MODIFICACIONES AL SIMULADOR SIM-ASYNC

Proyecto de Sistemas Informáticos
Facultad de Informática
Universidad Complutense de Madrid

Dirigido por Juan Lanchares Dávila,
Miguel Ángel Foruria Casajús y Daniel Blake Otero

CURSO: 2008-2009

Resumen

Sim-async modela la micro-arquitectura de un procesador superescalar totalmente asíncrono de 64-bits con ejecución fuera de orden y especulativa de instrucciones. El trabajo realizado consiste en un conjunto de modificaciones concretas realizadas al simulador asíncrono. Después de cada modificación se han realizado varias pruebas. Después de hacer que el número de unidades funcionales de enteros sea parametrizable se han realizado ocho pruebas para comprobar tiempos. Cada una de estas pruebas consiste en una evaluación del tiempo de finalización del último commit duplicando el número de unidades funcionales de enteros de uno a dos, de dos a cuatro y de cuatro a ocho. La frecuencia de utilización de cada unidad funcional de enteros se muestra en los resultados también. Después de estas pruebas se han ejecutado pruebas de los SPEC2000 en el simulador. En este caso las pruebas se han lanzado tanto en modo síncrono como en modo asíncrono para comparar los tiempos en ambos modos de funcionamiento.

Finalmente, se hace una evaluación general de todos los resultados comparando las pruebas realizadas con los resultados de ejecutar los SPEC2000 y se comparan también los resultados de ejecutar los SPEC2000 en ambos modos, síncrono y asíncrono.

Summary

Sim-async simulates de micro-arquitecture of a superscalar totally asynchronous 64 bit processor with out of order and speculative execution of instructions. The work done consists in a group of specific modifications made to the asynchronous Simulator. Several tests have been made after every modification. After getting the number of integer functional units Parameterized eight tests have been done to compare times. Each one of these tests consists in an assessment of the completion time of the last commit doubling the number of functional units of integers from one to two, from two to four and from four to eight. The frequency of use of each integer functional unit is shown in the results as well. After these tests have been carried out, tests of SPEC2000 have been executed in the simulator. In this case the tests have been executed in both synchronous and asynchronous mode to compare the times in both modes of operation.

Finally, a general assessment is made of all the tests carried out by comparing the results from tests with the results of running the SPEC2000 and comparing also the results of the SPEC2000 running in both modes, synchronous and asynchronous.

Palabras clave

Procesador

Simulación

Asíncrono

sim-async

Arquitectura

Microarquitectura

SPEC

Función de distribución

Sincronización

Comunicación

AUTORIZACIÓN DE USO

Por la presente, los abajo firmantes autorizan a la Universidad Complutense de Madrid a difundir y utilizar el contenido de esta memoria final del proyecto de Sistemas Informáticos llevado a cabo durante el curso 2007/2008 en la facultad de Informática siempre que sea con fines académicos y no comerciales.

Igualmente, se autoriza a la Universidad Complutense de Madrid a emplear con fines académicos y no comerciales el código desarrollado a lo largo de la realización del proyecto, la documentación incluida, las diferentes versiones desarrolladas y todo el material concerniente a la realización del presente proyecto de Sistemas Informáticos.

Madrid, a ___ de Septiembre de 2008.

Miguel Ángel Foruria Casajús
DNI: 52997291-R

Daniel Blake Otero
DNI: 70072878 C

ÍNDICE

INTRODUCCIÓN

Debido al actual nivel de integración y frecuencias de reloj en arquitecturas de microprocesadores, la sincronización de todos los elementos mediante una única fuente de reloj es una tarea extremadamente difícil. Como alternativa, los diseños totalmente asíncronos con circuitos que se auto-gestionan sustituyen la señal de reloj por protocolos de sincronización local.

Gracias a ello, las microarquitecturas asíncronas permiten llevar el diseño de procesadores un paso más allá de las limitaciones físicas que se presentan en la correcta distribución de la señal de reloj a todos los elementos de un sistema. De este modo, estos sistemas carecen de los problemas asociados a la distribución de la señal de reloj, y el tiempo de ejecución del circuito global corresponde a la ejecución del caso medio, ya que un nuevo cómputo empieza siempre justo después de terminar otro.

Al igual que ocurre en el campo de los sistemas síncronos, los diseñadores de sistemas asíncronos necesitan herramientas para probar y experimentar que abstraigan los detalles de implementación del hardware. Estas herramientas deben de ser capaces de modelar el retardo de las dependencias de datos de un sistema totalmente asíncrono a ese nivel de abstracción de la arquitectura, y además deben ser capaces de ejecutar aplicaciones completas.

Dentro del campo de los sistemas síncronos, una de las principales herramientas de simulación es la conocida con el nombre de “SimpleScalar”. Esta herramienta permite modificar la caché, el predictor de saltos o cualquier otro parámetro de la arquitectura, además de ser capaz de ejecutar benchmarks standard para obtener medidas comparables con cualquier tipo de datos relacionados con la ejecución y otras estadísticas.

Una vez vista la necesidad de obtener una herramienta de este tipo para sistemas asíncronos, uno de los problemas principales será saber como modelará la herramienta los retardos en el cómputo de las dependencias de datos de los módulos que forman un procesador asíncrono. Dado que los circuitos asíncronos toman diferentes cantidades de tiempo al hacer un mismo tipo de cómputo pero con diferentes valores de entrada, sería posible recoger una larga lista de valores de retardos para un circuito dado ejecutando simulaciones de bajo-nivel con un número de entradas representativo. De esa lista de retardos se puede obtener la función estadística de distribución que caracteriza el comportamiento del circuito tratado.

Sim-async es un simulador de una microarquitectura superescalar asíncrona. Esta herramienta será útil para el estudio de diferentes propuestas de arquitecturas asíncronas. Sim-async hace un estudio de los tiempos de dependencias de datos utilizando funciones de distribución que describen la probabilidad de que un cómputo dure un retardo determinado. Además, sim-async simula los retardos de todas las partes del hardware relacionadas con la comunicación asíncrona entre etapas.

Sim-async es el resultado de modificar el código fuente de SimpleScalar sustituyendo el núcleo que llevaba SimpleScalar por un motor de ejecución propio, que proporciona la funcionalidad de una microarquitectura adaptada al Alpha ISA.

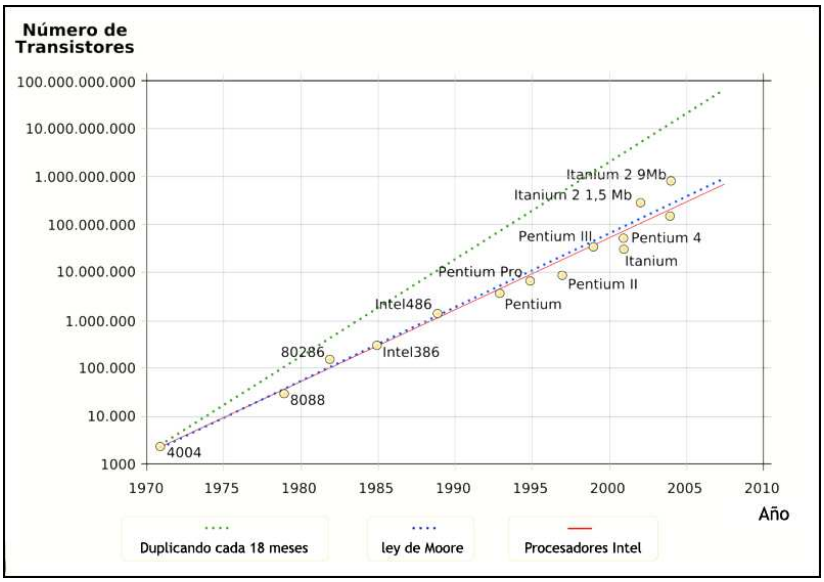
El presente trabajo gira por tanto en torno al funcionamiento y diseño de Sim-Async, así como a la labor de modificación y ampliación llevada a cabo sobre él durante el curso 2007-2008.

ESTADO DEL ARTE

Desde los años 70 con la aparición en el mercado de los primeros procesadores y su desarrollo comercial por parte de las grandes empresas, se hizo patente la necesidad de desarrollar simuladores que permitiesen a los investigadores desarrollar su trabajo sin que ello implicase un gasto económico y temporal excesivo. Bajo esta idea se desarrollaron los primeros simuladores de microarquitecturas que permitían el lanzamiento de ficheros de prueba y otorgaban a los investigadores un conjunto de datos con los cuales analizar las mejoras posibles en el rendimiento de los procesadores antes de su implementación final, ofreciendo así una ayuda muy valiosa en dicha tarea.

Sin embargo, el avance de la tecnología implica un nivel de integración cada vez mayor. Con ello los circuitos reducen cada cierto tiempo su tamaño físico, aumentando igualmente el número de transistores por unidad de superficie. En el año 1965, años antes de la creación de los primeros procesadores, el investigador Gordon Moore afirmo que el número de transistores por pulgada en circuitos integrados se duplicaría cada dos años. Esta es la conocida como "Ley de Moore", que se ha venido cumpliendo hasta el momento de un modo bastante escrupuloso, tal y como se puede observar mediante la siguiente gráfica:

Ley de Moore



Este aumento en el número de transistores por unidad de superficie, tiene una consecuencia casi inmediata. Para que todos los elementos de un procesador funcionen correctamente, la señal de reloj que los alimenta debe llegar a la vez a todos ellos. Sin embargo, esta señal de reloj es transportada por canales físicos que como es lógico cuentan con sus limitaciones, no pudiendo reducir infinitamente el tiempo de transmisión de una señal por dichos canales.

Ello plantea un cuello de botella en el desarrollo de procesadores que, viendo reducido su tamaño y potencia cada cierto tiempo, no pueden ver aumentada la frecuencia de reloj en los mismos términos debido al tiempo necesario para la transmisión de esta.

Con esta idea en mente, se hace necesario que los investigadores enfoquen sus miras a solucionar el problema de aumentar el rendimiento de los procesadores sabiendo de antemano que se comienza a alcanzar un límite en lo que al aumento de frecuencias de reloj se refiere.

Así, surgen diferentes tendencias en las investigaciones que llevan a caminos diferentes. Por un lado, una idea quizás más conservadora promovida por algunos investigadores de grandes empresas. Ante la dificultad de aumentar el rendimiento ampliando las frecuencias de reloj, aumentar el número de procesadores de un solo computador. De este modo, con frecuencias de reloj menores se consigue aumentar la productividad global del sistema al permitir que las tareas se lancen en paralelo sobre diferentes procesadores.

Otra rama de desarrollo, lo que promueve es un cambio radical en el concepto vigente de microarquitecturas comerciales. Así, la investigación en arquitecturas asíncronas plantea la posibilidad de desarrollar, a nivel comercial, procesadores que carezcan de señales de reloj para coordinar los diferentes elementos de un procesador. Con ello, se logra eliminar el problema que surge con la distribución de la señal de reloj a los elementos del sistema, sustituyendo esta por protocolos de comunicación entre estos que permiten el correcto funcionamiento global del sistema.

Sin embargo los investigadores requieren de herramientas que permitan, al igual que ocurre con los sistemas síncronos, realizar estudios de rendimiento y comprobar las mejoras que se producen en estos tipos de sistemas ante la modificación de sus elementos, parámetros de funcionamiento o implementación de cada fase de acción.

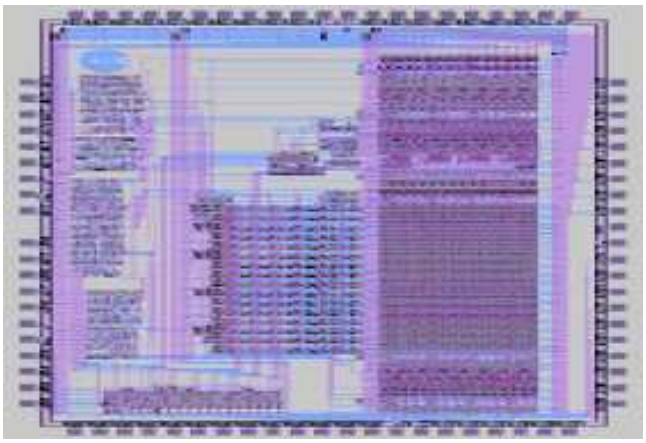
Haciendo un poco de historia, el desarrollo de sistemas asíncronos surge de modo más o menos paralelo al de los sistemas síncronos. A principios de la década de los 70, surgen los primeros sistemas asíncronos mediante el diseño de sumadores, memorias y otros elementos de proceso, incluidos computadores mas complejos como CHARM (diseñado para realizar cálculos relacionados con las probabilidades de los "Procesos de Markov") o el MMS-4 (Diseñado para ayudar en la manipulación de modelos moleculares).

Ya a finales de los años 70 y comienzos de los 80, nos encontramos con un nuevo sistema asíncrono completo: El DDM, desarrollado por la universidad de UTA. Este sistema es considerado como la primera máquina de tratamiento de datos operacional en el mundo asíncrono con un funcionamiento pleno.

Para finales de los años 80, surgió el primer procesador asíncrono de la era VLSI o, lo que es lo mismo, el primer procesador asíncrono propiamente dicho, integrado en un único circuito. El "Caltech" consistía en un procesador simple de 16 bits con un pipe

de 2 fases cuya intención no era la innovación en si misma si no servir como prueba de la validez de los diseños asíncronos y la línea de investigación de la empresa.

Layout del procesador asíncrono "Caltech"



A partir de este momento, durante la década de los 90 numerosas empresas y centros de investigación desarrollaron sus propios procesadores asíncronos, tales como el "NSR" de la propia universidad de UTAH, o el "Titac" del Instituto de Tecnología de Tokio.

En esta década, se fabrica también el procesador Amulet, desarrollado por la universidad de Manchester cuyo objetivo es la investigación de los procesadores asíncronos con el propósito de reducir el consumo de energía. Ganador de varios premios, este procesador consiste en una implementación asíncrona de la arquitectura ARM. algunas características de este procesador son:

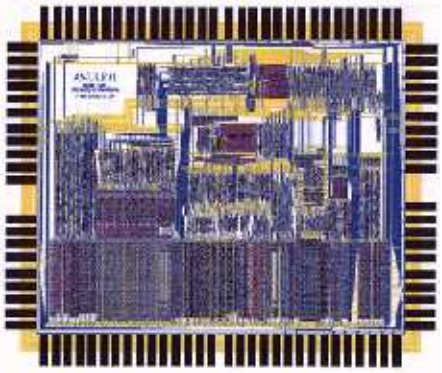
- Compatibilidad con el conjunto de instrucciones del ARM6.
- Transferencia de bloques completos de memoria.
- Tratamiento de instrucciones condicionales
- Retardos dependientes de los datos

El diseño permitió demostrar que los problemas habituales de procesadores síncronos, tales como excepciones, instrucciones de salto, interrupciones y demás, pueden ser solventadas igualmente con sistemas asíncronos.

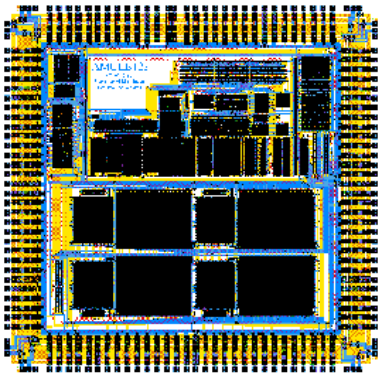
La siguiente modificación del procesador dio como resultado el Amulet-2, que consistía básicamente en una reimplementación del procesador anterior con algunas mejoras tales como:

- Predicción de saltos
- Tamaño dinámico del bus
- Micropipeline de 4 fases
- Modo "sleep" para ahorro de energía

Layout del microprocesador Amulet-1

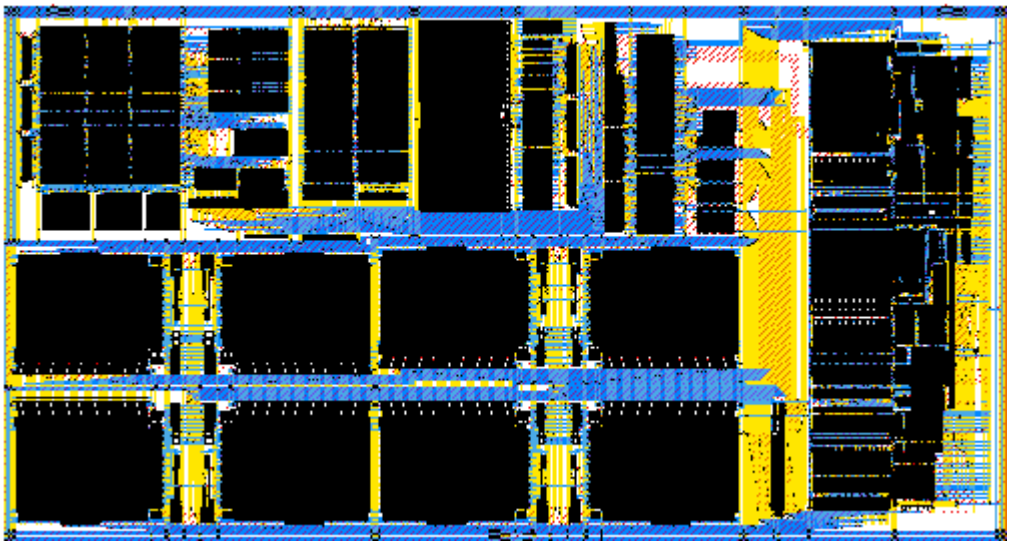


Layout del procesador Amulet-2e



Las siguientes investigaciones y modificaciones dieron como resultado los procesadores Amulet-3 y Amulet3i a comienzos de la primera década del 2000. La última versión de estos se fabricó en noviembre del 2000 con la intención de promover su uso finalmente comercial. Sin embargo, algunos pequeños fallos en el funcionamiento (tales como un 5% de gasto de energía adicional debido a la imposibilidad de desconexión de algunas unidades) dieron al traste con tales expectativas.

Layout del procesador Amulet 3i



A pesar de todo ello, la serie de procesadores "Amulet" son una clara muestra de cómo el diseño asíncrono puede solucionar los problemas típicos de un procesador y funcionar correctamente a gran escala. Según los tests realizados en esta gama de procesadores, con ellos se consigue un rendimiento muy similar al de un procesador de tipo síncrono que ejecute el repertorio de instrucciones ARM. Esto supone una clara promesa en lo que a la investigación en este sector se refiere.

También pone de manifiesto uno de los grandes vacíos existentes dentro del mundo de la investigación en sistemas asíncronos: La falta de herramientas de simulación de sistemas asíncronos.

Los investigadores en sistemas síncronos cuentan con numerosas y muy poderosas herramientas de simulación que permiten, sin necesidad de fabricar físicamente un diseño, probar el correcto funcionamiento de este o investigar en posibles modificaciones que den lugar a mejorar el rendimiento del procesador. Ello supone una clara ventaja con respecto al mundo asíncrono, en el que existe un gran vacío al respecto.

El simulador de una microarquitectura, consiste básicamente en un elemento software que permite la predicción del comportamiento y rendimiento del hardware. Así, el simulador para una microarquitectura dada, permite conocer de antemano resultados reales de rendimiento una vez aplicadas mejoras en el sistema. Los beneficios de disponer de este tipo de herramientas son claros.

Dentro del mundo síncrono, encontramos herramientas bastante poderosas tales como:

- OVPsim
- SIMICS (Simulador de un sistema completo)
- ML-RSIM
- WINMIPS-64 (Simulador de la arquitectura MIPS de 64 bits)
- M5 (simulador que soporta varias microarquitecturas: Alpha, MIPS, etc...)
- SimpleScalar

Y un sin fin más de herramientas. Sin embargo, en lo que se refiere a implementaciones asíncronas la lista, aunque existente, se ve notablemente reducida. Podemos encontrar simuladores tales como “ARAS” (Simulador de una microarquitectura ARM asíncrona) o “ASIM” (Simulador desarrollado en “Java” por la universidad de UTAH).

En este contexto surge “Sim-Async”, simulador desarrollado en “C” para su uso bajo sistemas operativos UNIX que implementa el repertorio de instrucciones Alpha. Sim-async presenta una serie de propuestas en el mundo de los procesadores asíncronos y permite la simulación de bancos de pruebas standard tales como son los SPEC-2000, además de conjuntos de pruebas propios.

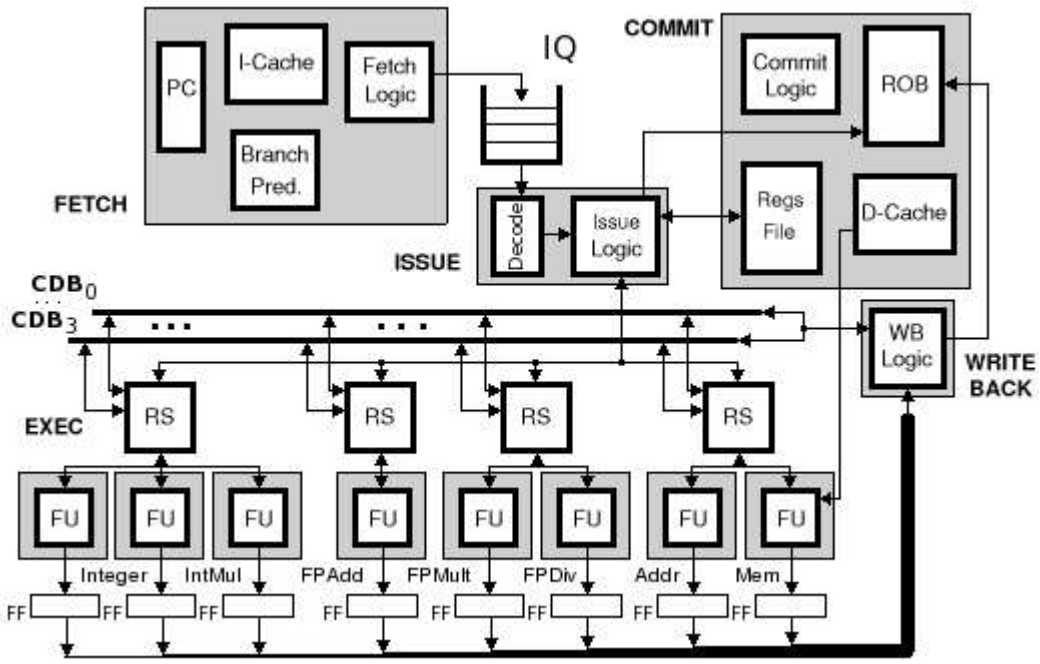
Su objetivo es el de servir como herramienta de simulación a los investigadores, paliando de algún modo y en su medida la falta de herramientas en este ámbito ya mencionada anteriormente.

SIM-ASYNC

2.1.- Microarquitectura

Sim-async modela la micro-arquitectura de un procesador superescalar totalmente asíncrono de 64-bits con ejecución fuera de orden y especulativa de instrucciones. El procesador se compone de cinco etapas: “fetch”, “issue”, “exec”, “write-back” y “commit”.

El esquema de la micro-arquitectura simulada es tal y como se observa a continuación. Como se puede observar, la etapa de ejecución está representada con algo más de detalle que las demás, aunque la lógica relacionada con la comunicación entre los diferentes módulos del sistema no está incluida en este esquema.



La implementación del procesador asíncrono es idéntica a la del síncrono, excepto que hay que sustituir la red del reloj por un grupo de componentes que permiten la comunicación de resultados entre módulos. A continuación se explica brevemente la funcionalidad de cada etapa:

Fetch: Un número parametrizable de instrucciones es leído de la caché de instrucciones teniendo en cuenta la predicción del salto. Las instrucciones son movidas a la cola de instrucciones (IQ), donde esperan a la etapa “Issue”. Sí alguna de las instrucciones en medio del grupo de “fetch” es un salto condicional tomado o un salto incondicional, entonces las siguientes instrucciones en el grupo serán descartadas.

Issue: **E**l diseño de la etapa "issue" es fundamental para obtener un elevado rendimiento en un procesador superescalar. La implementación escogida es la conocida como "instruction shelving" con buffer de reordenamiento (ROB) para la etapa "issue" porque desdobla la instrucción "issue" y el chequeo de dependencia.

Con este método el único factor que provocará el bloqueo de las instrucciones "issue" será la falta de entradas libres en las estaciones de reserva (ER, o "shelving buffers") o en el ROB, pero no las dependencias de datos, que son mucho más frecuentes.

Esta etapa decodifica y lanza en orden un número parametrizable de instrucciones de la cola de instrucciones (IQ) a su correspondiente Estación de Reserva (RS) y al Buffer de Reordenamiento (ROB). La etapa "issue" se ejecuta en orden debido a que preservar la consistencia de la secuencia para ejecutar "issue" fuera de orden requiere un mayor esfuerzo que ejecutar "issue" en orden. Además de que raramente se bloquea la etapa "issue" con "shelving", implementar "issue" fuera de orden tendría un beneficio poco significativo.

Ejecución: Las estaciones de reserva preservan las dependencias de datos manteniendo las etiquetas de las instrucciones que generarán los operandos pendientes, y guardaran valores esperando la ejecución en las unidades funcionales (FU).

Como se muestra en la figura anterior, la micro-arquitectura está provista de cuatro estaciones de reserva. La lógica del mensaje decide cual de las instrucciones listas de la estación de reserva será lanzada a "issue" a su correspondiente unidad funcional, teniendo en cuenta que cuanto más tiempo lleve la instrucción lista antes es lanzada a "issue".

Write-back: Una vez que el cómputo de cada unidad funcional haya finalizado, el resultado se almacena en el biestable de salida, provocado por una señal de captura, hasta que la etapa "write-back" se complete.

En esta etapa, la lógica de selección elige el resultado para ser distribuido a la estación de reserva y al buffer de reordenamiento a través del número de instancias (parametrizable) del bus de datos común (CDB), también enviando la etiqueta de las instrucciones que generaron cada resultado. La lógica de activación de cada estación de reserva compara las etiquetas entrantes con las etiquetas de las instrucciones pendientes efectuando la actualización de los valores en el caso de que las etiquetas coincidan.

Commit: Cada instrucción en el buffer de reordenamiento guarda el resultado para ser escrito al archivo de registro o a la memoria y al registro destino o a la dirección de memoria. Se retira un número parametrizable de instrucciones del buffer de reordenamiento manteniendo el orden del programa, y la predicción de saltos se comprueba cada vez que esta etapa se ejecuta. Se chequean interrupciones precisas y el "pipeline" se limpia y reestablece cuando se procesa un fallo en la predicción del salto.

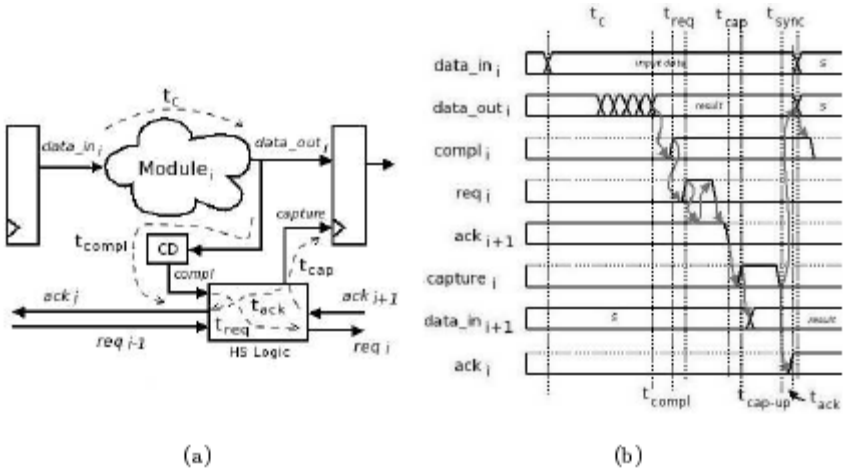
2.2.- Modelando la asincronía

Un dominio de sincronización consiste en todos los biestables activados por una misma señal y la lógica combinacional en todas sus entradas. En la figura anterior se pueden diferenciar doce dominios de sincronización que se han definido (las zonas sombreadas de la figura), donde la comunicación entre ellos se realiza utilizando un protocolo "handshake" de cuatro fases. A continuación se dan detalles sobre el modelado en función del tiempo de un determinado dominio y el protocolo de comunicación asumido.

2.2.1.- Modelado de un "Dominio de sincronización"

Sim-Async cuenta con un enfoque mezclado en el campo de sistemas asíncronos para describir el comportamiento temporal de los dominios. Utiliza un mecanismo de terminación de cómputo para detectar el fin del cómputo, y emplea un enfoque del retardo delimitado para modelar el comportamiento de la lógica de control.

En los sistemas asíncronos el retardo empleado en el cómputo de un dato, detectando la terminación del cómputo y comunicando el resultado al módulo receptor a través del protocolo de sincronización toma un valor diferente e impredecible para cada valor de la entrada. Ese retardo viene de la combinación de otros retardos que aparecen durante la operación de un módulo. Estos retardos vienen indicados por la línea de puntos en la siguiente figura. (a)



(a) (b)

(a) Esquema general de un dominio de sincronización. Las líneas de puntos se refieren a retardos.

(b) Cronograma que muestra los retardos de la lógica relacionado con un cómputo del módulo "i" y la comunicación de los resultados del siguiente dominio.

El retardo del cómputo, *tc*, es el retardo empleado por el módulo en el cómputo del dato de entrada y la generación de resultados. Es un retardo variable porque los circuitos asíncronos presentan un comportamiento dependiente de datos. En este simulador, cada etapa y unidad funcional de la microarquitectura recibe su propio *tc* como una función de distribución. Debido a esto, cuando el módulo realiza un cómputo, el simulador aleatoriamente selecciona un retardo de cómputo teniendo en cuenta la forma de la distribución. Aunque, el verdadero retardo para el cómputo de este dato no se obtiene, el comportamiento dependiente de datos del módulo se mantiene.

El retardo de detección de terminación, *tcompl*, corresponde al tiempo empleado por la lógica de detección de terminación (CD) en detectar una salida válida y asignando la señal compl. Este retardo es incluido como un parámetro de entrada constante en sim-async.

Se utiliza una codificación insensible a retardos y una lógica de detección de terminación debido a la variabilidad de *tc*. Debido a esto los módulos alternan entre un valor neutral o de sincronización (S) que no es ningún valor booleano, y la codificación de una salida válida. La generación de ese valor de sincronización toma *tsync* unidades de tiempo (t. u.) y, después de eso, el módulo está listo para recibir nuevos datos de entrada. Este retardo es también un parámetro de entrada de sim-async. La lógica que solicita la generación del valor de sincronización se omite en la anterior figura para mayor claridad.

El modelado del protocolo "handshake" se divide en dos retardos: retardo de solicitud, *treq,* que es el tiempo que transcurre desde el comienzo de la señal *compl* hasta el comienzo de la señal de solicitud, *reqi;* y el retardo de captura, *tcap*, que es el tiempo empleado desde el pulso de bajada de la señal *ack* del módulo receptor, *ack (i+1),* hasta el comienzo de la señal de captura. El tiempo empleado durante la comunicación es un retardo indeterminado que puede ser claramente obtenido solo con la simulación porque depende, mayormente, de la ocupación o variabilidad de las estructuras del módulo receptor en cada momento. Ambos *treq* y *tcap* están incluidos como parámetros constantes de entrada en sim-async.

Una vez que el protocolo se ha completado, la señal de captura se asigna como un pulso. Esta asignación no viola ninguna suposición de los tiempos porque se considera que *tcompl* es más largo que el retardo de iniciación (setup) del registro destino. Además, la longitud del pulso de captura, llamada en la figura anterior *tcap-up,* debe ser más elevada que la longitud del retardo "hold" del registro provocado por la señal de captura porque la generación del valor de sincronización es solicitado por el pulso de bajada de esa señal.

El retardo empleado desde que cae la señal de captura hasta que comienza la señal *acki* se denomina *tack,* también incluido como un parámetro de entrada de sim-async.

Todas las descripciones anteriores hacen referencia al gráfico anterior, en la figura izquierda (a) hay un esquema del funcionamiento de los módulos y la relación entre los tiempos y en la figura de la derecha (b) un diagrama temporal con los tiempos de activación y finalización de cada retardo.

2.3.- Protocolo de comunicación

La comunicación entre dominios se desarrolla mediante canales implementando un protocolo "handshake" de cuatro fases. La figura anterior (b) (de la derecha) muestra el cronograma de un ejemplo de comunicación entre el dominio i y sus vecinos. A continuación se explica esta comunicación utilizando los retardos definidos en la subsección previa.

El momento en el que el módulo empieza a operar es el instante en el que la señal *data_in*i propaga los datos de entrada. Luego, el módulo procesa estos datos y, después de un retardo de dependencia de datos, *tc,* el resultado se propaga a través de la señal *data_out*i. La señal *compl* se inicia después de *tcompl* t. u. y luego la lógica handshake activa la señal *reqi* para comenzar el protocolo de comunicación. El módulo receptor está listo para procesar nuevos datos porque *acki*+1 se ha activado. En ese momento, la lógica handshake desactiva la señal de solicitud (request) y espera a que termine *acki*+1. El módulo receptor desactiva la señal de acknowledge (reconocimiento) y el protocolo de comunicación termina. Después de eso la lógica handshake genera un pulso en la señal de captura. Durante la activación de la señal de captura (capture) el registro destino lanza los resultados del módulo y, en el pulso de bajada de la señal de captura, la lógica del módulo vuelve al valor de sincronización antes del siguiente cómputo. También el pulso de bajada de la señal de captura provoca la activación de *acki*, que indica que el módulo I está listo para recibir nuevos datos de entrada.

2.4.- Resultados experimentales

Para probar sim-async se han ejecutado los benchmarks SPEC2000 con diferentes configuraciones de tiempos de las opciones que ofrece el simulador. Después, se han comparado los resultados de estas ejecuciones con aquellos obtenidos del SimpleScalar original utilizando la misma configuración para la caché y el predictor de saltos. Las pruebas se ejecutaron parametrizando las etapas *fetch*, *issue*, *write-back* y *commit* para procesar hasta cuatro instrucciones cada vez que se ejecutan. La siguiente tabla (a) muestra, en inglés, la configuración arquitectónica de la microarquitectura y la tabla (b) muestra el caso peor que puede darse en los retardos de la ejecución de las etapas y de las operaciones funcionales en las simulaciones asíncronas.

Configuración arquitectónica de la microarquitectura en simulaciones:

Branch Predictor:	2-level PAg
Level 1	1024 entr, his 10
Level 2	1024 entr
BTB	4096 sets, 2-way
Instructions queue (IQ) size	100 entries
Integer RS queue size	6 entries
FP Addition RS queue size	3 entries
FP Mul, FP Div/Sqrt RS queue size	2 entries
Memory RS queue size	5 entries
Integer / FP Register File	32 / 32
ROB size	100 entries

(a)

Caso peor de los retardos de las etapas y operaciones de las unidades funcionales en las simulaciones asíncronas.

Stage / FU Operation	T. U.
Fetch, Issue, Int/Logic, WB, Commit	1000
IntMul	7000
MemLoad, FPAdd, FPMul	4000
FPDiv/Sqrt	30000

(b)

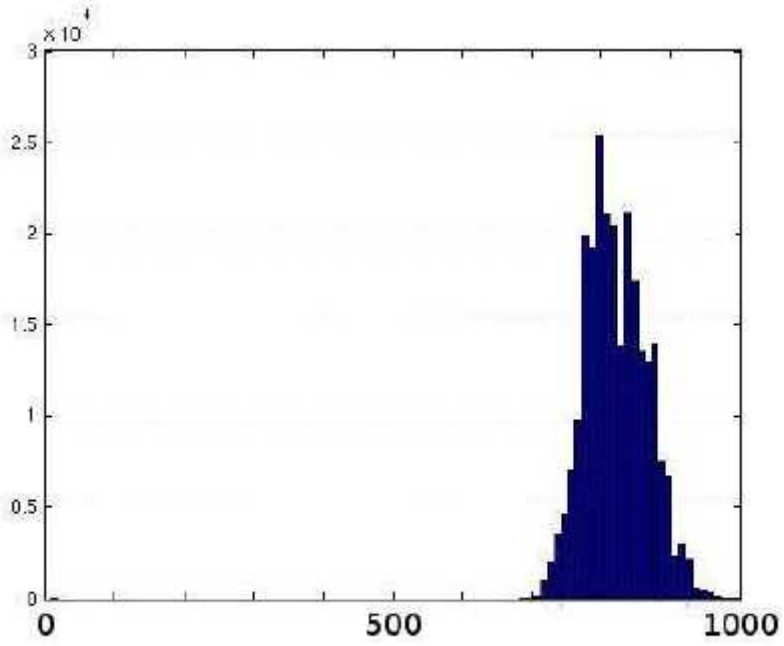
La primera configuración de los valores de tiempos probada era la que es totalmente asíncrona. En esta configuración asíncrona utilizamos dos funciones de distribución para caracterizar los retardos de ejecución de los módulos: Función del caso lento (SC) y función del caso medio (MC). Estas funciones fueron seleccionadas de la lista de simulaciones anotadas anteriormente por puertas de nivel de circuitos asíncronos relacionados, y fueron normalizadas a la misma cota superior (caso peor) de 1000 t. u. (unidades de tiempo).

La función del caso lento (SC), mostrada en la siguiente figura (a), cuyo retardo medio está cerca del peor retardo, representa un comportamiento lento porque la mayor parte de los datos toman un retardo elevado. No se han hecho suposiciones sobre la implementación de unidades funcionales, por lo que estaban caracterizadas individualmente a través de la función del caso lento (SC function).

De todas formas se consideró el uso de unidades funcionales de larga latencia sin segmentación para operaciones en punto flotante (FP operations) y multiplicaciones de enteros, para que la normalización de la función fuese convenientemente corregida a una cota superior mayor para estas unidades funcionales lentas y sin segmentación, de acuerdo con la tabla anterior (b)

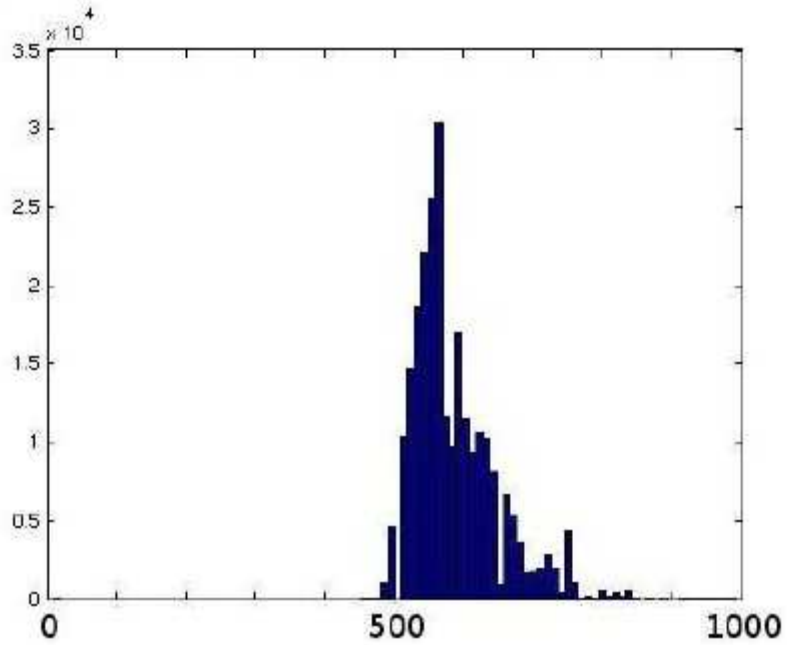
La función del caso medio (MC), mostrada en la siguiente gráfica (b), describe un comportamiento asíncrono donde el retardo medio está cerca de la mitad del peor retardo. Se ha utilizado esta función para caracterizar el resto de las etapas: *fetch*, *issue*, *write-back* y *commit*.

Función de distribución del caso lento (SC)



(a)

Función de distribución del caso medio (MC)



(b)

Es importante remarcar que el objetivo de estas funciones no es el de ser los verdaderos patrones de los módulos del procesador modelado. Se presentan estas funciones como ejemplos típicos de comportamientos de circuitos asíncronos obtenidos de simulaciones previas a bajo nivel.

Para establecer los retardos de la lógica de control, se ha considerado el trabajo de Cheng (10 de la bibliografía). En su trabajo, Cheng implementó un circuito de detección de terminación y sincronización (reinicio detección-terminación) de líneas de datos utilizando un protocolo handshake de cuatro fases y codificación de doble rail. Él obtuvo una media de 0,28 ns para el circuito de detección de terminación y 0,71 ns para la sincronización (reinicio). Considerando que el desarrollo de circuitos integrados digitales ha seguido la ley de Moore y ha mejorado en un 30% anualmente, los retardos de los circuitos que utilizan las tecnologías actuales podrían ser sobre 16 ps y 41 ps respectivamente.

Suponiendo que el camino crítico de los módulos del procesador estará por debajo de 1,25 ns (Esto equivale a un máximo de 800 Mhz de frecuencia en una versión síncrona), que hemos normalizado a 1000 unidades de tiempo (t. u.), los valores normalizados para *tcompl* y *tsync* teniendo en cuenta que la ampliación son 12,8 y 32 unidades de tiempo, respectivamente, que corresponden a retardos medios. En las simulaciones realizadas se han duplicado rigurosamente los retardos a 26 y 64 unidades de tiempo para todos los módulos. El resto de los retardos de la lógica de control, *treq*,

tcap y *tcap-up* fueron adaptados a 5, 5 y 10 unidades de tiempo, respectivamente, y *tack* fue considerado igual a *tsync.*

Se ha comprobado que las salidas obtenidas por sim-async al ejecutar los benchmarks SPEC2000 con la configuración asíncrona son idénticas a aquellas generadas por SimpleScalar para todos los benchmarks. Esto es, *bzip* genera el mismo archivo comprimido, *gcc* devuelve las mismas estadísticas de compilación, y así el resto. Además, se ha comparado el número de instrucciones commiteadas en ambos simuladores para la ejecución de esos benchmarks y solo difieren en un rango insignificante entre 0,21% y -0,012% (debida a la ligera diferencia en la implementación de las llamadas al sistema), como se muestra en la siguiente tabla (a). Por tanto, sim-async desarrolla simulaciones correctamente y ejecuta con éxito el Alpha ISA.

Número de instrucciones commiteadas para varios benchmarks SPEC2000 en SimpleScalar (sim-safe) y para sim-async con una configuración asíncrona.

SPEC	SimpleScalar	Async *Sim-async*	Diff (%)
ammp	45812883	45810845	-0.004
apsi	197579651	197612776	0.017
bzip	1819780172	1819780267	0.000
crafty	94419973	94420229	0.000
galgel	139306245	139310055	0.003
gap	82873902	82874407	0.001
gcc	2016139124	2016204817	0.003
gzip	601857009	601857104	0.000
lucas	19239488	19242782	0.017
mesa	1608605448	1608410610	-0.012
parser	268979662	269006191	0.010
perlbmk	205853718	205914747	0.030
sixtrack	11699655	11724227	0.210
swim	23557475	23562358	0.021
vortex	453666	454534	0.191

(a)

Diferencias medias entre las instrucciones ejecutadas y el uso de módulos de sim-async en configuraciones síncronas y asíncronas ejecutando los SPEC2000

Async vs. Synch	% Avg Diff
# Insn Exec	0.132
Use of Fetch	-42.076
Use of Issue	-61.993
Use of Int	-66.062
Use of IntMul	-99.894
Use of FPAdd	-95.665
Use of FPMul	-97.733
Use of FPDiv	-99.921
Use of Addr	-79.314
Use of Mem	-81.116
Use of WB	-52.324
Use of Commit	-73.852

(b)

Con el objetivo de probar que sim-async no solo ejecuta el Alpha ISA correctamente, sino que también modela correctamente el comportamiento asíncrono, se han hecho comparaciones entre las primeras simulaciones y aquellas que resultan de parametrizar sim-async para modelar un procesador síncrono. Esto es posible porque el comportamiento síncrono es un caso particular del comportamiento asíncrono. Esto es, en un procesador síncrono todos los módulos emplean el mismo tiempo en ejecutar un dato (el caso peor de la etapa más lenta) y el protocolo de comunicación emplea un retardo de cero unidades de tiempo debido a la señal de reloj.

Después, se establecen los parámetros *tcompl* , *treq*, *tcap*, *tcap-up*, *tsync* y *tack* a cero unidades de tiempo, y *tc* se fija a una distribución en la que todos los retardos son 1000 unidades de tiempo de duración, el caso peor de las simulaciones asíncronas, pero considerando las unidades funcionales más lentas (IntMul, FPAdd y FPMul/Div) como unidades totalmente segmentadas. La señal de captura solo se activa sí el módulo receptor está listo para aceptar el nuevo dato de entrada.

Las simulaciones síncronas se ejecutaron bajo la misma configuración de la arquitectura descrita para las simulaciones asíncronas, y se obtuvieron resultados de salida idénticos y también idéntico número de instrucciones commiteadas. Además, se tomaron estadísticas para medir el comportamiento asíncrono. Como muestra la tabla anterior (b), el número de instrucciones ejecutadas (incluyendo aquellas especulativas) es, en media, 0,132% mayor en la configuración asíncrona. Esto sucede porque los retardos medios de las etapas asíncronas son más cortos que el caso peor síncrono.

Aunque, el número de ejecuciones de los módulos asíncronos se reduce en relación a las simulaciones síncronas. Los rangos de reducción medios desde 42,076%

de la etapa *fetch* a 99,921% de la unidad funcional FPDiv, que permanece inactivo casi todo el tiempo (ver la tabla anterior (b)). Este comportamiento corresponde al esperado para un circuito asíncrono porque los módulos solo se ejecutan cuando hay "trabajo útil" para ejecutar.

Como una estadística adicional, el speedup alcanzado por la configuración asíncrona en relación a la síncrona es, de media, 1,135 para el SPEC2000.

Así, esta comparación entre ambas simulaciones (asíncrona y síncrona) verifica el correcto diseño del comportamiento asíncrono que sim-async desarrolla utilizando funciones de distribución para caracterizar el retardo de ejecución de los módulos de la microarquitectura.

DESARROLLO DEL PROYECTO

El presente proyecto gira en torno a la modificación y el estudio de rendimiento relativo al simulador de una implementación asíncrona de una microarquitectura superescalar de 64 bits. Este simulador consiste en una modificación del núcleo de la conocida aplicación "SimpleScalar" cuyo objetivo es el de simular microarquitecturas síncronas con un alto nivel de detalle.

Así, "sim-async" consiste principalmente en una modificación del núcleo de ejecución de "SimpleScalar", sustituyendo el motor de ejecución original por un motor totalmente nuevo que permite la simulación de una arquitectura asíncrona de 64 bits.

Este simulador se encuentra implementado en el conocido lenguaje de programación "C" bajo un Sistema Operativo de tipo UNIX y, más concretamente, bajo una distribución standard de "Linux".

Por tanto el desarrollo del proyecto requiere adquirir una cierta destreza en el manejo de este Sistema Operativo, así como de las técnicas de programación disponibles en él (librerías propias del sistema operativo).

Con todo ello, a lo largo de una serie de modificaciones en el código del simulador superescalar "sim-async", se debe ampliar la funcionalidad de este permitiendo la parametrización de las unidades funcionales del sistema. Igualmente, se debe ampliar la funcionalidad del sistema permitiendo el tratamiento de los protocolos de comunicación de un modo dinámico mediante funciones de distribución estadísticas, en lugar de ser calculado este tiempo de manera estática.

Al mismo tiempo que se lleva a cabo dicha ampliación de funcionalidad, se deben realizar pruebas que den como resultado un estudio del impacto que suponen, en términos de rendimiento y productividad, las mejoras introducidas.

3.1.- Unidades Funcionales

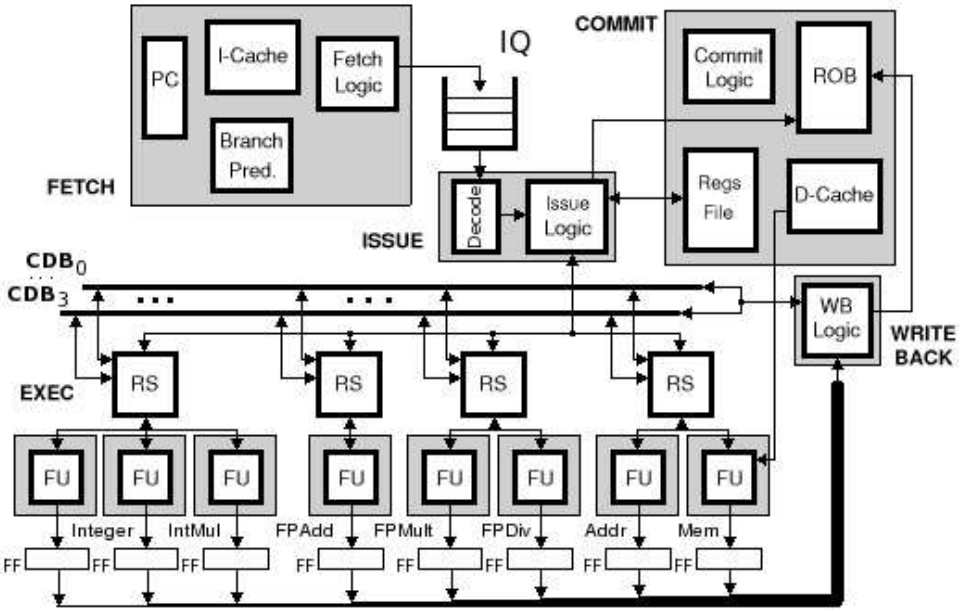
3.1.1.- Introducción

Sim-Async, como simulador de una microarquitectura superescalar de 64 bits, cuenta con una serie de unidades funcionales encargadas de realizar los cómputos necesarios para la correcta ejecución de las instrucciones del repertorio Alpha. Estas unidades funcionales se pueden resumir en:

- Unidades de operaciones con enteros:
    - Sumadores
    - Multiplicadores
    - Calculo de direcciones
    - Acceso a memoria

- Unidades de operaciones en Punto Flotante
    - Sumadores
    - Multiplicadores
    - Divisores

Esquema de la microarquitectura simulada por Sim-Async



Tal y como se observa en la figura, cada una de estas unidades funcionales recibe sus diferentes operandos desde las estaciones de reserva en las cuales se depositan desde la instrucción en la fase de "issue". Estas Estaciones de reserva almacenan los datos y comunican su disponibilidad a las unidades funcionales. De este modo, cuando una unidad funcional esta libre para ejecutar una nueva operación, comunica su disponibilidad y recoge los operandos, liberando así la estación de reserva para nuevos datos de entrada.

Además, el retardo de cada una de estas unidades funcionales se encuentra configurado mediante el uso de funciones de distribución estadísticas con datos obtenidos de pruebas a bajo nivel en unidades reales semejantes. Este hecho conlleva a una simulación mas precisa del tiempo de ejecución real de la microarquitectura propuesta.

Uno de los objetivos del presente trabajo consiste en ampliar la funcionalidad del simulador Sim-Async. Con dicho fin, se modificó el código del núcleo de ejecución del simulador de modo que el número de unidades funcionales disponibles en la ejecución sea indicado desde un fichero de configuración externo.

De este modo, el número de unidades funcionales de cada tipo podrá ser configurado externamente. Gracias a ello se podría probar la microarquitectura

propuesta con diferentes configuraciones en lo que a Unidades Funcionales se refiere. Se podría estudiar el rendimiento obtenido según el uso de las diferentes unidades funcionales de manera que se pueda optimizar el diseño de la microarquitectura propuesta por "sim-async".

3.1.2.- Estado inicial

Inicialmente el simulador cuenta con un número fijo de unidades funcionales:

1 unidad de multiplicación no segmentada
1 unidad de cálculo de direcciones no segmentada
1 unidad de acceso a memoria no segmentada
2 unidades de sumadores segmentada
1 unidad de sumadores segmentada
1 unidad de multiplicadores segmentada
1 unidad de divisores segmentada

3.1.3.- Modificaciones y pruebas

Al conseguir que el número de unidades funcionales enteras sea parametrizable se procedió a crear un fichero mediante el cual se configuraría el número de unidades funcionales en el simulador, de modo externo.

De este modo se permite la configuración del simulador desde el exterior, sin necesidad de modificar las variables dentro del propio código y recompilar el proyecto al completo.

Mediante estas ideas, se logró finalmente la ampliación de la funcionalidad del simulador "sim-async" mediante la parametrización de las unidades funcionales. Además, se incluye en el código de la aplicación la implementación de una serie de contadores que muestran al usuario el uso de cada una de las unidades funcionales empleadas por separado, permitiendo los pertinentes análisis de rendimiento observando el nivel de uso de cada una.

Pruebas sobre los cambios realizados:

El objetivo de estas modificaciones es realizar pruebas con el simulador para comprobar resultados y ver hasta que punto merece la pena aumentar el número de unidades funcionales de enteros en una arquitectura asíncrona. Se comprobarán los tiempos de finalización del último commit medidos en unidades de tiempo (t.u.) al ejecutar diversos tests que prueban la funcionalidad del simulador. Además, se comprobará la utilización de cada unidad funcional entera viendo la ocupación. Finalmente se realizarán las medias de todos los tiempos para ver de una manera más fiable la efectividad del aumento de la cantidad de unidades funcionales de enteros.

Con el objetivo de probar la rentabilidad de introducir un determinado número de unidades funcionales enteras en el simulador asíncrono, hemos realizado pruebas introduciendo cantidades fijas de 1, 2, 4 y 8 unidades funcionales de enteros mientras que el resto de unidades funcionales se mantenían con un valor constante de 1 por cada tipo de unidad funcional.

Para obtener resultados fiables para cada tipo de configuración, hemos ejecutado 8 tests (cada uno con las 4 configuraciones) que realizan diversos tipos de operaciones y hemos comparado el tiempo de finalización del último commit de cada uno de ellos medido en t.u. (unidades de tiempo).

Hemos representado los resultados en una gráfica para comparar la variación al utilizar las distintas cantidades de unidades funcionales de tipo entero (1, 2, 4 y 8).

Además de obtener estos valores de tiempo también se muestra la ocupación de cada unidad funcional de enteros, según cuantas haya. Esto es útil para comprobar rentabilidad de tener un determinado número de unidades funcionales enteras.

Finalmente hemos hallado la media de los tiempos de finalización del último commit de los 8 tests, para cada configuración diferente. De este modo obtenemos valores medios con los que podemos sacar conclusiones más generales.

Evaluación de los tests:

Test-math realiza 56894 operaciones de tipo entero que según la frecuencia de utilización de cada unidad funcional de enteros y el número de ellas se distribuyen de la siguiente manera:

- Para 1 unidad funcional de enteros:
  Ent FU1=56894

Y el tiempo de finalización del último commit es 12791780.

3. Para 2 unidades funcionales de enteros:
   Ent FU1=41983
   Ent FU2=16990

Y el tiempo de finalización del último commit es 12538480.

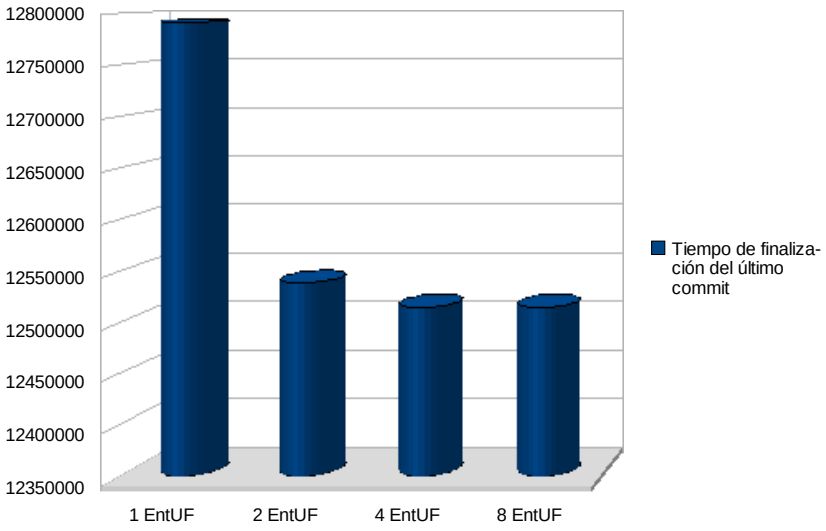
3. Para 4 unidades funcionales de enteros:
   Ent FU1=35739
   Ent FU2=15679
   Ent FU3=5873
   Ent FU4=1340

Y el tiempo de finalización del último commit es 12514480.

- Para 8 unidades funcionales de enteros:
  Ent FU1=35568
  Ent FU2=15635
  Ent FU3=5872

Ent FU4=1340
Ent FU5=216
Ent FU6=1
Ent FU7=0
Ent FU8=0

Y el tiempo de finalización del último commit es 12514480.



Anagram realiza 2624 operaciones de tipo entero que según la frecuencia de utilización de cada unidad funcional de enteros y el número de ellas se distribuyen de la siguiente manera:

- Para 1 unidad funcional de enteros:
  Ent FU1=2624

Y el tiempo de finalización del último commit es 570490.

- Para 2 unidades funcionales de enteros:
  Ent FU1=1930
  Ent FU2=832

Y el tiempo de finalización del último commit es 557790.

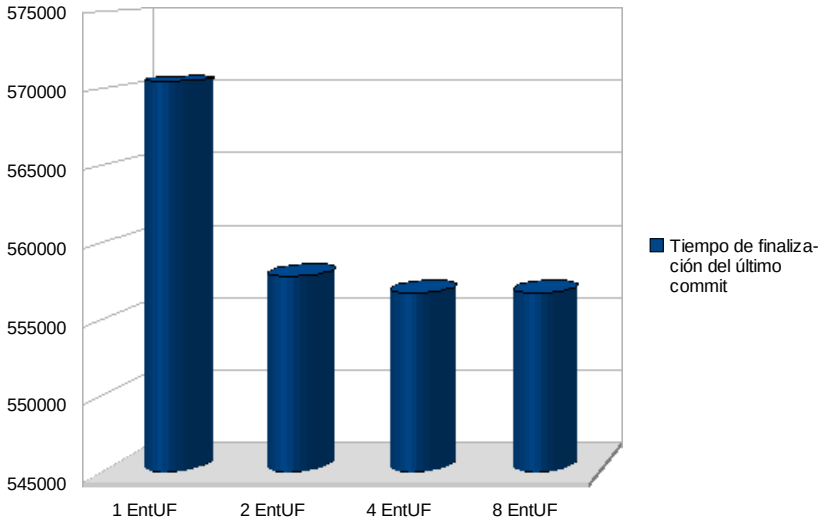
- Para 4 unidades funcionales de enteros:
  Ent FU1=1715
  Ent FU2=742
  Ent FU3=253
  Ent FU4=77

Y el tiempo de finalización del último commit es 556690.

- Para 8 unidades funcionales de enteros:
  Ent FU1=1703
  Ent FU2=740

Ent FU3=253
Ent FU4=77
Ent FU5=13
Ent FU6=1
Ent FU7=0
Ent FU8=0

Y el tiempo de finalización del último commit es 556690.



Test-fmath realiza 21465 operaciones de tipo entero que según la frecuencia de utilización de cada unidad funcional de enteros y el número de ellas se distribuyen de la siguiente manera:

- Para 1 unidad funcional de enteros:
  Ent FU1=21465

Y el tiempo de finalización del último commit es 4965920.

- Para 2 unidades funcionales de enteros:
  Ent FU1=15949
  Ent FU2=6282

Y el tiempo de finalización del último commit es 4878820.

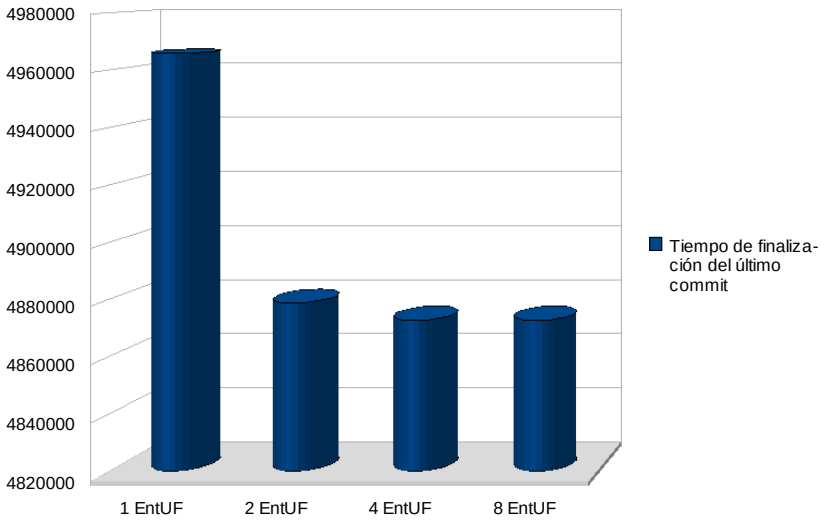
- Para 4 unidades funcionales de enteros:
  Ent FU1=13528
  Ent FU2=5748
  Ent FU3=2127
  Ent FU4=512

Y el tiempo de finalización del último commit es 4872720.

- Para 8 unidades funcionales de enteros:
  Ent FU1=13442

Ent FU2=5741
Ent FU3=2127
Ent FU4=512
Ent FU5=92
Ent FU6=1
Ent FU7=0
Ent FU8=0

Y el tiempo de finalización del último commit es 4872720.



Test-llong realiza 8109 operaciones de tipo entero que según la frecuencia de utilización de cada unidad funcional de enteros y el número de ellas se distribuyen de la siguiente manera:

- Para 1 unidad funcional de enteros:
  Ent FU1=8109

Y el tiempo de finalización del último commit es 1786030.

- Para 2 unidades funcionales de enteros:
  Ent FU1=5832
  Ent FU2=2507

Y el tiempo de finalización del último commit es 1750630.

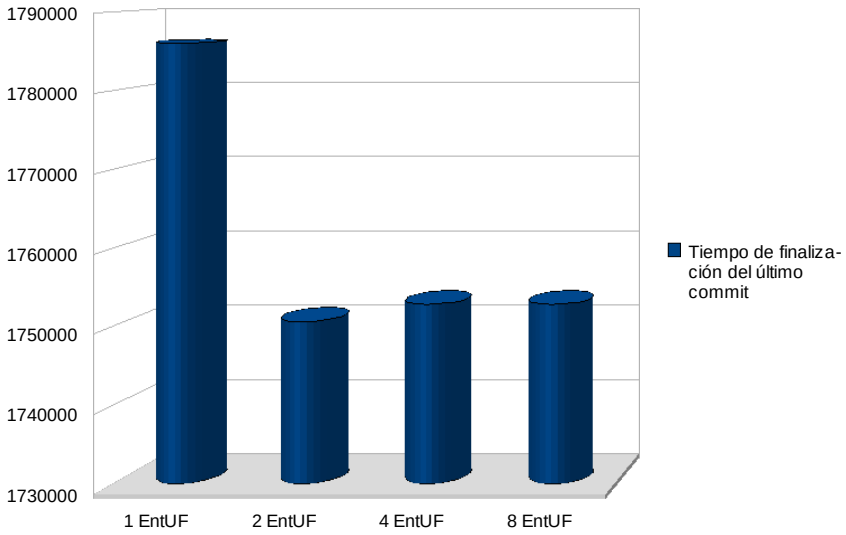
- Para 4 unidades funcionales de enteros:
  Ent FU1=5102
  Ent FU2=2248
  Ent FU3=868
  Ent FU4=175

Y el tiempo de finalización del último commit es 1752830.

- Para 8 unidades funcionales de enteros:
  Ent FU1=5088

Ent FU2=2242
Ent FU3=867
Ent FU4=175
Ent FU5=20
Ent FU6=1
Ent FU7=0
Ent FU8=0

Y el tiempo de finalización del último commit es 1752830.



T**est-args** realiza 6010 operaciones de tipo entero que según la frecuencia de utilización de cada unidad funcional de enteros y el número de ellas se distribuyen de la siguiente manera:

- Para 1 unidad funcional de enteros:
  Ent FU1=6010

Y el tiempo de finalización del último commit es 1397330.

- Para 2 unidades funcionales de enteros:
  Ent FU1=4373
  Ent FU2=2028

Y el tiempo de finalización del último commit es 1364830.

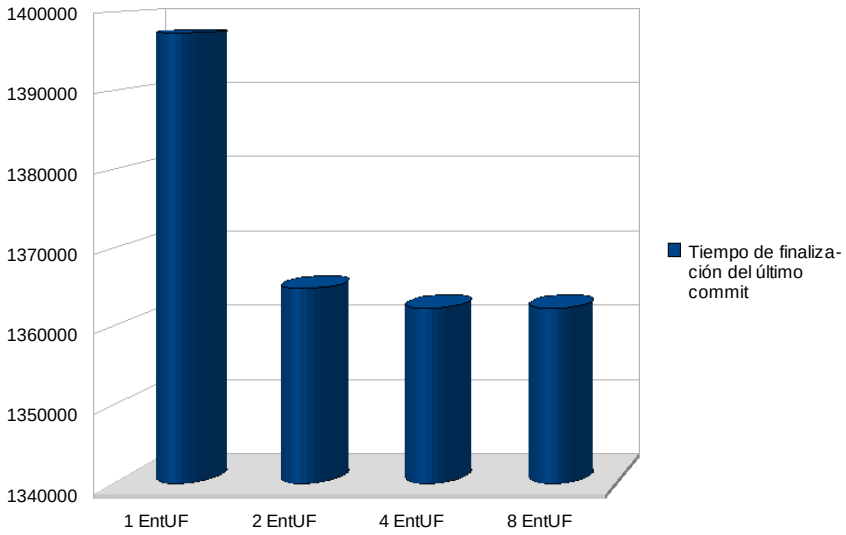
- Para 4 unidades funcionales de enteros:
  Ent FU1=3832
  Ent FU2=1824
  Ent FU3=674
  Ent FU4=152

Y el tiempo de finalización del último commit es 1362330.

- Para 8 unidades funcionales de enteros:
  Ent FU1=3800

Ent FU2=1815
Ent FU3=674
Ent FU4=152
Ent FU5=39
Ent FU6=4
Ent FU7=0
Ent FU8=0

Y el tiempo de finalización del último commit es 1362330.



T**est-lswlr** realiza 2650 operaciones de tipo entero que según la frecuencia de utilización de cada unidad funcional de enteros y el número de ellas se distribuyen de la siguiente manera:

- Para 1 unidad funcional de enteros:
  Ent FU1=2650

Y el tiempo de finalización del último commit es 573510.

- Para 2 unidades funcionales de enteros:
  Ent FU1=1966
  Ent FU2=832

Y el tiempo de finalización del último commit es 563010.

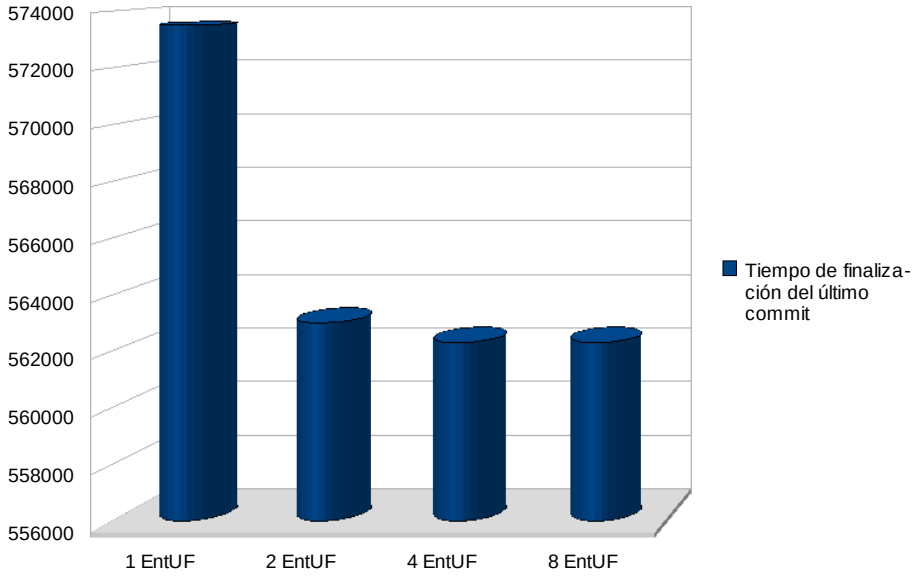
- Para 4 unidades funcionales de enteros:
  Ent FU1=1778
  Ent FU2=751
  Ent FU3=243
  Ent FU4=51

Y el tiempo de finalización del último commit es 562310.

- Para 8 unidades funcionales de enteros:
  Ent FU1=1772
  Ent FU2=745

Ent FU3=242
Ent FU4=51
Ent FU5=12
Ent FU6=1
Ent FU7=0
Ent FU8=0

Y el tiempo de finalización del último commit es 562310.



test-dirent realiza 4158 operaciones de tipo entero que según la frecuencia de utilización de cada unidad funcional de enteros y el número de ellas se distribuyen de la siguiente manera:

- Para 1 unidad funcional de enteros:
  Ent FU1=4158

Y el tiempo de finalización del último commit es 1031050.

- Para 2 unidades funcionales de enteros:
  Ent FU1=3004
  Ent FU2=1419

Y el tiempo de finalización del último commit es 1010950.

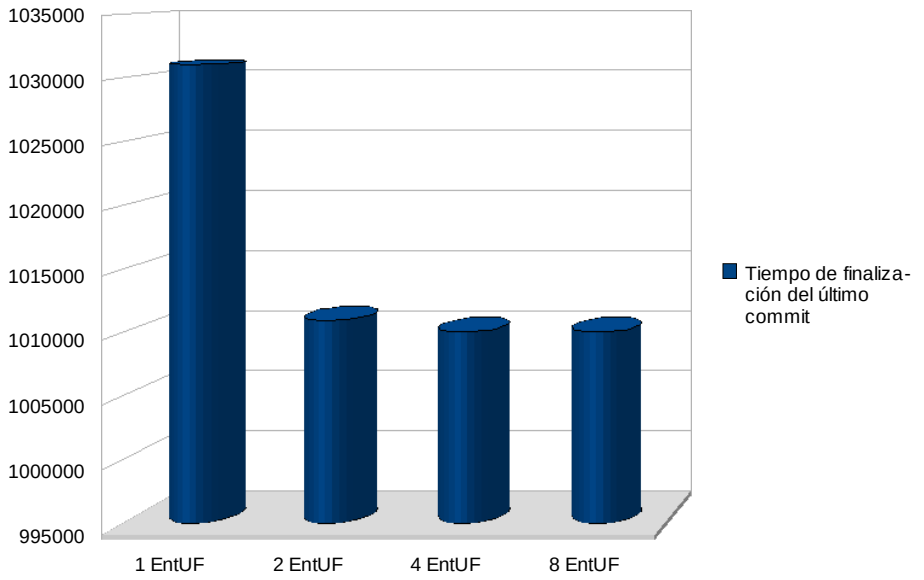
- Para 4 unidades funcionales de enteros:
  Ent FU1=2645
  Ent FU2=1279
  Ent FU3=455
  Ent FU4=93

Y el tiempo de finalización del último commit es 1010050.

- Para 8 unidades funcionales de enteros:
  Ent FU1=2642

Ent FU2=1279
Ent FU3=455
Ent FU4=93
Ent FU5=5
Ent FU6=1
Ent FU7=0
Ent FU8=0

Y el tiempo de finalización del último commit es 1010050.



Test-printf realiza 1040141 operaciones de tipo entero que según la frecuencia de utilización de cada unidad funcional de enteros y el número de ellas se distribuyen de la siguiente manera:

- Para 1 unidad funcional de enteros:
  Ent FU1=1040141

Y el tiempo de finalización del último commit es 240072970.

- Para 2 unidades funcionales de enteros:
  Ent FU1=762839
  Ent FU2=327917

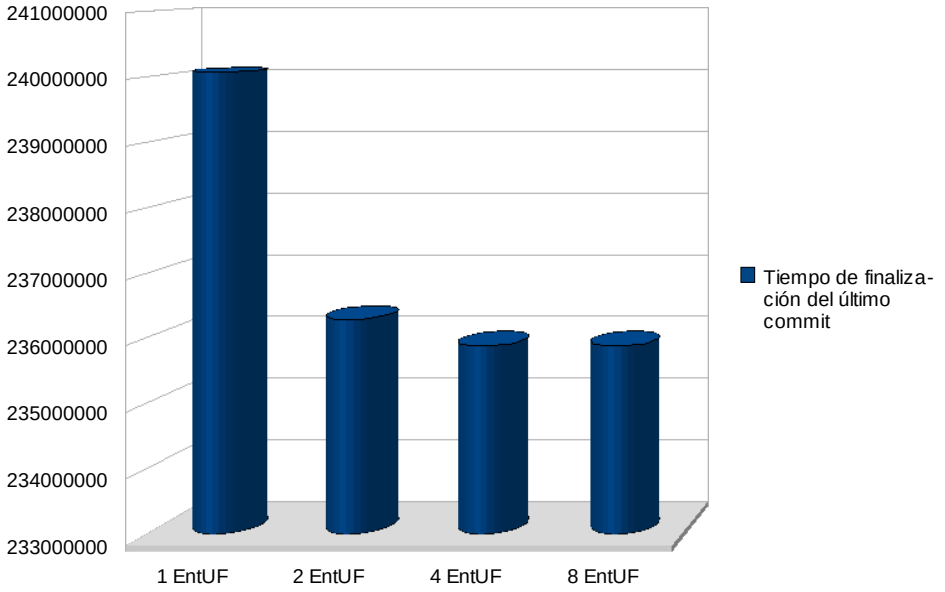
Y el tiempo de finalización del último commit es 236285270.

- Para 4 unidades funcionales de enteros:
  Ent FU1=668132
  Ent FU2=297870
  Ent FU3=114167
  Ent FU4=24086

Y el tiempo de finalización del último commit es 235882170.

- Para 8 unidades funcionales de enteros:
  Ent FU1=664390
  Ent FU2=295603
  Ent FU3=114166
  Ent FU4=24086
  Ent FU5=3797
  Ent FU6=2214
  Ent FU7=0
  Ent FU8=0

Y el tiempo de finalización del último commit es 235882170.



Media de tiempos de todos los tests.

Utilizando una unidad funcional de enteros obtenemos un tiempo medio de finalización del último commit de 32898635 unidades de tiempo.

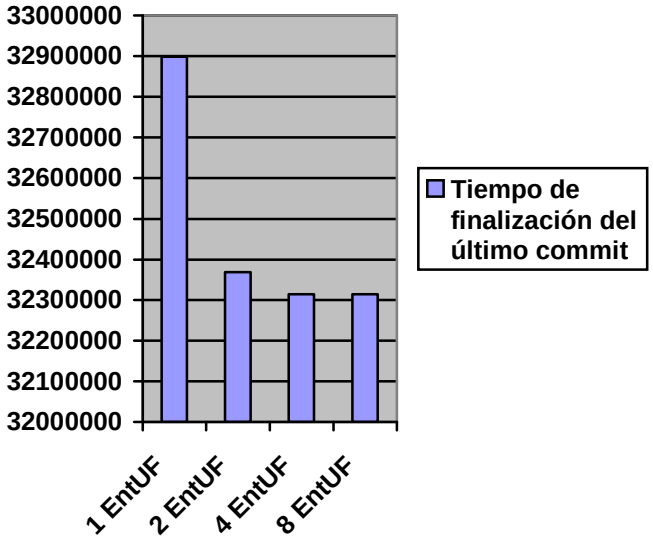
Utilizando dos unidades funcionales de enteros obtenemos un tiempo medio de finalización del último commit de 32368722,5 unidades de tiempo.

Utilizando cuatro unidades funcionales de enteros obtenemos un tiempo medio de finalización del último commit de 32314197,5 unidades de tiempo.

Utilizando ocho unidades funcionales de enteros obtenemos un tiempo medio de finalización del último commit de 32314197,5 unidades de tiempo, exactamente igual que utilizando cuatro unidades funcionales de enteros.

A continuación se muestra el gráfico con estos tiempos para comparar la efectividad de un número determinado de unidades funcionales de enteros según se va duplicando de 1 a 8.

Grafica final con los valores medios:



3.1.4.- Conclusión

Al observar los resultados obtenidos vemos que al aumentar el número de unidades funcionales de enteros de 1 a 2 se produce una notable mejora en tiempo. Al aumentarlo de 2 a 4 se produce también una mejora aunque mucho menor que la anterior. Y al aumentar de 4 a 8 los tiempos, medidos en esas unidades de tiempo, son los mismos. Es decir, que la mejora es tan pequeña que no se puede apreciar en las unidades de tiempo en las que se han medido los resultados.

Sí observamos la ocupación de cada unidad funcional de enteros al pasar de 1 a 2 se reduce considerablemente el uso de la primera unidad funcional de enteros. Al pasar de 2 a 4 la reducción es menor pero sigue habiendo una pequeña reducción en el caso de la primera unidad funcional de enteros a la vez que la hay en la segunda. Al pasar de 4 a 8 la reducción es mucho menor. Se reduce ligeramente el uso de las 4 primeras unidades funcionales de enteros y se reparte la carga en el resto.

Hay que recordar que la frecuencia de uso de una determinada unidad funcional de enteros es el número de veces que se quiere realizar una operación de enteros cuando las unidades anteriores están ocupadas y esta misma está libre. Según el diseño se ocupan las unidades funcionales de enteros en orden, es decir, que no se ocupa la EntUFi sí la EntUFj con j<i (siendo j >=1) está libre.

Podemos concluir diciendo que es rentable utilizar 2 unidades funcionales de enteros antes que 1, ya que la mejora es significativa. Y dependiendo del caso será mejor tener 4 antes que 2, porque hay una cierta mejora pero eso ya depende del coste y de otros factores de fabricación. Lo que es evidente es que con las pruebas ejecutadas no

es rentable utilizar 8 unidades funcionales de enteros antes que 4 porque la mejora es cero, es decir, el resultado es exactamente el mismo.

De los resultados de realizar las pruebas con 8 unidades enteras se puede observar que en ningún caso se usan las dos últimas. Podría ser debido a que no se dé ningún caso en que estén ocupadas las 6 primeras y no haga falta ocupar la séptima, lo cual es poco probable. Seguramente sea otro factor que impida la utilización de más unidades funcionales, posiblemente heredado de SimpleScalar.

Todo esto depende del uso que se le vaya a dar al procesador con respecto a operaciones enteras y del coste de añadir un determinado número de unidades funcionales de enteros. Es un compromiso entre estos dos factores.

3.2.- Protocolos de comunicación

3.2.1.- Estado inicial

Sim-Async, como simulador de una implementación asíncrona de un procesador, cuenta con un conjunto de protocolos de comunicación que posibilitan la conexión entre los diferentes módulos que lo componen.

Además, cuenta con la implementación tanto de un protocolo de dos fases como de uno de cuatro, permitiendo realizar diferentes estudios relativos a la mejora de rendimiento al aplicar uno u otro en diferentes condiciones de ejecución.

En el momento de comienzo de este proyecto, este sistema de comunicación se encontraba implementado en el simulador mediante variables que recibían un valor constante que determinase su tiempo de retardo. Sin embargo, al igual que ocurre con las unidades funcionales, para la correcta simulación del funcionamiento de la comunicación entre los diferentes módulos del sistema, se ha optado por el uso de funciones estadísticas de distribución. Con ello, se permite una aproximación al funcionamiento real del sistema a partir de un conjunto de valores reales tomados como muestra.

3.2.2.- Desarrollo

Se ha ampliado la funcionalidad del sistema incorporando el reconocimiento de las funciones de distribución por parte del protocolo de comunicación. Con ello, los valores de las variables que determinan los retardos de dicho protocolo son asignados ahora de manera estadística y variable. Con ello se consigue una mejora en el funcionamiento global del sistema (en términos de la simulación) ya que se aproxima más a una implementación real con tiempos de comunicación no fijos en diferentes ejecuciones de un mismo conjunto de datos.

Una vez logrado dicho proceso de ampliación se procedió al lanzamiento de algunas pruebas que permitiesen comprobar el funcionamiento de esta. Estas pruebas se realizaron con un pequeño fichero de pruebas que no supone la misma carga de trabajo que un SPEC. La decisión de utilizar este fichero de pruebas viene determinada por los valores y el objetivo de dichas pruebas.

Debido a que no se cuenta con los tiempos de retardo reales que suele invertir este tipo de protocolos, es imposible realizar un estudio comparativo entre implementaciones síncronas y asíncronas. Por el contrario, los actuales ficheros que contienen los valores de las funciones de distribución consisten en un conjunto de valores aleatorios tomados con el propósito de comprobar al menos el correcto funcionamiento de las ampliaciones realizadas.

Por tanto, a continuación se procede a comentar el resultado obtenido con estas pruebas, a falta de valores reales que permitan un estudio con una mayor profundidad.

- Protocolo de 2 fases

La implementación de este protocolo cuenta con dos variables, "tfv" y "tack", que determinan respectivamente el tiempo desde que un dato esta listo para ser enviado al siguiente módulo hasta que este se da por informado y su respuesta aceptando dicho traspaso de información.

En primer lugar se ha procedido con varias simulaciones sucesivas en modo "asíncrono" con valores fijos para los diferentes retardos de las Unidades Funcionales y el protocolo de comunicación. Estas pruebas se han realizado siempre con el mismo conjunto de datos (fichero de pruebas "anagram") con el propósito de observar posibles variaciones en el funcionamiento. Tras ellas se han recogido datos del tiempo de finalización del último commit para dichos lanzamientos.

- Prueba 1 - Tº de finalización del ultimo commit: 5493269
- Prueba 2 - Tº de finalización del ultimo commit: 5493269
- Prueba 3 - Tº de finalización del ultimo commit: 5493269
- Prueba 4 - Tº de finalización del ultimo commit: 5493269
- Prueba 5 - Tº de finalización del ultimo commit: 5493269

Con ello se puede considerar suficiente para determinar que el tiempo de ejecución es sin lugar a dudas idéntico en toda simulación que se realice con dicho conjunto de datos. Esto es, el tiempo de ejecución no fluctúa ni depende de ningún parámetro extra en estas ejecuciones.

A continuación se procede a modificar las condiciones de simulación con el fin de comprobar el correcto funcionamiento del protocolo de comunicación, así como su efecto sobre el funcionamiento global del sistema. Para dicho propósito, se mantuvo un tiempo de retardo fijo en las unidades funcionales y se asignó una función de distribución a cada señal el protocolo. El objetivo es el de comprobar la fluctuación en

los tiempos de finalización que debe ocasionar el protocolo de comunicación del sistema. Los resultados obtenidos son los siguientes:

Asignando una función de distribución al valor de la variable "tfv", se procedió a lanzar dos simulaciones que mostrasen el tiempo de ejecución del mismo conjunto de datos anterior. Los resultados obtenidos son:

- Prueba 1 - Tº de finalización del ultimo commit: 5498316
- Prueba 2 - Tº de finalización del ultimo commit: 5498590

Aunque se trata solo de dos pruebas, es suficiente para comprobar una fluctuación en el tiempo de ejecución de dicho conjunto de instrucciones. Debido a que el único valor variable en la configuración actual es el del protocolo de comunicación, esta fluctuación en los tiempos de finalización se debe a retardos producidos en la comunicación entre módulos.

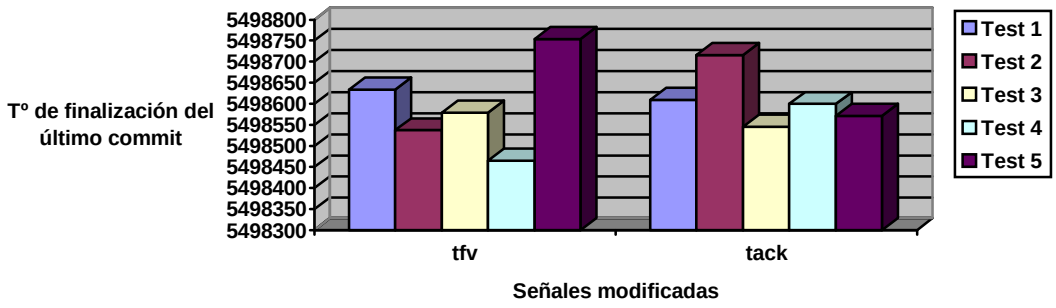
De igual modo, se procedió a fijar el tiempo de "tfv" con un valor constante, asignando una función de distribución a la variable "tack". Ejecutando de nuevo el mismo conjunto de datos, los tiempos recogidos son:

- Prueba 1 - Tº de finalización del ultimo commit: 5498512
- Prueba 2 - Tº de finalización del ultimo commit: 5498536

Con ello se puede comprobar que el tiempo de nuevo fluctúa a pesar de consistir en el mismo conjunto de datos ejecutados de forma sucesiva. Esta fluctuación, de nuevo debido a las condiciones de simulación, viene determinada por el retardo en la comunicación de los diferentes módulos. En esta ocasión debido al tiempo de respuesta del módulo que responde ante una señal "tfv" recibida.

La siguiente tabla muestra esta fluctuación de un modo gráfico, indicando el tiempo de finalización del último commit realizado, según la variable a la que se le ha asignado una función de distribución. En esta gráfica observamos que los tiempos de retardo que producen ambas señales son parecidos, además de observar la fluctuación en el tiempo de ejecución debida al retardo de cada variable indicada.

Tiempos de ejecución con el protocolo de 2 fases



- Protocolo de 4 fases

La implementación de este protocolo cuenta con 4 variables que representan las diferentes señales propias del protocolo "handshake" de 4 fases: "tfv", "tfn", "tack" y "tsync" que determinan los diferentes retardos del protocolo en cuestión.

Al igual que en el caso anterior, inicialmente se ha procedido con varias simulaciones sucesivas en modo "asíncrono" con valores fijos para los diferentes retardos de las Unidades Funcionales y el protocolo de comunicación. Estas pruebas, de nuevo, se han realizado con el fichero de pruebas "anagram" y de nuevo con el propósito de observar posibles variaciones en los retardos. Se han recogido datos del tiempo de finalización del último commit para dichos lanzamientos, produciendo los siguientes resultados:

- Prueba 1 - Tº de finalización del ultimo commit: 5497076
- Prueba 2 - Tº de finalización del ultimo commit: 5497076
- Prueba 3 - Tº de finalización del ultimo commit: 5497076
- Prueba 4 - Tº de finalización del ultimo commit: 5497076
- Prueba 5 - Tº de finalización del ultimo commit: 5497076

Tal y como se observa, bajo estas condiciones de simulación los tiempos de ejecución se mantienen constantes, tal y como era de esperar que sucediese. Esto es debido a que no existe ningún retardo variable asociado al sistema, por lo que no se posibilitan fluctuaciones temporales en la ejecución de instrucciones.

El siguiente paso de nuevo vuelve a ser observar el funcionamiento de cada señal de manera independiente, comprobando como afecta esta variación al funcionamiento del sistema. Para ello se modifican de nuevo las condiciones de simulación con el fin de comprobar el correcto funcionamiento del protocolo implementado, así como su efecto global. Para dicho propósito, se mantuvo un tiempo de retardo fijo en las unidades funcionales y se asignó una función de distribución a cada señal el protocolo. El objetivo es el de comprobar la fluctuación en los tiempos de finalización que debe ocasionar el protocolo de comunicación del sistema. Los resultados obtenidos son los siguientes:

Asignando una función de distribución al valor de la variable "tfv", se procedió a lanzar dos simulaciones que mostrasen el tiempo de ejecución del mismo conjunto de datos anterior. Los resultados obtenidos son:

- Prueba 1 - Tº de finalización del ultimo commit: 5502560
- Prueba 2 - Tº de finalización del ultimo commit: 5502432

Aunque se trata solo de dos pruebas, es suficiente para comprobar una fluctuación en el tiempo de ejecución de dicho conjunto de instrucciones (y por tanto del sistema). Debido a que el único valor variable en la configuración actual es el del protocolo de comunicación, esta fluctuación en los tiempos de finalización se debe a retardos producidos en la comunicación entre módulos.

A continuación se procedió a modificar la señal "tfn", asignándole una función de distribución de la que extraer su valor a la vez que se fijaba el del resto de elementos del sistema de manera constante. El lanzamiento del conjunto de datos produjo los siguientes resultados:

- Prueba 1 - Tº de finalización del ultimo commit: 5502622
- Prueba 2 - Tº de finalización del ultimo commit: 5502448

De igual manera, se procedió a fijar la función de distribución del resto de variables del protocolo, fijando todas las demás para comprobar el correcto funcionamiento de estas. Los resultados obtenidos en las pruebas ejecutadas para las variables restantes son:

Resultados para la variable "tack":

- Prueba 1 - Tº de finalización del ultimo commit: 5502661
- Prueba 2 - Tº de finalización del ultimo commit: 5502596

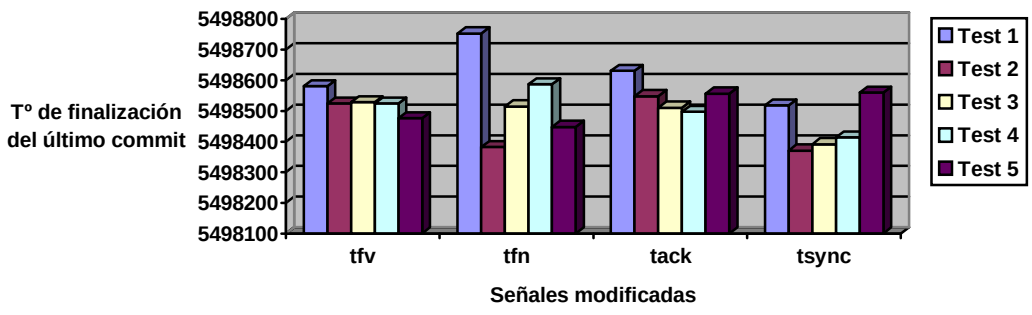
Resultados para la variable "tsync":

- Prueba 1 - Tº de finalización del ultimo commit: 5502441
- Prueba 2 - Tº de finalización del ultimo commit: 5502602

Con estas pruebas y sus resultados se observa que, efectivamente, el tiempo de ejecución del sistema se ve influido por los retardos existentes en las señales de comunicación entre los diferentes módulos.

La siguiente tabla muestra esta fluctuación de un modo gráfico, indicando el tiempo de finalización del último commit realizado, según la variable a la que se le ha asignado una función de distribución. En esta gráfica observamos que los tiempos de retardo que producen estas señales son parecidos, además de observar la fluctuación en el tiempo de ejecución debida al retardo de cada variable indicada.

Tiempos de ejecución con el protocolo de 4 fases



3.2.3.- Conclusión

Tras la ampliación realizada en la funcionalidad del simulador, y tras el lanzamiento de las pruebas indicadas, se puede asegurar el correcto funcionamiento del protocolo de comunicación en el sistema. Este protocolo, a diferencia de lo que ocurría inicialmente, ya no genera retardos constantes en todas sus señales. En su lugar, los retardos generados por las señales intercambiadas entre los diferentes módulos que se comunican entre si son obtenidos a partir de una función estadística de distribución.

Con ello se consigue una simulación más real del comportamiento de una microarquitectura asíncrona, tal y como se observa en la fluctuación de los tiempos de ejecución de los conjuntos de instrucciones empleados como pruebas.

La falta de retardos reales con las que construir las funciones de distribución impide un desarrollo más extenso de este estudio. Con dichos valores sería posible comparar el tiempo de ejecución del sistema, con los retardos generados por el protocolo de comunicación, con el tiempo de ejecución en modo síncrono. Con ello se podría realizar un análisis más extenso de la influencia de estos protocolos en el rendimiento del sistema así como la posible mejora que podría suponer.

3.3.- Pruebas

3.3.1.- Introducción

Las pruebas de rendimiento, ejecutadas bajo las actuales condiciones del simulador, se han realizado empleando el conjunto de "benchmarks" standard conocidos con el nombre de "SPEC 2000".

Utilizando un protocolo de comunicación de 4 fases, y con un número de unidades funcionales de la microarquitectura totalmente parametrizable, las siguientes páginas detallan las pruebas realizadas. El objetivo de las mismas es el estudio de rendimiento del procesador dependiendo del número de unidades de enteros disponibles en el mismo.

3.3.2.- Resultados experimentales

Mcf

- Simulación Asíncrona: Realiza 146.376.095 operaciones de tipo entero.
- Simulación Síncrona: Realiza 147.690.583 operaciones de tipo entero.

A continuación se muestra una comparación entre ambas simulaciones, mostrando el número de unidades funcionales y la frecuencia de utilización de cada una según la configuración:

- Para 1 unidad funcional de enteros:
  - F. de utilización de Ent FU1 (conf. Asíncrona): `146376095`
  - F. de utilización de Ent FU1 (conf. Síncrona): `147690583`

  - T. de finalización del último commit (conf. Asíncrona): `300859621133`
  - T. de finalización del último commit (conf. Síncrona): `580954848000`

- Para 2 unidades funcionales de enteros:
  - F. de utilización de Ent FU (conf. Asíncrona):
    ```
    Ent FU1 = 102116127
    Ent FU2 = 49569102
    ```
  - F. de utilización de Ent FU (conf. Síncrona):
    ```
    Ent FU1 = 108704836
    Ent FU2 = 44597164
    ```

  - T. de finalización del último commit (conf. Asíncrona): `300020595157`
  - T. de finalización del último commit (conf. Síncrona): `578356061000`

- Para 4 unidades funcionales de enteros:

F. de utilización de Ent FU (conf. Asíncrona):

```
Ent FU1 = 94251511
Ent FU2 = 41353311
Ent FU3 = 11705461
Ent FU4 = 4576675
```

F. de utilización de Ent FU (conf. Síncrona):

```
Ent FU1 = 96345323
Ent FU2 = 45576109
Ent FU3 = 9373024
Ent FU4 = 2597633
```

T. de finalización del último commit (conf. Asíncrona): `298826786688`
T. de finalización del último commit (conf. Síncrona): `576055417000`

- Para 8 unidades funcionales de enteros:

F. de utilización de Ent FU (conf. Asíncrona):

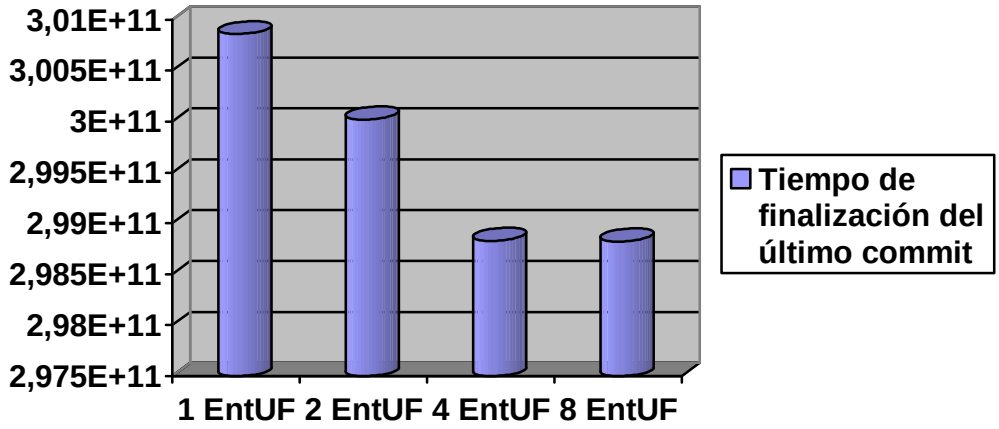
```
Ent FU1=94050971
Ent FU2=41210139
Ent FU3=11666999
Ent FU4=4554806
Ent FU5=476114
Ent FU6=3590
Ent FU7=0
Ent FU8=0
```

F. de utilización de Ent FU (conf. Síncrona):

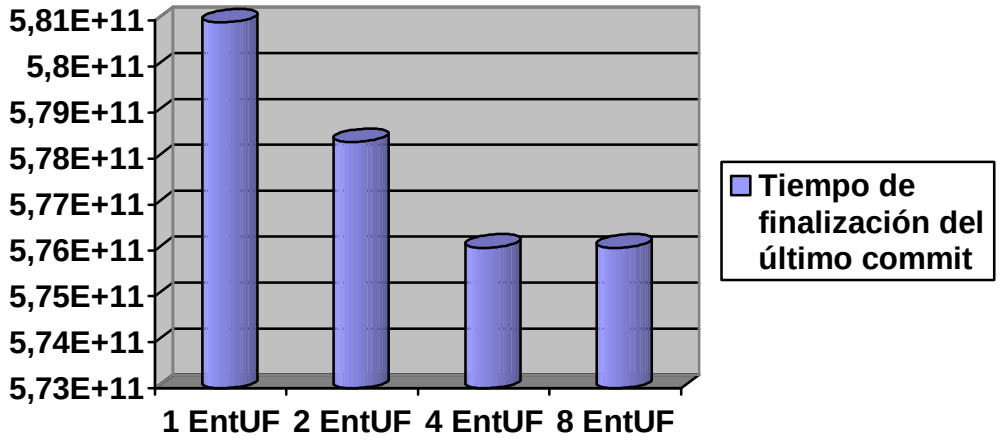
```
Ent FU1=96336737
Ent FU2=45576101
Ent FU3=9373024
Ent FU4=2597633
Ent FU5=8597
Ent FU6=7
Ent FU7=0
Ent FU8=0
```

T. de finalización del último commit (conf. Asíncrona): `298820705768`
T. de finalización del último commit (conf. Síncrona): `576055417000`

- **Ejecución asíncrona:**



- **Ejecución síncrona:**



Crafty

- Simulación Asíncrona: Realiza 83911950 operaciones de tipo entero.
- Simulación Síncrona: Realiza 84649212 operaciones de tipo entero.

A continuación se muestra una comparación entre ambas simulaciones, mostrando el número de unidades funcionales y la frecuencia de utilización de cada una:

- Para 1 unidad funcional de enteros:
  F. de utilización de Ent FU1 (conf. Asíncrona): `83911950`
  F. de utilización de Ent FU1 (conf. Síncrona): `84649212`

  T. de finalización del último commit (conf. Asíncrona): `102934492094`
  T. de finalización del último commit (conf. Síncrona): `199371835000`

- Para 2 unidades funcionales de enteros:
  F. de utilización de Ent FU (conf. Asíncrona):
  ```
  Ent FU1=57411544
  Ent FU2=30804418
  ```
  F. de utilización de Ent FU (conf. Síncrona):
  ```
  Ent FU1=60863622
  Ent FU2=28533440
  ```

  T. de finalización del último commit (conf. Asíncrona): `102058535512`
  T. de finalización del último commit (conf. Síncrona): `197368507000`

- Para 4 unidades funcionales de enteros:
  F. de utilización de Ent FU (conf. Asíncrona):
  ```
  Ent FU1=52470457
  Ent FU2=25788862
  Ent FU3=8548294
  Ent FU4=2205149
  ```
  F. de utilización de Ent FU1 (conf. Síncrona):
  ```
  Ent FU1=54811285
  Ent FU2=26000970
  Ent FU3=7650086
  Ent FU4=1981168
  ```

  T. de finalización del último commit (conf. Asíncrona): `101960493659`
  T. de finalización del último commit (conf. Síncrona): `197299741000`

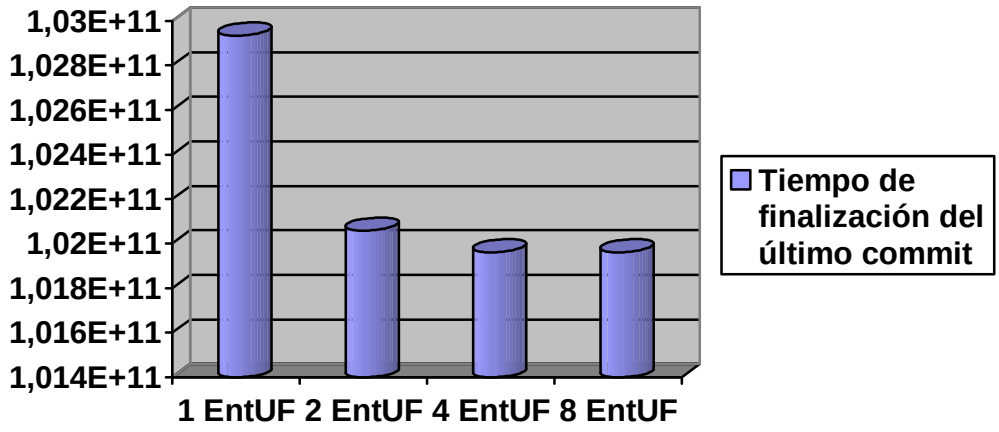
- Para 8 unidades funcionales de enteros:
  F. de utilización de Ent FU (conf. Asíncrona):
  ```
  Ent FU1=52324226
  Ent FU2=25652882
  Ent FU3=8488289
  Ent FU4=2191783
  Ent FU5=339962
  Ent FU6=23248
  Ent FU7=0
  Ent FU8=0
  ```

F. de utilización de Ent FU1 (conf. Síncrona):

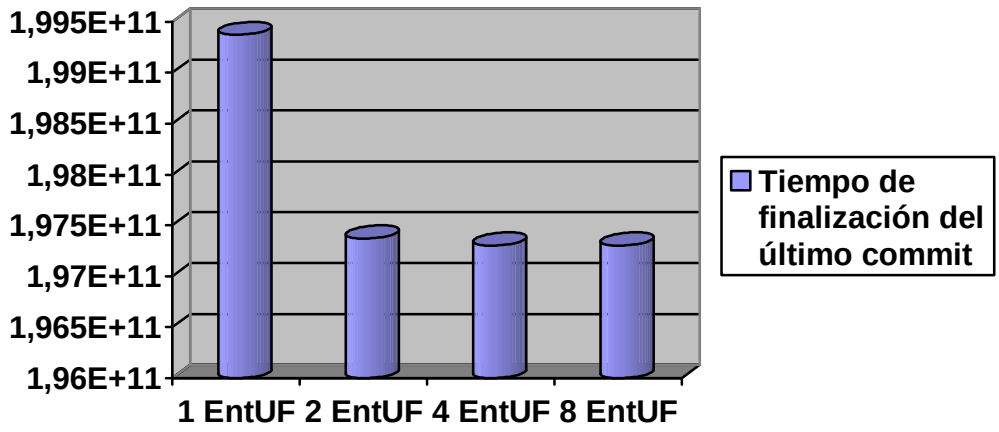
```
FU1=54475363
FU2=25974636
FU3=7650086
FU4=1981168
FU5=353597
FU6=25962
FU7=0
FU8=0
```

T. de finalización del último commit (conf. Asíncrona): `101960640934`
T. de finalización del último commit (conf. Síncrona): `197299745000`

- Ejecución asíncrona:



- Ejecución síncrona:



Gap

- Simulación Asíncrona: Realiza 46139178 operaciones de tipo entero.
- Simulación Síncrona: Realiza 46576158 operaciones de tipo entero.

A continuación se muestra una comparación entre ambas simulaciones, mostrando el número de unidades funcionales y la frecuencia de utilización de cada una:

- Para 1 unidad funcional de enteros:
  F. de utilización de Ent FU1 (conf. Asíncrona): `46139178`
  F. de utilización de Ent FU1 (conf. Síncrona): `46576158`

  T. de finalización del último commit (conf. Asíncrona): `117052847454`
  T. de finalización del último commit (conf. Síncrona): `223841712000`

- Para 2 unidades funcionales de enteros:
  F. de utilización de Ent FU1 (conf. Asíncrona):
  ```
  Ent FU1=33214369
  Ent FU2=14219567
  ```
  F. de utilización de Ent FU1 (conf. Síncrona):
  ```
  Ent FU1=34099525
  Ent FU2=13599797
  ```

  T. de finalización del último commit (conf. Asíncrona): `116730041811`
  T. de finalización del último commit (conf. Síncrona): `221488859000`

- Para 4 unidades funcionales de enteros:
  F. de utilización de Ent FU (conf. Asíncrona):
  ```
  Ent FU1=31519690
  Ent FU2=12351783
  Ent FU3=3081200
  Ent FU4=747729
  ```
  F. de utilización de Ent FU1 (conf. Síncrona):
  ```
  Ent FU1=31769888
  Ent FU2=12783708
  Ent FU3=2777430
  Ent FU4=641270
  ```

  T. de finalización del último commit (conf. Asíncrona): `116665033538`
  T. de finalización del último commit (conf. Síncrona): `221432710000`

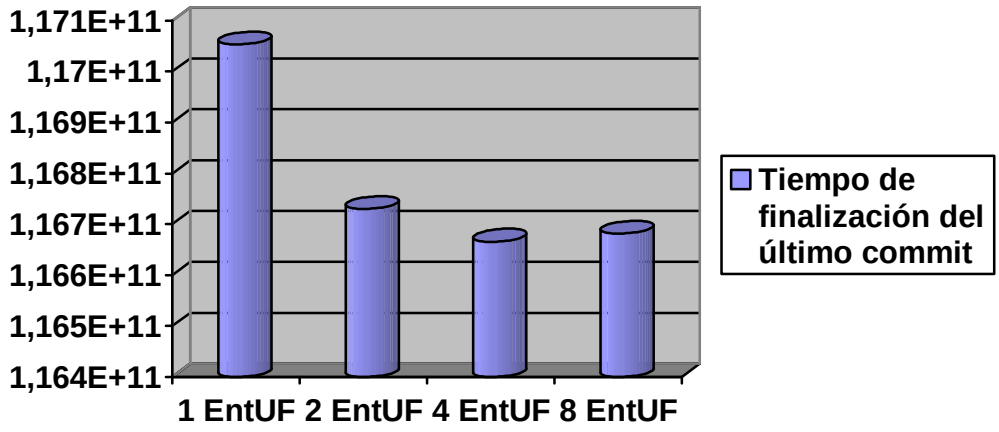
- Para 8 unidades funcionales de enteros:
  F. de utilización de Ent FU (conf. Asíncrona):
  ```
  Ent FU1=31349653
  Ent FU2=12189864
  Ent FU3=3015471
  Ent FU4=723430
  Ent FU5=196827
  Ent FU6=24136
  Ent FU7=0
  Ent FU8=0
  ```

F. de utilización de Ent FU (conf. Síncrona):

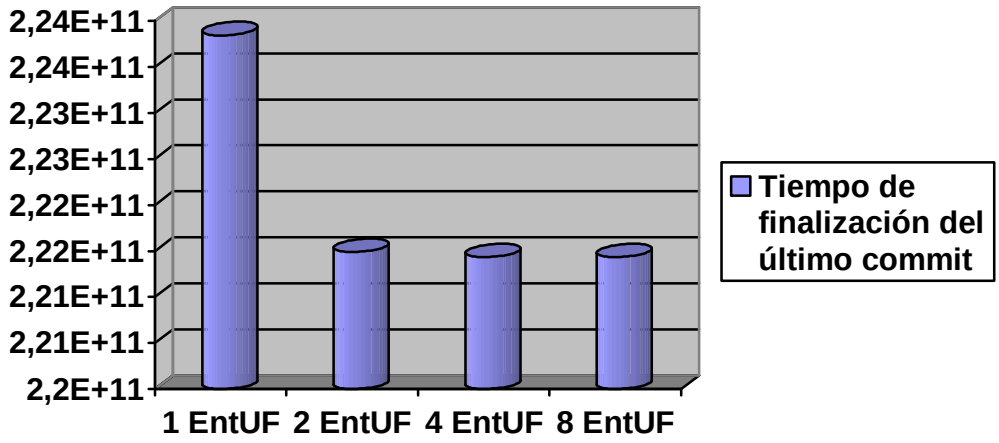
```
Ent FU1=31585506
Ent FU2=12751513
Ent FU3=2777430
Ent FU4=641270
Ent FU5=185835
Ent FU6=31313
Ent FU7=0
Ent FU8=0
```

T. de finalización del último commit (conf. Asíncrona): `116680804816`
T. de finalización del último commit (conf. Síncrona): `221432710000`

- Ejecución asíncrona:



- Ejecución síncrona:



Eon

- Simulación Asíncrona: Realiza 57186186 operaciones de tipo entero.
- Simulación Síncrona: Realiza 57503250 operaciones de tipo entero.

A continuación se muestra una comparación entre ambas simulaciones, mostrando el número de unidades funcionales y la frecuencia de utilización de cada una:

- Para 1 unidad funcional de enteros:
  F. de utilización de Ent FU1 (conf. Asíncrona): `57186186`
  F. de utilización de Ent FU1 (conf. Síncrona): `57503250`

  T. de finalización del último commit (conf. Asíncrona): `164197596253`
  T. de finalización del último commit (conf. Síncrona): `314087092000`

- Para 2 unidades funcionales de enteros:
  F. de utilización de Ent FU (conf. Asíncrona):
  ```
  Ent FU1=38925290
  Ent FU2=20080765
  ```
  F. de utilización de Ent FU (conf. Síncrona):
  ```
  Ent FU1=40989120
  Ent FU2=18018321
  ```

  T. de finalización del último commit (conf. Asíncrona): `164050842666`
  T. de finalización del último commit (conf. Síncrona): `313556630000`

- Para 4 unidades funcionales de enteros:
  F. de utilización de Ent FU (conf. Asíncrona):
  ```
  Ent FU1=35609214
  Ent FU2=16630349
  Ent FU3=5480221
  Ent FU4=1557800
  ```
  F. de utilización de Ent FU (conf. Síncrona):
  ```
  Ent FU1=38288169
  Ent FU2=16092253
  Ent FU3=4343344
  Ent FU4=618230
  ```

  T. de finalización del último commit (conf. Asíncrona): `163986586519`
  T. de finalización del último commit (conf. Síncrona): `313516240000`

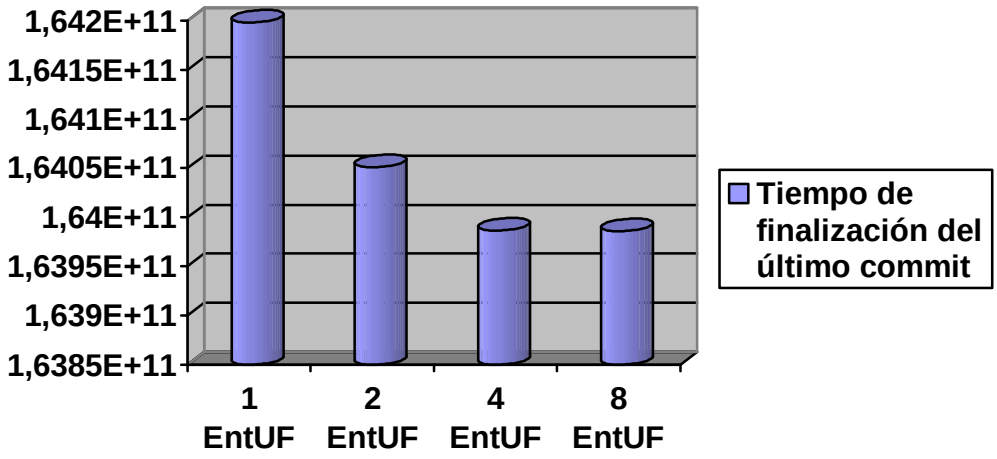
- Para 8 unidades funcionales de enteros:
  F. de utilización de Ent FU (conf. Asíncrona):
  ```
  Ent FU1=35459227
  Ent FU2=16471045
  Ent FU3=5404529
  Ent FU4=1536180
  Ent FU5=370294
  Ent FU6=40795
  Ent FU7=0
  Ent FU8=0
  ```

F. de utilización de Ent FU (conf. Síncrona):

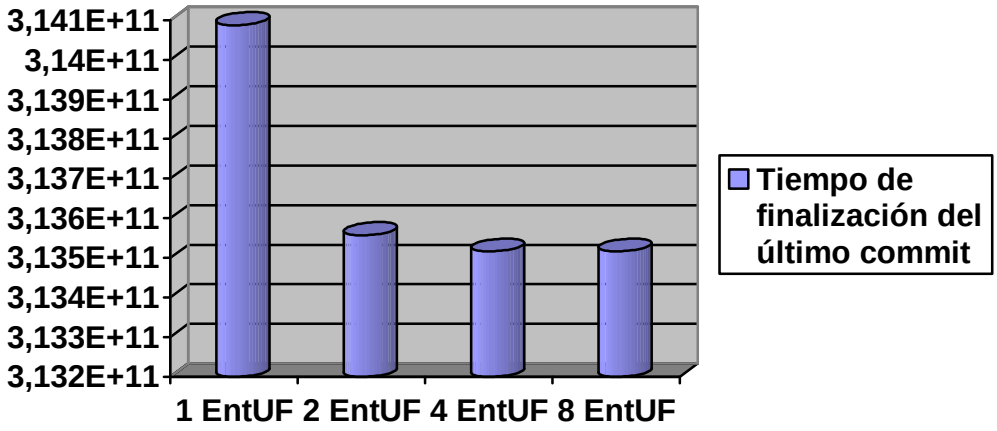
```
Ent FU1=38283880
Ent FU2=16089401
Ent FU3=4343344
Ent FU4=618230
Ent FU5=5271
Ent FU6=2447
Ent FU7=0
Ent FU8=0
```

T. de finalización del último commit (conf. Asíncrona): `163985849168`
T. de finalización del último commit (conf. Síncrona): `313516236000`

- Ejecución asíncrona:



- Ejecución síncrona:



Perlbmk

- Simulación Asíncrona: Realiza 136600383 operaciones de tipo entero.
- Simulación Síncrona: Realiza 137191347 operaciones de tipo entero.

A continuación se muestra una comparación entre ambas simulaciones, mostrando el número de unidades funcionales y la frecuencia de utilización de cada una:

- Para 1 unidad funcional de enteros:
  - F. de utilización de Ent FU1 (conf. Asíncrona): `136600383`
  - F. de utilización de Ent FU1 (conf. Síncrona): `137191347`

  - T. de finalización del último commit (conf. Asíncrona): `263384024481`
  - T. de finalización del último commit (conf. Síncrona): `495112066000`

- Para 2 unidades funcionales de enteros:
  - F. de utilización de Ent FU (conf. Asíncrona):
    ```
    Ent FU1=95880351
    Ent FU2=44463084
    ```
  - F. de utilización de Ent FU (conf. Síncrona):
    ```
    Ent FU1=102117084
    Ent FU2=39884217
    ```

  - T. de finalización del último commit (conf. Asíncrona): `261901692810`
  - T. de finalización del último commit (conf. Síncrona): `490088198000`

- Para 4 unidades funcionales de enteros:
  - F. de utilización de Ent FU (conf. Asíncrona):
    ```
    Ent FU1=88899822
    Ent FU2=37414328
    Ent FU3=11884182
    Ent FU4=2876393
    ```
  - F. de utilización de Ent FU (conf. Síncrona):
    ```
    Ent FU1=92764090
    Ent FU2=36288899
    Ent FU3=11390082
    Ent FU4=2497271
    ```

  - T. de finalización del último commit (conf. Asíncrona): `261432192154`
  - T. de finalización del último commit (conf. Síncrona): `489414493000`

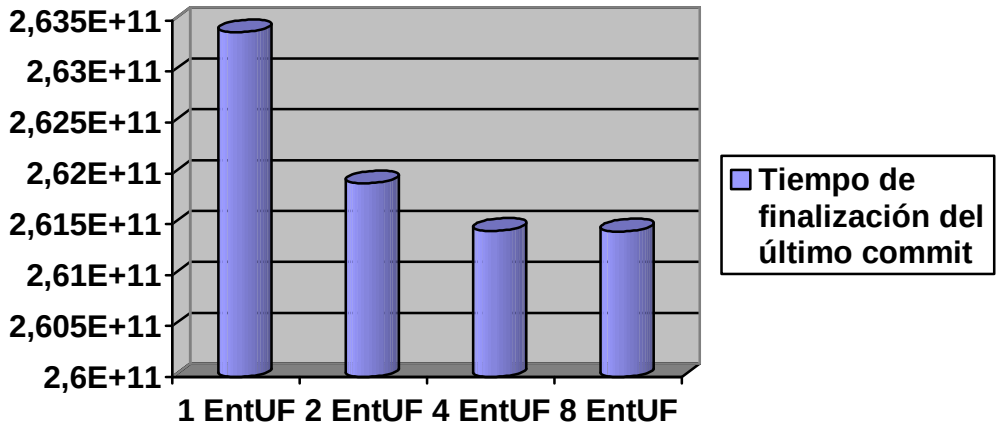
- Para 8 unidades funcionales de enteros:
  - F. de utilización de Ent FU (conf. Asíncrona):
    ```
    Ent FU1=88712321
    Ent FU2=37252400
    Ent FU3=11765785
    Ent FU4=2832761
    Ent FU5=483434
    Ent FU6=39260
    Ent FU7=0
    Ent FU8=0
    ```

F. de utilización de Ent FU (conf. Síncrona):

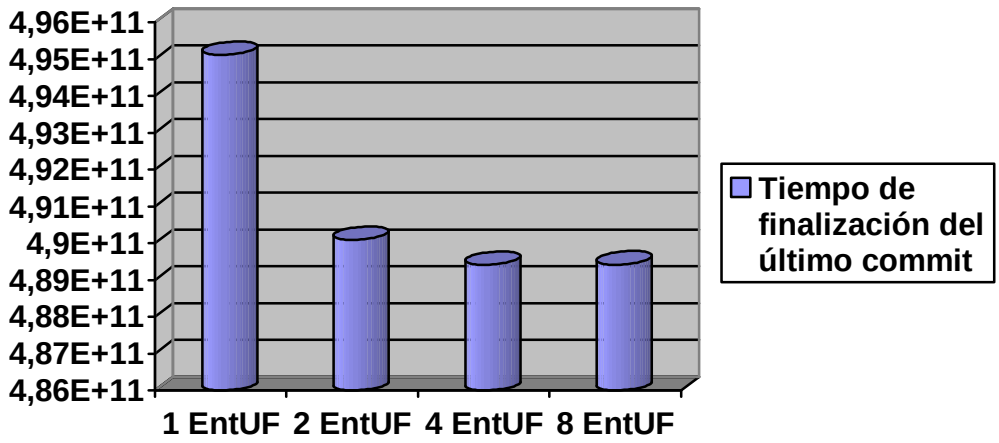
```
Ent FU1=92213784
Ent FU2=36151688
Ent FU3=11390201
Ent FU4=2497301
Ent FU5=689308
Ent FU6=93
Ent FU7=0
Ent FU8=0
```

T. de finalización del último commit (conf. Asíncrona): `261426419834`
T. de finalización del último commit (conf. Síncrona): `489419756000`

- Ejecución asíncrona:



- Ejecución síncrona:



Vortex

- Simulación Asíncrona: Realiza 342685 operaciones de tipo entero.
- Simulación Síncrona: Realiza 348778 operaciones de tipo entero.

A continuación se muestra una comparación entre ambas simulaciones, mostrando el número de unidades funcionales y la frecuencia de utilización de cada una:

- Para 1 unidad funcional de enteros:
  F. de utilización de Ent FU1 (conf. Asíncrona): `342685`
  F. de utilización de Ent FU1 (conf. Síncrona): `348778`

  T. de finalización del último commit (conf. Asíncrona): `551351579`
  T. de finalización del último commit (conf. Síncrona): `1068165000`

- Para 2 unidades funcionales de enteros:
  F. de utilización de Ent FU (conf. Asíncrona):
  ```
  Ent FU1=240438
  Ent FU2=125486
  ```
  F. de utilización de Ent FU (conf. Síncrona):
  ```
  Ent FU1=258379
  Ent FU2=113789
  ```

  T. de finalización del último commit (conf. Asíncrona): `548427664`
  T. de finalización del último commit (conf. Síncrona): `1056683000`

- Para 4 unidades funcionales de enteros:
  F. de utilización de Ent FU (conf. Asíncrona):
  ```
  Ent FU1=222059
  Ent FU2=105205
  Ent FU3=34150
  Ent FU4=8459
  ```
  F. de utilización de Ent FU (conf. Síncrona):
  ```
  Ent FU1=234029
  Ent FU2=105588
  Ent FU3=30827
  Ent FU4=6543
  ```

  T. de finalización del último commit (conf. Asíncrona): `547755031`
  T. de finalización del último commit (conf. Síncrona): `1055342000`

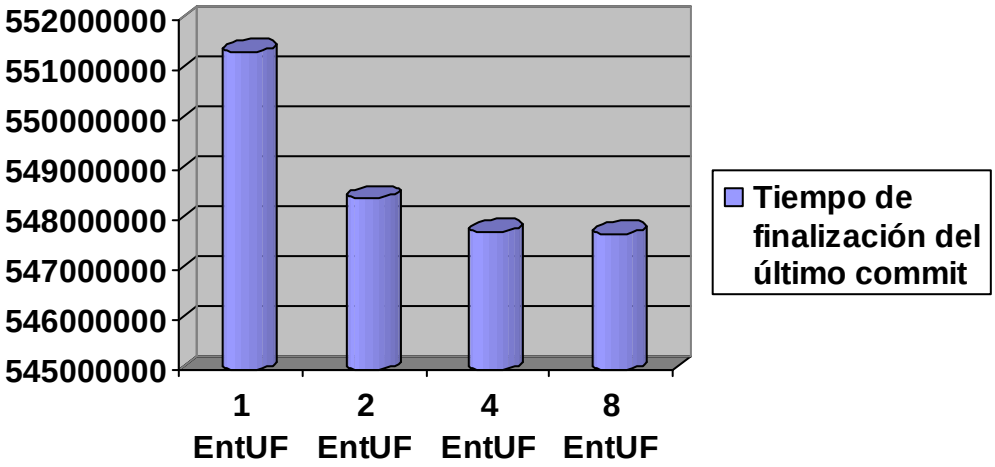
- Para 8 unidades funcionales de enteros:
  F. de utilización de Ent FU (conf. Asíncrona):
  ```
  Ent FU1=221358
  Ent FU2=104608
  Ent FU3=33924
  Ent FU4=8399
  Ent FU5=1463
  Ent FU6=206
  Ent FU7=0
  Ent FU8=0
  ```

F. de utilización de Ent FU (conf. Síncrona):

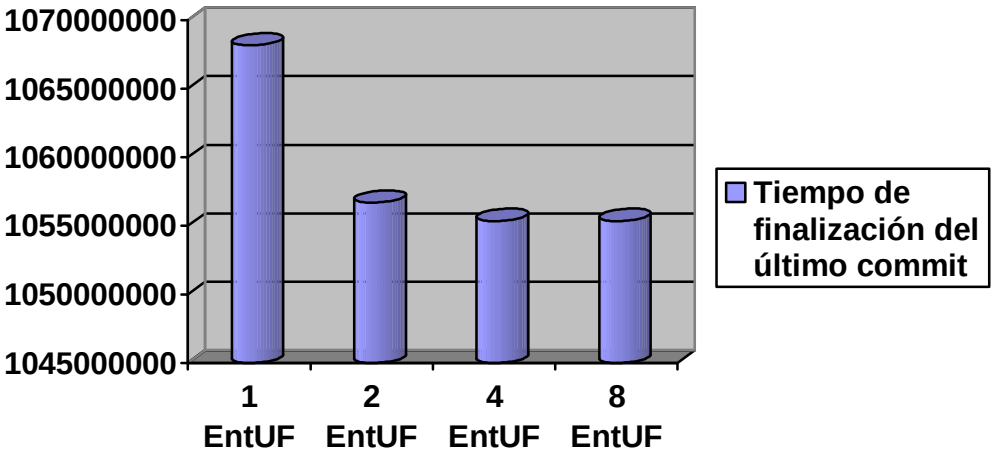
```
Ent FU1=233207
Ent FU2=105478
Ent FU3=30827
Ent FU4=6543
Ent FU5=1141
Ent FU6=12
Ent FU7=0
Ent FU8=0
```

T. de finalización del último commit (conf. Asíncrona): `547714747`
T. de finalización del último commit (conf. Síncrona): `1055342000`

- Ejecución asíncrona:



- Ejecución síncrona:



Vpr

- Simulación Asíncrona: Realiza 165081127 operaciones de tipo entero.
- Simulación Síncrona: Realiza 166923565 operaciones de tipo entero.

A continuación se muestra una comparación entre ambas simulaciones, mostrando el número de unidades funcionales y la frecuencia de utilización de cada una:

- Para 1 unidad funcional de enteros:
  F. de utilización de Ent FU1 (conf. Asíncrona): `165081127`
  F. de utilización de Ent FU1 (conf. Síncrona): `166923565`

  T. de finalización del último commit (conf. Asíncrona): `254251588514`
  T. de finalización del último commit (conf. Síncrona): `486446617000`

- Para 2 unidades funcionales de enteros:
  F. de utilización de Ent FU (conf. Asíncrona):
  ```
  Ent FU1=118116764
  Ent FU2=50279709
  ```
  F. de utilización de Ent FU (conf. Síncrona):
  ```
  Ent FU1=125426490
  Ent FU2=44951786
  ```

  T. de finalización del último commit (conf. Asíncrona): `250987548573`
  T. de finalización del último commit (conf. Síncrona): `479511189000`

- Para 4 unidades funcionales de enteros:
  F. de utilización de Ent FU1 (conf. Asíncrona):
  ```
  Ent FU1=110175584
  Ent FU2=41426877
  Ent FU3=14237375
  Ent FU4=3438243
  ```
  F. de utilización de Ent FU (conf. Síncrona):
  ```
  Ent FU1=114366889
  Ent FU2=41905874
  Ent FU3=13675928
  Ent FU4=2382210
  ```

  T. de finalización del último commit (conf. Asíncrona): `250485977070`
  T. de finalización del último commit (conf. Síncrona): `478202630000`

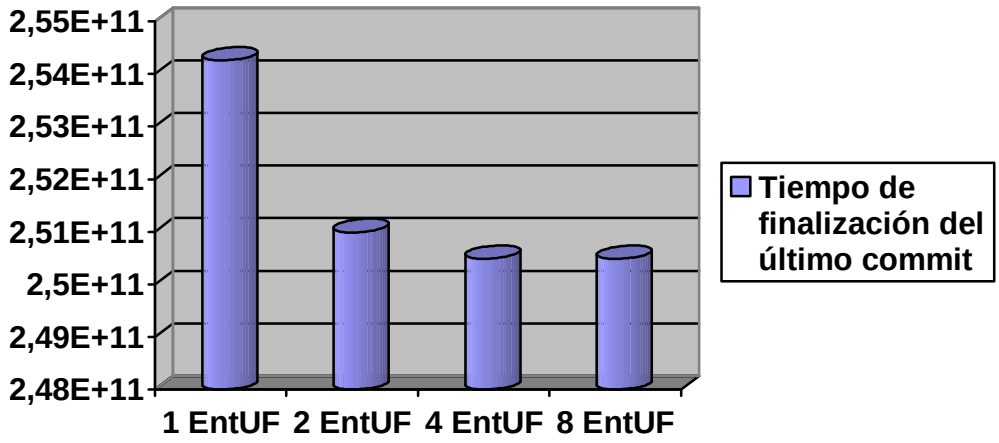
- Para 8 unidades funcionales de enteros:
  F. de utilización de Ent FU (conf. Asíncrona):
  ```
  Ent FU1=109971211
  Ent FU2=41280443
  Ent FU3=14154332
  Ent FU4=3422355
  Ent FU5=420392
  Ent FU6=37873
  Ent FU7=0
  Ent FU8=0
  ```

F. de utilización de Ent FU (conf. Síncrona):

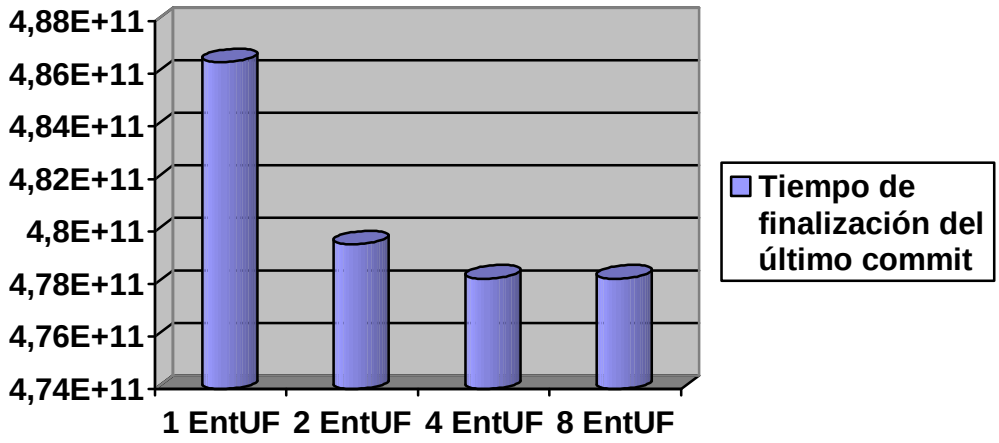
```
Ent FU1=113624633
Ent FU2=41905310
Ent FU3=13675928
Ent FU4=2382210
Ent FU5=789563
Ent FU6=561
Ent FU7=0
Ent FU8=0
```

T. de finalización del último commit (conf. Asíncrona): `250481814052`
T. de finalización del último commit (conf. Síncrona): `478202630000`

- Ejecución asíncrona:



- Ejecución síncrona:



Ammp

- Simulación Asíncrona: Realiza 47795078 operaciones de tipo entero.
- Simulación Síncrona: Realiza 36523232 operaciones de tipo entero.

A continuación se muestra una comparación entre ambas simulaciones, mostrando el número de unidades funcionales y la frecuencia de utilización de cada una:

- Para 1 unidad funcional de enteros:
  F. de utilización de Ent FU1 (conf. Asíncrona): `47795078`
  F. de utilización de Ent FU1 (conf. Síncrona): `36523232`

  T. de finalización del último commit (conf. Asíncrona): `54443106765`
  T. de finalización del último commit (conf. Síncrona): `105872870000`

- Para 2 unidades funcionales de enteros:
  F. de utilización de Ent FU (conf. Asíncrona):
  ```
  Ent FU1=33273817
  Ent FU2=16126824
  ```
  F. de utilización de Ent FU (conf. Síncrona):
  ```
  Ent FU1=18385218
  Ent FU2=7696555
  ```

  T. de finalización del último commit (conf. Asíncrona): `54366368506`
  T. de finalización del último commit (conf. Síncrona): `104630280000`

- Para 4 unidades funcionales de enteros:
  F. de utilización de Ent FU (conf. Asíncrona):
  ```
  Ent FU1=30587779
  Ent FU2=13586535
  Ent FU3=4447389
  Ent FU4=895323
  ```
  F. de utilización de Ent FU (conf. Síncrona):
  ```
  Ent FU1=31405510
  Ent FU2=14149734
  Ent FU3=7500887
  Ent FU4=1584167
  ```

  T. de finalización del último commit (conf. Asíncrona): `54161181167`
  T. de finalización del último commit (conf. Síncrona): `104379123000`

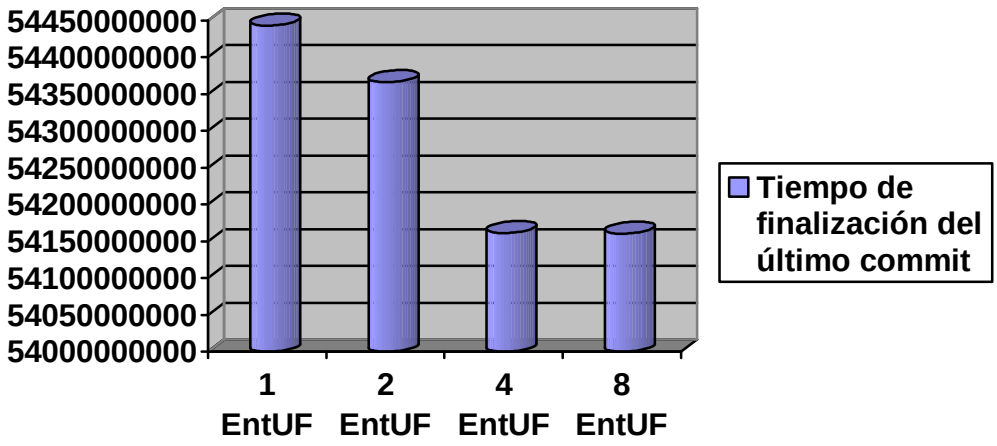
- Para 8 unidades funcionales de enteros:
  F. de utilización de Ent FU (conf. Asíncrona):
  ```
  Ent FU1=30577665
  Ent FU2=13574089
  Ent FU3=4439900
  Ent FU4=893578
  Ent FU5=26916
  Ent FU6=2956
  Ent FU7=0
  Ent FU8=0
  ```

F. de utilización de Ent FU (conf. Síncrona):

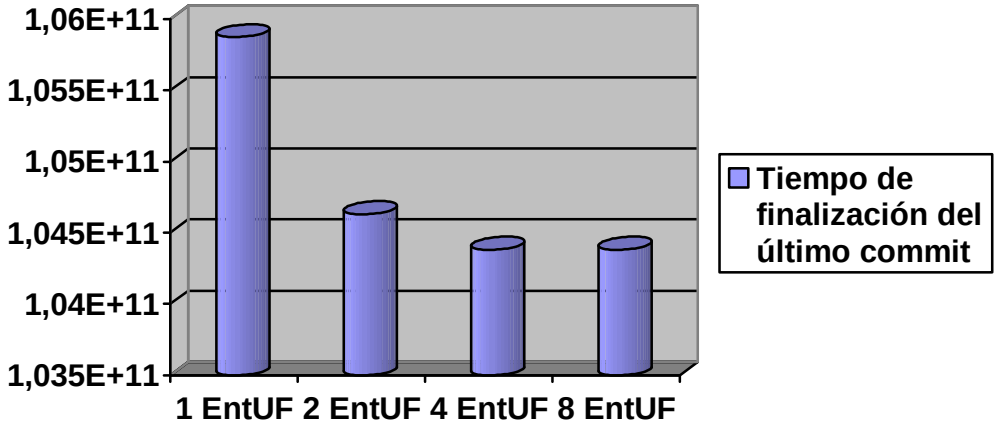
```
Ent FU1=31390181
Ent FU2=14149724
Ent FU3=7500887
Ent FU4=1584167
Ent FU5=15338
Ent FU6=10
Ent FU7=0
Ent FU8=0
```

T. de finalización del último commit (conf. Asíncrona): `54160304896`
T. de finalización del último commit (conf. Síncrona): `104379119000`

- Ejecución asíncrona:



- Ejecución síncrona:



Applu

- Simulación Asíncrona: Realiza 2225195 operaciones de tipo entero.
- Simulación Síncrona: Realiza 2229372 operaciones de tipo entero.

A continuación se muestra una comparación entre ambas simulaciones, mostrando el número de unidades funcionales y la frecuencia de utilización de cada una:

- Para 1 unidad funcional de enteros:
  F. de utilización de Ent FU1 (conf. Asíncrona): `2225195`
  F. de utilización de Ent FU1 (conf. Síncrona): `2229372`

  T. de finalización del último commit (conf. Asíncrona): `12469949800`
  T. de finalización del último commit (conf. Síncrona): `21893722000`

- Para 2 unidades funcionales de enteros:
  F. de utilización de Ent FU (conf. Asíncrona):
  ```
  Ent FU1=1521573
  Ent FU2=722833
  ```
  F. de utilización de Ent FU (conf. Síncrona):
  ```
  Ent FU1=1601971
  Ent FU2=647040
  ```

  T. de finalización del último commit (conf. Asíncrona): `12429083897`
  T. de finalización del último commit (conf. Síncrona): `21791424000`

- Para 4 unidades funcionales de enteros:
  F. de utilización de Ent FU (conf. Asíncrona):
  ```
  Ent FU1=1414334
  Ent FU2=619431
  Ent FU3=178744
  Ent FU4=34818
  ```
  F. de utilización de Ent FU (conf. Síncrona):
  ```
  Ent FU1=1492732
  Ent FU2=598994
  Ent FU3=144238
  Ent FU4=16956
  ```

  T. de finalización del último commit (conf. Asíncrona): `12422835156`
  T. de finalización del último commit (conf. Síncrona): `21786341000`

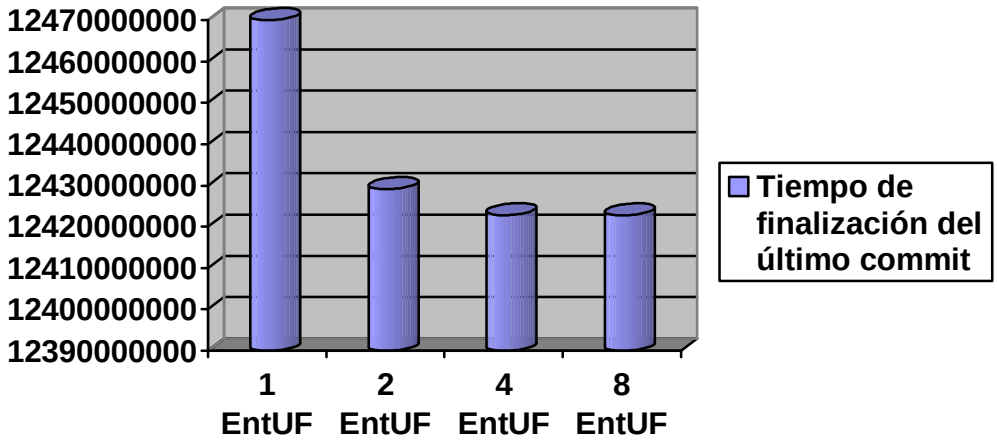
- Para 8 unidades funcionales de enteros:
  F. de utilización de Ent FU (conf. Asíncrona):
  ```
  Ent FU1=1412907
  Ent FU2=617697
  Ent FU3=177680
  Ent FU4=34336
  Ent FU5=4164
  Ent FU6=453
  Ent FU7=0
  Ent FU8=0
  ```

F. de utilización de Ent FU (conf. Síncrona):

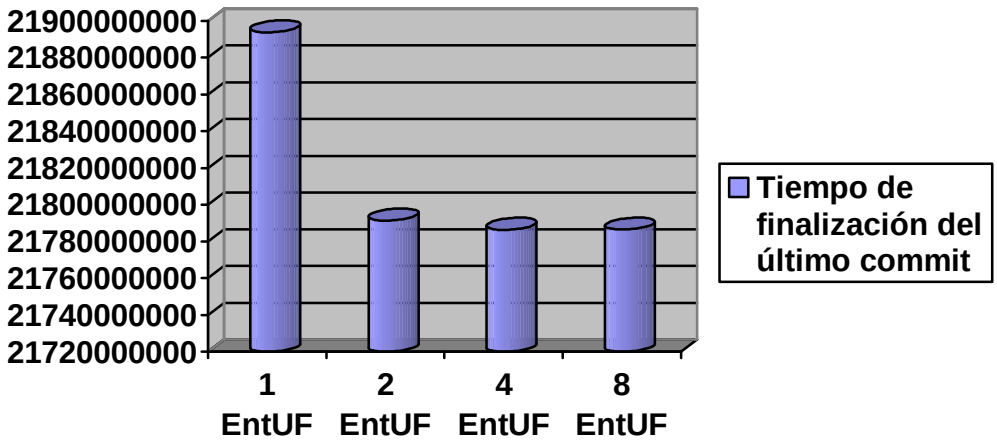
```
Ent FU1=1489036
Ent FU2=597498
Ent FU3=144240
Ent FU4=16956
Ent FU5=4950
Ent FU6=291
Ent FU7=0
Ent FU8=0
```

T. de finalización del último commit (conf. Asíncrona): `12422770407`
T. de finalización del último commit (conf. Síncrona): `21786355000`

- Ejecución asíncrona:



- Ejecución síncrona:



Lucas

- Simulación Asíncrona: Realiza 6021744 operaciones de tipo entero.
- Simulación Síncrona: Realiza 6044755 operaciones de tipo entero.

A continuación se muestra una comparación entre ambas simulaciones, mostrando el número de unidades funcionales y la frecuencia de utilización de cada una:

- Para 1 unidad funcional de enteros:
  F. de utilización de Ent FU1 (conf. Asíncrona): `6021744`
  F. de utilización de Ent FU1 (conf. Síncrona): `6044755`

  T. de finalización del último commit (conf. Asíncrona): `21299065319`
  T. de finalización del último commit (conf. Síncrona): `31783715000`

- Para 2 unidades funcionales de enteros:
  F. de utilización de Ent FU (conf. Asíncrona):
  ```
  Ent FU1=4035613
  Ent FU2=2051601
  ```
  F. de utilización de Ent FU (conf. Síncrona):
  ```
  Ent FU1=4209592
  Ent FU2=1918947
  ```

  T. de finalización del último commit (conf. Asíncrona): `21272277056`
  T. de finalización del último commit (conf. Síncrona): `31701114000`

- Para 4 unidades funcionales de enteros:
  F. de utilización de Ent FU (conf. Asíncrona):
  ```
  Ent FU1=3671086
  Ent FU2=1620266
  Ent FU3=619768
  Ent FU4=181776
  ```
  F. de utilización de Ent FU (conf. Síncrona):
  ```
  Ent FU1=3853890
  Ent FU2=1627998
  Ent FU3=554251
  Ent FU4=101228
  ```

  T. de finalización del último commit (conf. Asíncrona): `21267788310`
  T. de finalización del último commit (conf. Síncrona): `31701545000`

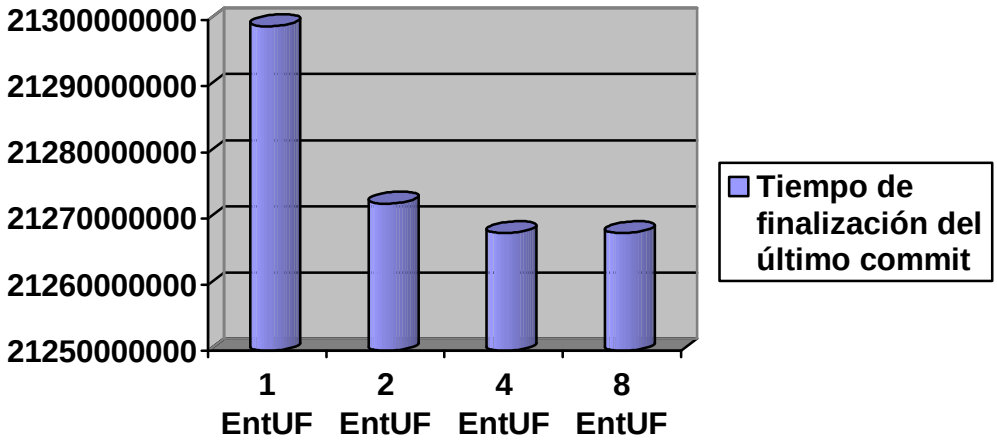
- Para 8 unidades funcionales de enteros:
  F. de utilización de Ent FU (conf. Asíncrona):
  ```
  Ent FU1=3654487
  Ent FU2=1596506
  Ent FU3=607908
  Ent FU4=178237
  Ent FU5=44730
  Ent FU6=11523
  Ent FU7=0
  Ent FU8=0
  ```

F. de utilización de Ent FU (conf. Síncrona):

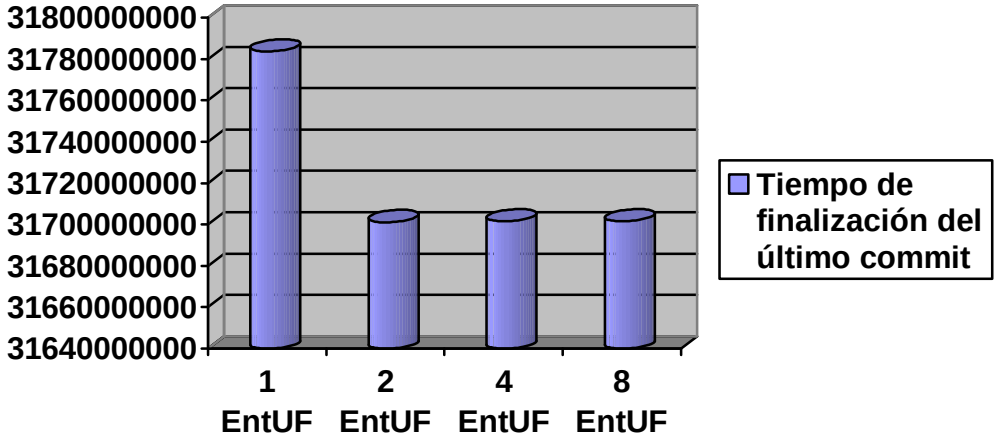
```
Ent FU1=3832366
Ent FU2=1608688
Ent FU3=554251
Ent FU4=101228
Ent FU5=23126
Ent FU6=19120
Ent FU7=0
Ent FU8=0
```

T. de finalización del último commit (conf. Asíncrona): `21267760166`
T. de finalización del último commit (conf. Síncrona): `31701545000`

- Ejecución asíncrona:



- Ejecución síncrona:



Mgrid

- Simulación Asíncrona: Realiza 3248689 operaciones de tipo entero.
- Simulación Síncrona: Realiza 3255924 operaciones de tipo entero.

A continuación se muestra una comparación entre ambas simulaciones, mostrando el número de unidades funcionales y la frecuencia de utilización de cada una:

- Para 1 unidad funcional de enteros:
  F. de utilización de Ent FU1 (conf. Asíncrona): `3248689`
  F. de utilización de Ent FU1 (conf. Síncrona): `3255924`

  T. de finalización del último commit (conf. Asíncrona): `17804202621`
  T. de finalización del último commit (conf. Síncrona): `25248695000`

- Para 2 unidades funcionales de enteros:
  F. de utilización de Ent FU (conf. Asíncrona):
  ```
  Ent FU1=2000151
  Ent FU2=1283848
  ```
  F. de utilización de Ent FU (conf. Síncrona):
  ```
  Ent FU1=2079803
  Ent FU2=1236681
  ```

  T. de finalización del último commit (conf. Asíncrona): `17530510988`
  T. de finalización del último commit (conf. Síncrona): `24630158000`

- Para 4 unidades funcionales de enteros:
  F. de utilización de Ent FU (conf. Asíncrona):
  ```
  Ent FU1=1745653
  Ent FU2=956652
  Ent FU3=429307
  Ent FU4=159063
  ```
  F. de utilización de Ent FU (conf. Síncrona):
  ```
  Ent FU1=1801740
  Ent FU2=1055606
  Ent FU3=363776
  Ent FU4=104811
  ```

  T. de finalización del último commit (conf. Asíncrona): `17474928036`
  T. de finalización del último commit (conf. Síncrona): `24600779000`

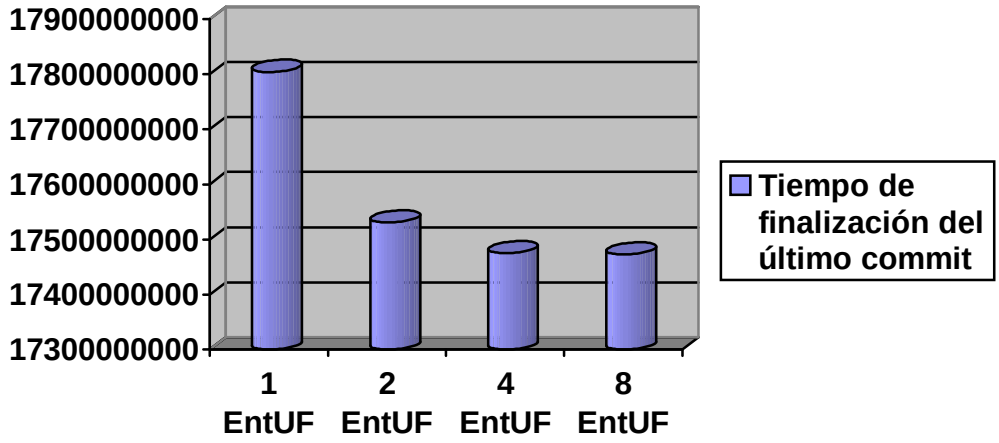
- Para 8 unidades funcionales de enteros:
  F. de utilización de Ent FU (conf. Asíncrona):
  ```
  Ent FU1=1732827
  Ent FU2=939747
  Ent FU3=418926
  Ent FU4=155943
  Ent FU5=37280
  Ent FU6=5174
  Ent FU7=0
  Ent FU8=0
  ```

F. de utilización de Ent FU (conf. Síncrona):

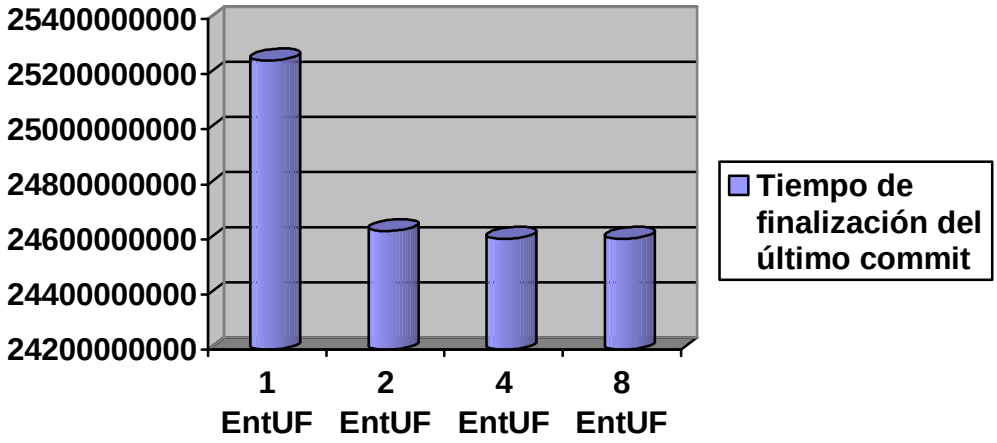
```
Ent FU1=1794788
Ent FU2=1046346
Ent FU3=363774
Ent FU4=104811
Ent FU5=11517
Ent FU6=4821
Ent FU7=0
Ent FU8=0
```

T. de finalización del último commit (conf. Asíncrona): `17472827859`
T. de finalización del último commit (conf. Síncrona): `24600765000`

- Ejecución asíncrona:



- Ejecución síncrona:



Sixtrack

- Simulación Asíncrona: Realiza 10470248 operaciones de tipo entero.
- Simulación Síncrona: Realiza 10571598 operaciones de tipo entero.

A continuación se muestra una comparación entre ambas simulaciones, mostrando el número de unidades funcionales y la frecuencia de utilización de cada una:

- Para 1 unidad funcional de enteros:
  F. de utilización de Ent FU1 (conf. Asíncrona): `10470248`
  F. de utilización de Ent FU1 (conf. Síncrona): `10571598`

  T. de finalización del último commit (conf. Asíncrona): `8372933152`
  T. de finalización del último commit (conf. Síncrona): `16077503000`

- Para 2 unidades funcionales de enteros:
  F. de utilización de Ent FU (conf. Asíncrona):
  ```
  Ent FU1=5838539
  Ent FU2=5392622
  ```
  F. de utilización de Ent FU (conf. Síncrona):
  ```
  Ent FU1=6358292
  Ent FU2=5332692
  ```

  T. de finalización del último commit (conf. Asíncrona): `7411358327`
  T. de finalización del último commit (conf. Síncrona): `13827193000`

- Para 4 unidades funcionales de enteros:
  F. de utilización de Ent FU (conf. Asíncrona):
  ```
  Ent FU1=4681576
  Ent FU2=3425513
  Ent FU3=2516206
  Ent FU4=913527
  ```
  F. de utilización de Ent FU (conf. Síncrona):
  ```
  Ent FU1=5781589
  Ent FU2=3669273
  Ent FU3=2554432
  Ent FU4=722445
  ```

  T. de finalización del último commit (conf. Asíncrona): `7180860532`
  T. de finalización del último commit (conf. Síncrona): `13789261000`

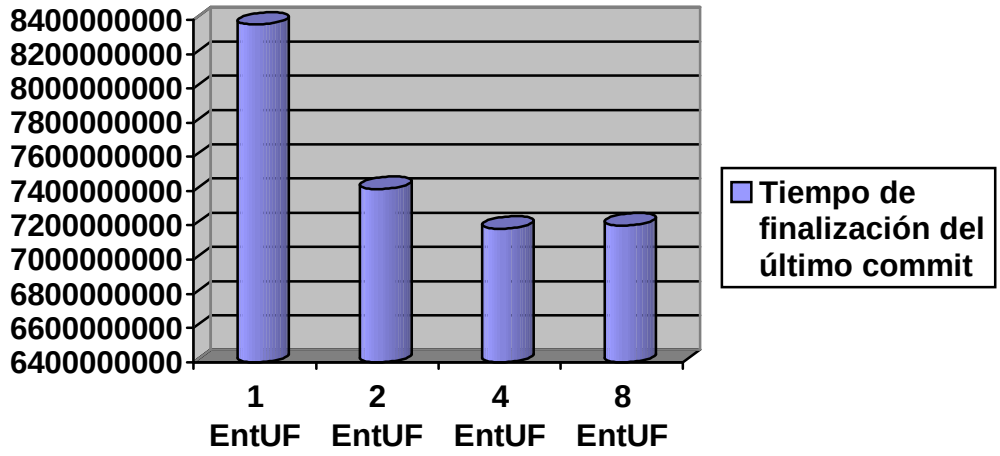
- Para 8 unidades funcionales de enteros:
  F. de utilización de Ent FU (conf. Asíncrona):
  ```
  Ent FU1=4615574
  Ent FU2=3368298
  Ent FU3=2498674
  Ent FU4=923499
  Ent FU5=159096
  Ent FU6=3453
  Ent FU7=0
  Ent FU8=0
  ```

F. de utilización de Ent FU (conf. Síncrona):

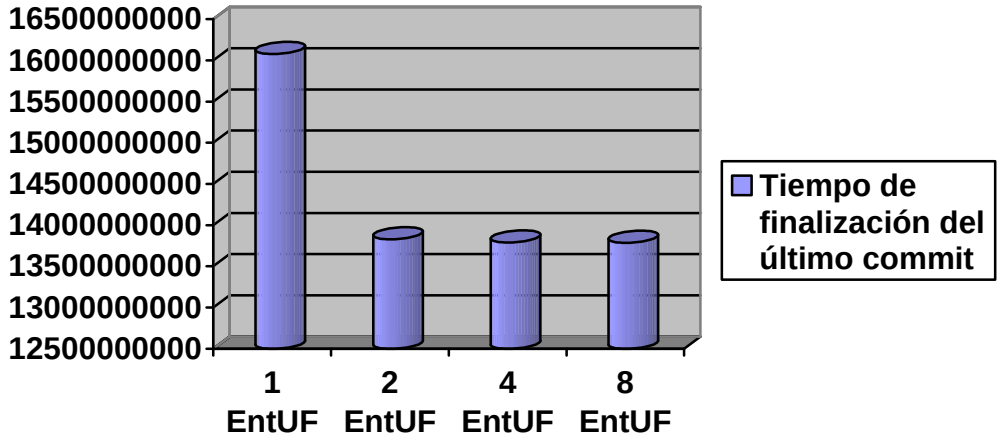
```
Ent FU1=5688861
Ent FU2=3667298
Ent FU3=2553878
Ent FU4=722448
Ent FU5=90044
Ent FU6=152
Ent FU7=0
Ent FU8=0
```

T. de finalización del último commit (conf. Asíncrona): `7199148461`
T. de finalización del último commit (conf. Síncrona): `13785289000`

- Ejecución asíncrona:



- Ejecución síncrona:



Swim

- Simulación Asíncrona: Realiza 10113268 operaciones de tipo entero.
- Simulación Síncrona: Realiza 9820551 operaciones de tipo entero.

A continuación se muestra una comparación entre ambas simulaciones, mostrando el número de unidades funcionales y la frecuencia de utilización de cada una:

- Para 1 unidad funcional de enteros:
  F. de utilización de Ent FU1 (conf. Asíncrona): `10113268`
  F. de utilización de Ent FU1 (conf. Síncrona): `9820551`

  T. de finalización del último commit (conf. Asíncrona): `20408233872`
  T. de finalización del último commit (conf. Síncrona): `35320954000`

- Para 2 unidades funcionales de enteros:
  F. de utilización de Ent FU (conf. Asíncrona):
  ```
  Ent FU1=6708606
  Ent FU2=3469508
  ```
  F. de utilización de Ent FU (conf. Síncrona):
  ```
  Ent FU1=7069609
  Ent FU2=2825404
  ```

  T. de finalización del último commit (conf. Asíncrona): `20257258326`
  T. de finalización del último commit (conf. Síncrona): `35112461000`

- Para 4 unidades funcionales de enteros:
  F. de utilización de Ent FU (conf. Asíncrona):
  ```
  Ent FU1=6150490
  Ent FU2=2884005
  Ent FU3=922065
  Ent FU4=220331
  ```
  F. de utilización de Ent FU (conf. Síncrona):
  ```
  Ent FU1=6360054
  Ent FU2=2657230
  Ent FU3=770091
  Ent FU4=109531
  ```

  T. de finalización del último commit (conf. Asíncrona): `20206472939`
  T. de finalización del último commit (conf. Síncrona): `35047765000`

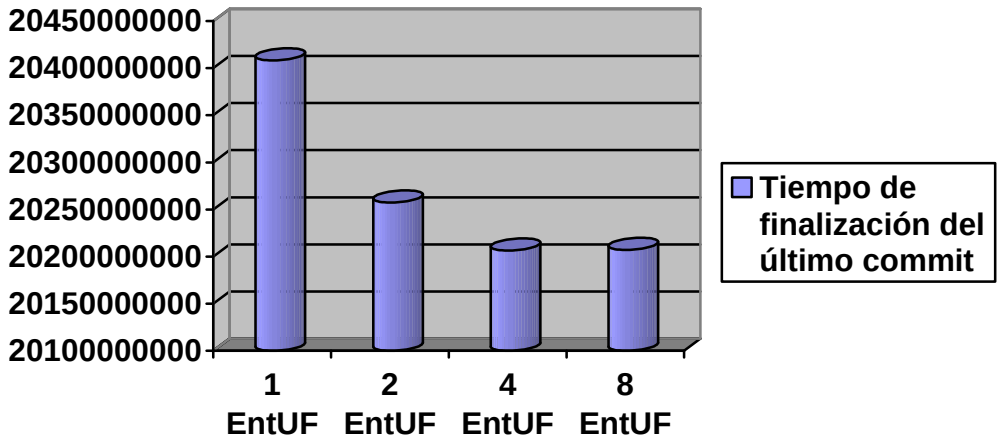
- Para 8 unidades funcionales de enteros:
  F. de utilización de Ent FU (conf. Asíncrona):
  ```
  Ent FU1=6137187
  Ent FU2=2868258
  Ent FU3=917001
  Ent FU4=219702
  Ent FU5=31481
  Ent FU6=3236
  Ent FU7=0
  Ent FU8=0
  ```

F. de utilización de Ent FU (conf. Síncrona):

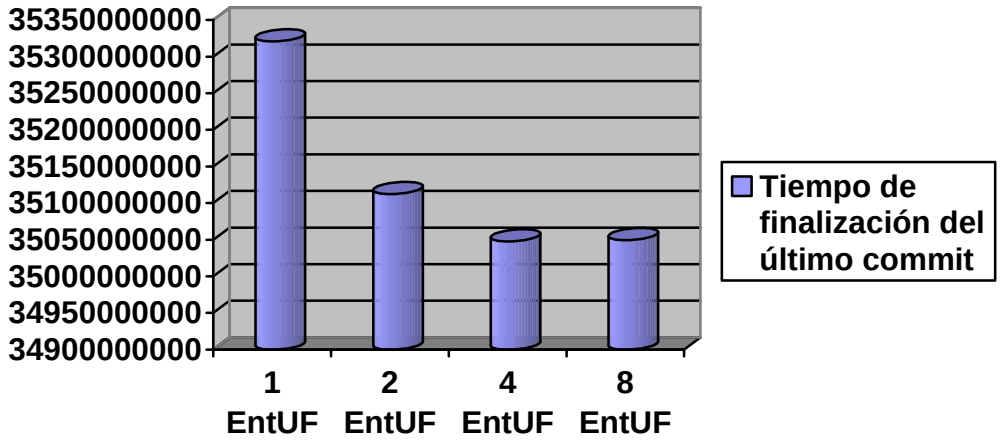
```
Ent FU1=6342061
Ent FU2=2640817
Ent FU3=770144
Ent FU4=109548
Ent FU5=34923
Ent FU6=55
Ent FU7=0
Ent FU8=0
```

T. de finalización del último commit (conf. Asíncrona): `20207050310`
T. de finalización del último commit (conf. Síncrona): `35049569000`

- Ejecución asíncrona:



- Ejecución síncrona:



Equake

- Simulación Asíncrona: Realiza 734824934 operaciones de tipo entero.
- Simulación Síncrona: Realiza 740859104 operaciones de tipo entero.

A continuación se muestra una comparación entre ambas simulaciones, mostrando el número de unidades funcionales y la frecuencia de utilización de cada una:

- Para 1 unidad funcional de enteros:
  F. de utilización de Ent FU1 (conf. Asíncrona): `734824934`
  F. de utilización de Ent FU1 (conf. Síncrona): `740859104`

  T. de finalización del último commit (conf. Asíncrona): `1181598734255`
  T. de finalización del último commit (conf. Síncrona): `2271120626000`

- Para 2 unidades funcionales de enteros:
  F. de utilización de Ent FU (conf. Asíncrona):
  ```
  Ent FU1=520175620
  Ent FU2=247530728
  ```
  F. de utilización de Ent FU (conf. Síncrona):
  ```
  Ent FU1=555453477
  Ent FU2=227577195
  ```

  T. de finalización del último commit (conf. Asíncrona): `1176370303052`
  T. de finalización del último commit (conf. Síncrona): `2252204302000`

- Para 4 unidades funcionales de enteros:
  F. de utilización de Ent FU (conf. Asíncrona):
  ```
  Ent FU1=477556502
  Ent FU2=201711209
  Ent FU3=80087611
  Ent FU4=20131880
  ```
  F. de utilización de Ent FU (conf. Síncrona):
  ```
  Ent FU1=508253756
  Ent FU2=199525847
  Ent FU3=71174831
  Ent FU4=17053102
  ```

  T. de finalización del último commit (conf. Asíncrona): `1175164806974`
  T. de finalización del último commit (conf. Síncrona): `2251148116000`

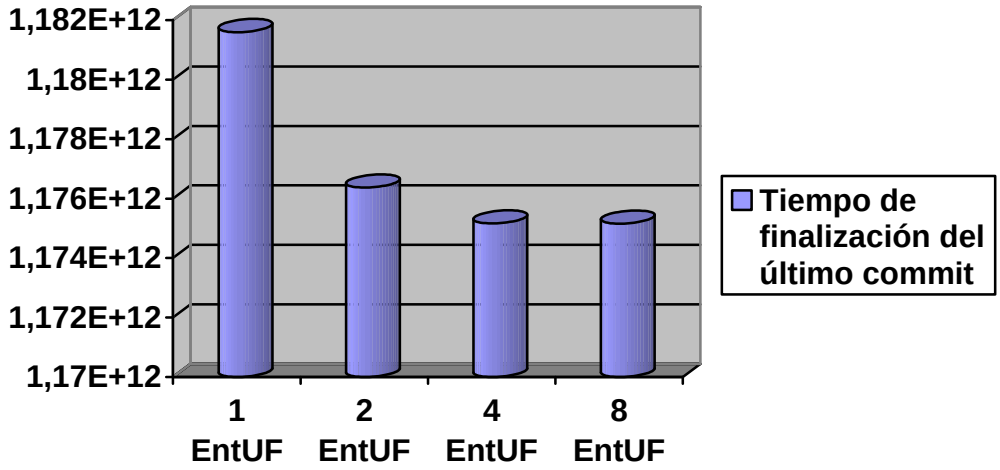
- Para 8 unidades funcionales de enteros:
  F. de utilización de Ent FU1 (conf. Asíncrona):
  ```
  Ent FU1=476463916
  Ent FU2=200522119
  Ent FU3=79519038
  Ent FU4=20065247
  Ent FU5=3003753
  Ent FU6=255290
  Ent FU7=0
  Ent FU8=0
  ```

F. de utilización de Ent FU (conf. Síncrona):

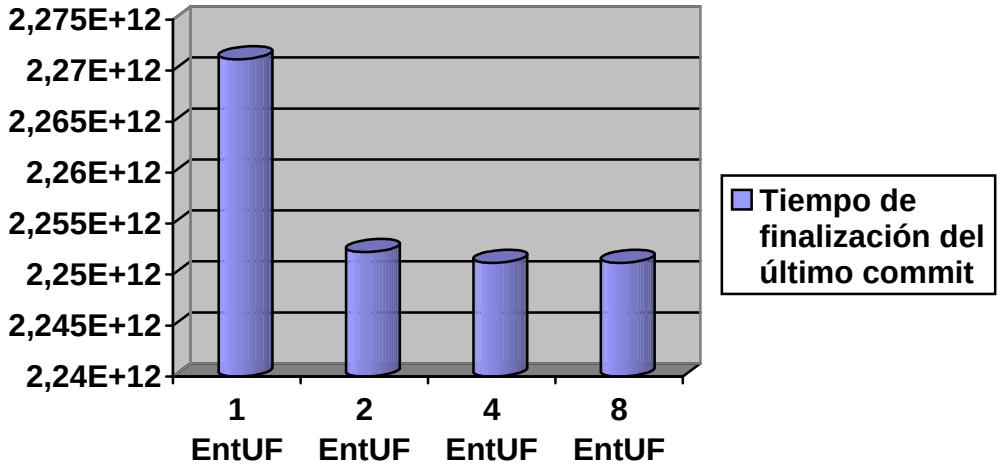
```
Ent FU1=505597454
Ent FU2=199525844
Ent FU3=71174831
Ent FU4=17053102
Ent FU5=2656325
Ent FU6=3
Ent FU7=0
Ent FU8=0
```

T. de finalización del último commit (conf. Asíncrona): `1175158067611`
T. de finalización del último commit (conf. Síncrona): `2251148116000`

- Ejecución asíncrona:



- Ejecución síncrona:



Mesa

- Simulación Asíncrona: Realiza 1372889257 operaciones de tipo entero.
- Simulación Síncrona: Realiza 1382162708 operaciones de tipo entero.

A continuación se muestra una comparación entre ambas simulaciones, mostrando el número de unidades funcionales y la frecuencia de utilización de cada una:

- Para 1 unidad funcional de enteros:
  F. de utilización de Ent FU1 (conf. Asíncrona): `1372889257`
  F. de utilización de Ent FU1 (conf. Síncrona): `1382162708`

  T. de finalización del último commit (conf. Asíncrona): `1573141991724`
  T. de finalización del último commit (conf. Síncrona): `3057733327000`

- Para 2 unidades funcionales de enteros:
  F. de utilización de Ent FU (conf. Asíncrona):
  ```
  Ent FU1=945812961
  Ent FU2=485297658
  ```
  F. de utilización de Ent FU (conf. Síncrona):
  ```
  Ent FU1=1005010506
  Ent FU2=440790884
  ```

  T. de finalización del último commit (conf. Asíncrona): `1555381708445`
  T. de finalización del último commit (conf. Síncrona): `3018351575000`

- Para 4 unidades funcionales de enteros:
  F. de utilización de Ent FU (conf. Asíncrona):
  ```
  Ent FU1=868566782
  Ent FU2=384423508
  Ent FU3=142128292
  Ent FU4=44839905
  ```
  F. de utilización de Ent FU1 (conf. Síncrona):
  ```
  Ent FU1=910222235
  Ent FU2=379503554
  Ent FU3=108887171
  Ent FU4=51121087
  ```

  T. de finalización del último commit (conf. Asíncrona): `1550747756258`
  T. de finalización del último commit (conf. Síncrona): `3015401388000`

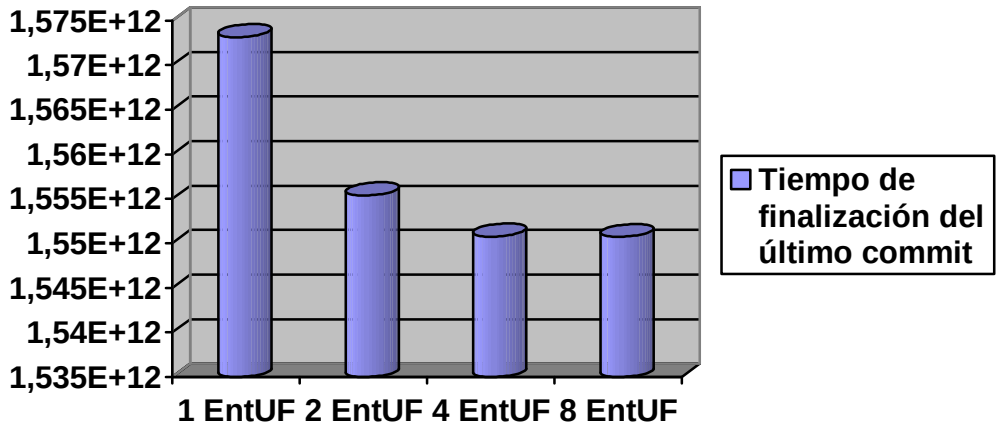
- Para 8 unidades funcionales de enteros:
  F. de utilización de Ent FU1 (conf. Asíncrona):
  ```
  Ent FU1=865828576
  Ent FU2=381130933
  Ent FU3=140343946
  Ent FU4=44356172
  Ent FU5=7786551
  Ent FU6=711949
  Ent FU7=0
  Ent FU8=0
  ```

F. de utilización de Ent FU (conf. Síncrona):

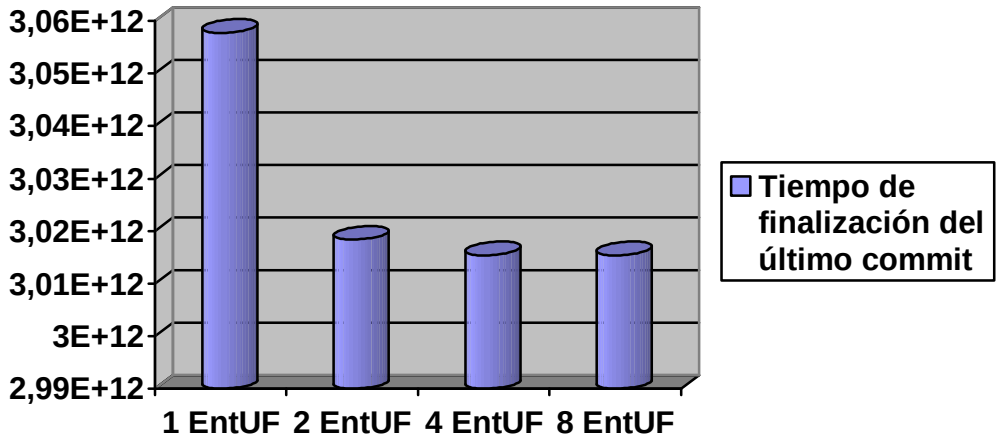
```
Ent FU1=910221985
Ent FU2=379503497
Ent FU3=108887195
Ent FU4=51121087
Ent FU5=456
Ent FU6=71
Ent FU7=0
Ent FU8=0
```

T. de finalización del último commit (conf. Asíncrona): `1550691721122`
T. de finalización del último commit (conf. Síncrona): `3015401792000`

- Ejecución asíncrona:



- Ejecución síncrona:



Bzip

- Simulación Asíncrona: Realiza 1653993585 operaciones de tipo entero.
- Simulación Síncrona: Realiza 1679971003 operaciones de tipo entero.

A continuación se muestra una comparación entre ambas simulaciones, mostrando el número de unidades funcionales y la frecuencia de utilización de cada una:

- Para 1 unidad funcional de enteros:
  F. de utilización de Ent FU1 (conf. Asíncrona): `1653993585`
  F. de utilización de Ent FU1 (conf. Síncrona): `1679971003`

  T. de finalización del último commit (conf. Asíncrona): `1986119477424`
  T. de finalización del último commit (conf. Síncrona): `3843177080000`

- Para 2 unidades funcionales de enteros:
  F. de utilización de Ent FU (conf. Asíncrona):
  ```
  Ent FU1=1154205132
  Ent FU2=620250658
  ```
  F. de utilización de Ent FU (conf. Síncrona):
  ```
  Ent FU1=1229503158
  Ent FU2=566223177
  ```

  T. de finalización del último commit (conf. Asíncrona): `1986508795247`
  T. de finalización del último commit (conf. Síncrona): `3824268149000`

- Para 4 unidades funcionales de enteros:
  F. de utilización de Ent FU (conf. Asíncrona):
  ```
  Ent FU1=1045914990
  Ent FU2=502992095
  Ent FU3=194194054
  Ent FU4=44630148
  ```
  F. de utilización de Ent FU (conf. Síncrona):
  ```
  Ent FU1=1091783925
  Ent FU2=508865769
  Ent FU3=175972266
  Ent FU4=23941073
  ```

  T. de finalización del último commit (conf. Asíncrona): `1984881832334`
  T. de finalización del último commit (conf. Síncrona): `3822912862000`

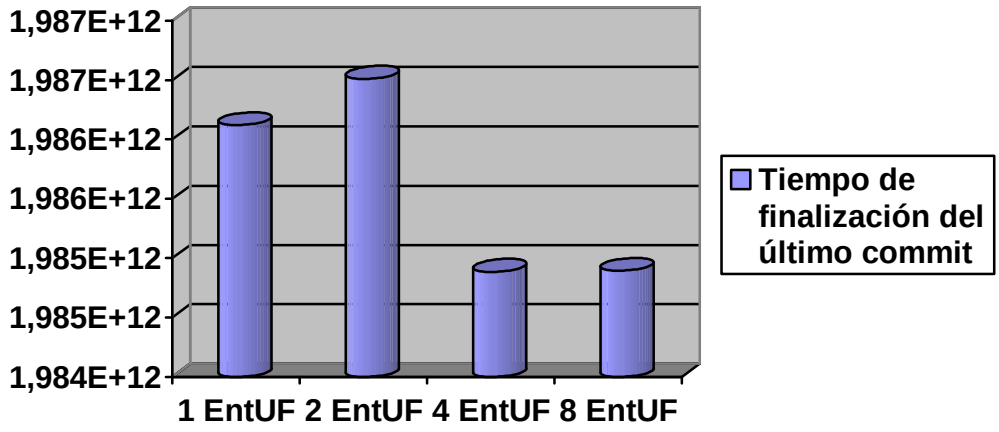
- Para 8 unidades funcionales de enteros:
  F. de utilización de Ent FU1 (conf. Asíncrona):
  ```
  Ent FU1=1043023649
  Ent FU2=500422430
  Ent FU3=193037657
  Ent FU4=44578236
  Ent FU5=6813958
  Ent FU6=174105
  Ent FU7=0
  Ent FU8=0
  ```

F. de utilización de Ent FU (conf. Síncrona):

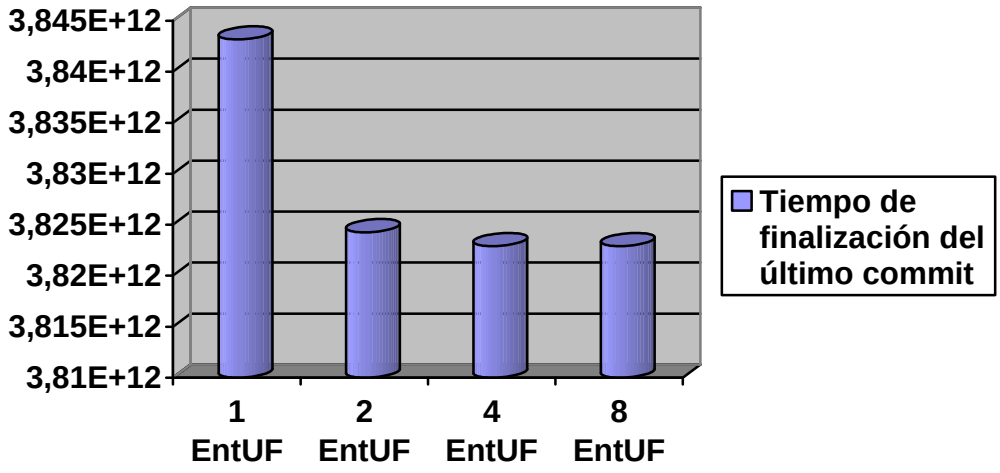
```
Ent FU1=1089798085
Ent FU2=508865696
Ent FU3=175972266
Ent FU4=23941073
Ent FU5=1990795
Ent FU6=16
Ent FU7=0
Ent FU8=0
```

T. de finalización del último commit (conf. Asíncrona): `1984892175889`
T. de finalización del último commit (conf. Síncrona): `3822912862000`

- Ejecución asíncrona:



- Ejecución síncrona:



Gzip

- Simulación Asíncrona: Realiza 639979514 operaciones de tipo entero.
- Simulación Síncrona: Realiza 641423101 operaciones de tipo entero.

A continuación se muestra una comparación entre ambas simulaciones, mostrando el número de unidades funcionales y la frecuencia de utilización de cada una:

- Para 1 unidad funcional de enteros:
  F. de utilización de Ent FU1 (conf. Asíncrona): `639979514`
  F. de utilización de Ent FU1 (conf. Síncrona): `641423101`

  T. de finalización del último commit (conf. Asíncrona): `639168853632`
  T. de finalización del último commit (conf. Síncrona): `1214232345000`

- Para 2 unidades funcionales de enteros:
  F. de utilización de Ent FU (conf. Asíncrona):
  ```
  Ent FU1=447440989
  Ent FU2=235042157
  ```
  F. de utilización de Ent FU (conf. Síncrona):
  ```
  Ent FU1=484715713
  Ent FU2=214686402
  ```

  T. de finalización del último commit (conf. Asíncrona): `634005752546`
  T. de finalización del último commit (conf. Síncrona): `1203657627000`

- Para 4 unidades funcionales de enteros:
  F. de utilización de Ent FU (conf. Asíncrona):
  ```
  Ent FU1=410757221
  Ent FU2=203067051
  Ent FU3=67147667
  Ent FU4=5221091
  ```
  F. de utilización de Ent FU (conf. Síncrona):
  ```
  Ent FU1=431352046
  Ent FU2=206841546
  Ent FU3=64170589
  Ent FU4=7868402
  ```

  T. de finalización del último commit (conf. Asíncrona): `632612876188`
  T. de finalización del último commit (conf. Síncrona): `1202833525000`

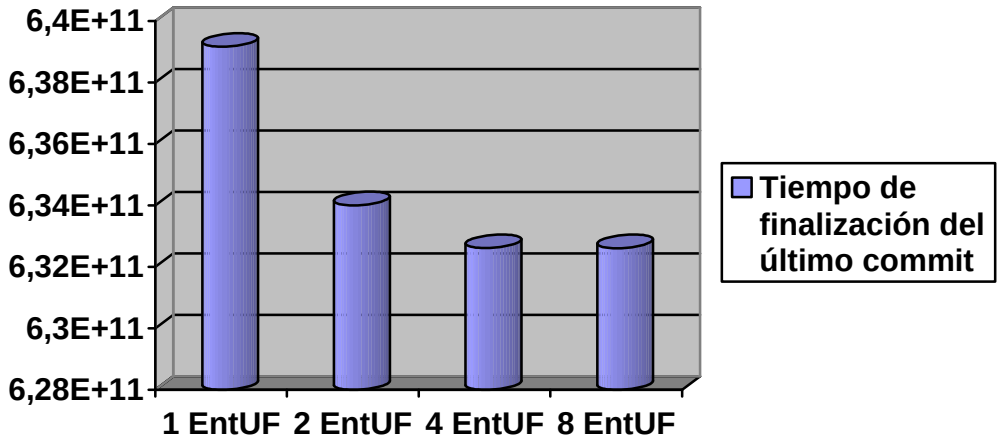
- Para 8 unidades funcionales de enteros:
  F. de utilización de Ent FU (conf. Asíncrona):
  ```
  Ent FU1=410484307
  Ent FU2=202844833
  Ent FU3=67061254
  Ent FU4=5192026
  Ent FU5=616319
  Ent FU6=1396
  Ent FU7=0
  Ent FU8=0
  ```

F. de utilización de Ent FU (conf. Síncrona):

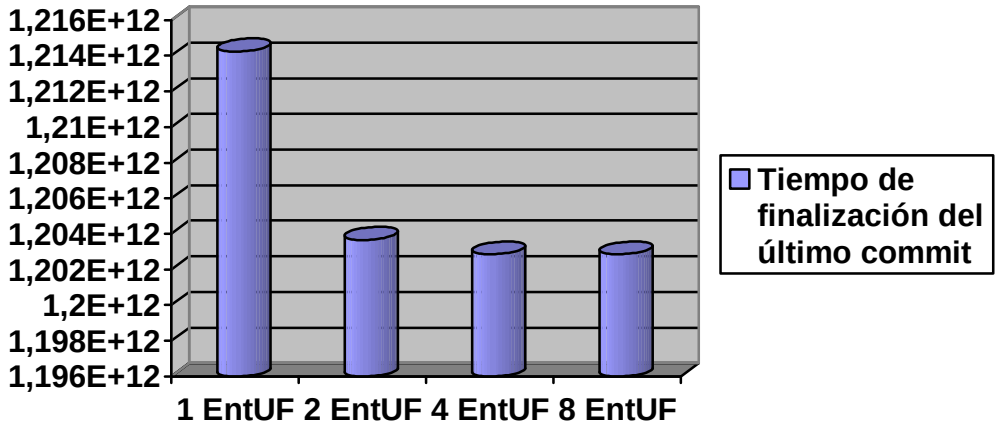
```
Ent FU1=431066649
Ent FU2=206841407
Ent FU3=64170589
Ent FU4=7868402
Ent FU5=287860
Ent FU6=130
Ent FU7=0
Ent FU8=0
```

T. de finalización del último commit (conf. Asíncrona): `632611489119`
T. de finalización del último commit (conf. Síncrona): `1202833525000`

- Ejecución asíncrona:



- Ejecución síncrona:



CONCLUSIÓN

Comparación de resultados. ***SPEC2000 en modo asíncrono y archivos de prueba.***

Al comparar los resultados de ejecutar los SPEC2000 con las pruebas que se realizaron con los archivos de prueba, en la etapa de modificación de la unidad funcional de enteros, se observa que los resultados obtenidos son similares. Una primera conclusión que se puede sacar es que estos archivos de prueba, que son más pequeños y más fáciles de ejecutar en un ordenador personal que las pruebas de los SPEC2000, realizan pruebas similares a los SPEC2000 ejecutándose en menos tiempo y son fiables para evaluar el funcionamiento del simulador.

Tanto en los resultados obtenidos al ejecutar las pruebas como en los resultados obtenidos al ejecutar los SPEC2000 vemos que en la mayoría de los casos la mejora principal en tiempo se produce al pasar de 1 a 2 unidades funcionales de enteros. Sin embargo, en las pruebas "Mcf" y "ammp" de los SPEC2000 los resultados difieren del resto. En estos 2 casos al pasar de 2 a 4 unidades funcionales de enteros la reducción es mayor que al pasar de 1 a 2. Esto debe ser debido a que este tipo de pruebas ejecuta un mayor número de operaciones de enteros en un espacio de tiempo menor que en el caso del resto de pruebas. La mayoría de las pruebas consiguen la mayor mejora en el cambio de 1 a 2 unidades funcionales de enteros, pero en este caso la mayor mejora se produce en el cambio de 2 a 4 unidades funcionales de enteros. Seguramente es debido a que el resto de pruebas no realizan tantas operaciones con enteros como las dos anteriores en un mismo espacio de tiempo.

Al contrario que en los resultados de ejecutar los tests, comparando los resultados de los tiempos en los SPEC2000 al aumentar de 4 a 8 el número de unidades funcionales de enteros no son los mismos. En muchos casos el tiempo disminuye ligeramente al pasar de 4 a 8. En otros casos aumenta ligeramente. Pero al igual que en los resultados obtenidos al ejecutar los tests se puede concluir que la mejora de 4 a 8 no merece la pena. El ahorro de tiempo que se consigue, en los casos en que el tiempo disminuye, no es proporcional al gasto que supone pasar de 4 a 8 unidades funcionales de enteros.

En la evaluación de la prueba "bzip" de los SPEC2000 se observa un dato curioso, que es que el tiempo de finalización del último commit cuando el simulador utiliza sólo una unidad funcional de enteros es ligeramente menor que el mismo tiempo cuando se utilizan dos unidades funcionales de enteros. También se observa que el número de instrucciones ejecutadas de tipo entero con una unidad funcional es ligeramente menor que el número de instrucciones ejecutadas, y esto sucede no sólo al comparar este resultado con el resultado obtenido con 2 unidades funcionales, sino también sucede al compararlo con el de 4 y el de 8. Sí observamos los ficheros de resultados vemos que el número de instrucciones commiteadas es prácticamente igual, pero el número de instrucciones ejecutadas difiere bastante. En el caso de más de una unidad funcional de enteros, puede ser debido a que "gzip", que es una prueba de compresión y descompresión, lanza todas las instrucciones que puede en el modo asíncrono mientras haya unidades funcionales de enteros libres y luego muchas de ellas

no se terminan debido a predicciones de salto equivocadas. Una hipótesis sería pensar que en el modo asíncrono, al tener más de una unidad funcional de enteros se lanzan más instrucciones antes de saber si el salto será tomado o no. Debido a esto, cuando la predicción es errónea el tiempo perdido es mayor y el número de instrucciones que se lanzaron a ejecutar y que se descartan es mayor. Esta sería una posible explicación a porque el tiempo de finalización del último commit en el caso en el que solo se utiliza una unidad funcional de enteros es menor que el tiempo de finalización del último commit en el caso en el que se utilizan dos. Al tener una única unidad funcional en un programa que ejecuta muchos saltos y que muchos de ellos harán descartar instrucciones que se lanzaron a ejecución el simulador es más rápido debido a que no lanza tantas instrucciones antes de saber si el salto será tomado o no y, consecuentemente, no se descartan tantas instrucciones cuando la predicción es errónea. Observando los resultados vemos que el caso en el que se utilizan dos unidades funcionales es el menos óptimo en tiempo, y los casos en los que se utilizan 4 y 8 unidades funcionales de enteros, a pesar de ejecutar un mayor número de instrucciones que el caso en el que se utiliza una unidad funcional de enteros, consiguen superar el tiempo de la prueba con una unidad funcional de enteros con una diferencia importante.

Comparando los resultados de utilizar 4 unidades funcionales de enteros con los de utilizar 8, en el caso anterior, al igual que en muchos otros, el tiempo empeora ligeramente. Esto puede ser debido a que la mejora obtenida al tener 8 unidades funcionales de enteros en lugar de 4 es mucho menor que el tiempo que se pierde al tener que organizar 8 unidades funcionales de enteros en vez de 4. Por lo tanto, el resultado obtenido en tiempo en el caso de 8 unidades funcionales de enteros es peor que el caso en el que sólo se utilizan 4 como se muestra en los resultados.

Tal y como sucedía con las pruebas al establecer 8 unidades funcionales enteras se puede observar que en ningún caso se usan las dos últimas. Las conclusiones que se pueden sacar son las mismas que se describieron anteriormente en el apartado de unidades funcionales.

Comparación de resultados. ***SPEC2000 en modo síncrono y SPEC2000 en modo asíncrono.***

Lo primero que se observa al comparar estos resultados es que los tiempos de finalización del último commit en la ejecución asíncrona son muy inferiores a los de la ejecución síncrona. En algunos casos el tiempo de ejecución en modo asíncrono es casi la mitad que el tiempo de ejecución en modo síncrono. Observando que esto se cumple en la mayoría de las pruebas podemos deducir que el modo asíncrono es bastante más rápido que el síncrono.

Se observa también que en la mayoría de los resultados obtenidos en las pruebas asíncronas se produce una disminución importante en tiempo al pasar de utilizar 1 unidad funcional de enteros a 2 y luego otra disminución en tiempo al pasar de utilizar 2 unidades funcionales de enteros a 4, aunque esta sea menor. Esto sucede en el modo síncrono con la diferencia de que la disminución en tiempo al pasar de utilizar 1 unidad

funcional de enteros a 2 es mayor en proporción a la disminución obtenida con los resultados de ejecutar el simulador en modo asíncrono. También se observa que al pasar de 2 a 4 unidades funcionales de enteros en el modo síncrono la mejora en el tiempo es inferior, en proporción, a la mejora obtenida en el modo asíncrono. Una causa de esto puede ser que en el modo síncrono se lanzan un número fijo de instrucciones antes de saber si el salto se toma o no, mientras que en el modo asíncrono puede no ser así. Por eso en el modo síncrono se produce la principal disminución de tiempo en el paso de 1 a 2 unidades funcionales de enteros, mientras que en el modo asíncrono no es tan grande esa disminución porque se están descartando más instrucciones debido a las predicciones de salto fallidas, ya que intenta ejecutar más instrucciones y una gran parte de las instrucciones son de tipo entero. De esto deducimos que en el modo síncrono se descartan menos instrucciones que en el modo asíncrono.

Evaluando resultados observamos que la prueba "Mcf" de los SPEC2000 da resultados distintos a la mayoría de las pruebas, tanto en modo asíncrono (lo vimos en el apartado anterior), como en modo síncrono. Como vimos en el apartado anterior, en este caso al pasar de 2 a 4 unidades funcionales de enteros la reducción es mayor que al pasar de 1 a 2. Como se dijo en el apartado anterior, esto debe ser debido a que este tipo de pruebas ejecuta un mayor número de operaciones de enteros en un espacio de tiempo menor que en el caso del resto de pruebas. La mayoría de las pruebas consiguen la mayor mejora en el cambio de 1 a 2 unidades funcionales de enteros, pero en este caso la mayor mejora se produce al pasar de 2 a 4 unidades funcionales de enteros. Seguramente es debido a que el resto de pruebas no realizan tantas operaciones con enteros como la anterior en un mismo espacio de tiempo. Sin embargo, al evaluar lo que sucede con la prueba "ammp" de los SPEC2000 observamos que, como se vio en el apartado anterior, en modo asíncrono al pasar de 2 a 4 unidades funcionales de enteros la reducción es mayor que al pasar de 1 a 2 unidades funcionales de enteros, lo cual difiere con la mayoría de resultados obtenidos al evaluar el resto de pruebas, pero en el modo síncrono no sucede esto. En el modo síncrono se observa que los resultados son similares a la mayoría de los resultados obtenidos al evaluar el resto de pruebas. Una causa posible es que esta prueba ejecute muchos saltos y se falle en muchas de las predicciones por lo tanto se descartan muchas instrucciones. Como en el modo síncrono el número de instrucciones especuladas es el mismo, el aumento del número de unidades funcionales produce una mejora en el tiempo. En el modo asíncrono puede que se estén especulando más instrucciones de las que se especularían en el modo síncrono, por tanto, si se están especulando operaciones enteras cuantas más unidades funcionales de enteros estén libres más tiempo se pierde si la predicción del salto es errónea. Además de lo comentado en el apartado anterior de que estas dos pruebas ejecutan un mayor número de operaciones enteras en un mismo espacio de tiempo, es decir, que tienen más "densidad" de operaciones enteras en algunas partes del programa que otras pruebas, no quiere decir necesariamente que tienen más operaciones enteras que el resto de pruebas.

De la misma manera que en los casos anteriores al comparar los resultados de utilizar 4 unidades funcionales de enteros, en el modo asíncrono, con los de utilizar 8, en la mayoría de los casos el tiempo empeora ligeramente mientras que en otros mejora. No podemos decir lo mismo al comparar estos mismos resultados en el modo síncrono

donde los tiempos de finalización del último commit son los mismos para la mayoría de las pruebas de los SPEC2000, tanto para 4 unidades funcionales de enteros como para 8. Además de esto al comparar la frecuencia de utilización de cada unidad funcional en el caso en el que hay 8 se observa que en la ejecución en modo síncrono las unidades funcionales más ocupadas son las 4 primeras mientras que las dos últimas se usan mucho menos en comparación con el uso que se les da en la ejecución en modo asíncrono. Esto puede ser debido a que la mejora obtenida al tener 8 unidades funcionales de enteros en lugar de 4 es mucho menor que el tiempo que se pierde al tener que organizar 8 unidades funcionales de enteros en vez de 4 en el caso de la ejecución en modo asíncrono. Por lo tanto, el resultado obtenido en tiempo en el caso de 8 unidades funcionales de enteros es peor que el caso en el que sólo se utilizan 4 como se muestra en la mayoría de los resultados, en modo asíncrono. Como esto depende de la distribución de las operaciones enteras no siempre se pierde tanto tiempo, lo cual explicaría porque hay casos en que si mejora en la ejecución en modo asíncrono. Lo que ocurre en el modo síncrono es que se emplea mucho más tiempo en terminar la prueba, por tanto, el tiempo durante el que se encuentran disponibles las 4 primeras unidades funcionales de enteros es mucho mayor, y no hace falta buscar en las siguientes. Esto explicaría porque varía tan poco el resultado de ejecutar el simulador en modo síncrono al pasar de 4 a 8 unidades funcionales de enteros.

Al igual que en las dos evaluaciones anteriores, tanto en el modo asíncrono como en el síncrono al establecer 8 unidades funcionales enteras se puede observar que en ningún caso se usan las dos últimas. Las conclusiones que se pueden sacar son las mismas que se describieron anteriormente en el apartado de unidades funcionales, es decir, que el simulador no está preparado para operar con más de 6 unidades funcionales de enteros.

Conclusiones Finales

Como conclusión de estas evaluaciones de resultados de ejecutar los ficheros de prueba del proyecto y las pruebas de los SPEC2000 se puede decir que lo óptimo es utilizar 4 unidades funcionales de enteros en lugar de 1, 2 ó 8 porque contrastando los resultados de todas las pruebas vemos que es la configuración óptima en tiempo por diversos motivos: podríamos considerar el utilizar 2 unidades funcionales de enteros en vez de 1 y la mejora es significativa, pero hay un caso en el que en vez de mejorar empeora. El cual nos hace descartar esta solución. Otra opción es utilizar 8 unidades funcionales de enteros, pero observamos que en muchos casos el tiempo empeora, aparte de que en el resto de casos la mejora es despreciable. Debido a estos motivos consideramos que la solución óptima es utilizar 4 unidades funcionales de enteros.

Todo esto depende del uso que se le vaya a dar al procesador con respecto a operaciones enteras y del coste de añadir un determinado número de unidades funcionales de enteros. Es un compromiso entre estos dos factores.

Por motivos evidentes de tiempo que se han comentado anteriormente consideramos que es mejor un sistema que cumple las características de este simulador en una versión asíncrona antes que en una versión síncrona.

BIBLIOGRAFÍA

- Principal fuente de documentación y código del proyecto para comenzar las modificaciones, pruebas y evaluación de resultados:

"Sim-async: an Architectural Simulator for Asynchronous Processor Modeling using Distribution Functions"

J. M. Colmenar (1), O. Garnica, J. Lanchares, J. I. Hidalgo,
G. Miñana, S. López (2)

(1) C. E. S. Felipe II, Complutense U. of Madrid
jmcolmenar©cesfelipesegundo.com

(2) Dept. of Computer Arch. and System Engineering,
Complutense University of Madrid
{ogarnica, julandan, hidalgo}@dacya.ucm.es,
guamiro@fdi.ucm.es, slopezal@dacya.ucm.es

- Fuentes de documentación secundarias:

1. D. Kearney, "Theoretical Limits on the Data Dependent Performance on Asynchronous Circuits," Proc. of Intl. Symposium on Aduanced Research in Asynchronous Circuits and Systems, pp. 201-207, 1999.

2. A. J. Martin, A. Lines, R. Manohar, M. Nystroem, P. Penzes, R. Southworth, and U. Cummings, "The Design of an Asynchronous MIPS R3000 Microprocessor," in Adu. Research in VLSI, pp. 164-181, 1997.

3. D. K. Arvind and R. D. Mullins, "A Fully Asynchronous Superscalar Architecture," in Proc. ofthe 1.999 Intl. Conf. on Parallel Architectures and Compilation Techniques (I. C. S. Press, ed.), pp. 17-22, 1999.

4. J. D. Garside, W. J. Bainbridge, A. Bardsley, D. M. Clark, D. A. Edwards, S. B. Furber, J. Liu, D. W. Lloyd, S. Mohammadi, J. S. Pepper, O. Petlin, S. Temple, and J. V. Woods, "AMULET3i - An Asynchronous System-on-Chip," in Proc. of the 6th Intl. Symposium on Aduanced Research in Asynchronous Circuits and Systems (I. C. S. Press, ed.), pp. 162-175, April 2000.

5. Q. Zhang and G. Theodoropoulos, "Modelling SAMIPS: a Synthesisable Asynchronous MIPS Processor," in Proc. ofthe 37th Annual Simulation Symposium, pp. 205-212, April 2004.

6. C. Chien, M. A. Franklin, 'I`. Pan, and P. Prabhu, “ARAS: Asynchronous RISC Architecture Simulator," Proc. ofthe 2nd Working Conference on Asynchronous Design Methodologies (ASYNC’.95}, 1995.

7. V. Rebello, On the Distribution of Control in Asynchronous Processor Architectures. PhD thesis, 1997.

8. 'I`. M. Austin, E. Larson, and D. Ernst, “SimpleScalar: An Infrastructure for Computer System Mode1ing," IEEE Computer Journal, vol. 35, 2, February 2002.

9. D. Sima, “Superscalar Instruction Issue,” IEEE Micro, vol. 17, pp. 28-39, Sep.-Oct. 1997.

10. F. Cheng, "Practical Design and Performance Evaluation of Completion Detection Circuits," in Proc. of the Intl. Conf. on Computer Design (I. C. S. Press, ed.), pp. 354-359, 1998.

11. A. J. Martin, "Asynchronous Datapaths and the Design of an Asynchornous Adder," Formal Methods in System Design, vol. 1, pp. 119-137, July 1992.

12. 'I`. H.-Y. Meng, R. W. Brodersen, and D. G. Messerschmitt, “Asynchronous Design for Programmable Digital Signal Processors,” IEEE Trans. on Signal Processing, vol. 39(4), pp. 939-952, 1991.

http://www.cs.utah.edu/classes/cs5810-elb/slides/dac99x4.pdf
http://intranet.cs.man.ac.uk/apt/projects/processors/amulet/
http://www.cs.utah.edu/classes/cs7940-010-rajeev/spr04/papers/nellans.pdf
http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.51.6915