el microprocesador atiende la misma por medio del manejador de interrupciones o excepciones [1].

Las interrupciones pueden ocurrir en cualquier momento durante la ejecución de un programa y se pueden activar por medio de hardware a través del pin de interrupciones o por un código que ejecute la instrucción INT.

Las excepciones ocurren cuando el microprocesador detecta una condición de error al ejecutar una instrucción, como por ejemplo una división por cero. El procesador también detecta condiciones de error tales como violaciones de protección, fallo en la paginación y fallas por errores internos de la máquina.

Cuando se recibe una interrupción o una excepción se detiene inmediatamente las tareas que se están realizando y se invoca a una rutina de atención de interrupción o excepción.

Las interrupciones pueden ocurrir tanto en el modo real como en el modo protegido, pero el manejo de la misma es diferente para estos modos.

Interrupción en modo real

- El contenido del registro de banderas se guarda en la pila.
- Se borran las banderas de interrupción (IF) y de atrapamiento (TF). Esto deshabilita la terminal INTR y la característica de atrapamiento o de paso individual.
- El contenido del registro del segmento de código (CS) se mete en la pila.
- El contenido del apuntador de instrucciones (IP) se mete en la pila.
- Se obtiene el contenido del vector de interrupción y luego se coloca en IP y en CS para que la siguiente instrucción se ejecute en el procedimiento de servicio de interrupciones direccionando el vector [1].

Interrupciones en modo protegido

Tienen las mismas asignaciones que en modo real, pero con la diferencia de que la tabla de vectores de interrupción es distinta, ya que se cuenta con un conjunto de 256 descriptores de interrupción que se almacenan en una tabla de descriptores de interrupción.

Por lo tanto las interrupciones externas, por software y las excepciones son manejadas por esta tabla de descriptores de interrupción (IDT). Los descriptores de interrupción proveen el acceso al manejador de interrupciones y de excepciones. Cada descriptor es de 8 bytes y tal como en el GDT, el IDT no es un segmento. La dirección lineal para la base del IDT está contenida en el registro IDTR [1].

Para acceder al manejador de interrupciones el procesador recibe un vector de interrupción, este contiene el índice dentro del IDT. Si se trata de una interrupción se llama al manejador de interrupciones para que la atienda.

3.2.1.6. Registros de modelo específico (MSRs)

Los registros MSRs son registros de 64 bits que proveen control de software y hardware como por ejemplo:

- Rendimiento y monitoreo del microprocesador.
- Establecimiento de los rangos de memoria para registros.
- Manejo de energía y temperatura.

- Propiedades del microprocesador.

Estos registros pueden leerse y escribirse utilizando las instrucciones de RDMSR y WRMSR respectivamente [1].

3.2.1.7. Controlador avanzado de interrupciones (APIC):

Este es un controlador que es utilizado para la multitarea. Entre sus principales funciones están:

- Recibir interrupciones desde el pin de interrupción del microprocesador generados por dispositivos locales (conectados directamente al microprocesador) y desde el I/O APIC (u otro controlador de interrupciones externo) [1].
- En múltiples procesadores envía y recibe mensajes desde interrupciones provenientes de otros procesadores [1].

El I/O APIC es parte del chipset. Su función principal es recibir interrupciones y eventos de dispositivos externos.

El local APIC consiste de un grupo de registros para controlar las interrupciones y mensajes que se generan, que pueden venir por ejemplo desde:

- Dispositivos locales, tales como un dispositivo conectado directamente al microprocesador.
- Dispositivos externos tales como un dispositivo que está conectado al I/O APIC.
- Interrupciones entre procesadores (IPIs), ya que un procesador puede utilizar IPIs para interrumpir otros procesadores conectados al mismo bus.

La figura 3.7 ilustra cómo se relacionan el local APIC y el I/O APIC en una conexión en la que solamente se tiene un procesador y dispositivos externos conectados al chipset.

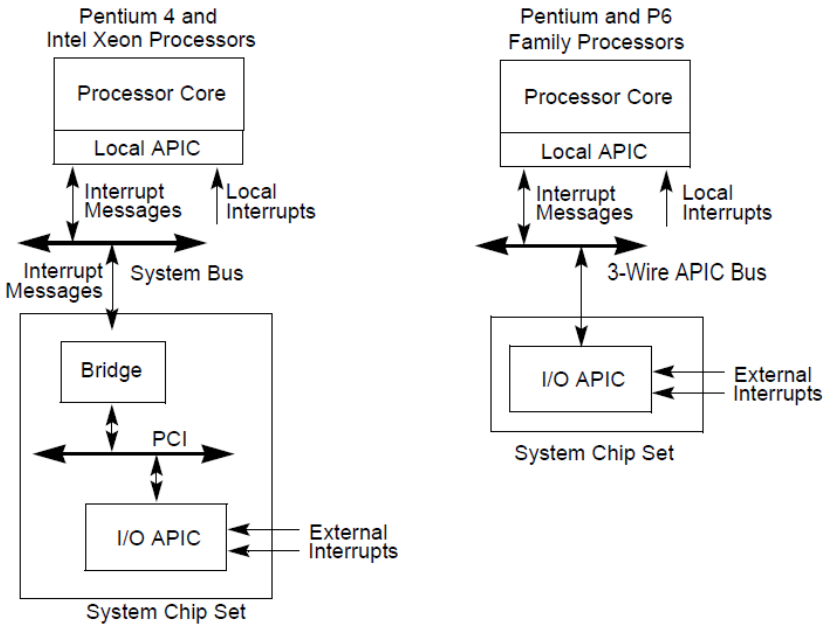

Figura 3.7. Relación entre el local APIC y el I/O APIC utilizando solo un procesador [1].

Como se observa en la figura 3.7, el I/O APIC está conectado al chipset y al I/O APIC se conectan dispositivos externos. El local APIC está conectado directamente en el procesador y puede recibir interrupciones locales de dispositivos conectados directamente al mismo o interrupciones de dispositivos externos que son enviadas desde el I/O APIC por el FSB.

Sin embargo como se mencionó anteriormente la utilidad principal que se le da a este dispositivo es para sistemas multiprocesadores. Un ejemplo de este tipo de arquitectura, se muestra en a figura 3.8:

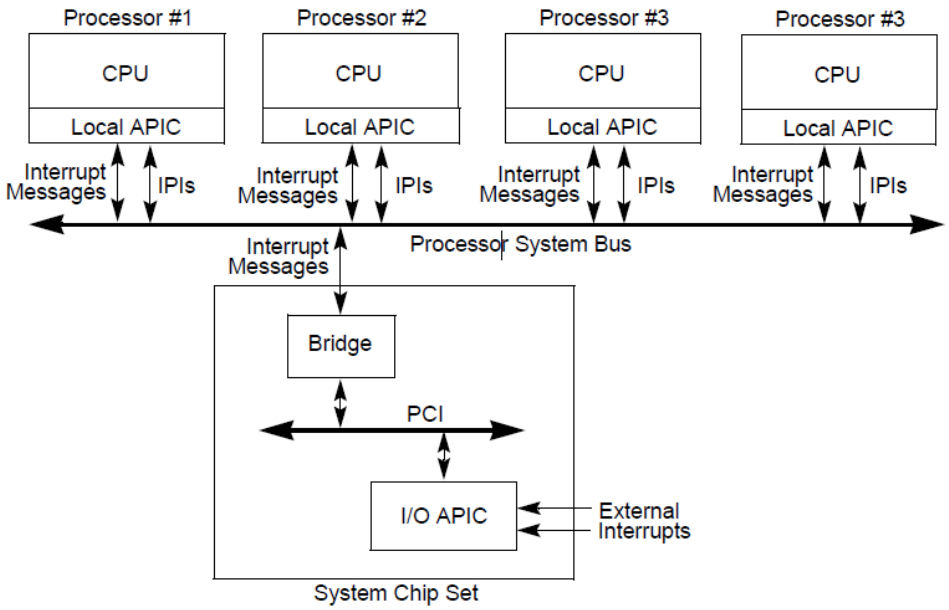

Figura 3.8. Local APICs y I/O APIC utilizados en una configuración de multiprocesadores [1].

En este caso se tienen 4 procesadores conectados en paralelo al mismo bus de mensajes y datos, cada uno posee un local APIC y en el chipset está el I/O APIC. Cuando un procesador desea manejar a otro u otros procesadores, lo debe hacer enviando IPIs, así el procesador central puede controlar a los demás y estos responden de igual manera permitiendo la sincronización del sistema.

3.2.2. Pruebas funcionales

El objetivo de las pruebas funcionales es validar la operación correcta de un sistema respecto a las características funcionales de este. Esto puede realizarse de dos maneras:

I. Haciendo una aproximación con un modelo de fallas y tratar de generar pruebas que detecten las fallas definidas en este modelo.
II. Obtener pruebas basadas en el comportamiento especificado sin hacer un modelo de fallas [2].

Debido a que el modelo del microprocesador para el que están escritas las pruebas utilizadas en este proyecto está basado en un modelo de fallas, el programador busca cubrir la mayor cantidad de fallas posible con cada prueba que escribe, ya que de esto

es que depende la calidad de la misma y el cumplimiento de la meta de cobertura que debe de tener cada unidad del microprocesador.

Por esta razón es que todas las pruebas optimizadas no deben de reducir el porcentaje de cobertura sobre el microprocesador. Para ello se debe de tener cuidado de que la herramienta elimine instrucciones que no aporten cobertura adicional, ya que las fallas detectadas por estas debieron haber sido cubiertas por otros patrones.

3.2.3 Generación de Pruebas

La generación de pruebas es un proceso para determinar el estímulo necesario para probar un sistema digital, depende principalmente del método de prueba empleado.

Puede estar orientada a fallas o a funciones. Las pruebas trabajadas en este proyecto están orientadas a fallas ya que su objetivo es ejercitar la mayor cantidad de fallas del modelo del microprocesador. [2]

3.2.4. Simulación de Pruebas

Debido a que la ejecución de pruebas sobre el modelo de fallas del microprocesador tarda mucho tiempo, los diseñadores de pruebas cuentan con simuladores que le permiten ejecutarlas, con lo que pueden chequear cada una de las transacciones y configuraciones que ocurren para verificar si se están realizando correctamente.

La ejecución de la prueba en estos simuladores es bastante rápida, y se pueden obtener de ella todos los accesos a memoria que se realizan, las escrituras y lecturas de registros, las instrucciones procesadas y el modo de operación sobre el que se está trabajando, entre muchas otras opciones.

Para este caso se utilizó un simulador por comportamiento llamado ARCHSIM [3], que basado en la arquitectura del microprocesador simula las instrucciones de la prueba de una forma rápida, fácil y accesible para analizar el comportamiento de estas sin tener que esperar su ejecución sobre el modelo total.

Capítulo 4: Procedimiento metodológico

En este capítulo se explican a grandes rasgos cada una de las etapas del diseño de ingeniería que se siguieron en el desarrollo del proyecto, las que una vez identificado y definido el problema, lograron llegar a la solución del mismo. Además se comenta como se implementó la solución y algunos problemas que surgieron durante el diseño y la validación de la misma.

4.1 Reconocimiento y definición del problema

Las pruebas utilizadas actualmente en los microprocesadores IA 32 e IA 64 son generadas por medio de herramientas enfocadas solamente en identificar errores en la lógica del microprocesador, pero descuidan el tiempo de ejecución de las mismas.

Debido a que sobre cada microprocesador se ejecuta una cantidad considerable de pruebas (entre 3000 hasta 7000 pruebas), reducir el tiempo en cada una generaría un ahorro bastante elevado para la empresa en la producción de microprocesadores.

El proyecto conlleva una parte de investigación, en la cual se analizaron los recursos disponibles, la cantidad de pruebas existentes para el modelo sobre el que se trabajaría y el tiempo que se tardaba ejecutando cada una de las pruebas en el modelo del microprocesador y así se definieron la meta y los objetivos del proyecto.

Una vez iniciado el proyecto se realizaron reuniones semanales con el asesor de la empresa para llevar un control sobre lo que iba avanzando el proyecto y los problemas que fueron surgiendo a lo largo del desarrollo del mismo.

4.2 Obtención y análisis de la información

La problemática se obtuvo a partir de entrevistas con personal del grupo encargado de escribir estas pruebas en componentes Intel de Costa Rica.

Se hizo un análisis de las pruebas existentes para los microprocesadores actuales y se determinaron varios patrones que podrían ser optimizados debidos a que su ejecución es innecesaria, ya que al no generar excitación adicional sobre la arquitectura del microprocesador no se producen más detecciones de errores.

También en el estudio de las pruebas, se encontraron varias que utilizan direcciones de memoria diferentes y además códigos de segmentos distintos. Debido a que la etapa de reset de las pruebas se demora bastante tiempo, una manera de optimizar aún más las pruebas es haciendo combinaciones entre ellas, es decir, hacer de dos pruebas una sola para así ahorrar el tiempo de configuración y de inicialización, que puede llegar a ser hasta el 30% o incluso más del tiempo total de ejecución de la misma.

Después se investigó cuánta ganancia adicional se obtendría en la producción de microprocesadores si estas optimizaciones se realizaban para determinar si el proyecto era viable o no para la empresa.

El impacto que tiene esta reducción del tiempo de prueba en el costo de ensamble y prueba de producto se puede ver de la siguiente forma: una reducción de 100 ms en el tiempo de prueba en determinado producto ahorra $5000 por cada millón de unidades probadas. Suponiendo una producción de tres millones de unidades por mes, el ahorro sería de $180000 al año.

En el caso del microprocesador utilizado para este proyecto, una reducción de al menos un 1% en el tiempo de prueba del conjunto de pruebas puede reducir alrededor de 200 ms, lo que hace que el ahorro en dinero sea considerable y que el proyecto sea factible.

4.3 Evaluación de las alternativas y síntesis de una solución

Debido a que el objetivo del proyecto es la disminución del tiempo de ejecución de las pruebas, la única manera de lograr esto es reduciendo las instrucciones redundantes e innecesarias presentes en las mismas, por lo que una parte importante del proyecto fue el estudio del código de ensamblador de cada una de las pruebas para determinar así los patrones que podrían ser modificados o removidos.

De esta manera se definió una serie de pasos a seguir en el proceso inicial del proyecto para encontrar así estos patrones:

- Tomar las mejores 40 pruebas hasta ese momento, basado en la cobertura que daba cada una sobre el modelo del microprocesador, y con estas iniciar el estudio y la búsqueda de patrones de optimización.
- Estudiar detalladamente la etapa de inicialización de cada una de las pruebas.
- Encontrar patrones que pudieran ser removidos o modificados en instrucciones más sencillas que consuman menos tiempo en la ejecución de la prueba.

En caso de que los patrones encontrados en una prueba puedan ser eventualmente removidos o sustituidos por otros más sencillos, se procede a verificar si los mismos se encuentran presentes en otras pruebas, de manera que sea factible realizar la optimización.

Los pasos descritos anteriormente duraron aproximadamente tres meses. Luego se inició con el diseño de la herramienta, para la que se definió un modelo flexible, de manera que cuando se encuentre una nueva optimización permita agregarla a la herramienta de manera sencilla para el usuario.

La herramienta trabaja sobre el código de ensamblador de la prueba, debido a que el set de instrucciones de los microprocesadores IA es muy similar. El trabajar de esta manera hace el proyecto útil en cualquier microprocesador Intel.

Una vez diseñado y definido el modelo para la herramienta se inició su programación, lo que tardó alrededor de un mes. Después se empezó la programación de los algoritmos para que realizaran cada una de las optimizaciones.

Para cada optimización se hizo un script que debía analizar si se puede sustituir o eliminar el código de la prueba actual por un nuevo código reducido. Para ello la herramienta debe de determinar exactamente qué parte debe modificarse para no cambiar código no deseado, ya que esto haría que la prueba falle la ejecución.

Cada una de las alternativas de optimización encontradas en la parte de investigación fueron analizadas con el asesor de la empresa para determinar si las mismas eran factibles y cumplían con las expectativas de lo que la empresa buscaba obtener.

4.4 Implementación de la solución

A continuación se describen los pasos que se siguieron para la ejecución del proyecto:

4.4.1 Estudio de la información

La primera etapa del proyecto consistió en estudiar la arquitectura del microprocesador sobre el que se trabajó, así como el set de instrucciones del mismo para poder entender las instrucciones de ensamblador presentes en cada una de las pruebas del mismo.

Además se estudió el lenguaje de programación Perl, ya que en este se programó la herramienta por las siguientes razones:

- Los programas hechos en Perl soportan cualquier plataforma.
- Las expresiones regulares que presenta hacen que funciones que en otros lenguajes tomarían extensas líneas de código se puedan hacer en pocas líneas de código en Perl.
- Muchas herramientas de Intel están programadas en este lenguaje, por lo que si se necesitara acoplar esta herramienta a alguna otra, la compatibilidad es más sencilla.

4.4.2. Estudio de las pruebas

La siguiente etapa consistió en tomar las 40 pruebas que aportaban más cobertura al microprocesador para estudiar detalladamente la etapa de inicialización, configuración y ejecución de cada una de ellas y determinar así patrones que pudieran ser reducidos.

Una vez identificados los patrones en una prueba se buscaban en las demás para determinar si era factible realizar la optimización o no, dependiendo del número de veces que estas estuvieran presentes en cada una de las pruebas.

4.4.3. Diseño de la herramienta

El diseño de la herramienta inició basado en los patrones encontrados y al proceso que se decidió utilizar para ejecutar cada una de las pruebas en la herramienta.

4.4.4. Diseño del algoritmo para optimizar los patrones encontrados

Con la herramienta programada se inició el diseño de cada uno de los algoritmos que se utilizaron para realizar las optimizaciones. Este paso fue complejo ya que la herramienta debe de entender el código de la prueba actual para poder hacer las modificaciones correctas en el código de la misma.

Para cada optimización se realizó un script. De esta manera cuando se tienen nuevas optimizaciones solamente se debe de agregar a la herramienta el nuevo script, así no se afecta el diseño de la misma ni de las otras optimizaciones.

4.4.5. Programación de la herramienta y de las optimizaciones

Una vez diseñada la herramienta y los algoritmos de optimización se continuó con la programación de los mismos. Lo primero que se programó fue la herramienta y luego se hizo la programación de dos optimizaciones, esto para iniciar las primeras pruebas y verificar que la herramienta realizaba correctamente los algoritmos. Aquí fue necesario corregir algunos errores tales como:

- Inicialmente la herramienta no configuraba correctamente el simulador de ensamblador, esto hacía que todas las simulaciones de las pruebas fallaran.
- Cuando la herramienta estaba ejecutando la simulación de las pruebas, algunas nunca terminaban (quedaban en un ciclo infinito), por errores propios de la prueba, por lo tanto la herramienta no continuaba la ejecución de las demás, ya que quedaba esperando el fin de la que se estaba simulando.
- No se generaba el archivo con los resultados de la simulación, esto porque no se estaba configurando una variable de ambiente necesaria.

4.4.6. Validación de la herramienta y las optimizaciones

Para la validación se tomó un grupo de pruebas que se corrían en la herramienta. Esta identificaba las optimizaciones y modificaba el código de la prueba original. El nuevo código se imprimía en una nueva prueba y se corría en el simulador de ensamblador. El resultado de la simulación se comparaba con el de la prueba original y se verificaba que el número de instrucciones de la prueba optimizada es menor al de la prueba original.

Para obtener el tiempo que se redujo de la prueba original se sigue lo mostrado en la figura 4.1.

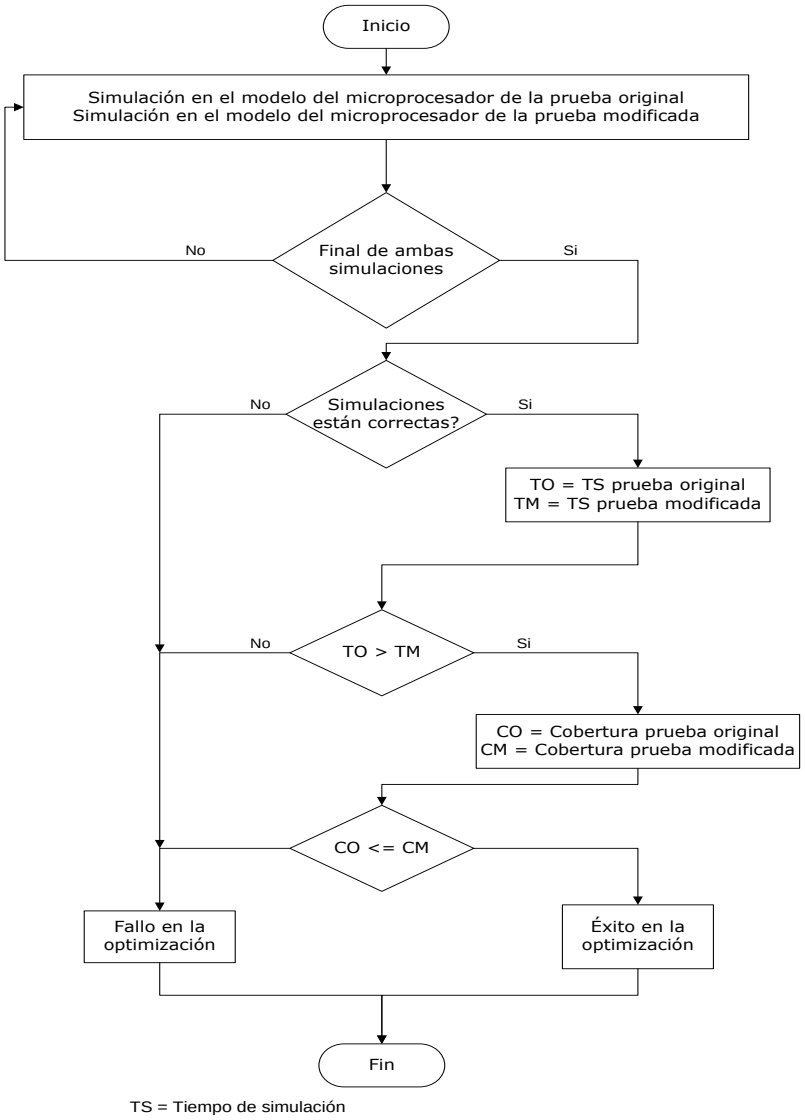

Figura 4.1. Diagrama de flujo del proceso seguido para determinar el tiempo de simulación reducido con la prueba optimizada.

Como ilustra la figura 4.1 para la obtención del tiempo reducido se simula tanto la prueba original como la optimizada en el modelo del microprocesador, cuando ambas simulaciones finalizan se verifica que ambas pasaron la simulación, y después se verifica si el tiempo de ejecución es menor en la prueba optimizada.

Inicialmente hubo problemas en este paso debido a que algunas de las pruebas optimizadas daban errores (se explican en la sección 4.4.7) cuando se simulaban en el modelo del microprocesador.

Después de verificar el tiempo de simulación se determinaba la cobertura que daban dichas pruebas sobre el modelo del microprocesador, en todos los casos la nueva prueba mantuvo el mismo porcentaje que la prueba original.

4.4.7. Pruebas finales y obtención de resultados

Una vez finalizada la validación se siguió con la obtención de resultados. Para ello se tomó un grupo de 135 pruebas que se optimizaron con la herramienta. Tanto las nuevas pruebas como las pruebas originales se enviaron a correr en el modelo del microprocesador y se siguió el diagrama de flujo de la figura 4.1 para cada una de ellas.

Esto tomó más tiempo debido a que hubo pruebas que fallaron la simulación. Se debió de revisar cada uno de los errores para solucionarlos y enviar a correr nuevamente las pruebas. Estos errores se mencionan a continuación:

- Cuando se eliminaba código de las pruebas, algunas cuando realizaban un salto (instrucción de *jmp*) caían en direcciones de memoria incorrectas y fallaban la simulación del modelo del microprocesador.
- La herramienta no identificaba correctamente las direcciones de memoria del código que debía optimizar. Por lo tanto eliminaba código que no pertenecía a la optimización haciendo que la prueba fallara o no realizara instrucciones necesarias.
- El modelo del microprocesador se cambió a otro servidor, al pasar la herramienta al mismo, las variables de ambiente del simulador de ensamblador no funcionaban correctamente, por lo tanto todas las pruebas fallaban la simulación.

Cuando los errores fueron solucionados se tomaron los resultados respectivos para hacer su respectivo análisis.

4.5. Revalidación y Rediseño

A pesar de que la herramienta funciona correctamente y que se cumplieron los objetivos propuestos, se pueden cambiar varias cosas que harían la herramienta aún más efectiva:

- La herramienta usa el simulador de ensamblador para ejecutar cada una de las pruebas y obtener cada uno de los datos que necesita de esta. Sin embargo la forma de correr el simulador varía entre los distintos modelos de cada microprocesador, por lo que cada cambio que se le haga a dicho simulador podría afectar la herramienta y por ende la optimización de las pruebas. Sería factible hacer que la herramienta no dependa de este simulador e interprete el código de las pruebas por sí sola, de esta manera se elimina esta dependencia.

- En este momento la ejecución de la herramienta se hace por línea de comando, para una mejor interacción con el usuario se podría hacer una interfaz gráfica de manera que se facilite visualizar las opciones que posee la misma.

Capítulo 5: Explicación detallada de la solución

Para el desarrollo de la solución del problema se consideraron distintas alternativas de diseño que se explicarán así como sus ventajas y desventajas. A través de este análisis se logró seleccionar la propuesta que mejor se adaptaba a los requisitos del proyecto y las condiciones de la empresa.

Finalmente se describirá detalladamente el funcionamiento de cada uno de los algoritmos necesarios para identificar optimizaciones y hacerlos efectivos en la prueba.

5.1 Análisis de soluciones y selección final

5.1.1 Optimización de pruebas analizando solo el código de cada prueba

La propuesta inicial consiste en que la herramienta analice el código de cada una de las pruebas sin ayuda de un simulador de ensamblador. Es decir se debe de tomar la prueba original y determinar el segmento de datos inicial para a partir de este seguir el código por toda la prueba e ir buscando cada una de las optimizaciones presentes en la misma. El algoritmo sigue un flujo como el de la figura 5.1.

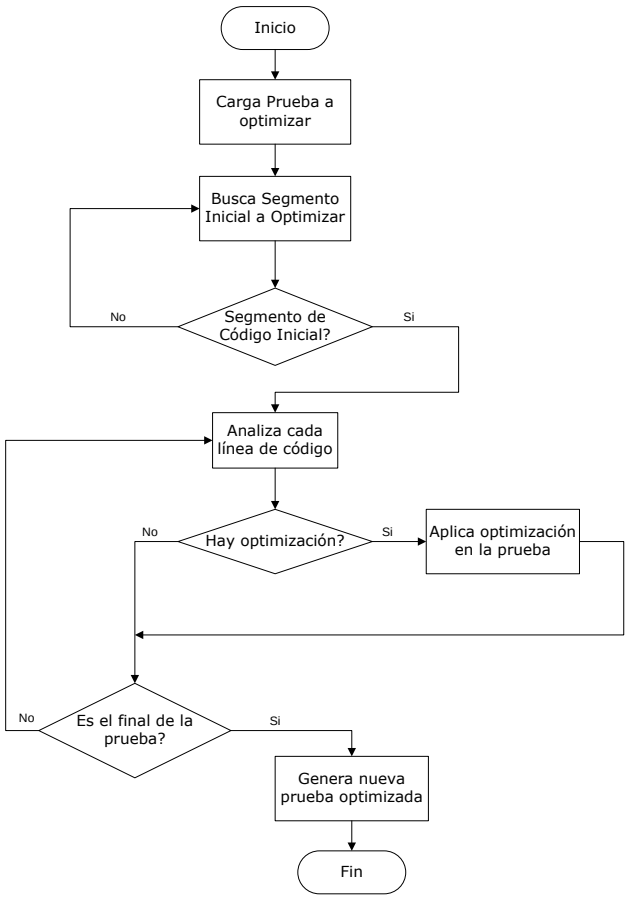

Figura 5.1. Diagrama de flujo de la primera solución propuesta.

Las ventajas que presenta este diseño son las siguientes:

- La herramienta es independiente de cualquier ambiente de ejecución, ya que solo necesita el código de la prueba para ejecutarse.
- No se necesita de ningún simulador de ensamblador para ejecutar la herramienta.

Por otro lado, sus desventajas son las siguientes:

- Se requiere de algoritmos complejos para poder determinar el segmento de código inicial manualmente y seguir el código de la prueba de la forma correcta.
- Si el algoritmo falla ya sea en la determinación del segmento de código inicial o en el seguimiento del código de la prueba, la lectura obtenida sería incorrecta y por ende la nueva prueba generada por la herramienta es errónea.
- La complejidad de los algoritmos harían la herramienta muy lenta, ya que se tarda mayor tiempo en el estudio y ejecución del código de cada prueba.

5.1.2. Optimización de pruebas utilizando un simulador de ensamblador

El diseño propuesto facilita la lectura de cada una de las pruebas, ya que por medio del simulador se puede determinar el segmento de código donde cada prueba inicia su ejecución y también se puede seguir más fácilmente el código de las mismas. La complejidad que lleva este modelo es que la herramienta debe de iniciar el simulador y ejecutarlo correctamente sobre la prueba de manera automática, sin embargo los algoritmos necesarios para ello son más sencillos que el del caso anterior.

El diagrama de flujo correspondiente a este diseño se presenta en la figura 5.2.

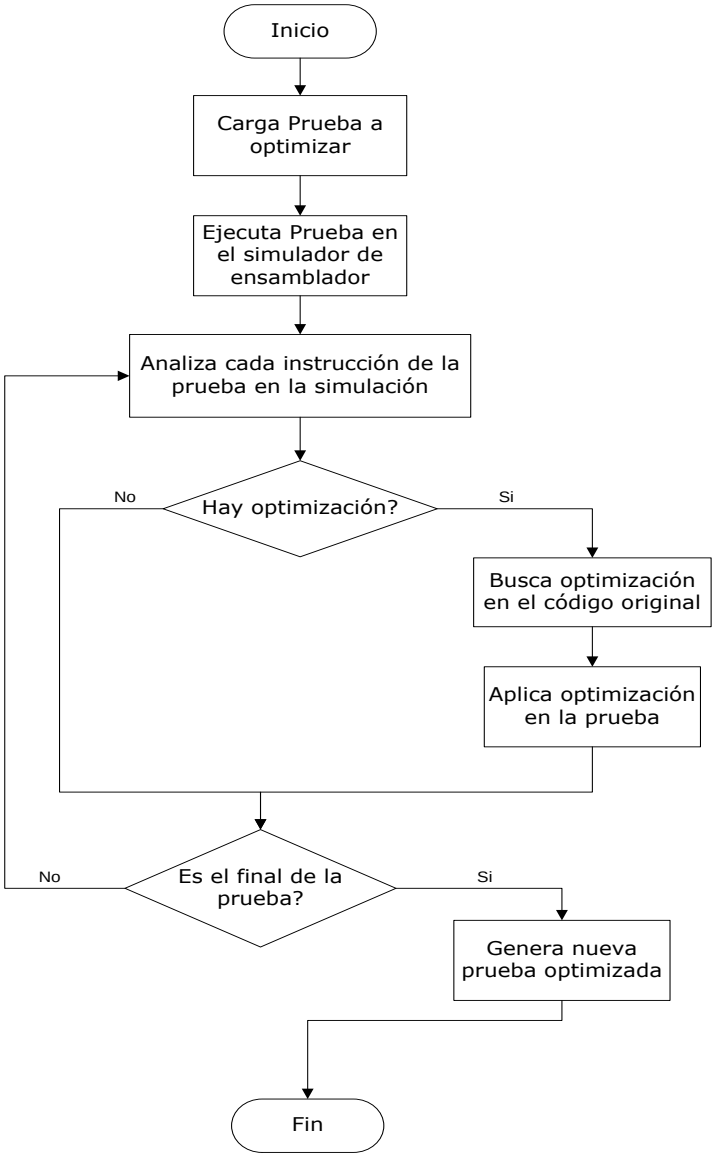

Figura 5.2. Diagrama de flujo de la segunda solución propuesta.

El diagrama de flujo es semejante al del caso 5.1.1 con la diferencia que no se trabaja directamente sobre la prueba, sino que se trabaja con la simulación de la misma. Por ello cuando se detecta una optimización, esta se debe de ir a buscar en la prueba original para poder aplicarla.

Las ventajas que presenta esta forma de optimización son las siguientes:

- Los algoritmos para la ejecución de la prueba son más sencillos.
- La ejecución de la prueba sobre la herramienta es más rápida debido a que el seguimiento de las instrucciones es más sencillo.

- Del simulador se puede extraer información como cambio en los registros del microprocesador, movimiento sobre los diferentes segmentos de código, cambios en el registro de banderas, etc.

Sin embargo también presenta desventajas tales como:

- Todo el estudio de la prueba se hace sobre la simulación de la misma. Esto hace que las optimizaciones encontradas deban buscarse en la prueba original. Para ello se debe diseñar y programar un algoritmo que identifique en qué segmento de código está la optimización y cuál posición de memoria física requiere para así identificar la optimización en la prueba.
- Depende de un ambiente de simulación para poder determinar las optimizaciones de cada una de las pruebas.

El problema que presenta este modelo es que en el simulador es fácil determinar la posición de memoria donde se encuentra el código de la optimización. Sin embargo en el ensamblador de la prueba no es fácil porque el código aún no ha sido compilado, por lo que las direcciones de memoria están en su formato relativo. Lo anterior vuelve difícil calcular las posiciones finales de las mismas.

5.1.3. Optimización de pruebas utilizando un simulador de ensamblador y los archivos de compilación de la prueba

Cuando se compila el ensamblador de cada una de las pruebas se generan varios archivos que presentan información de la misma. Uno de ellos contiene las posiciones de memoria de código donde se encuentran las instrucciones de la prueba, lo que permite determinar dónde está cada optimización encontrada en la simulación, haciendo los algoritmos de identificación de optimizaciones mucho más sencillos que los casos 5.1.1 y 5.1.2. El diagrama de flujo de este modelo se muestra en la figura 5.3.

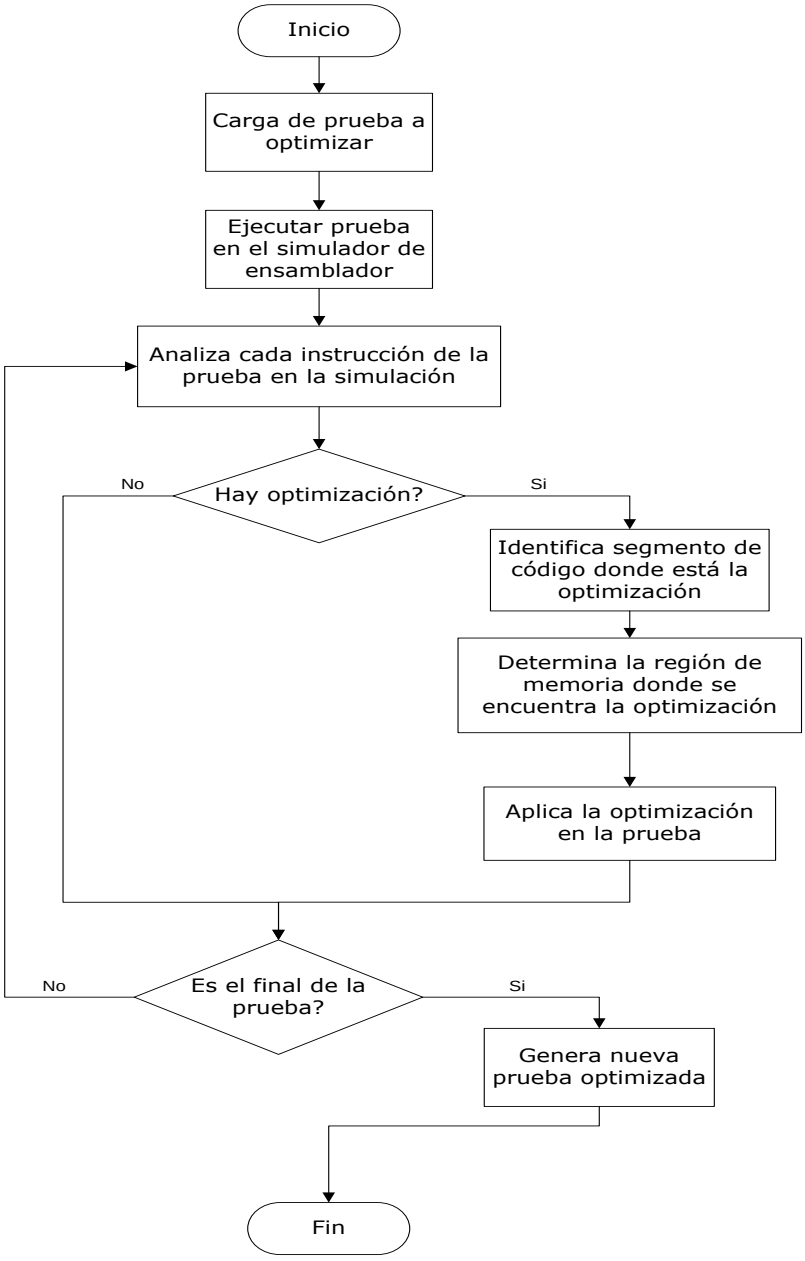

Figura 5.3. Diagrama de flujo de la tercera solución propuesta.

El diagrama es similar al del caso 5.1.2. Este modelo facilita la lectura y seguimiento de cada una de las instrucciones de la prueba, ya que lo hace con el simulador de ensamblador. De esta manera de la simulación se extrae fácilmente la dirección física de memoria del código donde inicia o termina una optimización y con los archivos de compilación se identifica qué parte específica de la prueba debe modificarse.

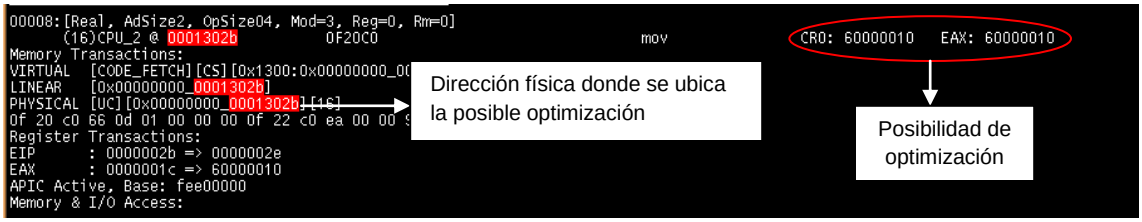

Figura 5.4. Identificación de una posible optimización en la prueba obtenida de su simulación.

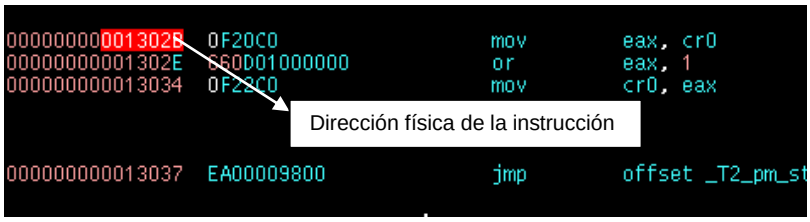

Figura 5.5. Ubicación de la optimización mostrada en la figura 5.4 en el código de la prueba.

La figura 5.4 ilustra la identificación de una posible optimización, al correr una simulación. Se identifica la dirección física donde se encuentra este código en la prueba.

En la figura 5.5 está el archivo de compilación del que se puede extraer el código de la prueba e identificar la dirección física de la instrucción que corresponde al inicio de la optimización identificada en la figura 5.4, la cual corresponde a registros de configuración (en la sección 5.2.3.2 se explica la misma). Así se identifica fácilmente la optimización con la simulación de la prueba y se identifica dónde específicamente debe modificarse ésta.

Las principales ventajas que presenta este modelo son las siguientes:

- La identificación de las optimizaciones en la prueba original se facilita ya que se sabe con certeza la dirección de memoria física y el segmento de código de cada una de las instrucciones.
- Los algoritmos para la ejecución y optimización de la prueba son más sencillos y por ende hacen la ejecución de las pruebas sobre la herramienta más rápida.
- Al igual que en la propuesta anterior, del simulador se puede extraer bastante información adicional de la prueba que se está ejecutando.

Entre las desventajas que presenta este modelo están:

- Al igual que la propuesta número 2 depende de un simulador para ejecutar las pruebas y determinar las optimizaciones.

- Depende de los archivos de compilación de la prueba, de manera que si esta no se puede compilar, no es posible realizar la optimización de la misma.

5.1.4. Elección de propuesta a implementar

Para elegir la propuesta a implementar se analizaron varios puntos, tales como complejidad de los algoritmos, duración del proyecto, recursos disponibles, viabilidad para la empresa, entre otros. Analizando cada una de las propuestas se concluye que la más viable es la propuesta número tres, ya que ofrece algoritmos adecuados para el tiempo dispuesto para el proyecto y se posee de las herramientas necesarias para poder realizar esta solución.

La propuesta número uno es una buena opción ya que no necesita de ningún simulador para determinar y realizar cada una de las optimizaciones; sin embargo la complejidad de los algoritmos hace que el tiempo que se necesita para realizar el proyecto exceda el disponible.

La propuesta dos tiene la ventaja de que no necesita de los archivos de compilación de la prueba; sin embargo si en el código no se puede determinar la dirección de memoria física donde se encuentran las instrucciones, el algoritmo para modificar la prueba original debe ser muy complejo.

A pesar de que la propuesta 3 depende de un simulador y de un compilador de ensamblador tal y como se mencionó anteriormente, fue la elegida por la empresa ya que requiere algoritmos menos complejos, es más factible tomando en cuenta el tiempo disponible y se cuenta con los programas y variables de ambiente necesarias para realizarla.

5.2 Descripción del software

La herramienta de software desarrollada recibe, por medio de línea de comando, las pruebas a realizar. A partir de la solicitud, evalúa para determinar si existen o no optimizaciones conocidas dentro de cada prueba. En la figura 5.6 se muestra un diagrama de bloques explicativo del programa en general.

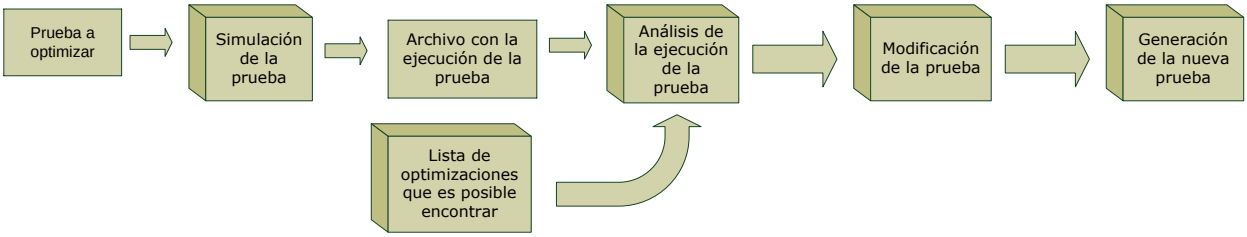

Figura 5.6 Diagrama de bloques de la herramienta para optimizar pruebas.

El flujo que sigue la herramienta mostrado en la figura 5.6 es el siguiente:

- La herramienta toma la prueba a optimizar.
- Se ejecuta la simulación de la prueba.
- Se guardan los resultados de la simulación en un archivo que utilizará para analizar todas las acciones que realiza la prueba durante su ejecución.
- En el bloque de la lista de optimizaciones están todos los casos que es posible encontrar, por lo que en el análisis de la prueba se revisa si en la misma existen patrones que se pueden eliminar o modificar.
- Si existen optimizaciones se realiza la modificación de la prueba.
- Por último se genera la nueva prueba optimizada.

De esta manera se tiene una prueba que hace las mismas funciones que la anterior pero su tiempo de ejecución es menor, ahorrando tiempo de prueba para los microprocesadores.

A continuación se explicarán cada uno de los bloques que conforman la herramienta de optimización de pruebas detalladamente.

5.2.1. Pruebas a optimizar

El primer bloque consiste en indicar las pruebas que se desean optimizar y el lugar donde se quiere que la herramienta escriba las nuevas pruebas optimizadas, esto se realiza mediante la línea de comandos necesaria para ejecutar el programa:

<path donde se encuentra el programa>

-path <path donde están las pruebas que se quieren optimizar>

-regression <path donde se quieren guardar las nuevas pruebas>

-total <número de pruebas que se quieren optimizar>

Al ejecutar esta línea de comando la herramienta comenzará a buscar las pruebas y a ejecutar sobre estas los algoritmos necesarios tanto para la ejecución como para la optimización de la misma.

5.2.2. Simulación de las pruebas

Para facilitar el seguimiento de las instrucciones de cada prueba se utiliza un simulador de ensamblador que no solo nos da información de la ejecución de la prueba sino que también entrega las direcciones de memoria física y virtual de cada una de las instrucciones ejecutadas. También se puede ver las modificaciones aplicadas a los registros de propósito general o los cambios en el registro de banderas.

Esto facilita en gran medida determinar cada uno de los cambios que hace la prueba sobre el microprocesador y así comprobar si existen o no patrones innecesarios que pueden optimizarse.

La simulación de la prueba se guarda en un archivo que posteriormente se usará para analizar en detalle cada uno de los cambios que realiza la prueba sobre el microprocesador.

Algoritmo para la simulación de la prueba

```
Simula_prueba () {
    Mientras (Pruebas a optimizar){
        Carga prueba a optimizar en la herramienta.
        Ejecuta prueba en el simulador de ensamblador.
        Mientras (Simulación de la prueba){
                Si (Tiempo de simulación == 30s){
                        Detiene la simulación.
                }
         }

        Si (Simulación es correcta){
                Genera archivo con los resultados de la  simulación
        }sino{
                Declara fallo en la simulación de esta prueba
       }
    } #fin pruebas
}# fin método
```

Figura 5.7. Pseudo-código para realizar la simulación de cada una de las pruebas.

Como se observa en la figura 5.7, si una prueba tarda mucho tiempo en la simulación significa que la misma nunca llega a su fin. Esto sucede por algún problema propio de la misma. La herramienta lo que hace en casos como este es detener la simulación y declarar un fallo en la ejecución de esta prueba, pero continua con la ejecución de las otras.

Esto solucionó un problema que se tenía inicialmente cuando una prueba no llegaba a su fin, ya que las otras no podían ser simuladas porque nunca se pasaba hacia la siguiente prueba y había que detener el simulador manualmente.

Cuando se termina la simulación de la prueba, se verifica que la ejecución de la misma esté correcta, revisando si todos los procesadores utilizados llegaron a la instrucción final de *hlt*. En este caso se genera el archivo con los resultados de la simulación que se usarán para la optimización de la prueba y, de no ser así, se declara un fallo en la simulación.

5.2.2.1 Archivo generado con la simulación de la prueba

El archivo que se genera posee información como la que se muestra en la figura 5.8.

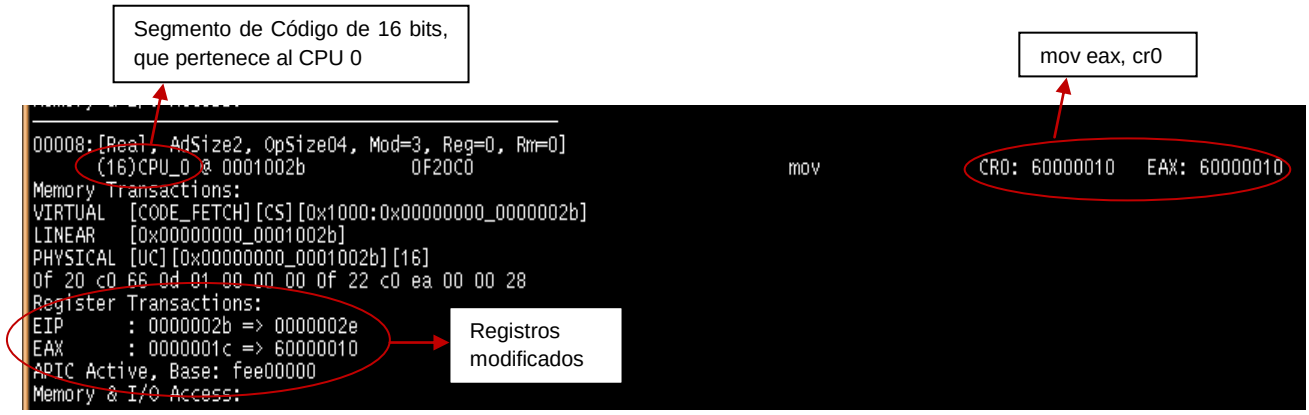

Figura 5.8. Ejemplo de información brindada en el archivo generado con la simulación de la prueba.

Como se aprecia en la imagen, para cada instrucción se tiene información de los registros que se están modificando, la memoria física, lineal y virtual donde se encuentra la instrucción, el CPU al que pertenece la instrucción, y si el segmento de código es de 16 bits, 32 bits ó 64 bits.

Estos datos permiten a la herramienta saber exactamente cuál sección del código se está estudiando y qué cambios hubo tanto en la memoria como en los

registros del microprocesador, lo que facilita determinar si existe o no una optimización en esta región específica.

5.2.3. Lista de optimizaciones que es posible encontrar y análisis de la ejecución de la prueba

Este es un punto importante, ya que el análisis que se haga sobre la prueba determina si es posible o no identificar las optimizaciones que se podrían encontrar en las pruebas.

En esta etapa se toma el archivo generado con la simulación de la prueba y se analiza cada una de las instrucciones y los cambios que estas hacen sobre los registros o las direcciones de memoria del microprocesador. A partir de ahí se pueden determinar las optimizaciones que hay en la prueba.

El algoritmo correspondiente a este bloque se muestra en la figura 5.9.

Algoritmo para el análisis de la prueba

```
Analiza_pruebas () {

    Mientras (Pruebas a optimizar){

        Carga prueba a optimizar en la herramienta.
        Carga simulación de la prueba.

        Si(La simulación es correcta){
            Si(Prueba es de tipo1){

                Modificación_de_jumps();
                Modificación_de_etiquetas();
            }

            Ejecuta subprogramas()
        }

    } # fin pruebas
} #fin método
```

Figura 5.9. Algoritmo para analizar la ejecución de la prueba.

La figura 5.9 se muestra que el algoritmo para el análisis pregunta si la prueba que se está analizando es de tipo 1, esto porque las pruebas son programadas usando dos tipos de programas. Las generadas con el tipo 0 no tienen problema cuando son modificadas por la herramienta de optimización. En las de tipo 1 el código de ensamblador no utiliza etiquetas para indicar el lugar de memoria al que deben de caer

los saltos, sino que se hace con el número en hexadecimal, tal como se muestra en la siguiente figura:

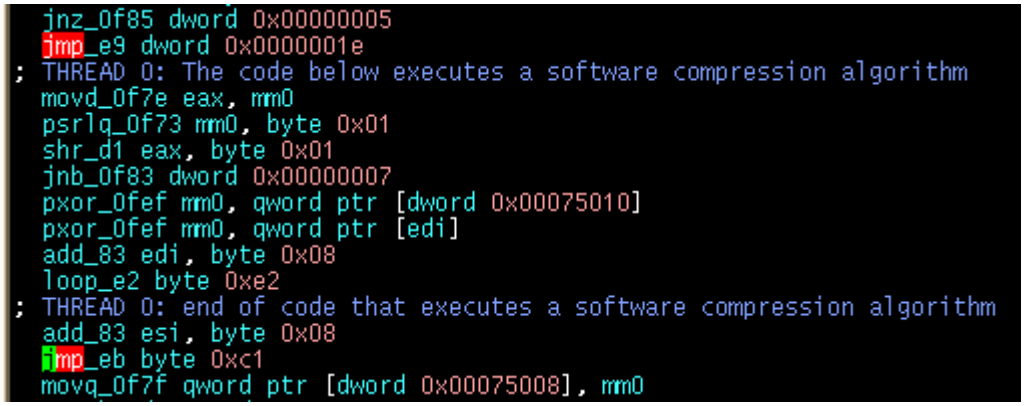

Figura 5.10. Ejemplo de saltos en pruebas de tipo 1.

Por ello si la simulación de la prueba en análisis terminó correctamente, y esta es de tipo 1, se ejecutan las funciones para modificar los saltos y las etiquetas en la prueba para que no se presente el problema mencionado en la sección 5.2.3.1

Después se cargan los subprogramas para identificar las optimizaciones y modificar la prueba original en caso necesario.

Cada subprograma corresponde a un grupo de algoritmos que tienen como fin identificar un tipo de optimización, por lo que cada uno es instanciado para que determine si en la prueba actual está o no su respectiva optimización, de ser así un algoritmo hace la modificación necesaria para que esta reducción quede aplicada. En esta etapa se pueden identificar alguna de las siguientes optimizaciones:

- Registros Generales
- CR.
- Interrupciones.
- Rdmsr.
- APIC
- APIC_4T.

Estas son las que permiten reducir las pruebas actuales, y se explicará en detalle cada una.

5.2.3.1. Identificación y modificación de saltos en pruebas de tipo 1

Este tipo de saltos generan un problema cuando se realiza la optimización, ya que al eliminar código de la prueba original cambian las posiciones de memoria del nuevo código. Por ello al realizarse el salto, se cae en una posición de memoria inválida o incorrecta, haciendo que la prueba falle o siga un camino equivocado al que debería realizar.

Para solucionar este problema se hizo un algoritmo que identifica cada uno de los saltos que hace la prueba. Según las posiciones de memoria del salto, se identifica si es un *short jmp* o un *long jmp*. Se cambian acordemente las posiciones a las que se debe saltar por una etiqueta y se busca el lugar donde cae el *jmp* para colocar el nombre de la etiqueta correspondiente.

Esto se realiza en toda la prueba y se genera una nueva prueba a la cual se le pueden eliminar instrucciones sin que existan problemas por caer en regiones de memoria inválidas o incorrectas.

El diagrama de bloques que sigue este algoritmo se ilustra en la figura 5.11:

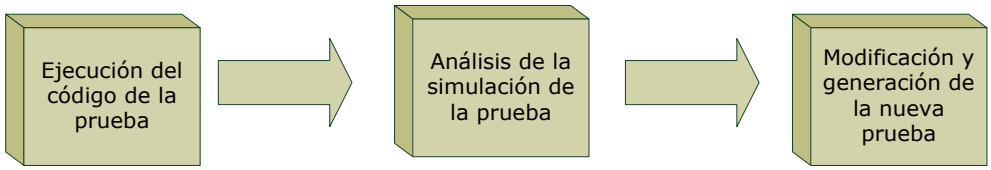

Figura 5.11. Diagrama de bloques del algoritmo para modificar los saltos en pruebas de tipo 1.

El pseudo código que ejecuta el primer bloque es el siguiente:

Algoritmo para estudiar el código de la prueba

```
determina_salto (){

    Carga el ensamblador de la prueba.
    Recorre las instrucciones de la prueba.

    Mientras (Ejecución del código de la prueba){

        Si(Hay jump){
            memory_jump[i] = posición de memoria del salto
            i++
        }
    }
    retorna memory_jmp
}
```

Figura 5.12. Algoritmo para determinar la posición en memoria de cada salto.

Este bloque carga el ensamblador de la prueba y va recorriendo cada una de las instrucciones para verificar si existen o no saltos en la misma. En caso de que se encuentre una instrucción de *jmp* se guarda la posición de memoria en la que esta se halla. Una vez que finaliza la ejecución del código de la prueba se retorna las posiciones de memoria donde se encontraron saltos en la prueba. Esta información será utilizada por el siguiente bloque.

El segundo bloque consiste en el análisis de la simulación de la prueba, tal como se muestra en la figura 5.13.

Algoritmo para determinar los destinos de cada salto

```
determina_destino(memory_jmp){

    Carga el ensamblador de la prueba.
    int jmp=0

    Mientras (Ejecución del código de la prueba){

        Si(jmp==1){
           destino[n] = posición de memoria del destino
            jmp=0
        }

        Si (Hay salto){
             jmp=1
             n = posición del jmp en memory_jump
        }sino{
             jmp=0
        }
    }
    retorna destino
}
```

Figura 5.13. Algoritmo para determinar el destino de cada salto de la prueba.

Este bloque utiliza la simulación para determinar el destino de cada uno de los saltos de la prueba. Para ello se recibe la pila con todos los saltos que se encontraron en el código de la prueba. Cuando la simulación se ejecuta, se pregunta si la variable *jmp* es igual a uno, porque cuando se encuentra una instrucción de salto la siguiente instrucción es el destino de ese salto. Así, si hay un *jmp* se hacen dos cosas:

- Se guarda en la variable n la posición que tiene ese salto en el arreglo.
- Se escribe un uno en la variable jmp.

De esta manera si jmp=1 estamos en el destino de uno de los saltos; este se guarda en el arreglo en la misma posición en la que está el salto en su respectiva pila para que sea fácil identificar el destino respectivo. Cuando finaliza la simulación se retorna la pila con los destinos, estos serán utilizados por el siguiente bloque para modificar la prueba. Además se elimina de la pila todos los saltos que no fueron ejecutados en la simulación.

Por último el tercer bloque es el encargado de modificar y generar la prueba con las etiquetas y saltos modificados, tal como ilustra la figura 5.14:

Algoritmo para la modificación y generación de la prueba

```
modifica_prueba_tipo1(memory_jmp, destino){
    Mientras (Ejecución del código de la prueba){
        Si(memory_jmp[0…n]==posición de memoria actual){
            verifica_tipo_de_salto()
            modifica_salto()
        }

        Si (destino[0…n]==posición de memoria actual){
            agrega_etiqueta_en_destino()
        }
    }
    genera_nueva_prueba()
}
```

Figura 5.14. Algoritmo del bloque de modificación y generación de la prueba.

Este bloque recibe la pila con las posiciones de memoria donde están los saltos y destinos que deben ser modificados. Primero se revisa si la posición actual pertenece a un salto. En caso de ser así se verifica el tipo de salto, si es corto o largo, y después se coloca la respectiva etiqueta para este salto.

Si la posición actual pertenece a un destino de un salto se coloca la etiqueta donde debe caer el mismo, para que no haya problema si se eliminan instrucciones producto de alguna optimización.

Finalmente se revisa si se ha llegado al fin de la prueba. En caso negativo se continúa con la búsqueda de saltos y destinos en la prueba y en caso afirmativo se genera la nueva prueba para que sea analizada por la herramienta.

Las pruebas también tienen otros tipos de saltos como los siguientes:

- Jnz, Jz, Je, Jne, Jc, Jnc.

5.2.3.2 Optimización de registros de control

Los registros de control CR0, CR3 y CR4 se utilizan para configurar distintos modos de operación del microprocesador. Por ejemplo, cuando se pasa de modo real a modo protegido se debe de configurar un bit del CR4. Para ello se carga el valor de este registro en un registro de propósito general, se hacen instrucciones lógicas como *and*, *or* o *xor* para escribir el registro y finalmente se mueve el valor a CR4.

Un ejemplo de esta configuración se ve en la figura 5.15:

```
;-------------------------------------------------------------
; Switch to LONG (64BIT) MODE
;-------------------------------------------------------------

; establish long mode page tables
;-------------------------------------------------------------

; enable paging
;-------------------------------------------------------------

; Set EFER NXE=>0, SCE=>1, LME=>1,
mov    ecx,  0xc0000080  ; Read EFER
rdmsr
or     eax, 0101h
and    eax, 0ffffffffffffff7ffh
wrmsr
; Set CR4 PSE=>0, PAE=>1,
mov    eax, cr4
or     eax, 020h
and    eax, 0fffffffffffffefh
mov    cr4, eax
 mov     eax, ((_linaddr _pagedir_SEGBEGIN) _AND 0FFFFF000h) _OR 0x0  ; PCD=0, PWT=0
 mov     cr3, eax
; Set CR0 PG=>1,
mov    eax, cr0
or     eax, 080000000h
and    eax, 0ffffffffffffffffh
mov    cr0, eax
```

Figura 5.15. Configuración de paginación y long mode por medio de los registros CR0 y CR4.

En la figura 5.15 se observa donde se mueve el valor de CR4 a EAX, seguidamente se hacen varias instrucciones lógicas y finalmente se escribe el valor de EAX a CR4. Por lo tanto esta optimización busca eliminar las instrucciones lógicas y escribir directamente el registro de configuración con el valor final del mismo, el cual se puede obtener del archivo de simulación de la prueba. En la imagen también se tiene un caso semejante con el registro CR0, el cual también puede ser optimizado.

El diagrama de bloques para este algoritmo está en la figura 5.16:

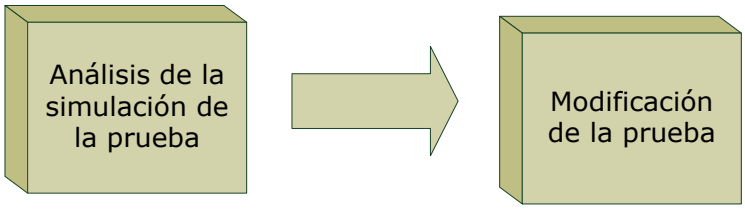

Figura 5.16. Diagrama de bloques del algoritmo para realizar la optimización de los registros de control.

Primero se hace un análisis de la simulación donde se busca si hay escrituras a los registros de configuración. En caso de ser afirmativo se revisa si hay instrucciones lógicas que pueden ser eliminadas de la misma. Si existen casos de optimización, se envían al bloque encargado de modificar la prueba para que haga los cambios respectivos.

El análisis de la simulación que se hace para el primer bloque es el que muestra en la figura 5.17.

Algoritmo buscar la optimización CR en la prueba

```
Identifica_optimización_CR(){
    read=0
    logic=0
    n=0
    Carga prueba a optimizar en la herramienta.
   Mientras (recorre simulación){
        Si (Lectura de un CR) {
           identifica_cr();
           guarda_PM_de_lectura()
           read=1
           logic=0
        }

        Si (Instrucción Lógica & read==1){
            logic=1
        }

        Si (Escritura de un CR & read==1 & logic==1){
            identifica_cr();
            guarda_PM_de_escritura()
            read=0
            logic=0
            Cr_opt[n]={read=>PM_lectura}
            Cr_opt[n]={write=>PM_escritura}
            Cr_opt[n]={cr=>número de registro}
            Cr_opt[n]={valor del registro}
            n++
        }
     }# fin simulación
     Retorna Cr_opt
  } #fin método
```

Nota PM => Posición de Memoria

Figura 5.17. Algoritmo para determinar las optimizaciones de CR.

El algoritmo carga la simulación de la prueba a analizar y se declaran tres variables necesarias para determinar el patrón que se busca. Cuando se está analizando la simulación, si el algoritmo encuentra que se dio una lectura de un registro de control, la variable read toma el valor de uno, esto para indicar que existe la posibilidad de que sea un caso de optimización. Además se identifica cuál es el registro que se está leyendo y se guarda la posición de memoria donde se ubica la lectura, porque en caso de ser una optimización esta es la referencia que nos permite encontrar el patrón dentro de la prueba.

Si ocurre una instrucción lógica y además la variable de read está activada y la instrucción pertenece al registro de propósito general que tiene el dato del registro CR, entonces se activa la variable logic para indicar que ya se hizo una lectura sobre ese registro y al menos una instrucción lógica.

Finalmente, cuando ocurre una escritura sobre un registro de configuración y ambas variables logic y read están activadas, entonces se trata de un caso de optimización, por lo que se guarda la posición de memoria donde está la escritura y se hacen cero las variables de read y logic.

Cr_opt es un arreglo de estructuras en el que se va guardando la información necesaria para hacer la optimización del registro de control:

- La dirección de memoria de la instrucción donde se hizo la lectura.
- La dirección de memoria de la instrucción donde se hizo la escritura.
- El nombre del registro de configuración y el de propósito general.
- El valor final del registro.

Por lo tanto, cada vez que se encuentra un caso de optimización se guardan los datos en esta estructura y además se suma la variable n, que es la encargada de ir guardando en una estructura diferente los nuevos datos de cada optimización de este tipo.

En la figura 5.18 se muestra cómo del archivo de simulación se toman los datos que se necesitan para realizar la optimización.

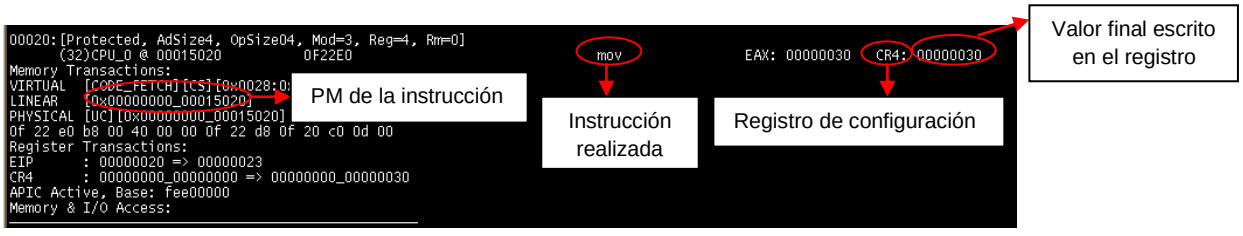

Figura 5.18. Escritura sobre el registro de configuración CR4 mostrada en el archivo de simulación.

El ejemplo de la figura 5.18 pertenece a la escritura del registro CR4. Como se observa, del archivo de simulación es posible extraer la información que se guarda en la variable Cr_opt cada vez que se encuentra un caso de optimización.

Una vez analizada la simulación, se retorna el arreglo de estructuras, en el que se guarda toda la información referente a las optimizaciones de este tipo encontradas en la prueba actual. El siguiente bloque toma de este arreglo los datos necesarios para realizar las modificaciones necesarias a la prueba. (ver figura 5.19).

Algoritmo para la modificación de la prueba

```
Modifica_prueba_por_CR(Cr_opt){

    read=0
     Carga prueba a optimizar en la herramienta.

    Mientras (recorre simulación){

        Si (CL == Cr_opt[0..n]{read}) {
            identifica_inicio_código_a_modificar
            read=1
        }sino{

              Si(read==1){

                  Si(CL == Cr_opt[0…n]{write}){
                      identifica_fin_codigo_a_modificar
                      elimina_instrucciones_innecesarias()
                      agrega_nuevo_código_optimizado()
                      read=0
                   }
              }
        }
    } # fin simulación

    retorna prueba_modificada

} #fin método
```

Nota PM => Posición de Memoria
CL => Posición de Memoria de la instrucción actual

Figura 5.19. Algoritmo aplicado para modificar la prueba a optimizar.

El método recibe el arreglo de estructuras enviado por el bloque anterior que se utiliza para determinar si la línea de código que se está analizando es parte o no de una optimización. En la figura 5.20 se muestran los elementos que contiene la estructura en la prueba.

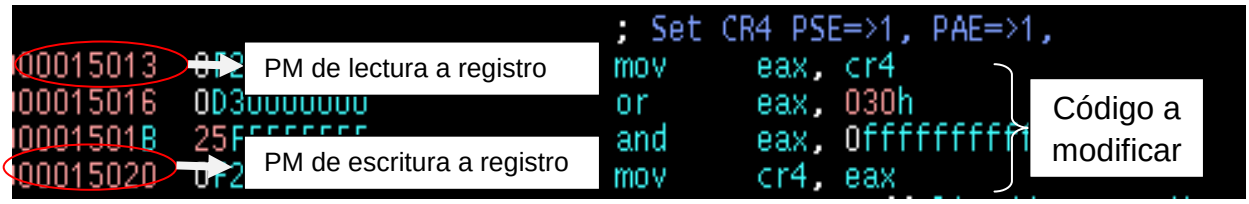

Figura 5.20. Datos obtenidos de la prueba que permiten hacer su optimización.

De esta manera cuando se recorre el código de la prueba, se verifica si la posición de memoria del código actual está guardada en alguna variable del arreglo de estructuras, ya que si se da la lectura de un registro de configuración indica el inicio del conjunto de instrucciones que pueden ser sustituidas por menos instrucciones.

Si la variable de read está activada, se analiza si esta instrucción pertenece a la escritura del registro de configuración correspondiente a la lectura que se había hecho previamente; de no ser así significa que todavía no se ha llegado a la instrucción donde se realizó la escritura del registro. Por el contrario si se cumple esta condición se procede a eliminar las instrucciones actuales para ser sustituidas por el nuevo patrón.

En la figura 5.21 se muestra el patrón que tiene la prueba original y el patrón por el que será sustituido.

Original:

```
15013: mov   EAX, CR4
15016: or    EAX, 030h
1501B: and   EAX, 0fffffffffh
15020: mov   CR4, EAX
```

Optimizado:

```
15013: mov   EAX, 0x0030h
15016: mov CR4, EAX
```

Figura 5.21. Patrón original y patrón optimizado correspondiente a los registros de configuración.

Como se observa para este caso se redujeron dos instrucciones, lo que puede parecer poco significativo, pero tomando en cuenta que en cada prueba hay al menos tres configuraciones y para las pruebas que utilizan varios microprocesadores se hacen todas estas configuraciones para cada microprocesador, la optimización toma importancia.

5.2.3.3. Optimización de registros MSR

Los registros MSR (Model Specific Register) se encuentran en microprocesadores Intel con capacidad para 64 bits, y se usan para controlar funciones de prueba, rastreo de ejecución, monitoreo de rendimiento y chequeo de errores en la máquina.

Durante la ejecución de la prueba es importante hacer configuraciones de control o monitoreo del estado del microprocesador que se hacen con lecturas y escrituras a estos registros.

Las instrucciones que se utilizan para realizar estas lecturas y escrituras son *rdmsr* y *wrmsr* respectivamente. Al igual que con los registros de configuración CR0, CR3 y CR4, cuando se necesita hacer una escritura, generalmente primero se hace una lectura del registro y seguidamente se hacen operaciones lógicas para escribir sólo los bits deseados.

```
; Set EFER NXE=>0, SCE=>1, LME=>1,
mov    ecx,  0xc0000080  ; Read EFER
rdmsr
or      eax, 0101h
and     eax, 0fffffffffffff7ffh
wrmsr
```

Figura 5.22. Ejemplo de una escritura a un registro MSR.

Como se aprecia en la figura 5.22, se hace una lectura al registro MSR 0xc0000080, se hacen dos operaciones lógicas y finalmente se hace la escritura del mismo.

El algoritmo utilizado para reducir este patrón de instrucciones es el mismo del tipo de optimización anterior porque la lógica es la misma; la única diferencia es la forma de escritura y de lectura para los registros MSR debido a que son de 64 bits.

La lectura se hace escribiendo en ECX la dirección del registro a leer, y se retorna el valor del mismo en los registros EDX y EAX, es decir $MSR[ECX] \rightarrow EDX{:}EAX$.

La escritura por lo tanto se hace escribiendo en ECX la dirección del registro a escribir, en EDX la parte alta del valor y en EAX la parte baja, o sea $EDX{:}EAX \rightarrow MSR[ECX]$.

Del archivo de simulación se toma la información necesaria para realizar la optimización, tal como se ve en la figura 5.23.

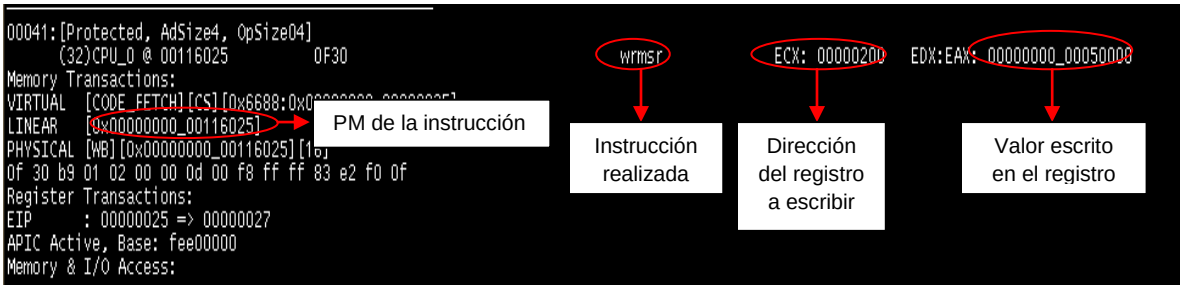

Figura 5.23. Escritura de un registro MSR mostrado en el archivo de simulación.

Al igual que en el tipo de optimización 5.2.3.2, se utiliza la información mostrada para determinar la posición exacta donde se ubica el patrón a optimizar dentro de la prueba original, en este caso se utiliza lo siguiente:

- Posición de memoria de la lectura del MSR.
- Posición de memoria de la escritura del MSR.
- Dirección del registro a escribir. (Valor de ECX).
- Valor escrito en el registro MSR. (Valor final de EDX y EAX).

Estos datos son suficientes para realizar la modificación de la prueba, utilizando los mismos algoritmos que se realizaron en la optimización para los registros de configuración.

La figura muestra el patrón de escritura en la prueba original y en la prueba optimizada.

Original:

```
00012: mov   ECX: c0000080
00013: rdmsr EDX: 00000000
             EAX: 00000000
00014: and eax, 0x000007ff
00015: or  eax, 0x000508
00016: xor_31 edx, edx
00017: wrmsr ECX: c0000080
```

Optimizado:

```
00008: mov   ECX: c0000080


00009: mov   EAX: 00000101
00010: mov   EDX: 00000000
00011: wrmsr ECX: c0000080
```

Figura 5.24. Patrón original y patrón optimizado correspondiente a los registros MSR.

Esta optimización redujo en este ejemplo dos instrucciones. Sin embargo la prueba original tiene una instrucción *rdmsr* y una instrucción *wrmsr* y cada una de estas instrucciones toma mucho más ciclos de CPU que la instrución *mov*. En la prueba optimizada hay solamente instrucciones *mov* y un *wrmsr*, por lo que el tiempo de ejecución reducido es significativo en una prueba que presente varios de estos casos.

Las pruebas que utilizan varios procesadores realizan la configuración para cada uno de los CPU. Esta optimización al encontrarse en la mayoría de las pruebas estudiadas, reduce una parte del tiempo de ejecución de cada una, por lo que el aporte de reducción de tiempo en el total de pruebas es importante.

5.2.3.4. Optimización por configuración de interrupciones

La mayoría de pruebas que se utilizan para los microprocesadores Intel se corren en modo protegido o modo de 64 bits. Esto hace necesario la configuración de las tablas de descriptores (globales, locales y de interrupción). Sin embargo la tabla de descriptores de interrupción es necesaria únicamente si la prueba realiza una o varias interrupciones durante su ejecución.

Existen una gran cantidad de pruebas que no ejecutan interrupciones durante su ejecución, por lo que esta configuración y la de su tabla de descriptores se pueden omitir; otras pruebas que utilizan multiprocesadores realizan interrupciones en solo un CPU por ejemplo, por lo que se puede eliminar la configuración de los demás.

Esta optimización ahorra el tiempo que toma el microprocesador haciendo la configuración de interrupciones.

```
      ; Load the new idt

      mov         eax, offset idtr_ptr0
      jmp SHORT idtr_load0
idtr_ptr0:
      dw          (_linaddr _idt_SEGEND - _linaddr _idt_SEGBEGIN) _AND 0FFFFh
      dq          _linaddr _idt_SEGBEGIN
idtr_load0:
      lidt  CS:[eax]
```

Figura 5.25. Configuración de la tabla de descriptores IDT.

En la figura 5.25 se tienen las instrucciones necesarias para cargar la tabla de descriptores IDT, para ello se guarda en el registro EAX la posición de memoria donde está definido la dirección de la tabla, y con la instrucción *lidt* se hace la configuración de la misma.

Con este patrón el microprocesador está listo para recibir interrupciones y haciendo uso de la tabla de descriptores ejecuta la acción necesaria. Además si se ejecutan interrupciones en modo protegido y en modo de 64 bits, se debe configurar el registro IDTR para cada uno de estos casos, ya que se debe de ajustar el direccionamiento del IDT a 32 bits o a 64 bits según corresponda.

Por esta razón hay pruebas que presentan configuración del IDT en el modo de 32 y de 64 bits, pero no ejecutan ninguna interrupción durante toda su ejecución, lo que hace más importante la optimización porque se eliminan varias configuraciones que el microprocesador toma tiempo realizándolas.

El diagrama de bloques que utiliza la herramienta para determinar si se puede eliminar la configuración de interrupciones y hacer la modificación de la prueba se ilustra en la figura 5.26.

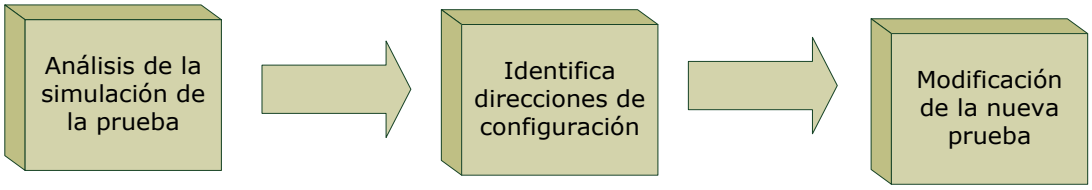

Figura 5.26. Diagrama de bloques del algoritmo para la optimización de configuración de interrupciones.

Este algoritmo hace un análisis de la simulación de la prueba, en la cual se busca si se hacen configuraciones del IDT o interrupciones durante su ejecución. De encontrarse instrucciones *lidt* en la simulación (configuración del IDT) sin que se den interrupciones, el siguiente bloque identifica la dirección de memoria donde está cada una de las configuraciones para que el último bloque haga las modificaciones necesarios en la prueba y quede así optimizada.

El algoritmo encargado de identificar la configuración del IDT y de las interrupciones se presenta en la figura 5.27.

Algoritmo para identificar la configuración de IDT e interrupciones

```
Identifica_interrupciones_y_IDT(){
        Carga prueba a optimizar en la herramienta.
        IDT=0
        Int=0
       Mientras (recorre simulación){
             Si (configuración de IDT) {
                  IDT=1
             }

            Si (hay interrupción?){
               Int=1
            }
       } # fin simulación
      Retorna IDT, Int
 } #fin método
```

Nota IDT => Tabla de descriptores de interrupción.

Figura 5.27. Algoritmo empleado para identificar la configuración del IDT e interrupciones en la ejecución de la prueba.

En este método se carga la simulación de la prueba y se recorre la misma, si se identifican configuraciones de IDT se pone en uno la variable IDT, y se si identifica alguna interrupción la variable Int cambia su valor a uno también. El método retorna la variable IDT e Int para que el siguiente bloque determine si es necesario o no hacer esta clase de optimización.

El algoritmo para el bloque encargado de identificar las direcciones de configuración es el siguiente:

Algoritmo para identificar las direcciones de configuración del IDT

```
Identifica_dirección_IDT(Int, IDT){
    Si(IDT==1 & Int==0){
        Carga prueba a optimizar en la herramienta.
        start=0
        n=0

        Mientras (recorre simulación){

            Si (configuración de IDT) {
               Identifica_PM_inicial
               lidt_opt[n]={start=PM_inicial}
               start=1
             }

            Si(start==1 & lidt){
               Identifica_PM_final
               lidt_opt[n]={end=PM_final}
               start=0
               n++
             }

         } # fin simulación

       retorna lidt_opt

   }#fin comparación
} #fin método
```

Nota PM => Posición de memoria

Figura 5.28. Algoritmo que identifica las direcciones de configuración del IDT.

En este bloque se reciben las variables IDT e Int. Si IDT es uno e Int es cero quiere decir que hubo configuraciones y no se dieron interrupciones, por lo que este caso de optimización aplica para la prueba.

Si se encuentra una configuración de IDT, se guarda en un arreglo de estructuras la posición en memoria de la instrucción inicial y se escribe un uno en la variable start. Este tipo de configuración termina con la instrucción de *lidt*, por eso cuando se encuentra la misma se hace cero la variable start, se guarda la posición de memoria final y se suma la variable n.

Como puede haber más de una configuración de interrupciones en una prueba, en caso de que estas puedan ser optimizadas, se utiliza la variable n para controlar la cantidad encontradas y para guardar estas en posiciones diferentes del arreglo.

Si se llega al fin de la simulación se retorna el arreglo de estructuras lidt_opt que será utilizado por el siguiente bloque para hacer las modificaciones respectivas en la prueba.

El algoritmo para el último bloque se presenta en la figura 5.29.

Algoritmo para eliminar las configuraciones innecesarias del IDT

```
Remueve_configuraciones_IDT(lidt_opt){
    Carga prueba a optimizar en la herramienta.
    start=0

    Mientras (recorre simulación){

        Si (CL == lidt_opt[0…n]{start}) {
            Identifica_inicio_configuracion_a_eliminar
            start=1
        }

        Si (start==1 & CL == lidt_opt[0…n]{end}) {
            Identifica_fin_configuracion_a_eliminar
            Elimina_instrucciones_innecesarias()
            start=0
        }

    } # fin simulación

    retorna Prueba_modificada

} #fin método
```

Nota PM => Posición de memoria
CL => Posición de memoria de la instrucción actual

Figura 5.29 Algoritmo que elimina de la prueba las configuraciones de IDT innecesarias.

La modificación de la prueba es bastante similar a la de las optimizaciones anteriores, ya que se recorre el ensamblador y cuando se encuentra que el código actual pertenece al inicio de una optimización, se hace start igual a uno para identificar el inicio de una configuración a eliminar. Cuando se encuentra el final del patrón a optimizar se elimina las instrucciones innecesarias para este caso y se continúa con la búsqueda de otros patrones.

A continuación se presenta un ejemplo de esta optimización en una prueba para microprocesadores intel:

Original:

```
11000  mov eax, offset gdtr_ptr0
11006  jmp SHORT gdtr_load0
gdtr_ptr0:
11008  dw _gdt_SEGEND _AND 0FFFFh
1100A  dd _linaddr _gdt_SEGBEGIN
gdtr_load0:
1100E  db 066h
1100F  lgdt PWORD PTR CS:[eax]
11014  mov   eax, offset idtr_ptr0
1101A  jmp SHORT idtr_load0
idtr_ptr0:
1101C  dw  (_linaddr _idt_SEGEND …
1101E  dq _linaddr _idt_SEGBEGIN
idtr_load0:
11026  lidt  CS:[eax]
```

Optimizado:

```
11000  mov eax, offset gdtr_ptr0
11006  jmp SHORT gdtr_load0
gdtr_ptr0:
11008  dw _gdt_SEGEND _AND 0FFFFh
1100A  dd _linaddr _gdt_SEGBEGIN
gdtr_load0:
1100E  db 066h
1100F  lgdt PWORD PTR CS:[eax]
```

Figura 5.30 Patrón original y optimizado correspondiente a la configuración de IDT.

La figura 5.29 muestra el patrón original donde se tiene la configuración de GDT (Global Descriptor Table) y IDT (Interruptor Descriptor Table). El patrón optimizado tiene únicamente la configuración de GDT, ya que en la prueba analizada el microprocesador nunca generó una interrupción, por lo que la tabla de interrupciones no es necesaria.

5.2.3.5. Optimización de registros de propósito general

Esta es la optimización que redujo la mayor cantidad de tiempo en cada una de las pruebas. La optimización consiste en analizar los movimientos que se hagan a los registros de propósito general, de manera que si se trata de escribir uno de estos, el algoritmo verifica el valor anterior de éste. En caso de ser el mismo se elimina la instrucción de la prueba original ya que no es necesaria.

Una gran cantidad de pruebas presentan escrituras a los registros de propósito general que no son necesarias, en la figura 5.31 se muestra un ejemplo de este caso:

```
### Register: NCDECS_CR_MSAC_0_2_0_PCI

;; CF8_CFC_read_creg(BUS => 0, DEVICE => 2, FUNCTION => 0, OFFSET => 98, SIZE => 8)
   mov eax, 0x80001060
   mov dx, 0x0CF8
   out dx, eax
   mov dx, 0x0CFE
   in al, dx
   ;; CF8_CFC_read_creg() done


   ;;  CF8_CFC_creg_write(BUS => 0, DEVICE => 2, FUNCTION => 0, OFFSET => 98, SIZE => 8, VALUE => 0)
   mov eax, 0x80001060
   mov dx, 0x0CF8
   out dx, eax
   mov dx, 0x0CFE
   mov al, 0
   out dx, al
   ;; CF8_CFC_creg_write() done
```

Figura 5.31. Ejemplo de optimización de registro de propósito general.

En la figura se puede ver que se hace un *mov* al registro EAX, y después se vuelve a hacer otro *mov* al mismo registro y con el mismo valor anterior. Este segundo movimiento al registro EAX no es necesario debido a que no ha habido ningún cambio sobre este registro, por lo que el mismo todavía mantiene el valor que se le va a escribir. Por lo tanto la herramienta debe de eliminar esta instrucción y ahorrar así el tiempo que tarda el microprocesador ejecutándola.

El diagrama de bloques para realizar esta optimización es el de la figura 5.32.

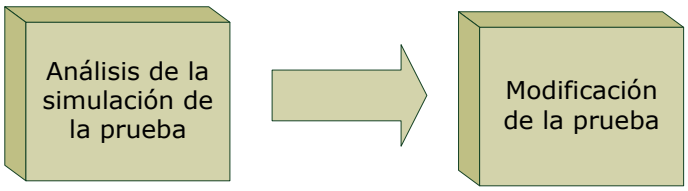

Figura 5.32. Diagrama de bloques realizado por el algoritmo para esta clase de optimización.

Esta optimización hace un análisis de la simulación de la prueba, en la que busca si existen escrituras innecesarias a registros, las que una vez identificadas son eliminadas de la prueba por el bloque modificación.

El algoritmo para el bloque encargado de hacer el análisis de la simulación es el de la figura 5.33.

Algoritmo para identificar optimizaciones por registros generales

```
Identifica_General_CR(){
    Carga prueba a optimizar en la herramienta.
    n=0

    Mientras (recorre simulación){
        Verifica_valor_de_registros

        Si (reg_value{ax} == valor_actual) {
            vax=1
        }sino{
            reg_value{ax} = valor_actual
            vax=0
        }

        Si (escritura a registro){
            Si(ax == 1){
                addr_code[n]=CL
                n++
            }
        }

    } # fin simulación

    retorna addr_code

} #fin método
```

Nota: CL => Posición de memoria de la instrucción actual

Figura 5.33. Diagrama de flujo para hacer el análisis de la simulación de la prueba.

El código de la figura 5.33 explica el análisis para el registro AX. Cada vez que se ejecuta una instrucción, se verifica el valor de los registros de propósito general. Así, se identifican los cambios en los mismos y se puede ir actualizando el valor correspondiente a estos.

La variable reg_value{ax} es una variable que contiene el valor del registro. Existe una variable para cada uno de los registros (reg_value{bx}, reg_value{cx}, reg_value{dx}). Si el valor cambió respecto a su valor anterior se modifica el mismo con el valor actual y la variable vax toma el valor de cero para indicar que hubo un cambio en el valor del registro.

Si la instrucción actual es una escritura a un registro, se revisa si la variable asociada a ese registro es uno, lo que quiere decir que el mismo tiene actualmente el valor que se desea escribirle. Por lo tanto se guarda en el arreglo addr_code la dirección de la instrucción actual, para que el bloque encargado de hacer la modificación borre esta instrucción de la prueba.

Se hace lo mismo para el resto de instrucciones y cuando se llega al final de la simulación se retorna el arreglo con todas las direcciones de memoria del código que debe removerse de la prueba.

El algoritmo para el bloque que hace la modificación de la prueba es el de la figura 5.34.

Algoritmo para eliminar la escritura innecesaria de registros generales

```
Elimina_escritura_a_CR(addr_code){

    Carga prueba a optimizar en la herramienta.

    Mientras (recorre simulación){

        Verifica_valor_de_registros

        Si (CL  == addr_code[0…n]) {
              elimina_escritura_innecesaria()
        }

    } # fin simulación

     retorna Prueba_modificada

} #fin método
```

Nota: CL => Posición de memoria de la instrucción actual

Figura 5.34. Algoritmo para hacer la modificación de la prueba.

Este bloque recibe el arreglo con todas las direcciones de memoria del código que deben removerse. Para ello se va recorriendo las instrucciones y se compara la dirección de memoria actual con las que están guardadas en el arreglo. Si alguna de estas coincide, se elimina la instrucción. Una vez que se revisa todo el código, se retorna la prueba modificada para que sea revisada por las demás optimizaciones de la herramienta.

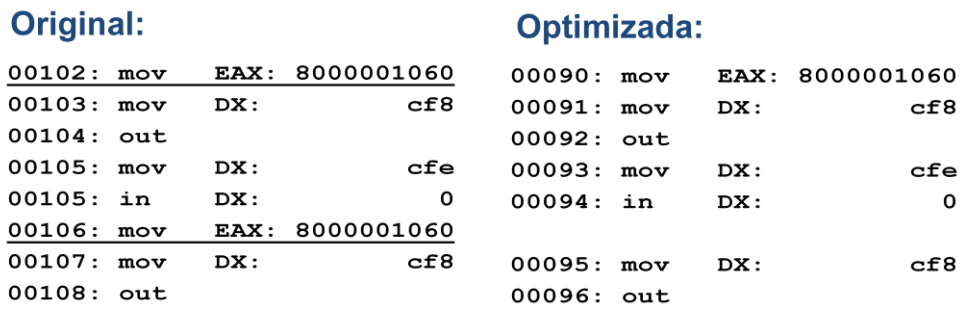

Figura 5.35. Patrón original y modificado utilizando esta optimización.

Como se ve en la figura 5.35 debido a que el registro EAX tenía el mismo valor en la línea 102 y 106 y no hubo modificaciones del mismo en las instrucciones anteriores, se puede eliminar la instrucción de la línea 106 del código original.

Casos como este se encuentran muchas veces en una misma prueba, ya que el test writer o las herramientas para escribir pruebas, no se preocupan por verificar los valores anteriores de cada registro sino que simplemente escriben en el mismo el valor que necesitan.

Por esta razón, esta optimización produjo el mayor ahorro de tiempo de ejecución por optimizaciones encontradas.

5.2.3.6. Optimización por configuración de APIC

El APIC (Advanced Programmable Interrupt Controller), es un controlador de interrupciones diseñado para el multiproceso, ya que entre sus funciones está sincronizar las instrucciones que realizan varios microprocesadores que trabajan en paralelo.

Sin embargo, existen pruebas que usan solamente un microprocesador y configuran el APIC innecesariamente.

Un ejemplo de esta configuración en una prueba es el de la figura 5.36:

Figura 5.36. Configuración del APIC en una prueba que utiliza solamente un microprocesador.

Eliminar la configuración del APIC ahorra todas las instrucciones de la prueba que se observan en la figura 5.36. Sin embargo comparado con las optimizaciones anteriores, esta se presenta en una menor cantidad de pruebas. Pero la misma no deja de ser importante ya que el tiempo de ejecución que ahorra es bastante considerable.

En la figura 5.37 se puede apreciar cómo escribir en el registro MSR 0x0000001b habilita el APIC.

Habilitación de APIC

```
75025  mov ecx, dword 0x0000001b
7502B  rdmsr
7502D  and eax, dword 0x000007ff
75033  or eax, dword 0x00050800
75039  xor edx, edx
7503C  wrmsr
```

Figura 5.37. Código necesario para habilitar el APIC en cada uno de los microprocesadores.

Por lo tanto, para determinar si la prueba está haciendo la habilitación, se debe de revisar si hay una escritura al registro anterior. Posteriormente se revisa si se está ejecutando solamente un microprocesador, porque en este caso es posible realizar la optimización.

El diagrama de bloques es el siguiente:

Figura 5.38. Diagrama de bloques utilizado para realizar la optimización del APIC.

La figura 5.38 explica esta optimización, que es igual a la optimización por configuración de interrupciones, ya que los pasos a seguir son los mismos pero ahora para la configuración del APIC. Primero se hace un análisis de la simulación de la prueba para determinar si esta optimización está presente. De ser así se busca la posición de memoria donde se encuentra dicha configuración y por último se procede a hacer la modificación necesaria.

El algoritmo para identificar si existe la configuración del APIC se muestra en la figura 5.39.

Algoritmo para identificar configuraciones de APIC

```
Identifica_configuración_APIC(){

    Carga prueba a optimizar en la herramienta.

    APIC=0
    cpu=0

    Mientras (recorre simulación){

        Si (Habilita APIC) {
            APIC=1
        }

        Si (sincronización con otro cpu){
            cpu=1
        }

    } # fin simulación

    Retorna APIC, cpu

} #fin método
```

Figura 5.39. Algoritmo para determinar si existe configuración de APIC.

Una vez cargada la simulación se busca si el APIC es configurado. Si es así se examina si este microprocesador está sincronizando o siendo sincronizado por otro microprocesador porque en este caso la configuración si es necesaria y no se puede dar la optimización.

El segundo bloque recibe las variables APIC y cpu, a partir de las que se determina si la optimización se puede o no hacer en la prueba actual, tal como se muestra en la figura 5.40.

Algoritmo para identificar la dirección de configuración del APIC

```
Identifica__dirección_config_APIC(APIC, cpu){
    Si(APIC==1 & cpu==0){
        Carga prueba a optimizar en la herramienta.
        start=0

        Mientras (recorre simulación){
            Si (configuración de APIC) {
                Identifica_PM_inicial
                APIC_opt={start=PM_inicial}
                start=1
            }

            Si(start==1 & fin de configuración){
                Identifica_PM_final
                APIC_opt={end=PM_final}
                start=0

            }

        } # fin simulación

      retorna APIC_opt

   }#fin comparación
} #fin método
```

Nota PM => Posición de memoria

Figura 5.40. Algoritmo para determinar la ubicación de la configuración del APIC.

Si la variable APIC es uno y la variable cpu es cero, quiere decir que hay configuración de APIC y no hay interacción con ningún otro CPU, por lo que la optimización es posible. El algoritmo busca la posición de memoria donde inicia la configuración, ya que a partir de aquí es donde está el código que se debe eliminar. Esta posición es guardada en la variable APIC_opt{start}, como se ve en la figura 5.40.

Cuando la herramienta determina el fin de la configuración del APIC, se guarda esta posición de memoria en la variable APIC_opt{end}, para que el bloque de modificación pueda hacer la eliminación del código.

Algoritmo para eliminar las configuraciones innecesarias del APIC

```
Remueve_configuraciones_APIC(APIC_opt){
    Carga prueba a optimizar en la herramienta.
    start=0

    Mientras (recorre simulación){

        Si (CL == APIC_opt{start}) {
            Identifica_inicio_configuracion_a_eliminar
            start=1
        }

        Si (start==1 & CL == APIC_opt{end}) {
            Identifica_fin_configuracion_a_eliminar
            Elimina_instrucciones_innecesarias()
            start=0
        }

    } # fin simulación

    retorna Prueba_modificada

} #fin método
```

Nota PM => Posición de memoria
CL => Posición de memoria de la instrucción actual

Figura 5.41. Algoritmo para eliminar la configuración del APIC de la prueba.

El algoritmo de la figura 5.41 elimina la configuración de la APIC, y es el mismo usado para la optimización por configuración de interrupciones. El algoritmo identifica la posición inicial y la posición final del código que necesita remover.

5.2.3.7. Optimización por configuración de APIC en pruebas que utilizan multiprocesadores

En las pruebas que utilizan multiprocesadores el APIC se encarga de sincronizar cada uno de CPU. Así cuando el primero de ellos inicia, configura el APIC para que este inicialice a los demás. Esto lo realiza identificando la dirección de cada microprocesador para hacer una inserción de SIPIs, que es una interrupción que se genera con el APIC para sincronizarse con los demás microprocesadores.

Esta interrupción puede enviarse por software (dentro de la prueba) o de manera externa (mediante línea de comando).

En las pruebas en que la inserción de SIPIs se hace por software, existe la posibilidad de identificar las direcciones de memoria donde inician los microprocesadores y así eliminar esta configuración de la prueba para realizarla por hardware, es decir enviarla por medio de línea de comando.

Para hacer la configuración por medio de software la prueba hace lo siguiente:

- Habilitar el APIC en cada microprocesador
- Identificar la posición de memoria donde debe empezar la ejecución el microprocesador.
- Identificar el ID de cada microprocesador que se desea inicializar para poder hacer el envío del SIPI.
- Hacer el envío del SIPI para la inicialización del microprocesador, esto se hace escribiendo el registro ICR (Interrupt Command Register).

El ejemplo de la configuración en una prueba original es el mostrado en la figura 5.42.

La configuración por hardware consiste en enviar de forma externa la interrupción, por lo que no se requieren los pasos descritos anteriormente.

Los algoritmos para la identificación del APIC y de las direcciones de configuración son muy semejantes al del caso 5.2.3.6, las diferencias son las siguientes:

- Se debe de identificar la dirección de memoria donde inicia cada microprocesador.
- Si es posible hacer la optimización se debe dejar el APIC habilitado pero las demás instrucciones pueden removerse.

Las instrucciones para configurar el APIC, junto con las relativas a la inserción de SIPIs para inicializar los demás micros se muestran en la figura 542.

Habilitación del APIC e inserción de SIPIs en prueba original

```
; THREAD 0: Enable APIC
        mov_b9 ecx, dword 0x0000001b
        rdmsr_0f32
        and_25 eax, dword 0x000007ff
        or_0d eax, dword 0x00050800
        xor_31 edx, edx
        wrmsr_0f30
        mov_ba dx, word 0x5000
        mov_8e es, dx
; THREAD 0: Enable APIC using SVR
        mov_b9 ecx, dword 0x000000f0
        mov_c7 dword ptr es:[ecx], dword 0x000001ff
; THREAD 0: Initialize ICR
;;figure out BSP's APIC ID
        mov_8b ebx, dword ptr es:[word 0x0020]
        and_81 ebx, dword 0x0f000000
; THREAD 0: find out if thread 1 was involved in the INIT interrupt
        mov_8b ebp, dword ptr [word 0x0008]
```

Habilitación del APIC e inserción de SIPIs en prueba original

```
        and_83 ebp, byte 0x02
; THREAD 0: if not skip sending a SIPI to this thread
       jz_0f84 word 0x0048
; THREAD 0: Determine APIC ID of AP by toggling ID bits based on thread number
       mov_89 eax, ebx
       xor_35 eax, dword 0x01000000
; THREAD 0: write AP's APIC ID to destination field in of ICR
       mov_8b ecx, dword ptr es:[word 0x0310]
       and_81 ecx, dword 0xf0ffffff
       or_09 ecx, eax
       mov_89 dword ptr es:[word 0x0310], ecx
; THREAD 0: Write 110 to Delivery Mode field and Thread start vector
       mov_8b ecx, dword ptr es:[word 0x0300]
       and_81 ecx, dword 0xfff0f800
       or_81 ecx, dword 0x0000069f
       mov_89 dword ptr es:[word 0x0300], ecx
; THREAD 0: Wait for delivery status bit to become idle
       mov_8b ecx, dword ptr es:[word 0x0300]
       and_81 ecx, dword 0x00001000
       jnz_75 byte 0xf1
; THREAD 0: find out if thread 2 was involved in the INIT interrupt
       mov_8b ebp, dword ptr [word 0x0008]
       and_83 ebp, byte 0x04
; THREAD 0: if not skip sending a SIPI to this thread
       jz_0f84 word 0x0048
; THREAD 0: Determine APIC ID of AP by toggling ID bits based on thread number
       mov_89 eax, ebx
       xor_35 eax, dword 0x02000000
; THREAD 0: write AP's APIC ID to destination field in of ICR
       mov_8b ecx, dword ptr es:[word 0x0310]
       and_81 ecx, dword 0xf0ffffff
       or_09 ecx, eax
       mov_89 dword ptr es:[word 0x0310], ecx
; THREAD 0: Write 110 to Delivery Mode field and Thread start vector
       mov_8b ecx, dword ptr es:[word 0x0300]
       and_81 ecx, dword 0xfff0f800
       or_81 ecx, dword 0x0000069e
       mov_89 dword ptr es:[word 0x0300], ecx
; THREAD 0: Wait for delivery status bit to become idle
       mov_8b ecx, dword ptr es:[word 0x0300]
       and_81 ecx, dword 0x00001000
       jnz_75 byte 0xf1
; THREAD 0: find out if thread 3 was involved in the INIT interrupt
        mov_8b ebp, dword ptr [word 0x0008]
       and_83 ebp, byte 0x08

; THREAD 0: if not skip sending a SIPI to this thread
       jz_0f84 word 0x0048
; THREAD 0: Determine APIC ID of AP by toggling ID bits based on thread number
       mov_89 eax, ebx
       xor_35 eax, dword 0x03000000
; THREAD 0: write AP's APIC ID to destination field in of ICR
       mov_8b ecx, dword ptr es:[word 0x0310]
       and_81 ecx, dword 0xf0ffffff
       or_09 ecx, eax
       mov_89 dword ptr es:[word 0x0310], ecx
; THREAD 0: Write 110 to Delivery Mode field and Thread start vector
       mov_8b ecx, dword ptr es:[word 0x0300]
       and_81 ecx, dword 0xfff0f800
       or_81 ecx, dword 0x0000069d
       mov_89 dword ptr es:[word 0x0300], ecx
; THREAD 0: Wait for delivery status bit to become idle
       mov_8b ecx, dword ptr es:[word 0x0300]
       and_81 ecx, dword 0x00001000
       jnz_75 byte 0xf1
```

Figura 5.42. Código para la habilitación del APIC e inserción de SIPIs en prueba original.

En la figura 5.43 se muestra el código que se debe dejar si se elimina la inserción de SIPIs y se hace de manera externa.

Habilitación del APIC en prueba optimizada

```
; THREAD 0: Enable APIC
        mov_b9 ecx, dword 0x0000001b
        rdmsr_0f32
        and_25 eax, dword 0x000007ff
        or_0d eax, dword 0x00050800
        xor_31 edx, edx
        wrmsr_0f30
        mov_ba dx, word 0x5000
        mov_8e es, dx
```

Figura 5.43. Habilitación del APIC en la prueba optimizada.

Como se observa en ambas figuras la optimización es bastante importante, sin embargo es aplicable solo en pruebas que usan varios microprocesadores y que sea posible obtener las direcciones de memoria donde inicia cada uno. Esta optimización será denominada en las otras secciones como optimización por APIC_4T.

5.2.4. Generación de la nueva prueba

Este es el bloque final, en el cual se genera la nueva prueba optimizada. Debido a que ya cada uno de los subprogramas de optimización realizaron la modificación necesaria sobre la prueba, este bloque lo que hace es simular la prueba para determinar que funciona correctamente.

El algoritmo utilizado es el mismo que se muestra en la figura 5.7, para hacer la simulación de la prueba original. La única diferencia es que se hace con la prueba optimizada.

Si no hay errores de simulación se guarda la prueba con el nombre *new_<nombre de la prueba original>.asm* y en caso de un error se declara como un fallo en la simulación por lo que la optimización es fallida.

Capítulo 6: Análisis de resultados

En este capítulo se presentan los resultados obtenidos con la herramienta de optimización, se muestran figuras y tablas que demuestran la eficacia de cada una de las optimizaciones encontradas y utilizadas en la solución del proyecto.

6.1. Selección de la muestra de datos

Cada microprocesador Intel tiene una meta de cobertura que generalmente es de un 85%, sin embargo para lograr ese porcentaje se necesitan bastantes pruebas (desde 3000 hasta 7000 dependiendo de la familia del microprocesador) tal y como se ha mencionado en las secciones anteriores.

El microprocesador trabajado no es la excepción y por esta razón se cuenta actualmente con más de 2000 pruebas y este número sigue incrementando semana a semana.

Los recursos disponibles para el proyecto no permiten hacer el análisis con todas las pruebas por las siguientes razones:

- La simulación de cada prueba en el modelo del microprocesador puede tardar desde un día hasta incluso una semana.
- No se cuenta con la cantidad de máquinas suficientes ni con el espacio en disco duro para poder enviar a simular todas las pruebas disponibles y obtener resultados en el tiempo requerido.

Por ello fue indispensable seleccionar una muestra de pruebas que se amolde a los recursos que se disponen y permita adquirir los resultados en el tiempo necesario. En el momento en que se tomó la muestra habían 1350 pruebas disponibles, por lo tanto se decidió escoger una muestra de un 10% del total, para que la misma fuera significativa y manejable.

Las pruebas están enfocadas para diferentes áreas del microprocesador, por esta razón se hizo un análisis de la cantidad que había por sector. Para abarcar de igual forma todas las regiones del microprocesador que se trabajan en Intel Costa Rica, para la muestra se seleccionó un 10% de las pruebas que había por unidad.

Además se utilizaron pruebas con una duración entre corta a media (entre 200 mil hasta 700 mil ciclos del microprocesador), ya que actualmente se dispone de pruebas que tardan hasta 5 millones de ciclos ejecutándose, en casos como este la reducción de tiempo es aún mayor, pero el tiempo de simulación es de varios días y esto no es compatible con el tiempo que se dispone para hacer el análisis de la muestra de pruebas.

En la tabla 6.1 se detalla las 135 pruebas seleccionadas, ordenadas según el sector del microprocesador que prueban.

Tabla 6.1. Muestra de pruebas seleccionada para ser analizada por la herramienta.

Región del microprocesador	Cantidad de pruebas
DMI	22
GFX	27
MMIO	8
PCIE	12
DDR	49
IMPH	12
Total	135

Las pruebas de la tabla 6.1 fueron ejecutadas en la herramienta y se midió el impacto que tuvo cada una de ellas. Los resultados se detallan en la tabla 6.2.

Tabla 6.2. Pruebas a las que fue posible aplicarle las optimizaciones encontradas.

Tipo de optimización	Pruebas optimizadas
Registros MSRs	126
Registros de control	135
Interrupciones	116
Registros generales	126
APIC	0
APIC_4T	19
Simulaciones fallidas	0
Total	135

Se hallaron optimizaciones por registros de control en todas las pruebas. Las optimizaciones de registros generales y registros MSR fueron aplicadas a 126 pruebas. Mientras que las optimizaciones por interrupciones fueron empleadas en 116 pruebas.

La optimización por APIC_4T fue posible aplicarla en 19 pruebas mientras que la de APIC no fue viable para ninguna de las pruebas.

En la figura 6.1 se exhibe gráficamente los datos de la tabla 6.2.

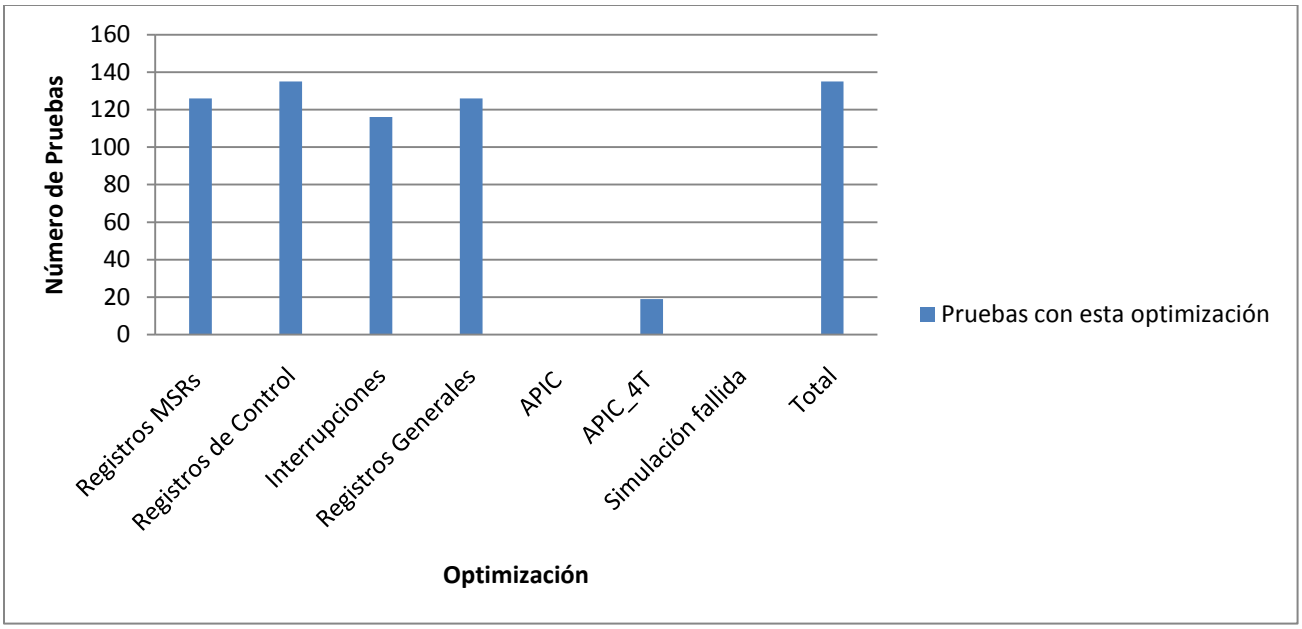

Figura 6.1. Número de optimizaciones que fueron encontradas en el conjunto de pruebas.

En la figura 6.1 se ilustra más claramente lo mencionado en el párrafo anterior. A pesar de que optimización por APIC no aparece en ninguna de las pruebas escogidas, esta no fue desechada por las siguientes razones:

- La mayoría de las pruebas de la muestra utilizan varios microprocesadores, por lo que no es posible aplicar la optimización.
- Se ahorran aproximadamente 60 instrucciones cuando se elimina una configuración por APIC. Por lo tanto cuando la herramienta se ejecute con la población de pruebas, aunque sean pocas a las que se les pueda aplicar la optimización, el tiempo disminuido será apreciable.

Por otro lado, como se mencionó en la sección 5.2.3.2, a pesar de que la optimización por registros de control disminuye pocas instrucciones (dos en el ejemplo mostrado en la figura 5.21) la misma está presente en todas las pruebas analizadas.

Como se verá en la sección 6.2, las optimizaciones por registros generales son las más importantes ya que disminuyeron el mayor tiempo de ejecución del conjunto de pruebas.

Cabe anotar, sin embargo, que fue necesario eliminar 11 pruebas de las 135, por las razones que se explican en la sección 6.3.3.

6.2. Evaluación del tiempo de ejecución de las pruebas optimizadas

Para analizar el impacto que aportaron las optimizaciones en el tiempo de ejecución de las pruebas, fue necesario simular en el modelo del microprocesador todas las pruebas con cada optimización por aparte y también con todas las optimizaciones aplicadas.

6.2.1. Tiempo de ejecución y ciclos reducidos con las optimizaciones aplicadas

De los resultados obtenidos se determinaron los ciclos reducidos por cada una de las optimizaciones en la muestra de pruebas. Para darse una idea del tiempo real ganado por optimización, se coloca una columna con la equivalencia ciclos-ms de un procesador Sandy Bridge con una frecuencia de reloj de 2 Gigahertz.

Tabla 6.3. Ciclos del microprocesador ejecutados para las pruebas originales y reducidas en todas las optimizaciones.

Optimización	Original (ciclos)	Optimizada (ciclos)	Ciclos reducidos	Tiempo reducido (ms)
Registros de control	42652191	42417226	234965	0,1174825
Interrupciones	42652191	42137825	514366	0,257183
Registros MSRs	42652191	42525366	126825	0,0634125
APIC__4T	42652191	42234047	418144	0,209072
Registros generales	42652191	36420190	6232001	3,1160005
Total	42652191	35125890	7526301	3,7631505

En la tabla 6.2 se observa que el tiempo total reducido por todas las optimizaciones es de 3.76 ms, correspondiente a 7.526 millones de ciclos. Esta reducción es bastante importante ya que cada prueba tarda entre 200 mil hasta 700 mil ciclos ejecutándose, por lo tanto en la muestra tomada, los ciclos eliminados equivalen a aproximadamente 15 pruebas de medio millón de ciclos cada una.

La figura 6.2 ilustra los resultados de la tabla 6.3:

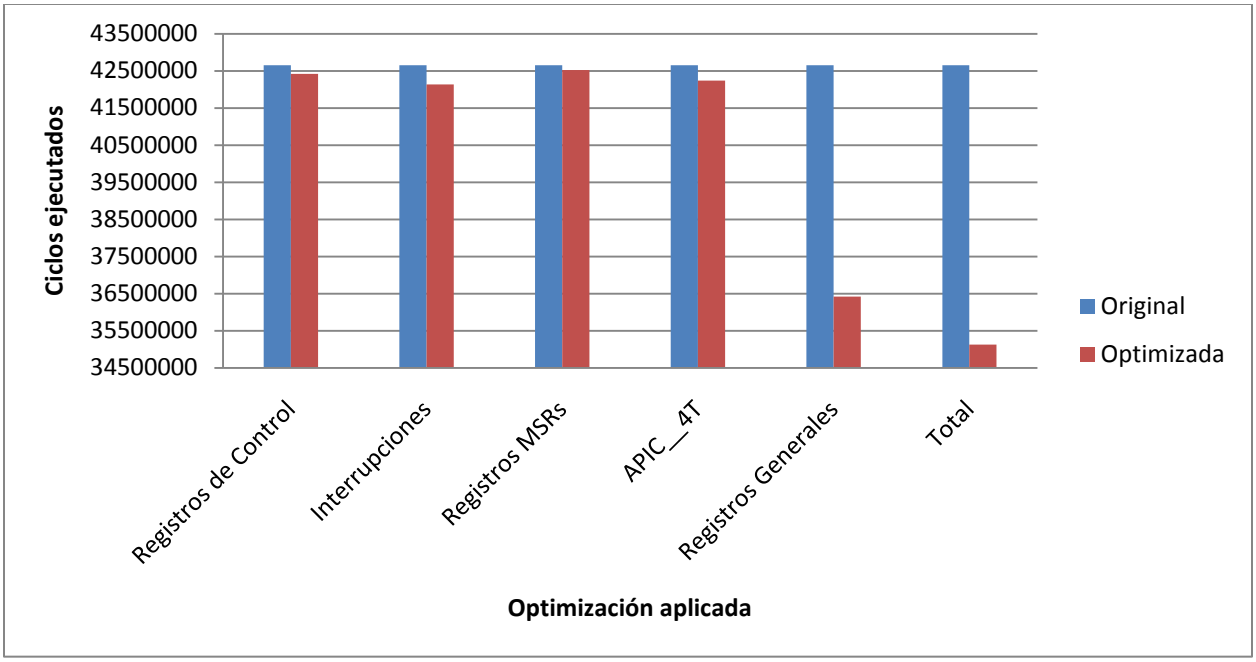

Figura 6.2. Ciclos ejecutados por el microprocesador en la prueba original versus ciclos de la prueba optimizada, por tipo de optimización y en total.

En la figura 6.2 se observa más claro la reducción de tiempo aportada por cada optimización. El mayor número de ciclos reducidos de las pruebas se debe a la optimización por registros generales. También hay una reducción importante por parte de las optimizaciones de interrupciones y APIC_4T.

A pesar de que las optimizaciones por registros de control y por registros MSRs están presentes en casi todas las pruebas, fueron las optimizaciones que redujeron menos tiempo.

Por su parte las optimizaciones por APIC_4T están presentes solamente en 19 pruebas pero redujo más tiempo que las mencionadas en el párrafo anterior, lo que indica la importancia de eliminar este patrón de configuración de la prueba cuando es posible, a pesar de su escasa presencia.

La figura 6.3 muestra el tiempo reducido por cada una de las optimizaciones así como el del total.

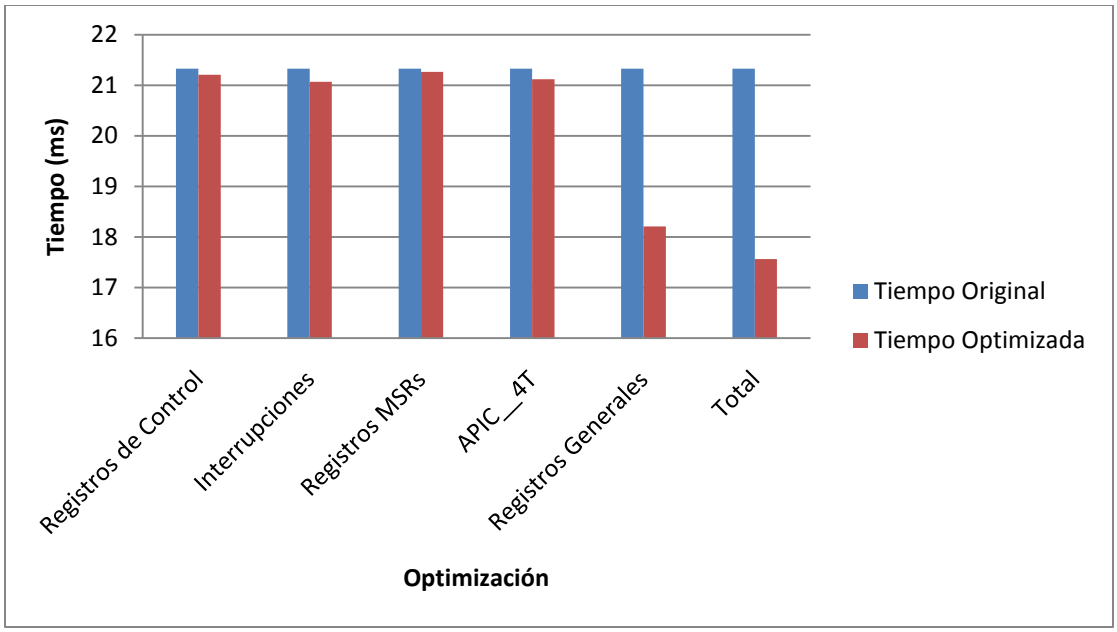

Figura 6.3. Tiempo total que tardó la ejecución de las pruebas originales y las modificadas por cada optimización y por el total de estas. .

En la figura 6.3 se aprecia que el tiempo total de la muestra de pruebas ejecutadas es cerca de 21 ms y el tiempo ahorrado es de 3.76 ms.

Suponiendo que se utilizan 5 mil pruebas para este microprocesador y siguiendo el patrón obtenido en esta muestra, el tiempo optimizado sería de 157 ms. Basado en lo mencionado en la sección 4.2, esta reducción ahorraría anualmente $280000 en manufactura, suponiendo una producción de 3 millones de unidades por mes.

Sin embargo, como se explicó en el punto 6.1, las pruebas con mayor tiempo de ejecución no se utilizaron para el análisis. Por lo tanto este resultado es pesimista, ya que cuando se agreguen estas pruebas las optimizaciones aumentarán y por ende el ahorro real será aún mayor.

6.2.2. Porcentaje de tiempo reducido con cada una de las optimizaciones

En la tabla 6.4 se muestra el porcentaje de tiempo reducido por cada optimización.

Tabla 6.4. Porcentaje de tiempo reducido de cada una de las optimizaciones sobre el total de las pruebas ejecutadas.

Optimización	Porcentaje de reducción
Registros de control	0,55%
Interrupciones	1,20%
Registros MSRs	0,29%
APIC__4T	0,98%
Registros generales	14,61%
Total	17,64%

Estos porcentajes de reducción se muestran también en la figura 6.4.

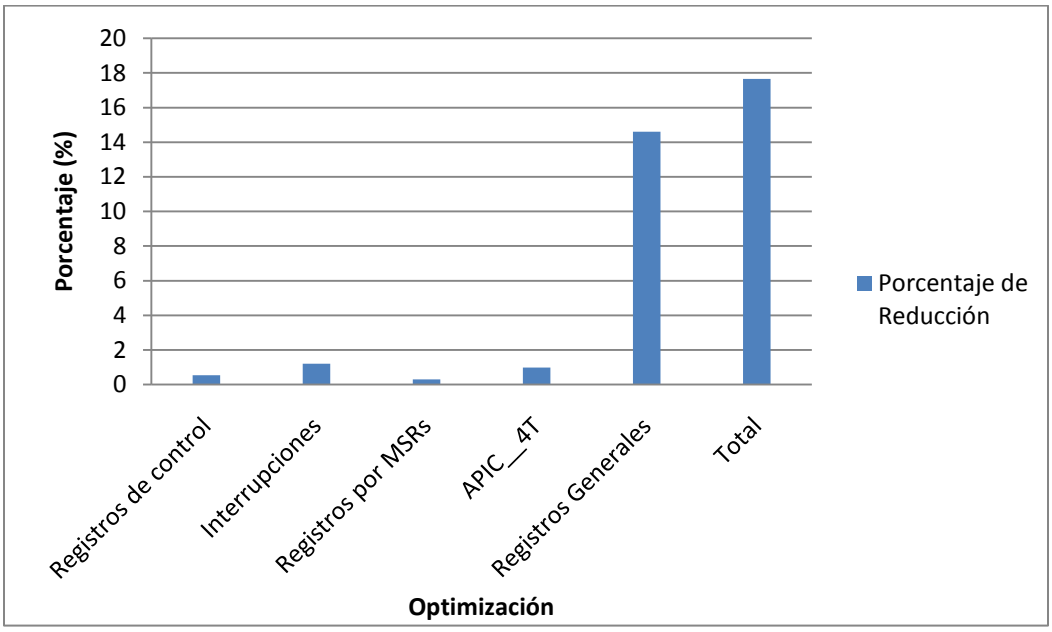

Figura 6.4. Porcentaje de reducción aportado por cada una de las optimizaciones sobre la muestra de pruebas.

Inicialmente se estaba obteniendo una reducción del tiempo total de ejecución del 3%, con el que se superaba el objetivo general del proyecto que es de un 1%. Sin embargo, en uno de los análisis de las pruebas se encontró que a muchos registros se les estaba escribiendo el valor que ya poseían, o sea que la escritura era innecesaria.

Basado en estos análisis se determinó la optimización por registros generales, ésta redujo un 14.61% el tiempo de ejecución de las pruebas. Con esto, el tiempo total reducido fue de un 17.64%, por lo tanto se superó 17 veces lo estipulado en el objetivo general.

Esta optimización se realiza en la etapa de ejecución de la prueba, por lo tanto existe un riesgo al eliminar instrucciones de esta parte (ver sección 6.3.1). Sin embargo, si la herramienta asimila que el patrón identificado no aporta cobertura a la prueba, se aplica la optimización.

Las demás optimizaciones se realizaron en la etapa de inicialización de la prueba. La perteneciente a interrupciones redujeron un 1.2% el tiempo de ejecución de la muestra de pruebas y el APIC_4T reduce un 0.98%.

Ambas optimizaciones pertenecen a configuraciones del microprocesador, y eliminan instrucciones más complejas tal como *lidt* para el caso de interrupciones y la inserción de SIPIs para el caso del APIC_4T. Por esta razón, a pesar de presentarse en menos ocasiones que las referentes a registros de control y registros de MSRs, reducen un porcentaje mayor de optimización porque en el caso de las optimizaciones por registros solo se eliminan instrucciones lógicas y *mov,* las cuáles toman menos ciclos de reloj para ejecutarse.

Seguidamente se presenta en la tabla 5.5 el aporte de reducción de cada optimización sobre el 100% de los ciclos reducidos.

Tabla 6.5. Porcentaje de reducción aportado por cada optimización sobre el total de ciclos reducidos.

Optimización	Porcentaje del Total Reducido
CR	3,12%
Interrupciones	6,83%
Rdmsr	1,68%
APIC__4T	5,55%
Registros generales	82,80%
Total	100%

La tabla 6.5 presenta el porcentaje reducido por cada optimización del total de tiempo que se logró reducir. En este se ve más claramente el impacto de la optimización de los registros generales sobre el tiempo total disminuido: un 82.80% de los casos.

La figura 6.5 presenta la gráfica para representar los resultados de la tabla 6.5.

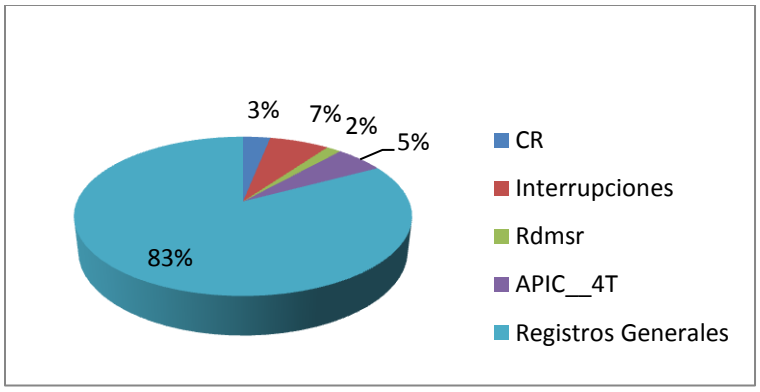

Figura 6.5. Gráfica porcentual que representa los resultados de la tabla 5.5.

La reducción por interrupciones aporta un 6.83%, lo que se debe a que todas las pruebas hacen la configuración para habilitarlas, pero muy pocas las ejecutan.

También como se ha venido mencionando aunque la optimización por APIC_4T está solamente en 19 pruebas, la reducción aportada es de un 5.55%.

También se reafirma que las reducciones por registros de control y de registros MSR a pesar de estar presentes en casi todas las pruebas son las que remueven menor cantidad de ciclos del microprocesador.

6.3 Comprobación del porcentaje de cobertura de las pruebas críticas

El porcentaje de cobertura está ligado a la cantidad de errores que puede detectar una prueba en el microprocesador. Como se ha explicado en secciones anteriores la cobertura es una restricción ineludible para la optimización.

6.3.1. Sectores de la prueba que aportan cobertura al microprocesador

La prueba posee áreas que aportan cobertura y áreas que no. Es sumamente importante que la herramienta de optimización elimine solamente instrucciones que no contribuyan con el porcentaje de cobertura de la prueba.

Por esta razón se hizo un análisis de las pruebas para determinar los sectores en los que se pueden hacer optimizaciones y en los que existe riesgo de hacer modificaciones porque se pueden remover instrucciones que aportan cobertura.

La figura 6.6 muestra las regiones de la prueba dónde es posible aportar cobertura al microprocesador y dónde es posible encontrar optimizaciones.

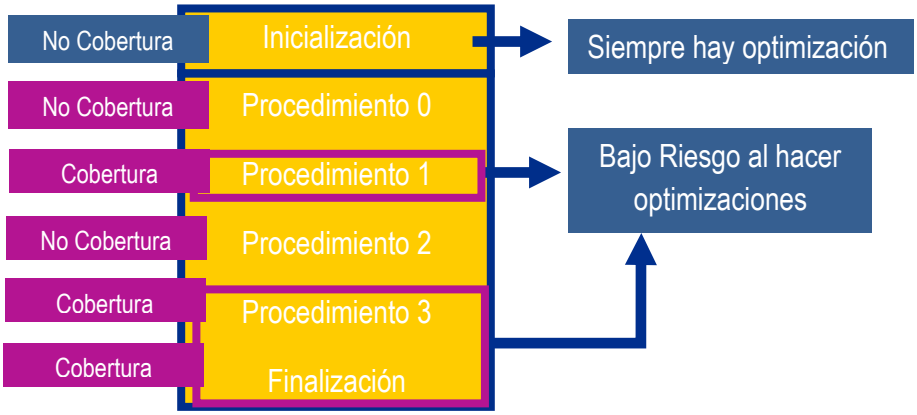

Figura 6.6. Regiones en que la prueba aporta cobertura al microprocesador y donde es posible encontrar optimizaciones.

Cada una de estas regiones implica distintas funciones y, por ende, están expuestas a distintos impactos debidos a las optimizaciones:

a. Inicialización:

En esta etapa se realizan las configuraciones necesarias para que la prueba pueda ejecutar las funciones que le corresponden. Además se configura el modo protegido o modo de 64 bits.

Por lo anterior, es posible aquí realizar optimizaciones por configuración de interrupciones, APIC, APIC_4T, registros de control y registros de MSR. Como esta etapa es solamente de inicialización no aporta nada de cobertura adicional a la prueba, por lo que no hay riesgo de eliminar o modificar instrucciones y que disminuyan la cobertura.

b. Procedimientos 0, 1, 2, etc:

Cuando la prueba finaliza la etapa de inicialización empieza la ejecución del código de la prueba. En esta sección hay partes que presentan cobertura y partes que no. Por ello se corre un bajo riesgo si se hacen optimizaciones en esta área ya que se podrían eliminar instrucciones necesarias para la prueba.

c. Finalización:

Generalmente aporta cobertura ya que esta sección es propia de la prueba, por lo tanto tiene un fin específico. Por eso no se recomienda hacer modificaciones en esta sección ya que hay una gran posibilidad de disminuir la cobertura.

6.3.2. Cobertura de las pruebas originales y optimizadas

Las pruebas optimizadas se simularon en el modelo de fallas del microprocesador. Así se comprueba el que las pruebas optimizadas tengan el mismo porcentaje de cobertura de la prueba original.

Sin embargo, debido a que la ejecución de las pruebas en el modelo de fallas tarda varios días o hasta semanas, no se pudo hacer la medición de cobertura a toda la muestra de pruebas, ya que todos los diseñadores de pruebas están constantemente utilizando este modelo.

En la tabla 6.6 se muestran los resultados de cobertura obtenido sobre el grupo de pruebas medidas.

Tabla 6.6. Porcentaje de cobertura para la prueba original y optimizada en el microprocesador.

	Nombre de la prueba	Original (%)	Optimizada (%)
1	Prueba 1	44,3	44,29
2	Prueba 5	38,7	38,68
3	Prueba 15	49,7	49,69
4	Prueba 23	27,08	27,08
5	Prueba 25	36,58	36,58
6	Prueba 36	36,17	36,17
7	Prueba 38	44,31	44,31
8	Prueba 44	38,46	38,46
9	Prueba 48	27,38	27,38
10	Prueba 56	49,69	49,68
11	Prueba 62	38,34	38,34
12	Prueba 73	26,81	26,81
13	Prueba 82	39,34	39,34
14	Prueba 94	20,51	20,5
15	Prueba 103	24,04	24,02
16	Prueba 114	19,43	19,42
17	Prueba 117	16,78	16,78
18	Prueba 122	24,56	24,56

Para visualizar mejor los datos expuestos en la tabla 6.6, se presenta la diferencia de cobertura entre la prueba original y la optimizada en la figura 6.7.

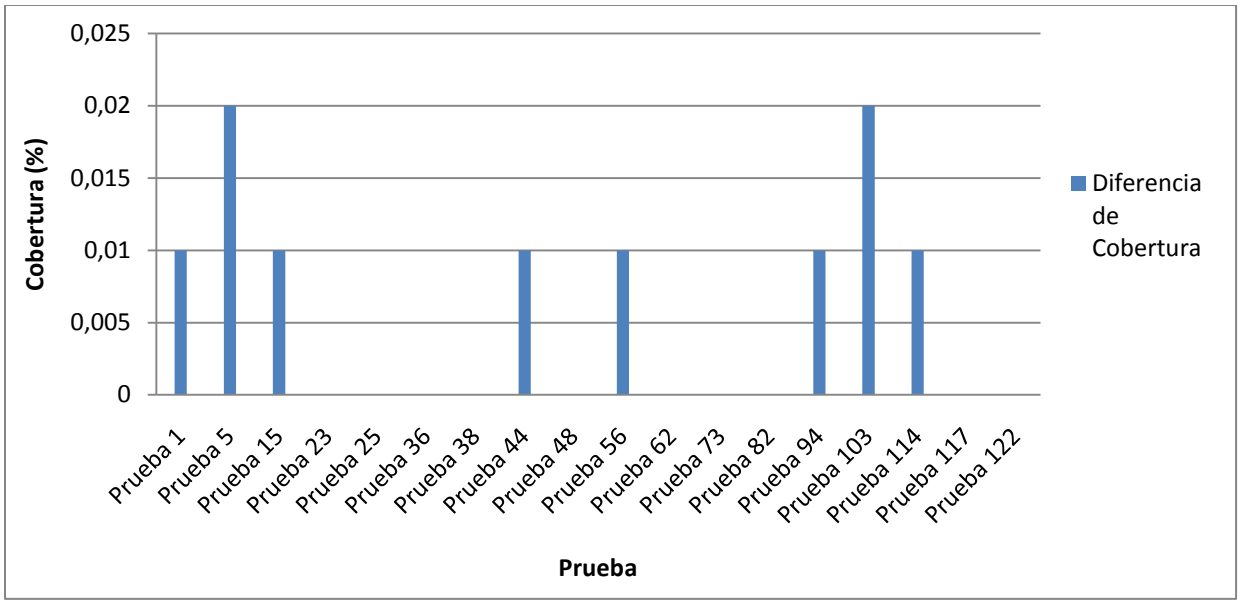

Figura 6.7. Diferencia entre la cobertura aportada por la prueba original y la optimizada sobre el modelo del microprocesador.

De la tabla 6.6 y la figura 6.7 se puede observar que todas las pruebas presentan la misma cobertura que la prueba original, porque aunque algunas presenten una diferencia, la misma es tan pequeña que se puede ignorar ya que la mayor diferencia es de un 0.02%.

Con esto se confirma que el proyecto cumple con los objetivos propuestos, tanto el general como sus objetivos específicos: la herramienta desarrollada es factible y funcional sobre un microprocesador Sandy Bridge y en futuros procesadores.

6.3.3. Pruebas fallidas por problemas de cobertura

Como se explicó en la sección 6.3.1, existe un bajo riesgo cuando se hacen optimizaciones en la parte de ejecución de la prueba porque se pueden eliminar instrucciones necesarias para la cobertura del microprocesador.

Por ejemplo, la optimización por registros generales se efectúa tanto en la etapa de inicialización como en la etapa de ejecución de la prueba. De hecho, por un problema de cobertura, nueve pruebas debieron eliminarse de la muestra, por lo que se vuelve necesario hacer un arreglo a esta optimización para eliminar la falla.

Las pruebas eliminadas se encargan de ejercitar la lógica de diferentes regiones de memoria. Sin embargo para hacer esto se hacen lecturas a cada una de las posiciones de memoria deseadas, es decir un movimiento de memoria a un registro de propósito general.

La mayoría de las posiciones de memoria leídas tenían un valor de cero, y el movimiento a registro se hacía siempre a EAX. Por lo tanto, cuando la herramienta hizo el estudio de la prueba, eliminó muchas de estas instrucciones porque se estaba escribiendo un cero a EAX cuando el mismo ya poseía este valor.

Cuando se obtuvieron los resultados de cobertura, estas pruebas lo habían reducido hasta en un 10% y fue así como se detectó el problema.

Este problema ya fue solucionado, sin embargo no había tiempo para volver a correr las pruebas y obtener los resultados de las mismas a tiempo.

Ahora la herramienta antes de eliminar una instrucción debido a esta optimización, primero verifica si es una lectura de memoria, en caso de ser así no se hace la optimización ya que las lecturas a memoria ejercitan lógica que da cobertura a la prueba.

6.4 Facilidades de expansión de la herramienta

Cuando se trata de verificar un circuito de las dimensiones de un microprocesador de Intel, donde además la observabilidad y controlabilidad están severamente limitadas por la extrema jerarquización del circuito, hace que proponer un algoritmo de optimización automatizado para dichos procedimientos de verificación y prueba sea una labor que requiere mucha investigación y trabajo.

Por ello, este proyecto inició la búsqueda para llegar a la optimización de las pruebas, sin embargo este es solo el principio, actualmente se está reduciendo el tiempo de ejecución de las pruebas en un 17%, pero se deben seguir buscando más patrones y alternativas que reduzcan la ejecución de la prueba para llevar la misma hasta el nivel óptimo. Por esta razón se seguirá trabajando en ello y las propuestas disponibles son las siguientes:

- El diseño actual de la herramienta permite agregar fácilmente patrones de reducción que sean encontrados en el futuro.
- Debido a que la etapa de reset de las pruebas toma mucho tiempo (entre 40 mil hasta 200 mil ciclos) una buena opción es determinar si existe la posibilidad de unir dos pruebas en una sola, y así se ahorran todos estos ciclos.
- Distribuir las instrucciones de una prueba entre varios CPU, así se trabaja de forma paralela y se ahorra bastante tiempo de ejecución.

Capítulo 7: Conclusiones y recomendaciones

7.1 Conclusiones

Las conclusiones obtenidas en el desarrollo de este proyecto son las siguientes:

- Se encontraron seis casos de optimización que permitieron reducir el tiempo de ejecución de cada una de las pruebas en un 17.64%.
- Se diseñó y generó una herramienta de software encargada de determinar y eliminar automáticamente los casos de optimización de cada una de las pruebas.
- Se comprobó que el porcentaje de cobertura de las pruebas originales es el mismo que el de su respectiva prueba optimizada.
- Se determinó las partes de la prueba que aportan y no aportan cobertura durante su ejecución en el microprocesador.
- Se aportó una herramienta de soporte al diseñador de pruebas para que automáticamente se genere una prueba más eficiente.

7.2 Recomendaciones

Después de la realización de este proyecto, se citan algunas recomendaciones que pueden facilitar el seguimiento del mismo:

- Eliminar la etapa de reset de varias pruebas mezclando dos o más en una sola, esto ahorraría hasta 100 mil ciclos por cada prueba fusionada.
- Estudiar cada una de las pruebas disponibles para extraer más casos posibles de optimización y agregar las mismas a la herramienta desarrollada.
- Buscar el ahorro de tiempo de ejecución distribuyendo las instrucciones de una prueba entre varios CPU.
- Ejecutar la herramienta en otros modelos de microprocesadores para extender el impacto que pueda tener en estos.

Bibliografia

[1] Intel, Intel 64 and IA 32 Architectures Software Developer's Manual, Volume 3ª.

[2] M. Abramovici, M. Breuer, A. Friedman. Digital Systems Testing and Testable Design. IEEE Press, Order N° PC041683.

[3] A. Antao, J. Brodersen. IEEE Behavioral Simulation for Analog System Design Verification, vol 3, 1995.

[4] B. Barry. Microprocesadores Intel. Séptima edición, Prentice Hall, 2006.

[5] RAE: Definición de microprocesador, [en línea] [citado el 2 de junio 2010]. Disponible en: http://buscon.rae.es/draeI/SrvltConsulta?TIPO_BUS=3&LEMA=microprocesador

[6] RAE: Definición de algoritmo, [en línea] [citado el 2 de junio 2010]. Disponible en: http://buscon.rae.es/draeI/SrvltConsulta?TIPO_BUS=3&LEMA=algoritmo

Apéndices

A.1. Glosario

Microprocesador

Circuito constituido por millares de transistores integrados en un chip, que realiza alguna determinada función de los computadores electrónicos digitales [5].

Algoritmo

Conjunto ordenado y finito de operaciones que permite hallar la solución de un problema [6].

Sandy Bridge

Microprocesador de 32 nm, que pertenece a la arquitectura IA-64 y utiliza cuatro núcleos.

Lenguaje de ensamblador

Lenguaje muy similar al de máquina, pero permite al programador utilizar códigos nemónicos en inglés en lugar de números binarios. Esto le facilita la programación del sistema. [4].

Modelo de Fallas

Se define como un universo de fallas en el cual cada falla es identificada individualmente por las pruebas [2]. Cada falla corresponde a un error que se podría identificar en el microprocesador. El modelo de fallas utilizado pertenece al del microprocesador Sandy Bridge.

Cobertura de errores de la prueba

Es la cantidad de errores que puede detectar una prueba en el microprocesador [2].

Porcentaje de cobertura

Es el porcentaje que se obtiene al dividir el número de fallas detectadas por una prueba entre el total de fallas del modelo [2]. En este proyecto indica qué tanta lógica del microprocesador es ejercitada con cada prueba.

A.2. Manual de Usuario

La herramienta desarrollada para optimizar pruebas es fácil de utilizar. El usuario debe seguir los siguientes pasos para poder utilizarla correctamente:

1. **Cargar las variables de ambiente para el simulador de ensamblador**

Existe un simulador de ensamblador para cada modelo de microprocesador, por esto es necesario cargar las variables de ambiente respectivas para simular la prueba en el modelo deseado.

2. **Cargar los argumentos necesarios para generar el reporte de resultados**

Para generar el reporte con los resultados de la simulación, es necesario cargar las variables de ambiente.

Estas permiten al simulador guardar el reporte con los resultados de la simulación y además indica que se guarden todas las transacciones de memoria y de registros que existan en la prueba para que puedan ser analizados por la herramienta.

3. **Ejecutar la línea de comando para ejecutar la herramienta**

Como se explico en el capítulo 4, la línea de comandos para ejecutar el programa es la siguiente:

<path de la herramienta> -path <localización de las pruebas originales>

-regression <lugar donde se van a guardar las pruebas optimizadas>

-total <Número de pruebas que se desea optimizar>

La herramienta comenzara a simular y optimizar las pruebas.

4. **Reporte de resultados**

Cuando la herramienta termina de ejecutar todas las pruebas, genera un reporte en el que se presentan todas las optimizaciones que se encontraron y un resumen general de todas las pruebas.

El reporte se presenta en la figura A.1:

Test Name	Optimization	Cpu Number	Optimization
1			
CRTW_TCTD_Imph_CRegs_test13.log	rdmsr	cpu0	1
CRTW_TCTD_Imph_CRegs_test13.log	rdmsr	cpu1	1
CRTW_TCTD_Imph_CRegs_test13.log	rdmsr	cpu2	1
CRTW_TCTD_Imph_CRegs_test13.log	rdmsr	cpu3	1
CRTW_TCTD_Imph_CRegs_test13.log	APIC	----	0
CRTW_TCTD_Imph_CRegs_test13.log	APIC_4T	cpu0	0
CRTW_TCTD_Imph_CRegs_test13.log	APIC_4T	cpu1	0
CRTW_TCTD_Imph_CRegs_test13.log	APIC_4T	cpu2	0
CRTW_TCTD_Imph_CRegs_test13.log	APIC_4T	cpu3	0
CRTW_TCTD_Imph_CRegs_test13.log	CR	CR0	1
CRTW_TCTD_Imph_CRegs_test13.log	CR	CR3	0
CRTW_TCTD_Imph_CRegs_test13.log	CR	CR4	1
CRTW_TCTD_Imph_CRegs_test13.log	Interrupts	cpu0	1
CRTW_TCTD_Imph_CRegs_test13.log	Interrupts	cpu1	1
CRTW_TCTD_Imph_CRegs_test13.log	Interrupts	cpu2	1
CRTW_TCTD_Imph_CRegs_test13.log	Interrupts	cpu3	1
CRTW_TCTD_Imph_CRegs_test13.log	Register	RAX	1
CRTW_TCTD_Imph_CRegs_test13.log	Register	RBX	0
CRTW_TCTD_Imph_CRegs_test13.log	Register	RCX	1
CRTW_TCTD_Imph_CRegs_test13.log	Register	RDX	1

Figura A.1. Reporte de resultados generado por la herramienta.

El reporte indica el tipo de optimización que está en cada prueba y para las pruebas que utilizan multiprocesadores señala en los procesadores que está presente.

También en el reporte se muestra un resumen general de todas las optimizaciones encontradas en el total de pruebas:

Optimization	Number of tests with this optimization
Rdmsr	14
CR Register	14
Interruptions	14
General Control Register	14
APIC	10
APIC_4T	2
Tests failing the simulation	1
Total tests analized	15

Figura A.2. Resumen generado por la herramienta.

En la figura A.2 se presenta el resumen mostrado en el reporte.

Finalmente las pruebas optimizadas se encuentran en el path de regresión.

A.3 Resultados de optimización de cada prueba

Tabla A.1. Porcentaje de ciclos reducido de las pruebas por la herramienta generada.

Prueba	Ciclos prueba original	Ciclos prueba optimizada	Ciclos reducidos	Porcentaje de ciclos reducidos
1	510547	480590	29957	5,87
2	510533	480546	29987	5,87
3	172733	140822	31911	18,47
4	172721	140830	31891	18,46
5	634562	476032	158530	24,98
6	634535	476052	158483	24,98
7	84196	56153	28043	33,31
8	54205	22971	31234	57,62
9	90612	35738	54874	60,56
10	51106	20885	30221	59,13
11	218549	158673	59876	27,40
12	465559	307783	157776	33,89
13	180638	167093	13545	7,50
14	186856	170872	15984	8,55
15	186891	169919	16972	9,08
16	130365	118635	11730	9,00
17	130362	118763	11599	8,90
18	414541	334697	79844	19,26
19	946985	766933	180052	19,01
20	753379	609595	143784	19,09
21	753304	610013	143291	19,02
22	753347	609714	143633	19,07
23	802594	648592	154002	19,19
24	718985	576533	142452	19,81
25	521559	392909	128650	24,67
26	521519	393021	128498	24,64
27	521529	393212	128317	24,60
28	521509	391751	129758	24,88
29	485850	434352	51498	10,60
30	485822	432919	52903	10,89
31	485837	432959	52878	10,88
32	523842	393962	129880	24,79
33	523807	393891	129916	24,80
34	523881	393778	130103	24,83
35	69062	58145	10917	15,81
36	68244	58012	10232	14,99

Prueba	Ciclos prueba original	Ciclos prueba optimizada	Ciclos reducidos	Porcentaje de ciclos reducidos
37	65437	53761	11676	17,84
38	919862	672924	246938	26,85
39	523815	394009	129806	24,78
40	662057	614060	47997	7,25
41	660367	613368	46999	7,12
42	658454	609376	49078	7,45
43	657632	608624	49008	7,45
44	656184	608110	48074	7,33
45	664210	615860	48350	7,28
46	659710	611828	47882	7,26
47	668486	619834	48652	7,28
48	665197	616578	48619	7,31
49	659152	612660	46492	7,05
50	662457	615043	47414	7,16
51	665837	617903	47934	7,20
52	656955	608817	48138	7,33
53	90345	65594	24751	27,40
54	90328	65644	24684	27,33
55	90358	65584	24774	27,42
56	90320	65756	24564	27,20
57	90387	65647	24740	27,37
58	90383	65520	24863	27,51
59	90371	65486	24885	27,54
60	90363	65677	24686	27,32
61	90367	65552	24815	27,46
62	391815	128248	263567	67,27
63	515663	424940	90723	17,59
64	233419	215164	18255	7,82
65	460081	350995	109086	23,71
66	688941	526835	162106	23,53
67	594635	460696	133939	22,52
68	536835	412477	124358	23,17
69	657812	547508	110304	16,77
70	444447	336686	107761	24,25
71	524363	419759	104604	19,95
72	745824	598379	147445	19,77
73	578790	444443	134347	23,21
74	455350	349995	105355	23,14
75	640496	492909	147587	23,04
76	517090	397431	119659	23,14
77	578767	443989	134778	23,29

Prueba	Ciclos prueba original	Ciclos prueba optimizada	Ciclos reducidos	Porcentaje de ciclos reducidos
78	640484	491983	148501	23,19
79	578770	445004	133766	23,11
80	640451	492034	148417	23,17
81	42287	32851	9436	22,31
82	42227	33073	9154	21,68
83	42248	32963	9285	21,98
84	42238	32953	9285	21,98
85	42207	32988	9219	21,84
86	42276	32787	9489	22,45
87	42287	32832	9455	22,36
88	42214	33172	9042	21,42
89	42290	32855	9435	22,31
90	80985	69860	11125	13,74
91	80998	69803	11195	13,82
92	80943	70040	10903	13,47
93	80925	70021	10904	13,47
94	80987	69849	11138	13,75
95	80911	69956	10955	13,54
96	80952	69973	10979	13,56
97	80989	69809	11180	13,80
98	80911	70058	10853	13,41
99	714303	576775	137528	19,25
100	714325	576702	137623	19,27
101	714384	576599	137785	19,29
102	159771	136604	23167	14,50
103	155578	102843	52735	33,90
104	232850	199333	33517	14,39
105	232832	199457	33375	14,33
106	175715	168944	6771	3,85
107	175719	168899	6820	3,88
108	175736	168871	6865	3,91
109	175707	169028	6679	3,80
110	175721	169003	6718	3,82
111	175755	168939	6816	3,88
112	175735	168952	6783	3,86
113	175743	168982	6761	3,85
114	175770	168895	6875	3,91
115	66699	56332	10367	15,54
116	66668	56473	10195	15,29
117	66638	56456	10182	15,28
118	66659	56392	10267	15,40

Prueba	Ciclos prueba original	Ciclos prueba optimizada	Ciclos reducidos	Porcentaje de ciclos reducidos
119	66637	56495	10142	15,22
120	66666	56469	10197	15,30
121	66616	56567	10049	15,08
122	66683	56362	10321	15,48
123	168417	158407	10010	5,94
124	182356	171233	11123	6,10
Total	42652191	42652191	7526301	17,64

Tabla A.2. Ciclos de ejecución reducidos con la optimización de CR.

Prueba	Ciclos prueba original	Ciclos prueba optimizada	Ciclos reducidos	Porcentaje de ciclos reducido
1	510547	508671	1876	0,37
2	510533	508680	1853	0,36
3	172733	170872	1861	1,08
4	172721	170886	1835	1,06
5	634562	632691	1871	0,29
6	634535	632684	1851	0,29
7	84196	82240	1956	2,32
8	54205	52318	1887	3,48
9	90612	88785	1827	2,02
10	51106	49290	1816	3,55
11	218549	216617	1932	0,88
12	465559	463660	1899	0,41
13	180638	178740	1898	1,05
14	186856	184967	1889	1,01
15	186891	184946	1945	1,04
16	130365	128485	1880	1,44
17	130362	128485	1877	1,44
18	414541	412604	1937	0,47
19	946985	945097	1888	0,20
20	753379	751425	1954	0,26
21	753304	751477	1827	0,24
22	753347	751444	1903	0,25
23	802594	800640	1954	0,24
24	718985	717089	1896	0,26
25	521559	519654	1905	0,37
26	521519	519678	1841	0,35
27	521529	519695	1834	0,35
28	521509	519693	1816	0,35

Prueba	Ciclos prueba original	Ciclos prueba optimizada	Ciclos reducidos	Porcentaje de ciclos reducido
29	485850	483928	1922	0,40
30	485822	483963	1859	0,38
31	485837	483991	1846	0,38
32	523842	521961	1881	0,36
33	523807	521904	1903	0,36
34	523881	521957	1924	0,37
35	69062	67147	1915	2,77
36	68244	66337	1907	2,79
37	65437	63560	1877	2,87
38	919862	917937	1925	0,21
39	523815	521995	1820	0,35
40	662057	660193	1864	0,28
41	660367	658494	1873	0,28
42	658454	656522	1932	0,29
43	657632	655770	1862	0,28
44	656184	654275	1909	0,29
45	664210	662334	1876	0,28
46	659710	657839	1871	0,28
47	668486	666596	1890	0,28
48	665197	663283	1914	0,29
49	659152	657292	1860	0,28
50	662457	660587	1870	0,28
51	665837	663976	1861	0,28
52	656955	655025	1930	0,29
53	90345	88426	1919	2,12
54	90328	88415	1913	2,12
55	90358	88464	1894	2,10
56	90320	88483	1837	2,03
57	90387	88486	1901	2,10
58	90383	88450	1933	2,14
59	90371	88412	1959	2,17
60	90363	88492	1871	2,07
61	90367	88462	1905	2,11
62	391815	389980	1835	0,47
63	515663	513764	1899	0,37
64	233419	231515	1904	0,82
65	460081	458116	1965	0,43
66	688941	687090	1851	0,27
67	594635	592726	1909	0,32
68	536835	534930	1905	0,35
69	657812	655939	1873	0,28

Prueba	Ciclos prueba original	Ciclos prueba optimizada	Ciclos reducidos	Porcentaje de ciclos reducido
70	444447	442571	1876	0,42
71	524363	522435	1928	0,37
72	745824	743934	1890	0,25
73	578790	576828	1962	0,34
74	455350	453427	1923	0,42
75	640496	638567	1929	0,30
76	517090	515174	1916	0,37
77	578767	576846	1921	0,33
78	640484	638545	1939	0,30
79	578770	576867	1903	0,33
80	640451	638520	1931	0,30
81	42287	40331	1956	4,63
82	42227	40363	1864	4,41
83	42248	40371	1877	4,44
84	42238	40355	1883	4,46
85	42207	40338	1869	4,43
86	42276	40313	1963	4,64
87	42287	40379	1908	4,51
88	42214	40378	1836	4,35
89	42290	40314	1976	4,67
90	80985	79064	1921	2,37
91	80998	79042	1956	2,41
92	80943	79090	1853	2,29
93	80925	79038	1887	2,33
94	80987	79011	1976	2,44
95	80911	79008	1903	2,35
96	80952	79081	1871	2,31
97	80989	79048	1941	2,40
98	80911	79069	1842	2,28
99	714303	712430	1873	0,26
100	714325	712423	1902	0,27
101	714384	712472	1912	0,27
102	159771	157879	1892	1,18
103	155578	153647	1931	1,24
104	232850	230988	1862	0,80
105	232832	230949	1883	0,81
106	175715	173852	1863	1,06
107	175719	173814	1905	1,08
108	175736	173801	1935	1,10
109	175707	173855	1852	1,05
110	175721	173896	1825	1,04

Prueba	Ciclos prueba original	Ciclos prueba optimizada	Ciclos reducidos	Porcentaje de ciclos reducido
111	175755	173848	1907	1,09
112	175735	173825	1910	1,09
113	175743	173831	1912	1,09
114	175770	173841	1929	1,10
115	66699	64740	1959	2,94
116	66668	64768	1900	2,85
117	66638	64711	1927	2,89
118	66659	64766	1893	2,84
119	66637	64754	1883	2,83
120	66666	64736	1930	2,90
121	66616	64771	1845	2,77
122	66683	64795	1888	2,83
123	168417	166578	1839	1,09
124	182356	180420	1936	1,06
Total	42652191	35125890	234965	0.55

Tabla A.3. Ciclos de ejecución reducidos con la optimización de MSR.

Prueba	Ciclos prueba original	Ciclos prueba optimizada	Ciclos reducidos	Porcentaje de ciclos reducidos
1	510547	509466	1081	0,21
2	510533	509407	1126	0,22
3	172733	171622	1111	0,64
4	172721	171641	1080	0,63
5	634562	633477	1085	0,17
6	634535	633465	1070	0,17
7	84196	83067	1129	1,34
8	54205	53127	1078	1,99
9	90612	89534	1078	1,19
10	51106	50056	1050	2,05
11	218549	217465	1084	0,50
12	465559	464470	1089	0,23
13	180638	179539	1099	0,61
14	186856	185734	1122	0,60
15	186891	185797	1094	0,59
16	130365	129210	1155	0,89
17	130362	129264	1098	0,84
18	414541	413433	1108	0,27
19	946985	945894	1091	0,12
20	753379	752211	1168	0,16

Prueba	Ciclos prueba original	Ciclos prueba optimizada	Ciclos reducidos	Porcentaje de ciclos reducidos
21	753304	752269	1035	0,14
22	753347	752237	1110	0,15
23	802594	801432	1162	0,14
24	718985	717805	1180	0,16
25	521559	520485	1074	0,21
26	521519	520423	1096	0,21
27	521529	520407	1122	0,22
28	521509	520442	1067	0,20
29	485850	484773	1077	0,22
30	485822	484701	1121	0,23
31	485837	484722	1115	0,23
32	523842	522787	1055	0,20
33	523807	522711	1096	0,21
34	523881	522792	1089	0,21
35	69062	67949	1113	1,61
36	68244	67121	1123	1,65
37	65437	64382	1055	1,61
38	919862	918739	1123	0,12
39	523815	522772	1043	0,20
40	662057	660928	1129	0,17
41	660367	659261	1106	0,17
42	658454	657321	1133	0,17
43	657632	656559	1073	0,16
44	656184	655074	1110	0,17
45	664210	663148	1062	0,16
46	659710	658613	1097	0,17
47	668486	667303	1183	0,18
48	665197	664053	1144	0,17
49	659152	658039	1113	0,17
50	662457	661391	1066	0,16
51	665837	664731	1106	0,17
52	656955	655805	1150	0,18
53	90345	89240	1105	1,22
54	90328	89251	1077	1,19
55	90358	89247	1111	1,23
56	90320	89237	1083	1,20
57	90387	89267	1120	1,24
58	90383	89258	1125	1,24
59	90371	89206	1165	1,29
60	90363	89274	1089	1,21
61	90367	89215	1152	1,27

Prueba	Ciclos prueba original	Ciclos prueba optimizada	Ciclos reducidos	Porcentaje de ciclos reducidos
62	391815	390709	1106	0,28
63	515663	514575	1088	0,21
64	233419	232347	1072	0,46
65	460081	458928	1153	0,25
66	688941	687890	1051	0,15
67	594635	593510	1125	0,19
68	536835	535767	1068	0,20
69	657812	656770	1042	0,16
70	444447	443308	1139	0,26
71	524363	523293	1070	0,20
72	745824	744722	1102	0,15
73	578790	577627	1163	0,20
74	455350	454238	1112	0,24
75	640496	639315	1181	0,18
76	517090	515961	1129	0,22
77	578767	577687	1080	0,19
78	640484	639394	1090	0,17
79	578770	577673	1097	0,19
80	640451	639336	1115	0,17
81	42287	41197	1090	2,58
82	42227	41155	1072	2,54
83	42248	41102	1146	2,71
84	42238	41144	1094	2,59
85	42207	41129	1078	2,55
86	42276	41175	1101	2,60
87	42287	41141	1146	2,71
88	42214	41158	1056	2,50
89	42290	41198	1092	2,58
90	80985	79887	1098	1,36
91	80998	79827	1171	1,45
92	80943	79876	1067	1,32
93	80925	79889	1036	1,28
94	80987	79842	1145	1,41
95	80911	79837	1074	1,33
96	80952	79868	1084	1,34
97	80989	79838	1151	1,42
98	80911	79843	1068	1,32
99	714303	713256	1047	0,15
100	714325	713291	1034	0,14
101	714384	713243	1141	0,16
102	159771	158628	1143	0,72

Prueba	Ciclos prueba original	Ciclos prueba optimizada	Ciclos reducidos	Porcentaje de ciclos reducidos
103	155578	154459	1119	0,72
104	232850	231726	1124	0,48
105	232832	231779	1053	0,45
106	175715	175715	0	0,00
107	175719	175719	0	0,00
108	175736	175736	0	0,00
109	175707	175707	0	0,00
110	175721	175721	0	0,00
111	175755	175755	0	0,00
112	175735	175735	0	0,00
113	175743	175743	0	0,00
114	175770	175770	0	0,00
115	66699	65579	1120	1,68
116	66668	65546	1122	1,68
117	66638	65580	1058	1,59
118	66659	65542	1117	1,68
119	66637	65507	1130	1,70
120	66666	65595	1071	1,61
121	66616	65503	1113	1,67
122	66683	65501	1182	1,77
123	168417	167343	1074	0,64
124	182356	181282	1074	0,59
Total	42652191	42525366	126825	0,29

Tabla A.4. Ciclos de ejecución reducidos con la optimización de interrupciones.

Prueba	Ciclos prueba original	Ciclos prueba optimizada	Ciclos reducidos	Porcentaje de ciclos reducidos
1	510547	510547	0	0,00
2	510533	510533	0	0,00
3	172733	172733	0	0,00
4	172721	172721	0	0,00
5	634562	629674	4888	0,77
6	634535	629643	4892	0,77
7	84196	79290	4906	5,83
8	54205	49355	4850	8,95
9	90612	85751	4861	5,36
10	51106	46224	4882	9,55
11	218549	213653	4896	2,24
12	465559	460615	4944	1,06

Prueba	Ciclos prueba original	Ciclos prueba optimizada	Ciclos reducidos	Porcentaje de ciclos reducidos
13	180638	175791	4847	2,68
14	186856	181978	4878	2,61
15	186891	181964	4927	2,64
16	130365	125420	4945	3,79
17	130362	125458	4904	3,76
18	414541	409689	4852	1,17
19	946985	942020	4965	0,52
20	753379	748442	4937	0,66
21	753304	748494	4810	0,64
22	753347	748454	4893	0,65
23	802594	797699	4895	0,61
24	718985	714045	4940	0,69
25	521559	516611	4948	0,95
26	521519	516606	4913	0,94
27	521529	516676	4853	0,93
28	521509	516699	4810	0,92
29	485850	480987	4863	1,00
30	485822	480972	4850	1,00
31	485837	480975	4862	1,00
32	523842	518953	4889	0,93
33	523807	518960	4847	0,93
34	523881	518966	4915	0,94
35	69062	64196	4866	7,05
36	68244	63301	4943	7,24
37	65437	60539	4898	7,49
38	919862	914977	4885	0,53
39	523815	518960	4855	0,93
40	662057	662057	0	0,00
41	660367	660367	0	0,00
42	658454	658454	0	0,00
43	657632	657632	0	0,00
44	656184	656184	0	0,00
45	664210	664210	0	0,00
46	659710	659710	0	0,00
47	668486	668486	0	0,00
48	665197	665197	0	0,00
49	659152	659152	0	0,00
50	662457	662457	0	0,00
51	665837	665837	0	0,00
52	656955	656955	0	0,00
53	90345	85409	4936	5,46

Prueba	Ciclos prueba original	Ciclos prueba optimizada	Ciclos reducidos	Porcentaje de ciclos reducidos
54	90328	85442	4886	5,41
55	90358	85416	4942	5,47
56	90320	85485	4835	5,35
57	90387	85462	4925	5,45
58	90383	85405	4978	5,51
59	90371	85419	4952	5,48
60	90363	85491	4872	5,39
61	90367	85457	4910	5,43
62	391815	386922	4893	1,25
63	515663	510761	4902	0,95
64	233419	228595	4824	2,07
65	460081	455159	4922	1,07
66	688941	684030	4911	0,71
67	594635	589738	4897	0,82
68	536835	531976	4859	0,91
69	657812	652977	4835	0,74
70	444447	439539	4908	1,10
71	524363	519467	4896	0,93
72	745824	740970	4854	0,65
73	578790	573890	4900	0,85
74	455350	450429	4921	1,08
75	640496	635542	4954	0,77
76	517090	512145	4945	0,96
77	578767	573806	4961	0,86
78	640484	635554	4930	0,77
79	578770	573819	4951	0,86
80	640451	635583	4868	0,76
81	42287	37304	4983	11,78
82	42227	37354	4873	11,54
83	42248	37343	4905	11,61
84	42238	37357	4881	11,56
85	42207	37321	4886	11,58
86	42276	37303	4973	11,76
87	42287	37327	4960	11,73
88	42214	37392	4822	11,42
89	42290	37329	4961	11,73
90	80985	76050	4935	6,09
91	80998	76029	4969	6,13
92	80943	76011	4932	6,09
93	80925	76054	4871	6,02
94	80987	76088	4899	6,05

Prueba	Ciclos prueba original	Ciclos prueba optimizada	Ciclos reducidos	Porcentaje de ciclos reducidos
95	80911	76034	4877	6,03
96	80952	76058	4894	6,05
97	80989	76084	4905	6,06
98	80911	76074	4837	5,98
99	714303	709432	4871	0,68
100	714325	709404	4921	0,69
101	714384	709498	4886	0,68
102	159771	154848	4923	3,08
103	155578	150606	4972	3,20
104	232850	232850	0	0,00
105	232832	232832	0	0,00
106	175715	170807	4908	2,79
107	175719	170804	4915	2,80
108	175736	170806	4930	2,81
109	175707	170880	4827	2,75
110	175721	170828	4893	2,78
111	175755	170846	4909	2,79
112	175735	170862	4873	2,77
113	175743	170894	4849	2,76
114	175770	170824	4946	2,81
115	66699	61789	4910	7,36
116	66668	61775	4893	7,34
117	66638	61732	4906	7,36
118	66659	61739	4920	7,38
119	66637	61762	4875	7,32
120	66666	61770	4896	7,34
121	66616	61765	4851	7,28
122	66683	61767	4916	7,37
123	168417	163562	4855	2,88
124	182356	177474	4882	2,68
Total	42652191	42137825	514366	1,21

Tabla A.5. Ciclos de ejecución reducidos con la optimización de registros generales.

Prueba	Ciclos prueba original	Ciclos prueba optimizada	Ciclos reducidos	Porcentaje de ciclos reducido
1	510547	505527	5020	0,98
2	510533	505525	5008	0,98
3	172733	165777	6956	4,03
4	172721	165729	6992	4,05

Prueba	Ciclos prueba original	Ciclos prueba optimizada	Ciclos reducidos	Porcentaje de ciclos reducido
5	634562	483876	150686	23,75
6	634535	483865	150670	23,74
7	84196	64144	20052	23,82
8	54205	30786	23419	43,20
9	90612	43504	47108	51,99
10	51106	28633	22473	43,97
11	218549	166585	51964	23,78
12	465559	315715	149844	32,19
13	180638	174937	5701	3,16
14	186856	178761	8095	4,33
15	186891	177885	9006	4,82
16	130365	126615	3750	2,88
17	130362	126642	3720	2,85
18	414541	342594	71947	17,36
19	946985	774877	172108	18,17
20	753379	617654	135725	18,02
21	753304	617685	135619	18,00
22	753347	617620	135727	18,02
23	802594	656603	145991	18,19
24	718985	584549	134436	18,70
25	521559	400836	120723	23,15
26	521519	400871	120648	23,13
27	521529	401021	120508	23,11
28	521509	399444	122065	23,41
29	485850	442214	43636	8,98
30	485822	440749	45073	9,28
31	485837	440782	45055	9,27
32	523842	401787	122055	23,30
33	523807	401737	122070	23,30
34	523881	401706	122175	23,32
35	69062	66039	3023	4,38
36	68244	65985	2259	3,31
37	65437	61591	3846	5,88
38	919862	680857	239005	25,98
39	523815	401727	122088	23,31
40	662057	639096	22961	3,47
41	660367	638349	22018	3,33
42	658454	634446	24008	3,65
43	657632	633649	23983	3,65
44	656184	633161	23023	3,51
45	664210	640751	23459	3,53

Prueba	Ciclos prueba original	Ciclos prueba optimizada	Ciclos reducidos	Porcentaje de ciclos reducido
46	659710	636779	22931	3,48
47	668486	644939	23547	3,52
48	665197	641640	23557	3,54
49	659152	637671	21481	3,26
50	662457	639974	22483	3,39
51	665837	642891	22946	3,45
52	656955	633923	23032	3,51
53	90345	73554	16791	18,59
54	90328	73520	16808	18,61
55	90358	73531	16827	18,62
56	90320	73511	16809	18,61
57	90387	73593	16794	18,58
58	90383	73556	16827	18,62
59	90371	73562	16809	18,60
60	90363	73509	16854	18,65
61	90367	73519	16848	18,64
62	391815	136082	255733	65,27
63	515663	432829	82834	16,06
64	233419	222964	10455	4,48
65	460081	359035	101046	21,96
66	688941	534648	154293	22,40
67	594635	468627	126008	21,19
68	536835	420309	116526	21,71
69	657812	555258	102554	15,59
70	444447	344609	99838	22,46
71	524363	427653	96710	18,44
72	745824	606225	139599	18,72
73	578790	452468	126322	21,83
74	455350	357951	97399	21,39
75	640496	500973	139523	21,78
76	517090	405421	111669	21,60
77	578767	451951	126816	21,91
78	640484	499942	140542	21,94
79	578770	452955	125815	21,74
80	640451	499948	140503	21,94
81	42287	40880	1407	3,33
82	42227	40882	1345	3,19
83	42248	40891	1357	3,21
84	42238	40811	1427	3,38
85	42207	40821	1386	3,28
86	42276	40824	1452	3,43

Prueba	Ciclos prueba original	Ciclos prueba optimizada	Ciclos reducidos	Porcentaje de ciclos reducido
87	42287	40846	1441	3,41
88	42214	40886	1328	3,15
89	42290	40884	1406	3,32
90	80985	77814	3171	3,92
91	80998	77899	3099	3,83
92	80943	77892	3051	3,77
93	80925	77815	3110	3,84
94	80987	77869	3118	3,85
95	80911	77810	3101	3,83
96	80952	77822	3130	3,87
97	80989	77806	3183	3,93
98	80911	77805	3106	3,84
99	714303	584566	129737	18,16
100	714325	584559	129766	18,17
101	714384	584538	129846	18,18
102	159771	144562	15209	9,52
103	155578	110865	44713	28,74
104	232850	224319	8531	3,66
105	232832	224366	8466	3,64
106	175715	175715	0	0,00
107	175719	175719	0	0,00
108	175736	175736	0	0,00
109	175707	175707	0	0,00
110	175721	175721	0	0,00
111	175755	175755	0	0,00
112	175735	175735	0	0,00
113	175743	175743	0	0,00
114	175770	175770	0	0,00
115	66699	64321	2378	3,57
116	66668	64388	2280	3,42
117	66638	64347	2291	3,44
118	66659	64322	2337	3,51
119	66637	64383	2254	3,38
120	66666	64366	2300	3,45
121	66616	64376	2240	3,36
122	66683	64348	2335	3,50
123	168417	166175	2242	1,33
124	182356	179125	3231	1,77
Total	42652191	36420190	6232001	14,61

Tabla A.5. Ciclos de ejecución reducidos con la optimización de APIC de 4 core.

Prueba	Ciclos prueba original	Ciclos prueba optimizada	Ciclos reducidos	Porcentaje de ciclos reducido
1	510547	488567	21980	4,31
2	510533	488533	22000	4,31
3	172733	150750	21983	12,73
4	172721	150737	21984	12,73
5	634562	634562	0	0,00
6	634535	634535	0	0,00
7	84196	84196	0	0,00
8	54205	54205	0	0,00
9	90612	90612	0	0,00
10	51106	51106	0	0,00
11	218549	218549	0	0,00
12	465559	465559	0	0,00
13	180638	180638	0	0,00
14	186856	186856	0	0,00
15	186891	186891	0	0,00
16	130365	130365	0	0,00
17	130362	130362	0	0,00
18	414541	414541	0	0,00
19	946985	946985	0	0,00
20	753379	753379	0	0,00
21	753304	753304	0	0,00
22	753347	753347	0	0,00
23	802594	802594	0	0,00
24	718985	718985	0	0,00
25	521559	521559	0	0,00
26	521519	521519	0	0,00
27	521529	521529	0	0,00
28	521509	521509	0	0,00
29	485850	485850	0	0,00
30	485822	485822	0	0,00
31	485837	485837	0	0,00
32	523842	523842	0	0,00
33	523807	523807	0	0,00
34	523881	523881	0	0,00
35	69062	69062	0	0,00
36	68244	68244	0	0,00
37	65437	65437	0	0,00
38	919862	919862	0	0,00
39	523815	523815	0	0,00

Prueba	Ciclos prueba original	Ciclos prueba optimizada	Ciclos reducidos	Porcentaje de ciclos reducido
40	662057	640014	22043	3,33
41	660367	638365	22002	3,33
42	658454	636449	22005	3,34
43	657632	635542	22090	3,36
44	656184	634152	22032	3,36
45	664210	642257	21953	3,31
46	659710	637727	21983	3,33
47	668486	646454	22032	3,30
48	665197	643193	22004	3,31
49	659152	637114	22038	3,34
50	662457	640462	21995	3,32
51	665837	643816	22021	3,31
52	656955	634929	22026	3,35
53	90345	90345	0	0,00
54	90328	90328	0	0,00
55	90358	90358	0	0,00
56	90320	90320	0	0,00
57	90387	90387	0	0,00
58	90383	90383	0	0,00
59	90371	90371	0	0,00
60	90363	90363	0	0,00
61	90367	90367	0	0,00
62	391815	391815	0	0,00
63	515663	515663	0	0,00
64	233419	233419	0	0,00
65	460081	460081	0	0,00
66	688941	688941	0	0,00
67	594635	594635	0	0,00
68	536835	536835	0	0,00
69	657812	657812	0	0,00
70	444447	444447	0	0,00
71	524363	524363	0	0,00
72	745824	745824	0	0,00
73	578790	578790	0	0,00
74	455350	455350	0	0,00
75	640496	640496	0	0,00
76	517090	517090	0	0,00
77	578767	578767	0	0,00
78	640484	640484	0	0,00
79	578770	578770	0	0,00
80	640451	640451	0	0,00

Prueba	Ciclos prueba original	Ciclos prueba optimizada	Ciclos reducidos	Porcentaje de ciclos reducido
81	42287	42287	0	0,00
82	42227	42227	0	0,00
83	42248	42248	0	0,00
84	42238	42238	0	0,00
85	42207	42207	0	0,00
86	42276	42276	0	0,00
87	42287	42287	0	0,00
88	42214	42214	0	0,00
89	42290	42290	0	0,00
90	80985	80985	0	0,00
91	80998	80998	0	0,00
92	80943	80943	0	0,00
93	80925	80925	0	0,00
94	80987	80987	0	0,00
95	80911	80911	0	0,00
96	80952	80952	0	0,00
97	80989	80989	0	0,00
98	80911	80911	0	0,00
99	714303	714303	0	0,00
100	714325	714325	0	0,00
101	714384	714384	0	0,00
102	159771	159771	0	0,00
103	155578	155578	0	0,00
104	232850	210850	22000	9,45
105	232832	210859	21973	9,44
106	175715	175715	0	0,00
107	175719	175719	0	0,00
108	175736	175736	0	0,00
109	175707	175707	0	0,00
110	175721	175721	0	0,00
111	175755	175755	0	0,00
112	175735	175735	0	0,00
113	175743	175743	0	0,00
114	175770	175770	0	0,00
115	66699	66699	0	0,00
116	66668	66668	0	0,00
117	66638	66638	0	0,00
118	66659	66659	0	0,00
119	66637	66637	0	0,00
120	66666	66666	0	0,00
121	66616	66616	0	0,00

Prueba	Ciclos prueba original	Ciclos prueba optimizada	Ciclos reducidos	Porcentaje de ciclos reducido
122	66683	66683	0	0,00
123	168417	168417	0	0,00
124	182356	182356	0	0,00
Total	42652191	42234047	418144	0,98