Informe de Diseño - Proyecto 3

Sebastián Quesada Rojas, *estudiante,* Ingeniería en Computadores, *Instituto Tecnológico de Costa Rica*
Mario Carranza Castillo, *estudiante,* Ingeniería en Computadores, *Instituto Tecnológico de Costa Rica*
Repositorio:*https://github.com/sebasqr22/Proyecto-3-Arqui-1*

1 Identificación de las variables

Para las iteraciones que se realizaron, se cambiaron algunos parámetros del archivo *runGEM5.sh*. Para el cambio de los CPUs, el parámetro **–cpu-type**, en este se seleccionaron los siguiente:

- MinorCPU: Se utilizó este CPU ya que permite una rápida iteración de experimentos y ajustes.
- NonCachingSimpleCPU: Se utilizó este modelo de CPU ya que no posee memoria caché y nos fue interesante ver de cerca su funcionamiento.

Para el caso del cambio en el branch predictor, se cambió el parámetro **–bp-type**, donde las opciones son:

- BiModeBP
- TournamentBP
- LocalBP
- LTAGE
- TAGE

Para el caso de la caché, se cambiaron dos parámetros. Se cambió el tamaño de la caché 1d y de la caché 1i. Para eso se cambiaron los parámetros **–l1d_size** y **–l1d_size**. Los valores que se utilizaron son:

- 2kB y 2kB
- 8kB y 8kB
- 16kB y 16kB
- 128kB y 128kB
- 256kB y 256kB

Para las políticas de reemplazo se cambió el parámetro **–replacement_policy**. Los valores utilizados fueron:

- lru
- fifo
- random
- mru
- lfu

2 Benchmarks

Para el caso del SPEC, se decidió utilizar el benchmark se decidió utilizar el **458.sjeng**. Este benchmark fue inspirado en el programa de ajedrez llamado sjeng y programado en C. Este es bastante útil pues se utiliza para poder medir el rendimiento de aplicaciones que tienen una carga muy intensa de cálculos y búsquedas. [4]

Para el caso del PARSEC, se decidió utilizar el **blackscholes**. Este benchmark es de gran utilidad, pues está diseñado para para aprovechar al máximo el paralelismo. Este mismo tiene funciones logarítmicas y exponenciales, por lo que se utiliza el punto flotante de alta precisión. Está programado en C y C++ y su principal objetivo es poder evaluar una arquitectura y su comportamiento con operaciones pesadas de puntos flotantes. [4]

3 Solución Ingenieril

Se comenzará por utilizar el sistema operativo Ubuntu 18.04.6 LTS. Luego se procederá a instalar GEM5. Para la realización de este proyecto, se decidió por utilizar ARM y RISC-V, esto debido al conocimiento previo que se tenía así como la facilidad de los mismos.

Como se explicó en el punto [2], se utilizaron dos benchmarks de SPEC y otro de PARSEC. Se comenzó con la instalación de SPEC, de donde se seleccionó el 458.sjeng. Se configuró el Makefile con el compilador de ARM y de RISC-V respectivamente. Luego en el archivo de configuración **runGEM5.sh** se definieron los parámetros a utilizar. Este mismo proceso se hizo en el benchmark de PARSEC utilizando blackscholes.

Las iteraciones se definieron primero cambiando 3 parámetros; estos de tamaño de caché, políticas de reemplazo y tipo de branch predictor. Por cada uno de estos se hicieron 5 cambios. Luego, se procedió a hacer las mismas iteraciones pero cambiando el tipo de CPU, los cuales se mencionan en el punto [1].

Para visualizar los resultados obtenidos, se creó un script de python utilizando Matplotlib. De esta manera, el script determina los valores que se quieren utilizar, los extrae del archivo *stats.txt* (archivo con datos de salida que brida GEM5) y crea los gráficos correspondientes.

4 Evaluación de los benchmarks utilizados y resultados

4.1 RISC-V

4.1.1 PARSEC

Se comenzó analizando el PARSEC. Utilizando el MinorCPU, para el cambio de branches se obtuvo:

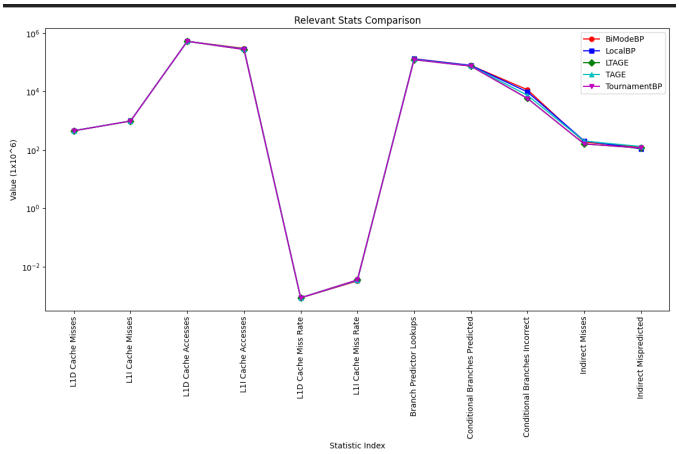

Fig. 1. Gráfica de RISC-V, PARSEC, MinorCPU, cambio de branch predictor

Obteniendo como mejor resultado el TAGE.

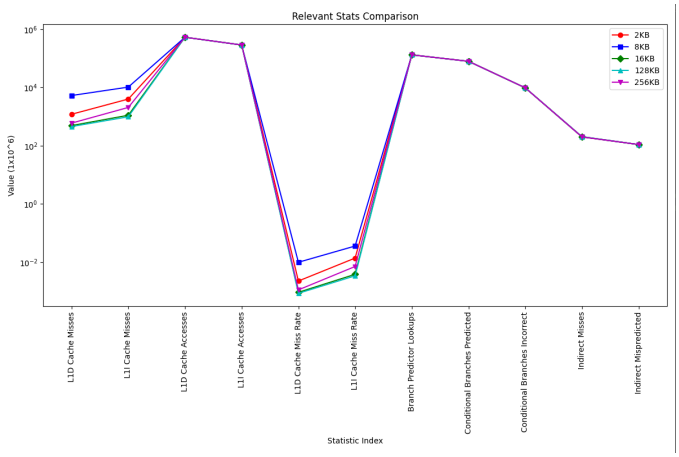

Fig. 2. Gráfica de RISC-V, PARSEC, MinorCPU, cambio de caché size

Cambiando el tamaño de la caché, se obtuvo que el mejor resultado fue utilizando 256kb.

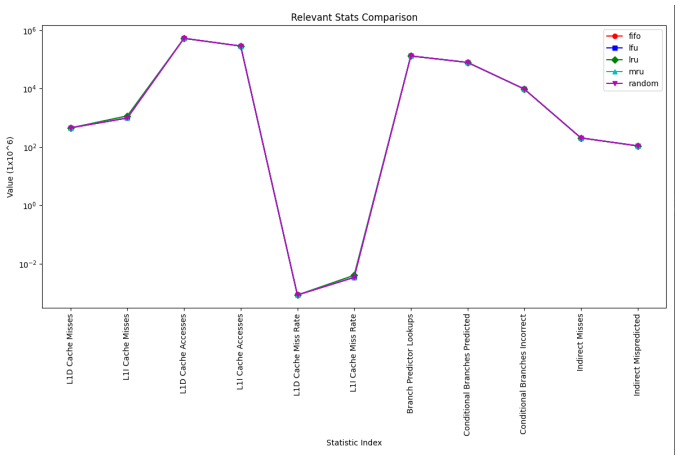

Fig. 3. Gráfica de RISC-V, PARSEC, MinorCPU, cambio de políticas de reemplazo

De esta manera, se obtiene que la peor fue el Random y la mejor fue LFU.

Utilizando NonCachingSimpleCPU se obtuvo:

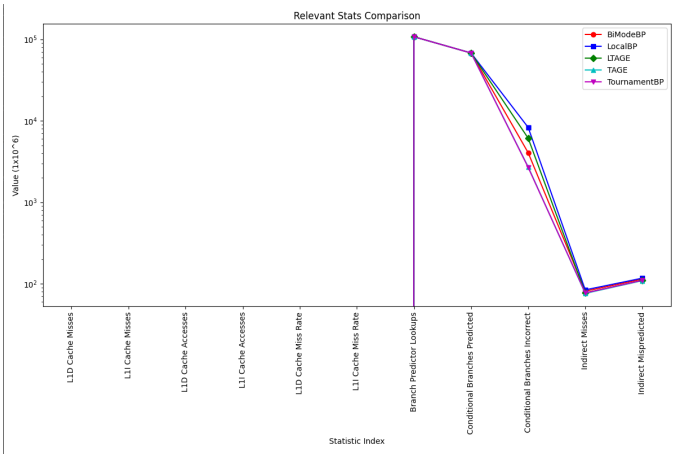

Fig. 4. Gráfica de RISC-V, PARSEC, NonCachingSimpleCPU, cambio de branch predictor

Donde se obtuvo que el mejor también fue el TAGE.

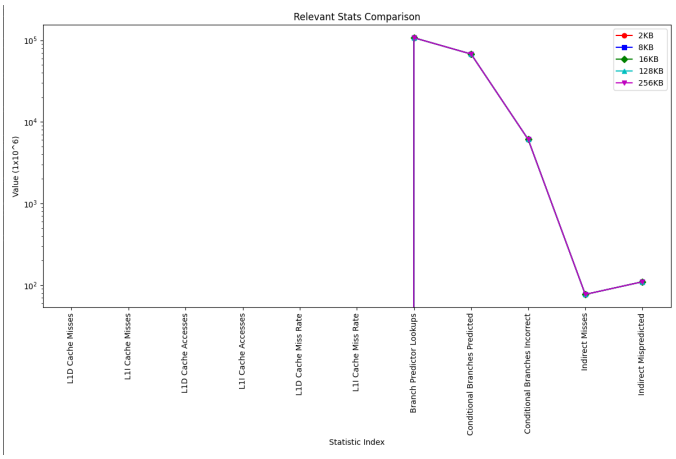

Fig. 5. Gráfica de RISC-V, PARSEC, NonCachingSimpleCPU, cambio de caché size

Aquí se observa que el mejor resultado fue utilizando 256kb.

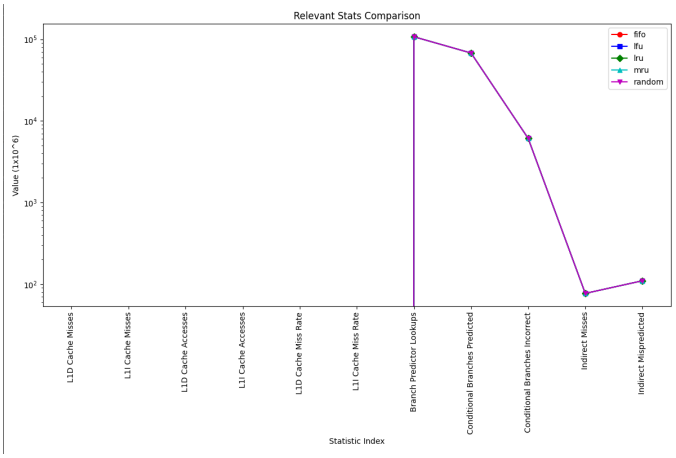

Fig. 6. Gráfica de RISC-V, PARSEC, NonCachingSimpleCPU, cambio de políticas de reemplazo

4.1.2 Iteraciones en SPEC

Para MinorCPU:

Para NonCachingSimpleCPU:

4.2 ARM

4.2.1 PARSEC

Para MinorCPU:

Para NonCachingSimpleCPU:

4.3 SPEC:

Para MinorCPU:

Para NonCachingSimpleCPU:

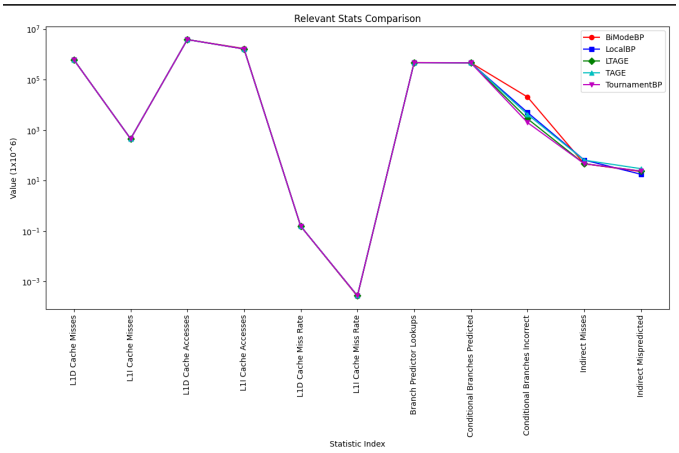

Fig. 7. Gráfica de RISC-V, SPEC, MinorCPU, cambio de branch predictor

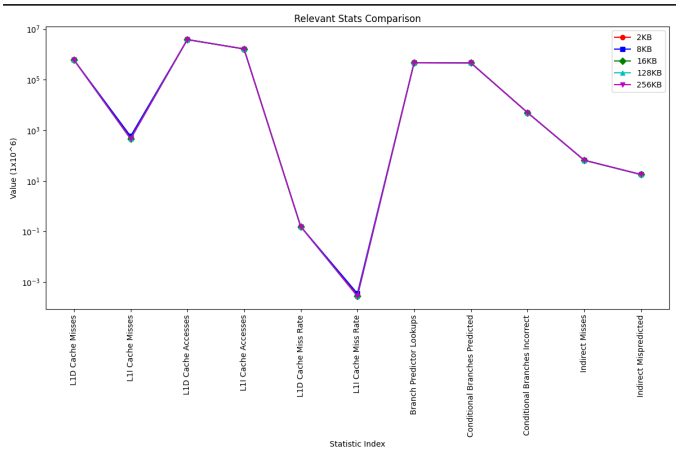

Fig. 8. Gráfica de RISC-V, SPEC, MinorCPU, cambio de caché size

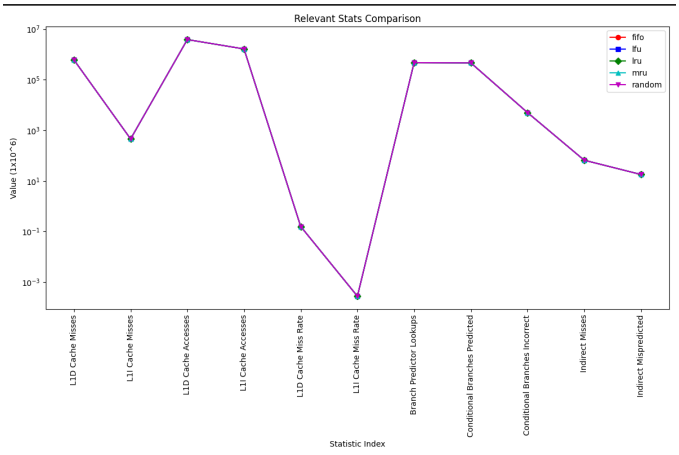

Fig. 9. Gráfica de RISC-V, SPEC, MinorCPU, cambio de políticas de reemplazo.

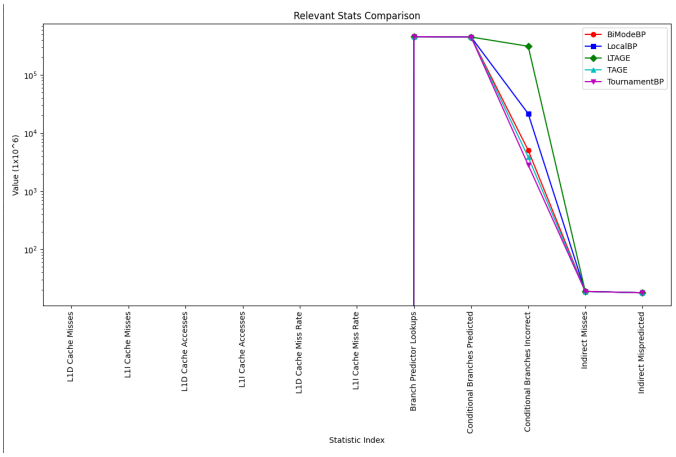

Fig. 10. Gráfica de RISC-V, SPEC, NonCachingSimpleCPU, cambio de branch predictor

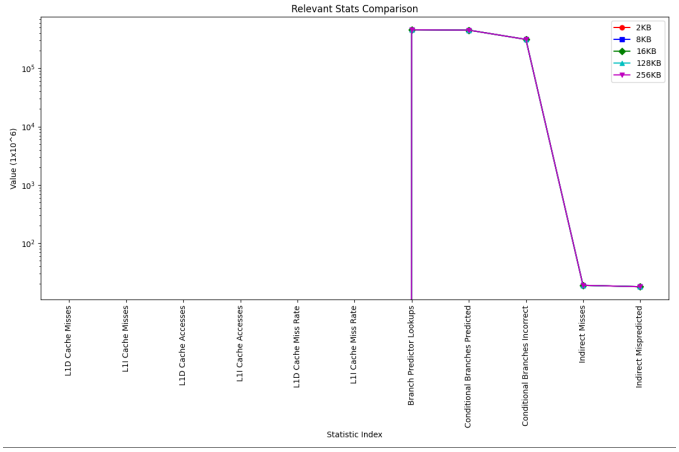

Fig. 11. Gráfica de RISC-V, SPEC, NonCachingSimpleCPU, cambio de caché size

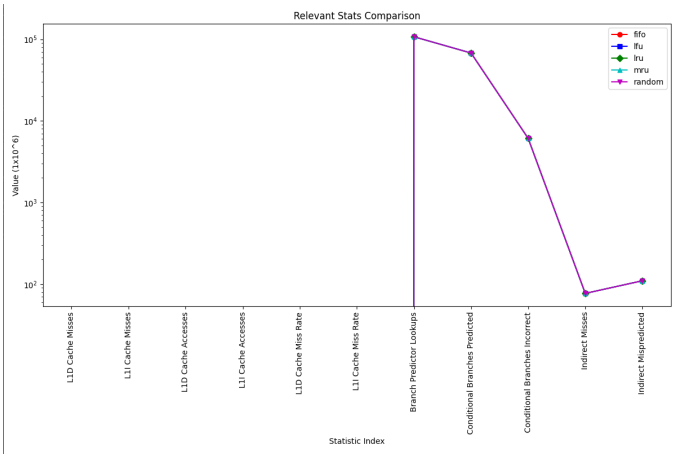

Fig. 12. Gráfica de RISC-V, SPEC, NonCachingSimpleCPU, cambio de políticas de reemplazo

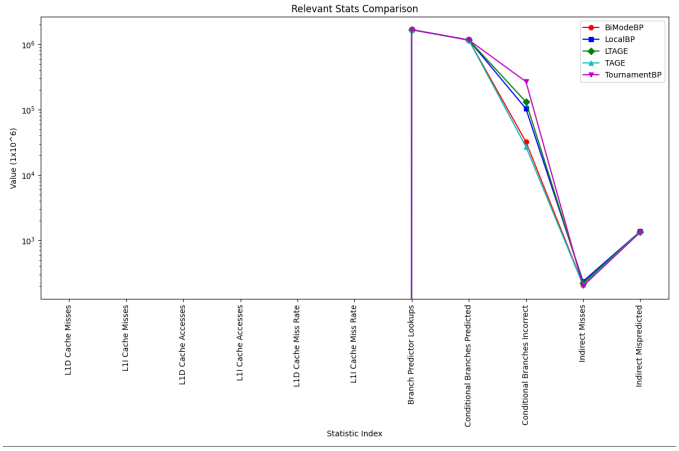

Fig. 13. Gráfica de ARM, PARSEC, MinorCPU, cambio de branch predictor

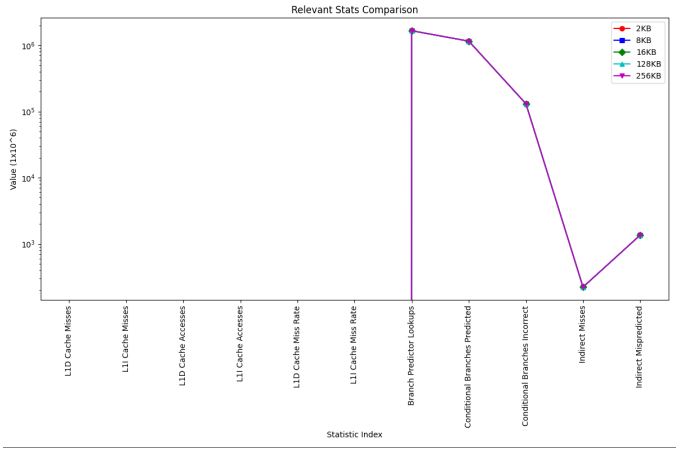

Fig. 14. Gráfica de ARM, PARSEC, MinorCPU, cambio de caché size

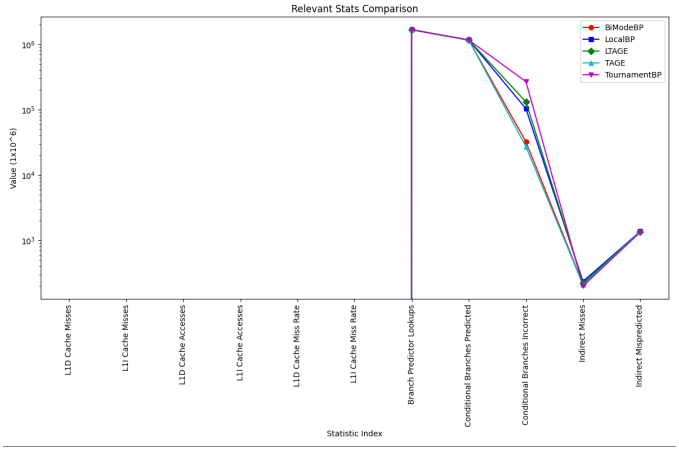

Fig. 15. Gráfica de ARM, PARSEC, MinorCPU, cambio de políticas de reemplazo

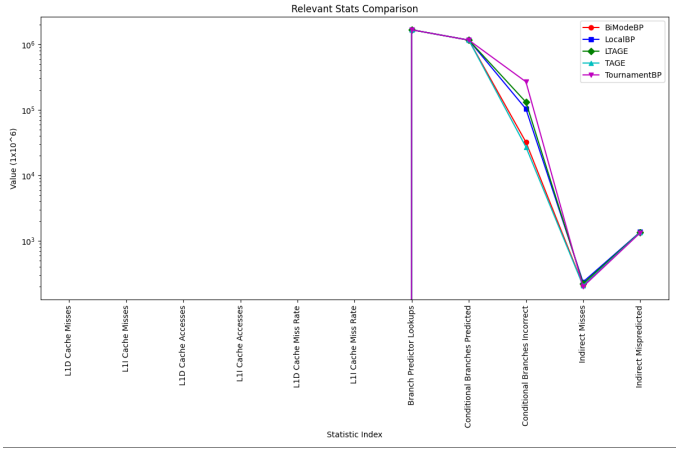

Fig. 16. Gráfica de ARM, PARSEC, NonCachingSimpleCPU, cambio de branch predictor

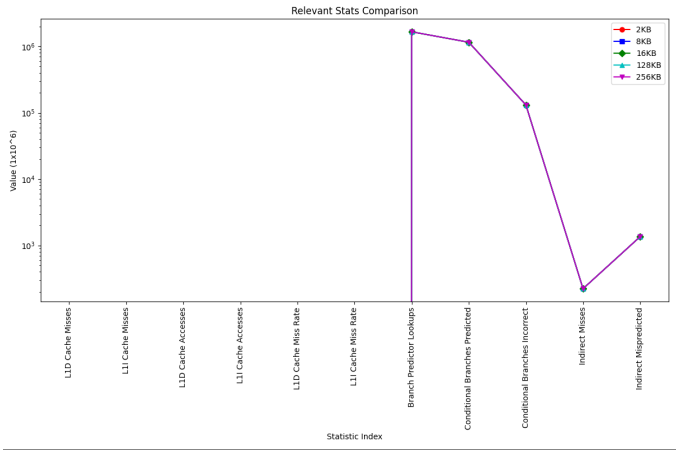

Fig. 17. Gráfica de ARM, PARSEC, NonCachingSimpleCPU, cambio de caché size

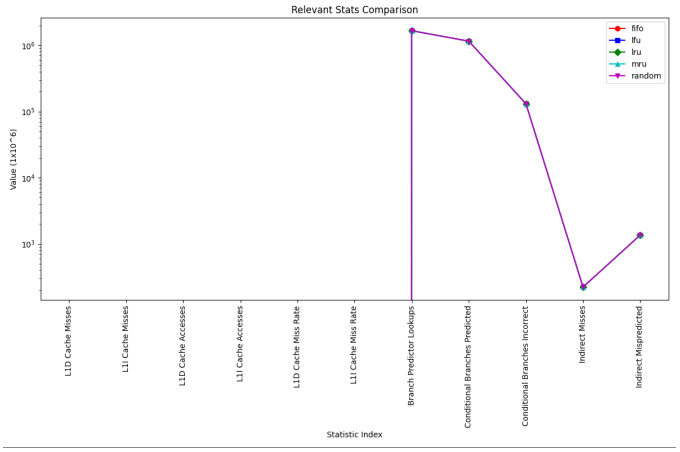

Fig. 18. Gráfica de ARM, PARSEC, NonCachingSimpleCPU, cambio de políticas de reemplazo

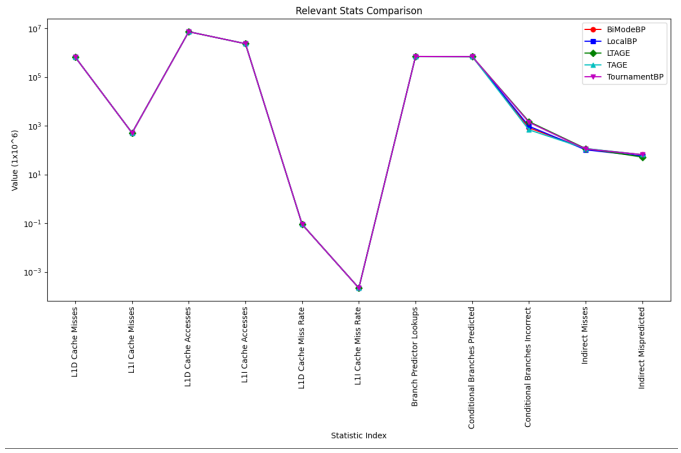

Fig. 19. Gráfica de ARM, SPEC, MinorCPU, cambio de branch predictor

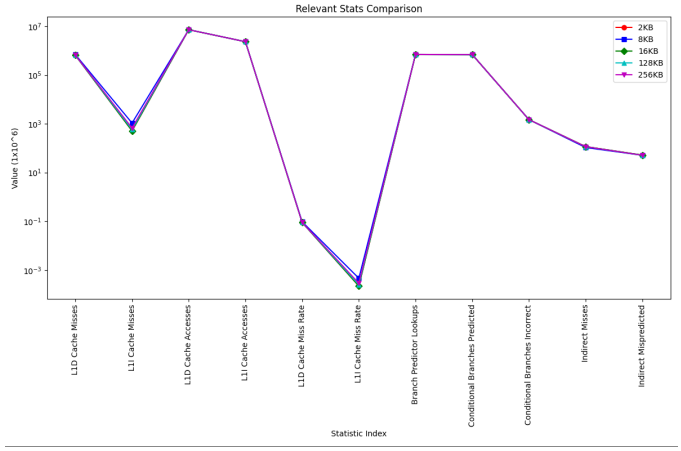

Fig. 20. Gráfica de ARM, SPEC, MinorCPU, cambio de caché size

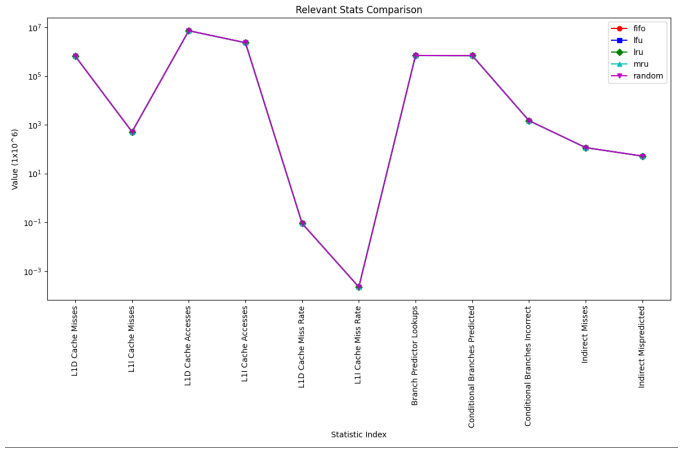

Fig. 21. Gráfica de ARM, SPEC, MinorCPU, cambio de políticas de reemplazo

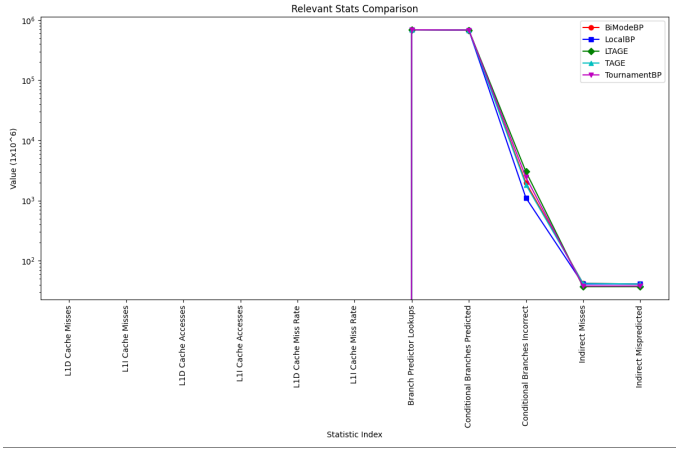

Fig. 22. Gráfica de ARM, SPEC, NonCachingSimpleCPU, cambio de branch predictor

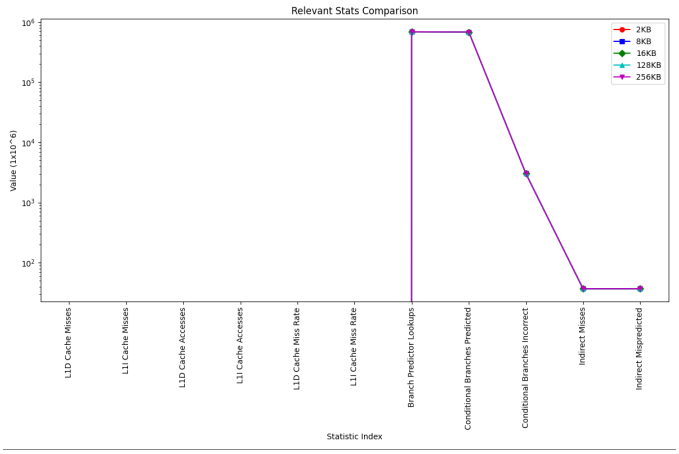

Fig. 23. Gráfica de ARM, SPEC, NonCachingSimpleCPU, cambio de caché size

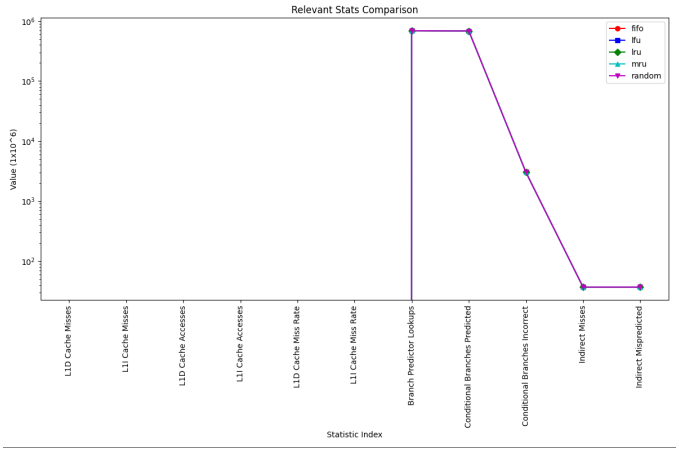

Fig. 24. Gráfica de ARM, SPEC, NonCachingSimpleCPU, cambio de políticas de reemplazo

REFERENCES

[1] D.Barbier, ”Introducing the Latest SiFive RISC-V Core IP Series”, (s.d), (s.d), February, p13, 2018.

[2] msyksphinz, ”RISC-V Instruction Set Specifications” (s.d). (s.d). 2019.

[3] “458.sjeng: SPEC CPU2006 Benchmark Description”. SPEC - Standard Performance Evaluation Corporation. Accedido el 13 de junio de 2024. [En línea]. Disponible: https://www.spec.org/auto/cpu2006/Docs/458.sjeng.html

[4] “Blackscholes/src . a76467d2cd3e94720545ab0609c69fb1e614bac7 . benchmarks / parsec-ompss . GitLab”. GitLab. Accedido el 13 de junio de 2024. [En línea]. Disponible:

https://pm.bsc.es/gitlab/benchmarks/parsec-ompss/-/tree/a76467d2cd3e94720545ab0609c69fb1e614bac7/blackscholes/src