Introducción

Para ponernos en contexto, el predictor más avanzado que implementamos en la práctica era el gshare. Para intentar mejorarlo, investigué leyendo uno de los libros que se encuentra en la bibliografía de la asignatura: *«Computer Architecture: A quantitative approach» de John L. Hennessy y David A. Patterson.* En internet se encuentra colgada la quinta edición de forma gratuita. A partir de la página 164, y en adelante, encontré un predictor conocido como Tournament Predictor, y las especificaciones correspondientes para poder implementarlo.

Explicación Tournament Predictor

Los predictores de torneo surgieron como respuesta a las limitaciones de los predictores estándar de 2 bits que solo utilizan información local. Este consiste en usar múltiples predictores, uno basado en información global y otro en información local. Además, estos se combinan mediante un selector, implementado como un contador saturado de 2 bits, que elige el predictor más eficaz basado en predicciones recientes. De esta manera, se mejora el rendimiento. No obstante, si ambos predictores aciertan, o fallan, no se hace nada.

Más adelante, en el libro de la bibliografía, se explica cómo fue usado en el procesador Intel Core i7, con un modelo parecido al de la práctica. Así que intenté implementarlo como especificaba el libro, cogiendo como referencia el código escrito en la sesión 2, de la práctica 3.

Implementación

En el libro, se explica lo siguiente:

- El tournament predictor se implementa como un contador saturado de 2 bits
- Un contador local de doble nivel. El primer nivel incluye una tabla para la historia de 1024 con 10 bits de entrada. Cada entrada de 10 bits corresponde a los 10 resultados de bifurcación más recientes de la entrada. Es decir, si la rama se tomó 10 o más veces seguidas, la entrada en la tabla histórica local será todo 1s. Si la rama se toma y no se toma alternativamente, la entrada del historial consistirá en una alternancia de 0s y 1s. Este historial de 10 bits permite descubrir y predecir patrones de hasta 10 ramas. La entrada seleccionada de la tabla de historial local se utiliza para indexar una tabla de 1.000 entradas formada por contadores saturados de 3 bits (segundo nivel), que proporcionan la predicción local
- La historia global
- Un contador global de 2 bits (usamos el gshare de la práctica)

Inicialmente, en la práctica teníamos un contador global implementado con gshare de 2 bits, y una historia global. Así que solo faltaba completar el contador local de 3 bits, su historia local de 10 bits de entrada, y el selector, un contador saturado de 2 bits. Así como sus inicializaciones, predicciones y actualizaciones. En el código C, he añadido comentarios comentando cada sección, pero de manera resumida, los bits 0-1 del selector se los asignamos al contador local y los bits 2-3 al global. Si el contador global toma taken y el salto fue tomado, incrementamos en el selector favoreciendo al global. Lo mismo con el local, pero si ambos fallan o aciertan, no hacemos nada.

Por otro lado, para actualizar la historia local tuve que recurrir a un código que encontré en GitHub, pues no sabía como tratar los 10 bits de entrada. Como pude observar, se usa una máscara de bits para hacerlo. Intenté buscar información acerca de ello, pero no pude encontrar mucho. Más tarde le vi más sentido, pues tenemos que ir actualizando los bit antiguos, pero sigo sin

entender del todo cómo funciona. Observé que si no le restábamos 1, daba error de segmentación al acceder a zonas de memoria prohibidas, por lo que supuse que es una manera de mantener el tamaño a 10 bits constantes.

Resultados

Con el código que realicé, se obtuvieron los siguientes resultados:

	Gshare (Práctica)		**Tournament predictor**	
	lbm	**wrf**	**lbm**	**wrf**
Prediction Accuracy	96.6566%	94.8463%	96.8863%	97.4897%
IPC	0.367326	0.55629	0.367899	0.573505

- **Para la aplicación lbm:**
  - Mejora en la precisión: **0.2297 %**
  - Mejora en el IPC: $\frac{0.367899}{0.367326}$ = 1.001559922 = **0.1559 %**
- **Para la aplicación wrf:**
  - Mejora en la precisión: **2.6434 %**
  - Mejora en el IPC: $\frac{0.573505}{0.55629}$ = 1.030946089 = **3.0946 %**
- **Tamaño del predictor en KB**
  - **Contador global (1024 entradas de 2 bits cada una):** 1024 x 2
  - **Historia global (1 entrada de 64 bits):** 64
  - **Contador local (1024 entradas de 3 bits cada una):** 1024 x 3
  - **Historia local (1024 entradas de 10 bits cada una):** 1024 x 10
  - **Selector de torneo (1024 entradas de 2 bits cada una):** 1024 x 2

  En total = 17472 / (8 * 1024) = **2.1328 KB**

Comparación para todas las aplicaciones:

	gcc	**mcf**	**lbm**	**wrf**	M.**Armónica**
IPC.Tournament	0.122034	0.172378	0.367899	0.573505	0.2167
IPC.Gshare	0.121783	0.172162	0.367326	0.55629	0.21575
Mejora	**0.2061 %**	**0.1254 %**	**0.1559 %**	**3.0946 %**	**0.4403 %**

Reflexión

Al principio, intenté programarlo con un contador local de 2 bits y no tuve en cuenta los 10 bits de entrada en la historia local, como se menciona en el libro. En ese caso conseguí mejorar el wrf. Pero en la aplicación de lbm, empeoraba. Por ello intenté seguir la configuración del libro, a ver si conseguía mejores resultados para esta última

Con la implementación del libro, mejoró más el IPC en general. En realidad es bastante obvio, pues el uso de un contador de 3 bits es mejor que uno de 2 bits, al presentar más estados. Lo mismo al usar los 10 bits de entrada en la historia local, ya que permite descubrir y predecir hasta patrones de 10 ramas.

Mis conclusiones acerca del empeoramiento de lbm con la antigua implementación están relacionadas con que el predictor no estaba bien configurado para la aplicación, y daba más predicciones incorrectas. Asimismo, cuando probaba diferentes funciones hash más sencillas,

mejoraba al mismo porcentaje que se obtuvo en la práctica inicialmente, por lo que no era una mejora significativa.

De todas maneras, hay predictores que funcionan mejor con unos programas y peor con otros. Estoy segura de que hay mejores formas para conseguir resultados óptimos pues, mientras investigaba como hacer la tarea, leí múltiples implementaciones de predictores, e incluso con redes neuronales y machine learning. Esto último fue bastante fascinante de descubrir, pero se requiere un nivel bastante avanzado.

Para finalizar, esta tarea me ha resultado bastante interesante, aunque también un tanto laboriosa de realizar, pues no conseguía mejoras significativas, o tenía problemas en realizar ciertas implementaciones en el programa. También estoy segura de que mi código puede ser mejorado aún más, incluso pienso que me puedo haber equivocado en alguna parte implementándolo. No obstante, creo que se ha podido obtener una buena mejora (sobretodo en wrf), para los pocos conocimientos que tenía.

Bibliografía

- *«Computer Architecture: A quantitative approach 5th edition» de John L. Hennessy y David A. Patterson*
- *https://en.wikipedia.org/wiki/Branch_predictor*
- *https://github.com/ajgupta93/gshare-and-tournament-branch-predictor*