Laboratorio 1: Simulación de microarquitectura

27 de septiembre 2024

1st Camilo Álvarez Muñoz
dept. Ingeniería electrónica
Universidad de Antioquia
Medellín, Colombia
camilo.alvarezm@udea.edu.co

2nd Daniel Cano Restrepo
dept. Ingeniería electrónica
Universidad de Antioquia
Medellín, Colombia
daniel.canor1@udea.edu.co

***Abstract*—This assignment explores the impact of varying microarchitectural parameters on the performance and energy efficiency of a processor. Using the Gem5 simulator, a series of simulations will be performed to study how design choices such as cache size, pipeline width, and reorder buffer (ROB) capacity influence workload performance. McPAT will be used to estimate power consumption, and the Energy-Delay Product (EDP) will be computed to evaluate the trade-off between performance and energy. Scripts will automate the simulation process, facilitating design space exploration (DSE) through brute force and heuristic strategies.**

***Index Terms*—Cache, Energía, Gem5, McPat, Microarquitectura, Parámetros, Rendimiento, Scripts, Simulaciones, Unidades funcionales.**

I. Introducción

En este trabajo se busca explorar cómo los diferentes parámetros microarquitectónicos de un procesador afectan tanto el rendimiento como el consumo de energía. Se utilizó el simulador Gem5, que es ampliamente utilizado en la investigación de microarquitecturas de procesadores. Con este simulador, se realizaron una serie de simulaciones iniciales utilizando los parámetros generales del procesador para obtener un perfil de desempeño del procesador al ejecutar 6 programas distintos. A partir de este perfil, se construyó un espacio de diseño, el cual contiene distintos parámetros del procesador a ser cambiados para analizar su impacto en el rendimiento mediante el simulador, y también, el consumo energético del procesador mediante la herramienta McPAT. Para la exploración del espacio de diseño, se realizó un script el cual implementa un algoritmo de búsqueda metaheurística conocido como Recocido Simulado (Simulated annealing en inglés) inspirado en el proceso de enfriamiento de los metales, donde se permite que el sistema explore soluciones subóptimas al principio (alta temperatura) para evitar quedarse atrapado en óptimos locales, y luego se reduce gradualmente la probabilidad de aceptar peores soluciones (baja temperatura), buscando así una solución óptima global. A través de este algoritmo, exploramos con facilidad las posibles configuraciones del procesador y medimos el desempeño de cada una al ejecutar los programas, para así analizar cuáles son las mejores en términos de rendimiento, energía o EDP.

II. Objetivos

- Comprender y cuantificar el efecto de modificar los principales parámetros microarquitectónicos de un procesador.
- Desarrollar un script para la automatización de las simulaciones de distintas microarquitecturas de procesadores y explorar el espacio de diseño mediante el algoritmo de búsqueda Recocido simulado.
- Realizar análisis de potencia y consumo energético del procesador mediante la herramienta McPAT.

III. Marco teórico

Gem5:

Gem5 es un simulador de sistemas y microarquitecturas de procesadores ampliamente utilizado en la investigación para evaluar el rendimiento y comportamiento energético de microprocesadores. Ofrece una plataforma flexible que permite simular diferentes arquitecturas de CPU, subsistemas de memoria y dispositivos I/O. Gem5 combina simulación a nivel de ciclo con simulación a nivel funcional, lo que permite capturar tanto el rendimiento como el comportamiento detallado de un sistema.

McPAT:

McPAT (Multicore Power, Area, and Timing) es una herramienta que estima el consumo de potencia, el área física y el tiempo de respuesta de procesadores multicore. Utiliza modelos analíticos basados en parámetros arquitectónicos (como el tamaño de caché, ancho de bus, frecuencia de reloj, etc.) para proporcionar una estimación detallada de la energía consumida por los diferentes componentes del procesador.

Primero, a través de Gem5 se generan estadísticas para cada simulación (stats.txt) y parámetros arquitectónicos (config.json). Luego, estos archivos son indicados mediante una línea de comando al programa de python gem5toMcPAT_cortexA76.py suministrado por el profesor, este programa entrega un archivo config.xml. Para ejecutar McPAT, simplemente se le indica al programa la ubicación de este archivo .xml generado por cada simulación y a través de la consola obtendremos un reporte con detalles como la potencia estática, dinámica y el área de los componentes del procesador.

Energy Delay Product:

El Energy-Delay Product (EDP) es una métrica que se utiliza para evaluar la eficiencia energética de un procesador, teniendo en cuenta tanto el consumo de energía como el rendimiento. Se define como el producto entre el consumo total de energía y el tiempo de ejecución o latencia de un sistema. En nuestro caso, utilizando Gem5 y McPAT, se calcula como:

$$EDP = Energy \cdot sytem.cpu.cpi$$

Donde la energía se calcula mediante los resultados de McPAT con la siguiente fórmula: $Energy = (Totalleakage + RuntimeDynamic) \cdot sytem.cpu.cpi$. Un EDP más bajo indica un sistema más eficiente en términos de consumo energético y rendimiento.

Recocido Simulado:

El recocido simulado (*simulated annealing*) es un algoritmo de optimización inspirado en el proceso de enfriamiento de metales. Emula cómo los átomos se ordenan al enfriar lentamente un material para minimizar su energía y lograr una estructura estable. En optimización, este método permite buscar soluciones en un espacio amplio, aceptando soluciones peores al inicio (cuando la "temperatura" es alta) para evitar quedar atrapado en óptimos locales y gradualmente afinando la búsqueda hacia el óptimo global. [1]

- **Temperatura inicial** (T_0): valor de temperatura que controla la probabilidad de aceptar soluciones peores en las primeras iteraciones.
- **Factor de enfriamiento** (α): determina la reducción de la temperatura en cada iteración, con $\alpha < 1$.
- **Número de iteraciones por temperatura** (n): cantidad de intentos por cada nivel de temperatura antes de reducirla.
- **Criterio de parada**: condición para finalizar el proceso, como alcanzar una temperatura mínima o un número máximo de iteraciones.

IV. Procedimiento

Con el fin de entender el funcionamiento del simulador Gem5, se realizaron unas 6 simulaciones del procesador con sus parámetros generales, ejecutando cada uno de los programas (h264_dec,h264_enc, jpg2k_dec, jpg2k_enc, mp3_dec y mp3_enc) propuestos en la carpeta Workloads suministrada por el profesor. El fin de estas simulaciones es analizar la información entregada por Gem5 en el archivo stats.txt una vez finalizada la simulación del procesador para así seleccionar ciertos parámetros de la microarquitectura del procesador y cuantificar su impacto en la eficiencia y consumo energético del procesador.

En el archivo stats.txt de cada simulación, podemos encontrar información muy importante como el IPC (Instrucciones por ciclo), el CPI (ciclos por instrucción) y el IC (Intruction count o cantidad de instrucciones). Con estas métricas podemos medir de forma sencilla el desempeño del procesador, ya sea observando que al aplicar un cambio en la microarquitectura el CPI del procesador aumente o que el ICP reduzca, o también, calculando el "CPU time" que relaciona todas estas métricas en una misma ecuación. Aunque Gem5 nos entregue todas estas métricas generales, también nos da otra información detallada como la cantidad de fallos del "Branch predictor", el porcentaje de predicciones erróneas de la memoria cache, y el porcentaje de los distintos tipos de instrucciones ejecutadas por el procesador.

Dado que analizar a detalle todas las estadísticas del archivo stats.txt puede resultar laborioso, una estrategia útil es centrarse en los tipos de instrucciones ejecutadas. En el archivo podemos encontrar la métrica "system.cpu.commit.committedInstType_0::" que nos dice el porcentaje y el conteo de todas las instrucciones que fueron ejecutadas por el procesador según las diferentes clases instrucciones que hay. También, con la métrica "system.cpu.statFuBusy::" podemos conocer la cantidad de veces en las que se intentó ejecutar un tipo de instrucción en específico y no se encontraban disponibles unidades funcionales para poder procesar la instrucción. Por ejemplo, podríamos observar que un 45% de las instrucciones que ejecutó el procesador fueron de tipo punto flotante (float) y que un 52% de las veces, las unidades funcionales se encontraban ocupadas, generando un cuello de botella en el procesador, ya que las instrucciones float son un gran porcentaje de las operaciones que se realizan, por esta razón sería lógico añadir más unidades funcionales para las instrucciones tipo float y analizar si la simulación entrega mejores resultados o no.

Entonces, con el fin de realizar un perfil de clases de instrucciones de cada programa, se realizó la simulación de los 6 programas con los parámetros por defecto del procesador. Se observaron los siguientes resultados:

Para el programa h264_dec:

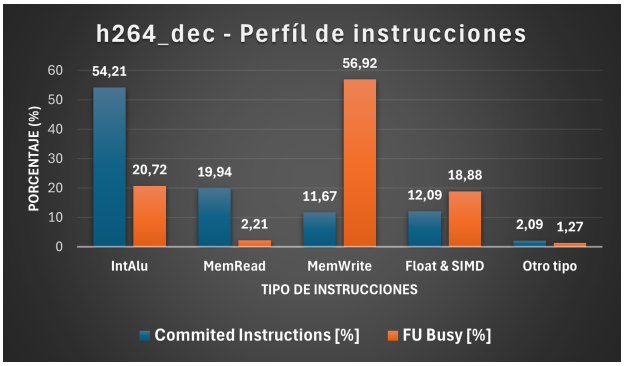

Fig. 1. Perfil de instrucciones programa h264_dec.

Para el programa h264_enc:

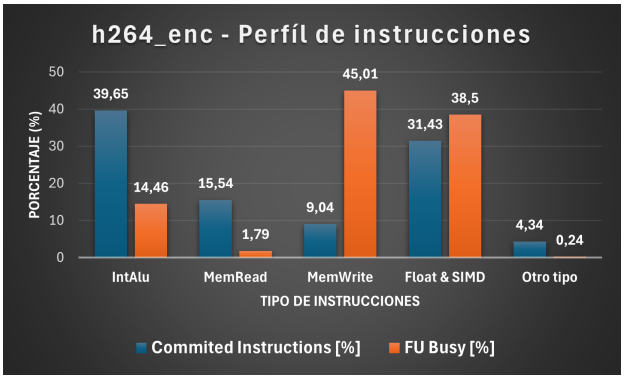

Fig. 2. Perfil de instrucciones programa h264_enc.

Para el programa jpeg2k_dec:

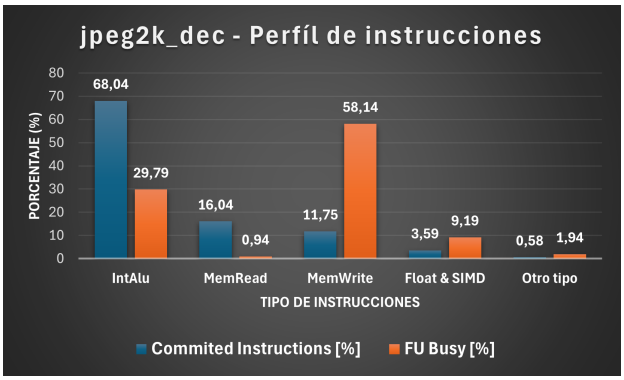

Fig. 3. Perfil de instrucciones programa jpeg2k_dec.

Para el programa jpeg2k_enc:

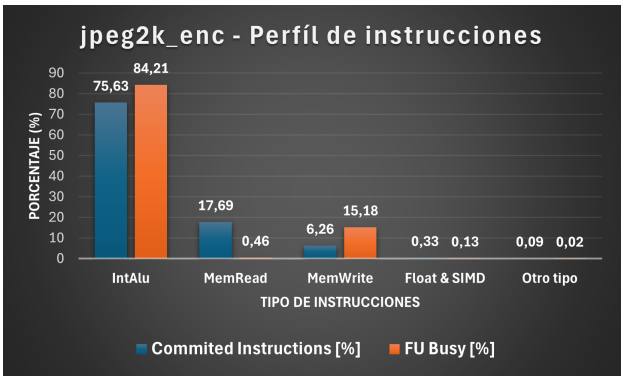

Fig. 4. Perfil de instrucciones programa jpeg2k_enc.

Para el programa mp3_dec:

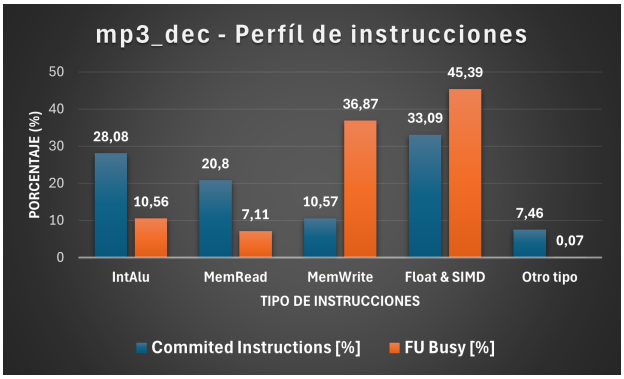

Fig. 5. Perfil de instrucciones programa mp3_dec.

Para el programa mp3_enc:

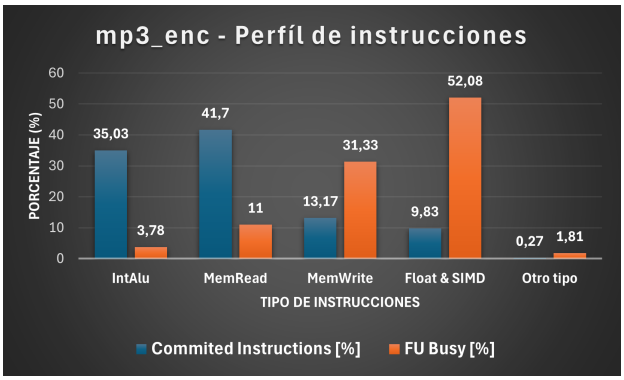

Fig. 6. Perfil de instrucciones programa mp3_enc.

En todos los programas el tipo de instrucciones que más ejecuto el procesador son: IntAlu (enteros), MemRead (lecturas de memoria), MemWrite (Guardado de memoria) y float_simd (punto flotante y "Una instrucción, múltiples datos", las cuales comparten las mismas unidades funcionales). Para cada tipo de instrucciones se observa una barra de color azul (Commited Instructions), que representa el porcentaje total de instrucciones de ese tipo que fueron ejecutadas por el procesador, y en la barra de color naranja observamos el porcentaje de veces que las unidades funcionales (UF o FU en inglés) para ese tipo se encontraron ocupadas.

En general, las instrucciones de tipo IntAlu y MemWrite tienden a tener una alta proporción de instrucciones ejecutadas en todos los programas, siendo más pronunciado en jpeg2k_enc (75.63%) y jpeg2k_dec (68.04%). Por otro lado, se observa una alta ocupación de las unidades de punto flotante (Float & SIMD): en h264_enc (38.5%), mp3_dec (45.39%) y mp3_enc (52.08%). A excepción del caso de jpeg2k_enc, que muestra una mínima actividad en Float & SIMD (0.33%). Las instrucciones enteras (IntAlu) están presentes tanto en cantidad de instrucciones como en ocupación de las UF en todos los programas, superando en todos los casos más del 28% de las instrucciones.

Programa	Instruction Count	CPI	IPC	CPUtime	Potencia	EDP
h264_dec	261123615	0.9389	1.0650	0.1167	2.9291	2.7502
h264_enc	314287521	1.0093	0.9907	0.1510	2.6414	2.6661
jpg2k$_dec$	388181042	1.1969	0.8354	0.2212	4.0201	4.8117
jpg2k$_enc$	230470462	0.9201	1.0867	0.1009	2.3766	2.1868
mp3$_dec$	230472847	0.8976	1.1140	0.0985	2.3470	2.1067
mp3$_enc$	238353249	1.1998	0.8334	0.1361	2.3960	2.874
Promedio:	**277148122.7**	**1.0271**	**0.9875**	**0.1374**	**2.7850**	**2.8994**

TABLE I
RESULTADOS DE SIMULACIONES CON PARÁMETROS GENERALES.

La demanda de diferentes tipos de instrucciones varía según la naturaleza del programa, pero, gracias a este análisis, podemos empezar a definir nuestro espacio de diseño, ya que vale la pena minimizar los casos donde las UF de los distintos tipos de instrucciones se encuentran ocupadas y observar si esto tiene impacto en el desempeño del procesador. Por lo tanto, se explorará el desempeño del procesador variando la cantidad de unidades funcionales que tienen las instrucciones de tipo IntAlU, Load, Store y floating-point SIMD.

Además, se seleccionaron otros parámetros clave, como el tamaño y la asociatividad de las cachés de instrucciones y datos, para explorar cómo afectan el rendimiento y la eficiencia energética. El tamaño de las cachés influye en la tasa de fallos, afectando la latencia de acceso a la memoria y el número de accesos a la memoria principal, mientras que la asociatividad impacta la tasa de conflictos. Al probar diferentes combinaciones de estos parámetros, buscamos identificar configuraciones óptimas que minimicen los fallos de caché y maximicen el rendimiento del sistema, aunque esto pueda implicar un mayor consumo energético.

Parámetro	Descripción	Valores		
`--num_fu_intALU`	# de UF para instrucciones enteros ALU	2	4	6
`--num_fu_read`	# de UF para instrucciones de lectura	2	4	6
`--num_fu_write`	# de UF para instrucciones de escritura	2	4	6
`--num_fu_FP_SIMD_ALU`	# de UF para instrucciones de punto flotante y SIMD	1	2	4
`--l1i_size`	Tamaño de la caché de instrucciones L1	64kB	128kB	256kB
`--l1i_assoc`	Asociatividad de la cache de instrucciones L1	4	8	16
`--l1d_size`	Tamaño de la cache de datos L1	64kB	128kB	256kB
`--l1d_assoc`	Asociatividad de la cache de datos L1	4	8	16

TABLE II
ESPACIO DE DISEÑO ELEGIDO.

Esto corresponde a $3^8 = 6561$ combinaciones. Debido a que realizar las simulaciones de los seis programas simultáneamente toma alrededor de 22 minutos, no es factible analizar todas las posibilidades mediante fuerza bruta. Por lo tanto, la exploración de las microarquitecturas de procesadores en el espacio de diseño se llevó a cabo utilizando una técnica de optimización conocida como recocido simulado. Esta técnica permite analizar menos combinaciones, pero acercándonos a una microarquitectura de procesador que proporcione una mejor compensación entre rendimiento y consumo energético, evaluada mediante el producto energía-retardo (EDP).

En el código, primero se definen parámetros como el factor de enfriamiento y la temperatura inicial, que establece la probabilidad de aceptar configuraciones de procesador con un menor EDP al inicio del proceso. A medida que avanza la simulación, esta temperatura disminuye (enfriamiento), reduciendo la probabilidad de aceptar configuraciones subóptimas. Luego, se especifica la cantidad de configuraciones a iterar y analizar.

El proceso detallado inicia con la primera iteración del programa, donde se define una configuración aleatoria del procesador, por ejemplo: "`--num_fu_intALU`=2 `--num_fu_read`=4, `--num_fu_write`=2, `--num_fu_FP_SIMD_ALU`=4, `--l1i_size`=64kB, `--l1i_assoc`=16, `--l1d_size`=128kB, y `--l1d_assoc`=4". Luego, con la función "eval_state", se inicia en Gem5 la ejecución simultánea de las seis simulaciones del procesador utilizando esta configuración. Al finalizar las seis simulaciones, se iteran las carpetas que contienen los resultados (stats.txt) de cada programa, extrayendo el IPC, CPI e Instruction Count. Con estos valores se calcula el CPU_Time, que se escribe en el archivo "Results.CSV".

A continuación, se ejecuta la función "mcpat()", que analiza los archivos de salida de cada programa: stats.txt y config.json, generando un archivo .xml. Este archivo es procesado por McPAT para calcular el área del procesador, la potencia dinámica, la potencia de fuga y otra información relevante sobre el consumo energético del procesador simulado. De estos resultados, se extraen los valores de "Runtime Dynamic" y "Total Leakage", permitiendo calcular el producto energía-retardo (EDP) de la siguiente forma:

$$Energy = (TotalLeakage + RuntimeDynamic) \cdot CPI$$

$$EDP = Energy \cdot CPI$$

Estos valores de Energía y EDP también se guardan en el archivo "Results.CSV" para cada simulación. Luego, se almacena en el archivo "AvgResults.csv" el promedio de las estadísticas de las seis simulaciones. Este promedio de EDP es nuestra función objetivo, pues se utiliza para comparar si una configuración en el espacio de diseño es superior a otra.

Tras la extracción y el almacenamiento de resultados, la temperatura se reduce de acuerdo con el factor de enfriamiento, y se genera una nueva configuración mediante la función "generar_nuevo_punto", que modifica aleatoriamente un parámetro de la configuración inicial. Se inicia una nueva ronda de simulaciones en Gem5 con esta configuración y, al completarse, se obtienen los datos de rendimiento y consumo energético como en la iteración anterior. La técnica de recocido simulado evalúa si esta nueva configuración es

mejor que la anterior en términos de la función objetivo (menor EDP). Si es mejor, se actualiza el mejor estado; en caso contrario, se toma una decisión probabilística basada en la temperatura actual para aceptar o rechazar el cambio, permitiendo escapar de mínimos locales y mejorar la exploración en las primeras etapas de la optimización.

Este proceso se repite hasta que la temperatura desciende a un umbral predefinido, indicando el final de las iteraciones. Al concluir todas las rondas, el programa muestra cuál de todas las configuraciones del espacio de diseño obtuvo el mejor EDP. Además, el archivo "AvgResults.csv" mostrará el desempeño promedio de todas las iteraciones, y si se requiere información detallada de una configuración específica del procesador ejecutando un programa, se podrá analizar el archivo "Results.CSV".

V. Resultados y análisis de resultados

Para realizar la correcta exploración del espacio de diseño, se ejecutó el programa en 4 equipos diferentes (Nos referiremos a estos como Equipo 1, Equipo 2, Equipo 3 y Equipo 4) donde cada uno realizo 45, 50, 50 y 55 iteraciones respectivamente. Cada equipo empezó desde un estado de parámetros diferente, y esperamos observar como el recocido simulado intenta encontrar la microarquitectura de mejor EDP en cada equipo.

Para ejecutar el experimento, primero se realizaron 15 iteraciones al azar y se calculó la desviación estándar del EDP. Esto permitió definir una temperatura inicial con una probabilidad de cambio del 90%, utilizando dos veces la desviación estándar. La decadencia de la temperatura fue del 95% por iteración, y cada temperatura se evaluó solo una vez, debido al tiempo prolongado que requiere ejecutar la función objetivo (entre 20 y 30 minutos).

Para el Equipo 1 el algoritmo fue interrumpido a las 45 iteraciones, pero para los 3 Equipos restantes se completaron satisfactoriamente las iteraciones planteadas (50, 50 y 55). Una vez terminadas, el recocido simulado finalizó y entregó los siguientes resultados:

Para el Equipo 2:
La mejor configuración hallada fue: `--num_fu_intALU=6` `--num_fu_read=6`, `--num_fu_write=2`, `--num_fu_FP_SIMD_ALU=1`, `--l1i_size=64kB`, `--l1i_assoc=8`, `--l1d_size=128kB`, y `--l1d_assoc=8`". Con un EDP de 3,2427.

Para el Equipo 3:
La mejor configuración hallada fue: `--num_fu_intALU=4` `--num_fu_read=4`, `--num_fu_write=4`, `--num_fu_FP_SIMD_ALU=1`, `--l1i_size=256kB`, `--l1i_assoc=16`, `--l1d_size=256kB`, y `--l1d_assoc=8`". Con un EDP de 3,3503.

Para el Equipo 4:
La mejor configuración hallada fue: `--num_fu_intALU=4` `--num_fu_read=4`, `--num_fu_write=4`, `--num_fu_FP_SIMD_ALU=2`, `--l1i_size=128kB`, `--l1i_assoc=16`, `--l1d_size=256kB`, y `--l1d_assoc=4`". Con un EDP de 3,1883.

Es importante señalar que la función de cambio de estado ajusta los índices de la lista con una distancia de Hamming promedio de 1.3. Los resultados de este experimento se muestran en la Figura 7.

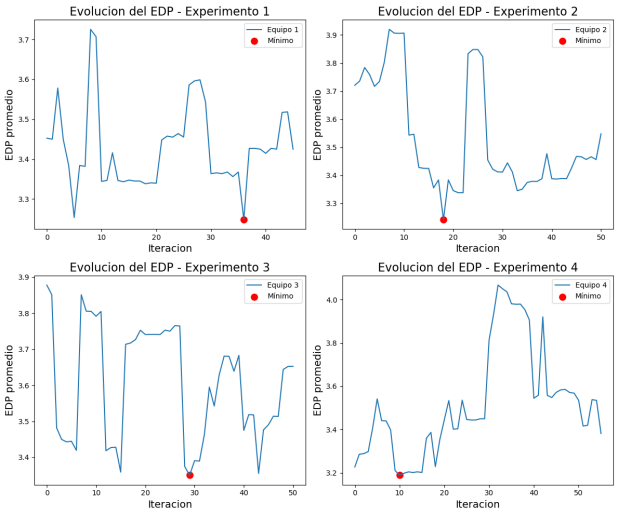

Fig. 7. Progreso de los experimentos.

El comportamiento de las gráficas resulta algo caótico debido a la aleatoriedad inherente de este método de optimización. Además, es importante destacar que para que esta heurística sea efectiva, los cambios en el estado deben provocar variaciones pequeñas en la función objetivo. Sin embargo, en múltiples ocasiones, un solo cambio a alguno de los parámetros del procesador generó variaciones grandes en la función objetivo (EDP), provocando cierta inconsistencia en los resultados. A pesar de esto, el algoritmo encontró un mínimo local en el EDP.

También se evaluó cómo estos cambios afectaban cada programa específico. Para simplificar el análisis, se realizó una correlación que mide el impacto de optimizar el promedio sobre cada programa individual, obteniendo los siguientes resultados:

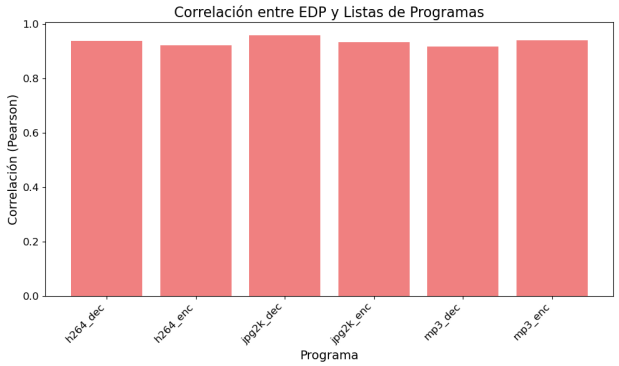

Fig. 8. Correlación de los programas con el EDP promedio.

Esto indica que los parámetros modificados en el experimento influyen de manera similar en todos los programas, en términos prácticos.

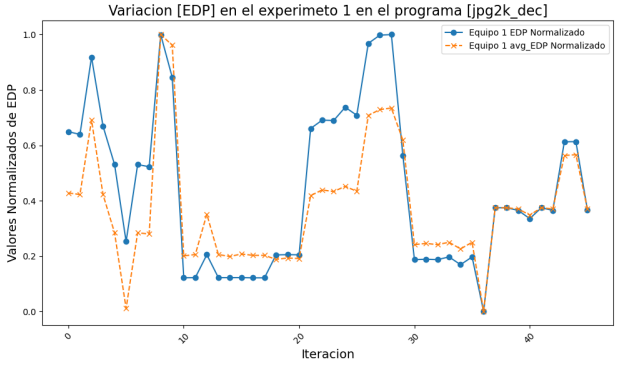

Fig. 9. Variación de EDP.

Como se muestra en la Figura 9, a medida que cambia el promedio de los resultados, los programas siguen un comportamiento similar.

Con estos resultados, se generó un espacio de compromiso (*trade-off*) en el que se examinan la rapidez (CPI) y la energía consumida por instrucción. La Figura 10 muestra los casi 200 experimentos realizados y sus distintos resultados.

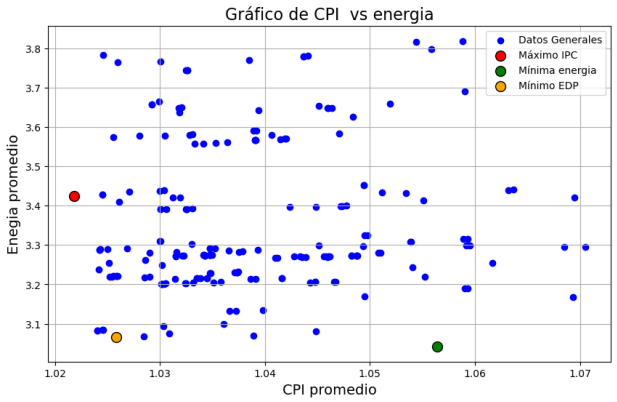

Fig. 10. Espacio de resultados.

No se observa un patrón claro en este espacio. Aunque cabría esperar que un aumento en el consumo energético se refleje en un incremento del rendimiento, los resultados muestran un comportamiento caótico. Además, los cambios en el CPI son mínimos, lo que sugiere que ajustar estos parámetros no mejora significativamente el rendimiento del procesador, aunque pueden ser de interés en otros contextos.

Cuando se encontró el ”mínimo” EDP, se corrió la simulación de nuevo con una temperatura mucho más baja para que el algoritmo se asemejara más a un algoritmo greedy. Esto se hizo para determinar si este punto es un óptimo local o si se tuvo mala suerte al alejarse de ese resultado. Se encontró lo siguiente:

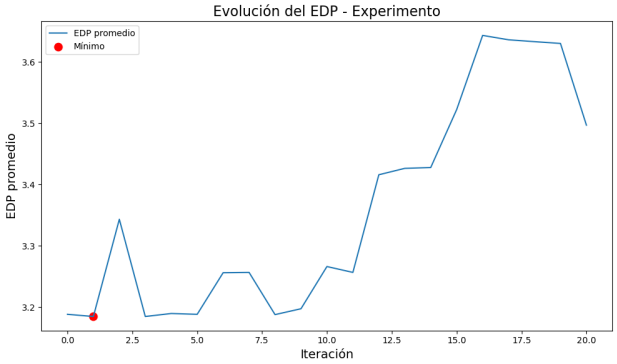

Fig. 11. Localidad mínima EDP.

Se encontró un valor más pequeño que el anterior, aunque no por mucho, y debido a su comportamiento, este debería ser un mínimo local, lo cual indica el buen funcionamiento del algoritmo. Este punto no se explora en este informe, ya que el experimento se realizó únicamente para verificar si el mínimo de EDP era un mínimo local, lo que suele ser el objetivo de este tipo de algoritmos.

Finalmente, analizando las 200 iteraciones de configuraciones del procesador, encontramos que las siguientes fueron

las que presentaron mayor IPC, menor consumo energético y menor EDP:

Mejores configuraciones	Parámetros			
	`--num_fu_intALU=`	`--num_fu_read=`	`--num_fu_write=`	`--num_fu_FP_SIMD_ALU=`
	4	4	2	4
	`--l1i_size=`	`--l1i_assoc=`	`--l1d_size=`	`--l1d_assoc=`
Mayor IPC = 0.996336	256kB	16	256kB	8
	`--num_fu_intALU=`	`--num_fu_read=`	`--num_fu_write=`	`--num_fu_FP_SIMD_ALU=`
	4	2	4	2
	`--l1i_size=`	`--l1i_assoc=`	`--l1d_size=`	`--l1d_assoc=`
Menor Energía = 3.042321	64kB	4	64kB	4
	`--num_fu_intALU=`	`--num_fu_read=`	`--num_fu_write=`	`--num_fu_FP_SIMD_ALU=`
	6	4	2	4
	`--l1i_size=`	`--l1i_assoc=`	`--l1d_size=`	`--l1d_assoc=`
Menor EDP = 3.188300	128kB	4	256kB	4

TABLE III
MEJORES CONFIGURACIONES EN RENDIMIENTO, ENERGÍA Y EDP.

Se obtuvo un mayor IPC (0.996336) en una configuración que usó una memoria cache de datos e instrucciones de 256kB y de alta asociatividad entre ambas. Y como era de esperarse, la configuración que usa menos energía es aquella en que ambas memorias solo tienen un tamaño de 64kB. Al aumentar la complejidad de la memoria, aumenta el consumo energético del procesador.

Cuando realizamos el perfilado de instrucciones, obtuvimos un CPU Time promedio de 0.1374 segundo, en cambio, cuando realizamos el recocido simulado obtuvimos un tiempo de 0.144157 segundos. Esto se debe principalmente a los cambios en los parámetros del procesador que afectan la cantidad de recursos y la estructura de la memoria caché. Ya que a comparación de los parámetros generales, para nuestra mejor configuración aumentamos considerablemente las unidades funcionales de 3 clases de instrucciones, aumentando el paralelismo (generando un mayor rendimiento), pero también aumentando el consumo energético. También se aumentó el tamaño de la caché L1 de instrucciones y datos, lo que puede implicar un tiempo de acceso ligeramente mayor debido a la mayor complejidad de administración de una memoria caché más grande, aunque también se esperaría que esto redujera las fallas de caché. Algo importante a mencionar, durante el perfilado de instrucciones, utilizamos como Branch Predictor un predictor Bimodal (de un funcionamiento más sencillo), y, en cambio, en los experimentos realizados con el recocido utilizamos un predictor de "Tournament", el cual añade una mayor complejidad al procesador a costo de reducir las predicciones erroneas.

Todos los resultados obtenidos, así como los programas de Python desarrollados, pueden ser encontrados en el repositorio: https://github.com/camiloam20/simulation-gem5

VI. CONCLUSIONES

- **Desafíos en la Optimización del Procesador**: Dada la cantidad de parámetros que se pueden ajustar en el procesador y el tiempo requerido para realizar simulaciones de múltiples programas, es necesario el uso de heurísticas, lo cual evidencia la dificultad de encontrar el óptimo en este tipo de problemas.

- **Complejidad de la Función Objetivo**: Los cambios abruptos que se observan en la figura 7 indican que el EDP es algo inestable con los cambios realizados (a pesar de ser pequeños), lo cual implica que la heurística puede tender a fallar, lo que hace que hallar el óptimo sea una tarea complicada.

- **Parámetros de la microarquitectura:** En nuestro análisis, exploramos el desempeño del procesador modificando las propiedades de su memoria caché L1 y su cantidad de unidades funcionales. Esto no significa que hayan sido los únicos parámetros a modificar; con Gem5 también podemos realizar cambios a las memorias L2 y L3, o realizar cambios al ancho de instrucciones que puede recibir una etapa del pipeline, o también, modificar el predictor de saltos. Alterando estos parámetros del procesador podríamos llegar a obtener un mejor EDP, pero esto a costo de necesitar explorar espacio de diseño más grande.

- **Correlación entre Programas**: Aunque los parámetros se seleccionaron para mejorar todos los programas a la vez, una correlación superior al 90% sugiere que los programas tienden a seguir patrones similares. Esto implica que optimizar uno puede llevar a la optimización de otro. No se sostiene que el hardware especializado no tenga utilidad; más bien, se considera que un procesador de propósito general puede ser viable en este contexto.

- **Energía vs. Rendimiento**: Dado que no se observó una relación clara entre la energía y el rendimiento, pensamos que esto puede deberse a que elegimos incorrectamente los parámetros o a que la variación es tan pequeña que la relación no se hace evidente, como si hubiéramos hecho tanto zoom que los detalles de ruido no nos permitieran verla.

REFERENCES

[1] Algoritmo de Recocido Simulado (2024) Wikipedia. Available at: https://es.wikipedia.org/wiki/Algoritmo_de_recocido_simulado (Accessed: 27 October 2024).