Herramienta de perfilado para obtención de tiempos de ejecución en un procesador basado en RISC V

Arturo Salas Delgado
Área de Ingeniería en Computadores
Tecnológico de Costa Rica
Cartago, Costa Rica
Email: artsaldel@tec.ac.cr

***Resumen*—En este artículo se muestra el proceso de implementación y los resultados de una herramienta de perfilado para la obtención de tiempos de ejecución sobre un microprocesador con arquitectura *pipeline* y multiciclo basada en RISC-V. Esta herramienta tiene la capacidad de soportar distintos escenarios, donde se tendrá que ejecutar diferentes programas escritos en C estándar con funciones aritméticas simples, usando o no instrucciones de punto flotante, crear su similar en código binario y verificar su correcto funcionamiento en el procesador basado en RISC V que estará implementado sobre una placa de desarrollo de *hardware*, en este caso, Zedboard o Nexys 3.**

Palabras clave: RISC-V, ISA, perfilado, punto flotante, pipeline, multiciclo, placa de desarrollo, *speed down*.

Introducción

Investigadores y estudiantes de la escuela de Ingeniería Electrónica del Tecnológico de Costa Rica, junto con las áreas de Ingeniería en Computadores e Ingeniería en Mecatrónica, colaboran con científicos del Departamento de Neurociencia del Centro Médico Erasmus, en Rotterdam, Países Bajos, para el desarrollo de dispositivos electrónicos que apoyen la creación de sustitutos para zonas dañadas del cerebro humano.

Es por esto que los ingenieros involucrados trabajan para la resolución de múltiples problemas en la utilización de dispositivos de *hardware* que emulen el proceso cognitivo. Se utilizan FPGAs (*Field Programmable Gate Array*) para la creación de distintos bloques de *hardware*, de manera personalizada, para la realización de tareas complejas. De acá surge la necesidad de la creación de un nuevo procesador basado en la arquitectura RISC-V (*Reduced Instruction Set Computer*) para la ejecución de programas escritos en código C estándar.

Consigo, los diseños de arquitecturas de procesadores y los ISAs (*Instruction Set Architecture*) en su mayoría son de carácter privativo y es necesario de un pago monetario para poder ser utilizados. RISC-V (pronunciado 'risk-five') es un ISA abierto y gratuito que permite una nueva era de innovación de procesadores a través de la colaboración estándar abierta. Nacido en la academia y la investigación, RISC-V ISA ofrece un nuevo nivel de libertad de *software* y *hardware* extensible y gratuito en arquitectura, allanando el camino para los próximos 50 años de diseño e innovación informática [1]. Su creación empezó en el año 2010 como una iniciativa *Open Source* que consistía en el desarrollo de un ISA, para que la persona que lo desee lo pueda utilizar con fines ingenieriles.

El conjunto de Instrucciones de RISC-V a pesar de ser *Open Source*, tener herramientas para la compilación de código y varios simuladores, no

cuenta con *software* libre para obtener estadísticas o perfilados sobre algún procesador con dicha arquitectura, es por esto que se ve en la necesidad de emplear conocimientos ingenieriles para el desarrollo de esta nueva herramienta de perfilado basada en la arquitectura RISC-V.
En general, la herramienta de perfilado busca brindar una opción sencilla para la generación de estadísticas, específicamente tiempos de ejecución, para una microarquitectura predefinida y para brindar una mejora en su rendimiento al momento de utilizar un conjunto de instrucciones RISC V.

Descripción de la solución

La herramienta diseñada consta de tres etapas: la generación de código binario a partir de un programa escrito en C estándar, la optimización de ese código binario mediante la utilización de una biblioteca de aproximación de flotantes y por último, la generación de estadísticas sobre un procesador basado en la arquitectura RISC-V ejecutando ese programa, específicamente tiempos de ejecución. Se explica en detalle cada etapa a continuación:

- **Generación de código binario basado en RISC V:** El usuario tendrá un código escrito en C estándar, con instrucciones aritméticas básicas como sumas, restas, multiplicaciones y divisiones. De ahí, la herramienta se encarga de la generación de código binario basado en el set de instrucciones de RISC V, versión RV32I, a partir del código en C estándar escrito por el usuario.

- **Optimización de código binario:** La utilización de instrucciones en punto flotante para la arquitectura RISC V puede traer consigo inconvenientes de compilación. Es por esto que mediante el uso de una biblioteca llamada *softfloat* el *toolchain* de RISC-V aplica una estrategia de soporte de flotantes mediante una aproximación por enteros utilizando rutinas especializadas para esta transformación.

- **Generación de estadísticas:** Una vez que se tiene el código binario generado, se pretende que este sea ejecutado por un procesador predefinido basado en la arquitectura RISC V. Es aquí donde la herramienta se encarga de la colección de datos, como tiempos de ejecución, para ser mostrados al usuario y poder obtener conclusiones cuantitativas sobre el proceso.

En la figura 1 se muestra el flujo de trabajo de la herramienta, incluyendo las etapas mencionadas anteriormente:

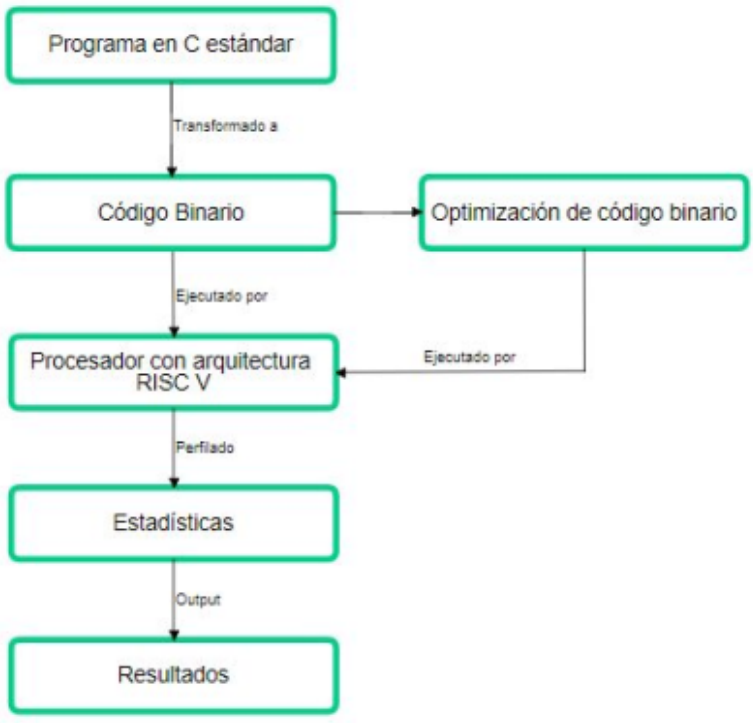


Figura 1. Flujo de trabajo del script de la herramienta de perfilado

Además, es importante mencionar que se usó la versión 2.0 de RISC-V por las siguientes razones:

- El ISA es dividido en una base de enteros con varias extensiones del estándar, comparado con versiones anteriores. [2]
- Los formatos de instrucción están reorganizados para hacer que la codificación inmediata sea más eficiente. [2]
- La base del ISA se ha definido para tener un sistema de memoria *little-endian*, con *big-endian* o *bi-endian* como variantes no estándar. [2]

Análisis de Resultados

Una vez completado el diseño e implementación de la herramienta de perfilado, se obtiene una serie de archivos de texto. Estos archivos son explicados a continuación:

Archivo ejecutable: Este archivo representa al código binario que puede ser ejecutado por un procesador basado en el set de instrucciones de RISC-v versión RV32I.

Archivo de tiempos de ejecución: Este archivo de salida representa a la simulación ejecutada por el usuario, donde se muestra un dato aproximado de la ejecución del programa ingresada en la arquitectura seleccionada (*pipeline* o multiciclo).

Archivo de configuración de arquitectura *pipeline*: Este archivo representa la configuración de la arquitectura *pipeline* sobre el cuál se realiza la simulación del programa ingresado por el usuario. Los datos contenidos en este archivo son:

- **Tipo:** Representa el tipo de arquitectura sobre el cuál se realiza la simulación, para este caso el valor siempre es ”pipeline”.
- **Versión:** Este dato representa la versión de RISC-V sobre la cuál se realiza la simulación.
- **CPI:** Es la medida del rendimiento de un procesador, en este caso, el número medio de ciclos de reloj por instrucción en un programa o fragmento sobre la arquitectura *pipeline*. Es el inverso multiplicativo de las instrucciones por ciclo.
- **Frecuencia:** Es la frecuencia sobre el cuál opera el procesador *pipeline* en el cuál se realiza la simulación.

Archivo de configuración de arquitectura multiciclo: Este archivo representa la configuración de la arquitectura multiciclo sobre el cuál se realiza la simulación del programa ingresado por el usuario. Los datos contenidos en este archivo son:

- **Tipo:** Representa el tipo de arquitectura sobre el cuál se realiza la simulación, para este caso el valor siempre es ”multicycle”.
- **Versión:** Este dato representa la versión de RISC-V sobre el cuál se realiza la simulación.
- **CPI Carga:** Son los ciclos por instrucción para las instrucciones de carga a memoria pertenecientes al ISA de RISC-V.
- **CPI Almacenamiento:** Son los ciclos por instrucción para las instrucciones de almacenamiento en memoria pertenecientes al ISA de RISC-V.
- **CPI Aritmética:** Son los ciclos por instrucción para las instrucciones aritméticas pertenecientes al ISA de RISC-V.
- **CPI Control Flujo:** Son los ciclos por instrucción para las instrucciones de control de flujo pertenecientes al ISA de RISC-V.
- **CPI Llamada al Sistema:** Son los ciclos por instrucción para las instrucciones que hacen llamadas al sistema pertenecientes al ISA de RISC-V.
- **Frecuencia:** Es la frecuencia sobre el cuál opera el procesador multiciclo en el cuál se realiza la simulación.

Se realizó una serie de pruebas donde se tiene una configuración *pipeline* tal como se muestra en la figura 2, y una configuración multiciclo tal como se muestra en la figura 3.

Prueba 1: En esta prueba se simula el programa en C mostrado en la figura 4, donde se tiene

```
Tipo=pipeline
Version=rv32i
CPI=1
Frecuencia(hz)=50000000
```

Figura 2. Configuración para la arquitectura pipeline

```
Tipo=multicycle
Version=rv32i
CPI Carga=5
CPI Almacenamiento=5
CPI Aritmética=4
CPI Control Flujo=3
CPI Llamada al sistema=3
Frecuencia(hz)=50000000
```

Figura 3. Configuración para la arquitectura multiciclo

una serie de instrucciones aritméticas de punto flotante y de punto fijo. Una vez realizada la simulación en una arquitectura *pipeline* y en una arquitectura multiciclo se obtiene un tiempo de ejecución de 0.044ms y 0.15ms respectivamente.

```c
#include <stdio.h>

int SumInt(int,int);
int SubInt(int,int);
int MultInt(int,int);
int DivInt(int,int);

float SumFloat(float,float);
float SubFloat(float,float);
float MultFloat(float,float);
float DivFloat(float,float);

void main()
{
    int sumInt = SumInt(1, 2);
    int multInt = MultInt(2, 3);
    int subInt = SubInt(5, 2);
    int divInt = DivInt(3, 3);

    float sumFloat = SumFloat(2.3, 1.7);
    float subFloat = SubFloat(9.2, 2.1);
    float multFloat = MultFloat(2.9, 3.1);
    float divFloat = DivFloat(2.1, 6.2);

    for( int i = 0 ; i < 10; i++)
    {
        SumInt(1,i);
    }
}

int SumInt(int a, int b){ return a + b; }
float SumFloat(float a, float b){ return a + b; }
int SubInt(int a, int b){ return a - b; }
float SubFloat(float a, float b){ return a - b; }
int MultInt(int a, int b){ return a * b; }
float MultFloat(float a, float b){ return a * b; }
int DivInt(int a, int b){ return a / b; }
float DivFloat(float a, float b){ return a / b; }
```

Figura 4. Programa en C para primera prueba

Prueba 2: En esta prueba se simula el programa en C mostrado en la figura 5 sobre una arquitectura *pipeline*, el cual consta de un algoritmo con un *for* anidado dentro de otro, usando punto fijo o punto flotante para sus variables, donde se va incrementando el valor de N para aumentar el número de ciclos. Los resultados de la simulación se muestran en la figura 6 y graficados en la figura 7.

```c
int N = 5;

void IcNeighbors(float VD[N],float IC[N],float CONN[N][N])
{
    float V = 0;//////////////////////
    int i,j;
    float Facc = 0;//////////////////////
    float Vacc = 0;//////////////////////
    float f = 0;//////////////////////
    for(i=0;i<N;i++)
    {
        for(j=0;j<N;j++)
        {
            V = VD[i]-VD[j];
            f = V*34.0;
            Facc = Facc + f*CONN[i][j];
            Vacc = Vacc + V*CONN[i][j];
        }
        IC[i] = 20.0*Facc + 80.0*Vacc ;
    }
}

int main()
{
    float VD[N];
    float IC[N];
    float CONN[N][N];
    IcNeighbors(VD, IC, CONN);
}
```

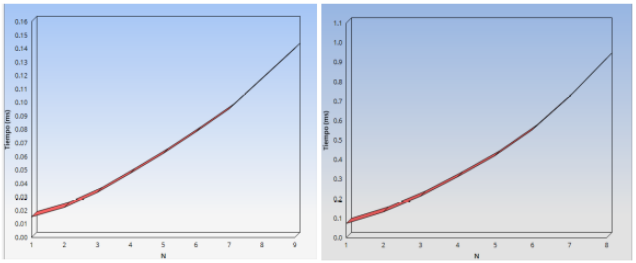
Figura 5. Programa en C para segunda prueba

N	Tiempo de ejecución - Punto flotante (ms)	Tiempo de ejecución - Punto fijo (ms)
1	0.031	0.015
2	0.071	0.022
3	0.131	0.033
4	0.211	0.047
5	0.311	0.062
6	0.419	0.078
7	0.550	0.095
8	0.719	0.117
9	0.918	0.140

Figura 6. Tabla de resultados de la simulación sobre una arquitectura *pipeline*

Prueba 3: En esta prueba se simula el programa en C mostrado en la figura 5 sobre una arquitectura multiciclo, mismo programa usado en la prueba 2. Los resultados de la simulación se muestran en la figura 8 y graficados en la figura 9.

En las pruebas anteriores, realizando simulaciones sobre dos diferentes arquitecturas basadas en RISC-V (*pipeline* y multiciclo) con una misma frecuencia de operación, en este caso una frecuencia de 50 MHz, se puede comprobar que los



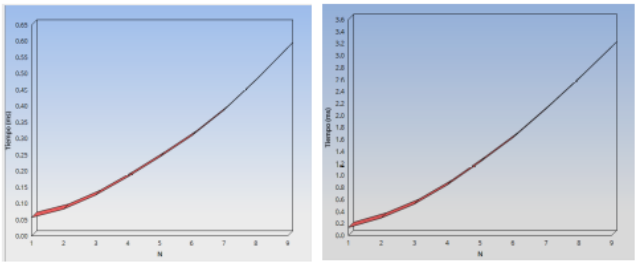
Punto fijo Punto flotante

Figura 7. Gráficos de resultados de simulación sobre una arquitectura *pipeline*

N	Tiempo de ejecución - Punto flotante (ms)	Tiempo de ejecución - Punto fijo (ms)
1	0.129	0.056
2	0.284	0.082
3	0.518	0.125
4	0.833	0.180
5	1.219	0.243
6	1.624	0.307
7	2.100	0.385
8	2.602	0.478
9	3.137	0.579

Figura 8. Tabla de resultados de la simulación sobre una arquitectura multiciclo

tiempos de ejecución sobre un mismo programa son distintos. En todas las pruebas los tiempos de ejecución en la arquitectura *pipeline* fueron menores que en la arquitectura multiciclo. Se realiza un análisis de *Speed Down* en ambas arquitecturas, específicamente sobre las pruebas 2 y 3; en la figura 10 se muestran los resultados de *Speed Down* usando punto fijo y en la figura 11 se



Punto fijo Punto flotante

Figura 9. Gráficos de resultados de simulación sobre una arquitectura multiciclo

muestran los resultados de *Speed Down* usando punto flotante, mostrados también en el gráfico de la figura 12. Estos datos refieren a que los tiempos de ejecución difieren en una factor igual al *Speed Down* para el mismo programa simulado en las dos arquitecturas, donde se demuestra que la arquitectura *pipeline* es más veloz al ejecutar un mismo programa, determinando un mejor desempeño en cuanto a velocidad de procesamiento.

N	Tiempo Ejecución Multiciclo (ms)	Tiempo ejecución Pipeline (ms)	Speeddown
1	0.056	0.015	3.73
2	0.082	0.022	3.73
3	0.125	0.033	3.78
4	0.180	0.047	3.82
5	0.243	0.062	3.91
6	0.307	0.078	3.93
7	0.385	0.095	4.05
8	0.478	0.117	4.08
9	0.579	0.140	4.14
			AVG = 3.90

Figura 10. Tabla de resultados de *Speed Down* usando punto fijo

N	Tempo Ejecución Multiciclo (ms)	Tiempo ejecución Pipeline (ms)	Speeddown
1	0.129	0.031	4.16
2	0.284	0.071	4.00
3	0.518	0.131	3.95
4	0.833	0.211	3.94
5	1.219	0.311	3.92
6	1.624	0.419	3.86
7	2.100	0.550	3.81
8	2.602	0.719	3.61
9	3.137	0.918	3.42
			AVG = 3.86

Figura 11. Tabla de resultados de *Speed Down* usando punto flotante

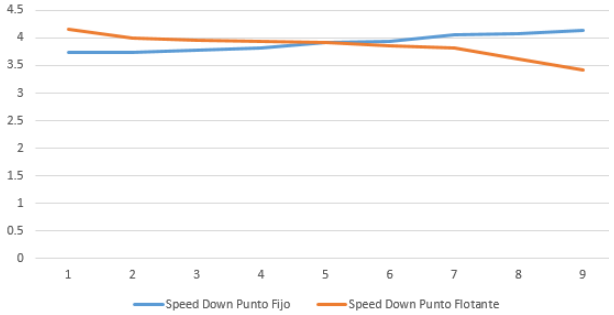


Figura 12. Gráfico de resultados de *Speed Down* usando punto fijo y punto flotante

Conclusiones

Tomando en cuenta los resultados obtenidos, se identifica que la arquitectura *pipeline* basada en RISC-V posee un mayor desempeño en términos de velocidad de procesamiento comparado con la arquitectura multiciclo basada en RISC-V. Esto puede cambiar en caso de realizar variaciones en los archivos de configuración de ambas arquitecturas, por ejemplo, cambiando la frecuencia de operación o cambiando los ciclos por instrucción. Es por esto que la herramienta está creada para realizar especulaciones con criterio dependiendo de las características de la arquitectura de los procesadores.

Con la implementación de esta herramienta se puede tener un análisis cuantitativo de tiempos, por lo que su uso es ideal para hacer supuestos antes de ejecutar un programa en un microprocesador basado en RISC-V implementado en una tarjeta desarrollo, donde los tiempos de síntesis son altos al momento de hacer cambios en la memoria de instrucciones. Además, esta herramienta facilita al usuario entender la arquitectura RISC-V y su set de instrucciones, considerado como un aspecto importante en el ámbito ingenieril.

Las funcionalidades de este sistema pueden ser extendidas, es por esto que es creado como una herramienta *Open Source*. Se puede integrar nuevas arquitecturas de simulación además de *pipeline* y multiciclo, donde solamente sería necesario integrar la configuración de la arquitectura y su cálculo de tiempos de ejecución. Al ser un sistema modular, se considera una posible integración de nuevas arquitecturas a futuro.

Referencias

[1] R. V, «Risc-v isa», https://riscv.org/risc-v-isa/, Accesado: 28-05-2018.

[2] A. Waterman, Y. Lee, D. A. Patterson y K. Asanović, «The risc-v instruction set manual, volume i: User-level isa, version 2.1», reporte técnico UCB/EECS-2016-118, EECS Department, University of California, Berkeley, Mayo 2016.