(19) OFICINA ESPAÑOLA DE PATENTES Y MARCAS

ESPAÑA

(11) Número de publicación: **2 764 954**

(51) Int. CI.:

G06F 12/0815 (2006.01)
G06F 9/30 (2008.01)
G06F 9/38 (2008.01)
G06F 9/46 (2006.01)

(12) TRADUCCIÓN DE PATENTE EUROPEA T3

(86) Fecha de presentación y número de la solicitud internacional: **11.03.2015 PCT/EP2015/055019**

(87) Fecha y número de publicación internacional: **17.09.2015 WO15135967**

(96) Fecha de presentación y número de la solicitud europea: **11.03.2015 E 15710158 (5)**

(97) Fecha y número de publicación de la concesión europea: **04.12.2019 EP 3117323**

(54) Título: **Aumento de protocolo de coherencia para indicar estado de transacción**

(30) Prioridad:

14.03.2014 US 201414212217

(45) Fecha de publicación y mención en BOPI de la traducción de la patente:
05.06.2020

(73) Titular/es:

INTERNATIONAL BUSINESS MACHINES CORPORATION (100.0%)
New Orchard Road
Armonk, New York 10504, US

(72) Inventor/es:

SCHWARZ, ERIC, MARK;
BUSABA, FADI, YUSUF;
GSCHWIND, MICHAEL KARL;
SLEGEL, TIMOTHY;
SALAPURA, VALENTINA;
JACOBI, CHRISTIAN y
CAIN III, HAROLD, WADE

(74) Agente/Representante:

ISERN JARA, Jorge

DESCRIPCIÓN

Aumento de protocolo de coherencia para indicar estado de transacción

Antecedentes

La presente invención se refiere en general a protocolos de solicitud y respuesta, y más específicamente, a aumento de protocolo de coherencia para indicar estado de transacción.

El número de núcleos de unidad de procesamiento central (CPU) en un chip y el número de núcleos de CPU conectados a una memoria compartida continúa creciendo significativamente para soportar la demanda de capacidad de carga de trabajo creciente. El número creciente de CPU que cooperan para procesar las mismas cargas de trabajo impone una carga significativa en la escalabilidad de software; por ejemplo, las colas compartidas o estructuras de datos protegidas por semáforos tradicionales se vuelven puntos calientes y conducen a curvas de cambio de escala de n-sentidos sublineales. De manera tradicional, esto se ha contrarrestado implementando bloqueo de mayor resolución en software, y con interconexiones de latencia inferior/ancho de banda superior en hardware. Implementar el bloqueo de alta resolución para mejorar el cambio de escala de software puede ser muy complicado y propenso a errores, y en las frecuencias de las CPU de hoy en día, las latencias de interconexiones de hardware están limitadas por la dimensión física de los chips y sistemas, y por la velocidad de la luz.

Se ha introducido la memoria transaccional de hardware (HTM, o en este análisis, simplemente TM), en la que un grupo de instrucciones - denominadas una transacción - operan de una manera atómica en una estructura de datos en memoria, como se observa por otras unidades de procesamiento central (CPU) y el subsistema de E/S (la operación atómica también es conocida como bloqueo concurrente o serializada en otra bibliografía). La transacción se ejecuta de manera optimista sin obtener un bloqueo, pero puede necesitar abortar y reintentar la ejecución de transacción si una operación, de la transacción de ejecución, en una localización de memoria entra en conflicto con otra operación en la misma localización de memoria. Anteriormente, se han propuesto implementaciones de memoria transaccionales de software para soportar Memoria Transaccional (TM) de software.

De acuerdo con la presente invención, se proporciona ahora un método implementado por ordenador para implementar un protocolo de coherencia, comprendiendo el método: enviar una solicitud de datos a un procesador remoto; recibir, por un procesador, una respuesta del procesador remoto, incluyendo la respuesta un estado de transacción de una transacción remota en el procesador remoto, en el que el estado de transacción recibido en la respuesta del procesador remoto incluye: un tipo de interferencia en el procesador remoto provocada por una transacción solicitante que se ejecuta en el procesador, un número de ciclos de reloj de trabajo que se han realizado por la transacción remota antes de que se aborten en el procesador remoto, y una indicación de si se provocó una restauración en el procesador remoto enviando la solicitud al procesador remoto; y añadir, por el procesador, el estado de transacción de la transacción remota en el procesador remoto en una tabla de rastreo de interferencia de transacción; en el que el procesador es un procesador separado del procesador remoto. Ejemplos de sistemas de gestión de memoria informática convencionales se describen en los documentos US 7.421.544 B1 y US 2010/332721 A1.

Sumario

La invención se expone en las reivindicaciones independientes adjuntas. Se definen realizaciones ventajosas por las reivindicaciones dependientes adjuntas. Las realizaciones o ejemplos de la siguiente descripción que no están cubiertos por las reivindicaciones adjuntas se consideran que no son parte de la invención de acuerdo con esta descripción.

Breve descripción de las varias vistas de los dibujos

La materia objeto que se considera como realizaciones se señala particularmente y se reivindica de manera distinta en las reivindicaciones en la conclusión de la memoria descriptiva. Las características anteriores y otras, y las ventajas de las realizaciones son evidentes a partir de la siguiente descripción detallada tomada en conjunto con los dibujos adjuntos en los que:

La Figura 1 representa un diagrama de bloques esquemático de un procesador múltiple (CPU)/entorno de Memoria Transaccional de núcleo de ejemplo de acuerdo con una realización;

La Figura 2 representa un diagrama de bloques esquemático que ilustra un procesador transaccional de acuerdo con una realización;

La Figura 3 representa un diagrama de bloques esquemático de componentes ejemplares de un procesador transaccional (CPU) mostrado en las Figuras 1 y 2, de acuerdo con una realización;

La Figura 4 representa un diagrama de bloques esquemático de un sistema informático que tiene componentes como los sistemas de múltiples procesadores mostrados en las Figuras 1-3, para permitir solicitudes y respuestas

en un entorno de memoria transaccional de hardware de acuerdo con una realización;

La Figura 5 representa una solicitud y respuesta de protocolo ejemplar de acuerdo con una realización;

La Figura 6 representa una solicitud de protocolo ejemplar de acuerdo con una realización;

La Figura 7 representa un diagrama de flujo de generación de solicitud de protocolo por un procesador que realiza una solicitud de datos de acuerdo con una realización;

La Figura 8 representa un diagrama de flujo de manejo de solicitud por el procesador de recepción/remoto que recibe la solicitud y envía una respuesta de acuerdo con una realización;

La Figura 9 representa un diagrama de flujo que ilustra el manejo de transacción por un procesador de acuerdo con una realización;

La Figura 10 representa la solicitud de protocolo y una nueva respuesta de protocolo de acuerdo con una realización;

La Figura 11 representa la solicitud de escritura de protocolo y una nueva respuesta de acuerdo con una realización;

La Figura 12 representa un diagrama de flujo que ilustra el manejo de solicitud de coherencia por el procesador de recepción/remoto que recibe la solicitud de acuerdo con una realización;

La Figura 13 representa un diagrama de flujo que ilustra el origen de la solicitud de protocolo y procesamiento por el procesador solicitante de acuerdo con una realización;

La Figura 14 representa un diagrama de flujo que ilustra el manejo de transacción por un procesador de acuerdo con una realización;

La Figura 15 representa un diagrama de flujo que ilustra cómo el procesador responde a indicación de interferencia en una tabla de almacenamiento de interferencia de transacción de rastreo local de acuerdo con una realización;

La Figura 16 representa un diagrama de flujo que ilustra cómo el procesador responde a indicación de interferencia en la tabla de almacenamiento de interferencia de transacción de rastreo local de acuerdo con una realización;

La Figura 17 representa un método para manejo de protocolo de coherencia de acuerdo con una realización; y

la Figura 18 representa un medio legible por ordenador de acuerdo con una realización.

Descripción detallada

Los sistemas de múltiples procesadores usan protocolos de coherencia para mantener la consistencia entre todas las cachés en un sistema de memoria compartida distribuido. Cuando se realiza una solicitud de datos de una cierta caché, la caché emite los datos y actualiza su estado de que ya no tiene los datos, o que los datos no se mantienen de manera exclusiva. Si un procesador está en una ejecución transaccional, y se solicitan los datos de su caché que son una parte de la transacción, el procesador abortará la transacción, y enviará los datos.

No se proporciona información de si una solicitud ha provocado otra transacción para abortar. En algunos casos, será deseable notificar al solicitante original de si una solicitud impactó otra transacción, para proporcionar realimentación y permitir al originador del solicitante adaptar su ejecución, por ejemplo, para detectar escenarios de bloqueo activo y tratar otros escenarios de degradación de rendimiento.

De acuerdo con la realización, el protocolo de coherencia se amplía para incluir información adicional acerca del estado de transacción. Cuando un procesador está en ejecución transaccional, una solicitud de coherencia puede provocar su ejecución para abortar, por ejemplo, puesto que los datos son parte de la lectura o escritura de transacción establecidas, y se detecta un conflicto. La solicitud de protocolo de coherencia se amplía con información adicional que él (el procesador que recibe la solicitud de coherencia) abortó una transacción durante la ejecución transaccional de acuerdo con las realizaciones.

"Power ISA™ Versión 2.07 publicado el 22 de mayo de 2013 de IBM® enseña una arquitectura de conjunto de instrucciones (ISA) de ordenador de conjunto reducido de instrucciones (RISC) de ejemplo". También, "z/Architecture Principles of Operation" SA22-7832-09 (agosto de 2012) de IBM® enseña una arquitectura de conjunto de instrucciones CISC (ordenador de conjunto de instrucciones completas)").

Históricamente, un sistema informático o procesador tenía únicamente un único procesador (también conocido como

unidad de procesamiento o unidad de procesamiento central). El procesador incluía una unidad de procesamiento de instrucciones (IPU), una unidad de ramal, una unidad de control de memoria y similares. Tales procesadores eran capaces de ejecutar un único hilo de un programa a la vez. Se desarrollaron sistemas operativos que podrían compartir en tiempo un procesador despachando un programa a ejecutarse en el procesador para un periodo de tiempo, y a continuación despachando otro programa a ejecutarse en el procesador para otro periodo de tiempo. A medida que la tecnología evoluciona, se añadieron las cachés de subsistema de memoria a menudo al procesador así como la traducción de dirección dinámica compleja que incluye memorias intermedias laterales de traducción (TLB). La misma IPU a menudo se denominó como un procesador. A medida que la tecnología continuó evolucionando, un procesador entero, podría empaquetarse como un único chip o molde de semiconductores, un procesador de este tipo se denominó como un microprocesador. A continuación se desarrollaron procesadores que incorporaban múltiples IPU, tales procesadores se denominaron como multiprocesadores. Cada procesador de este tipo de un sistema informático de múltiples procesadores (procesador) puede incluir cachés individuales o compartidas, interfaces de memoria, bus de sistema, mecanismo de traducción de dirección y similares. Los emuladores de máquina virtual y arquitectura de conjunto de instrucciones (ISA) añadieron una capa de software a un procesador, que proporcionó a la máquina virtual con múltiples "procesadores virtuales" (también conocido como procesadores) mediante el uso de intervalos de tiempo de una única IPU en un único procesador de hardware. A medida que la tecnología evolucionó adicionalmente, se desarrollaron procesadores de múltiples hilos, que posibilitan que un único procesador de hardware tenga una única IPU de múltiples hilos para proporcionar una capacidad de ejecución de manera simultánea hilos de diferentes programas, por lo tanto cada hilo de un procesador de múltiples hilos aparece para el sistema operativo como un procesador. A medida que la tecnología evolucionaba adicionalmente, era posible poner múltiples procesadores (teniendo cada uno una IPU) en un único chip o molde de semiconductores. Estos procesadores se denominaron como núcleos de procesador o simplemente núcleos. Por lo tanto los términos tales como procesador, unidad de procesamiento central, unidad de procesamiento, microprocesador, núcleo, núcleo de procesador, hilo de procesador, hilo, por ejemplo, a menudo se usan de manera intercambiable. Los de las realizaciones en el presente documento pueden ponerse en práctica por cualquiera o todos los procesadores incluyendo aquellos conocidos mencionados anteriormente, sin alejarse de las enseñanzas en el presente documento. En las que se usa el término "hilo" o "hilo de procesador" en el presente documento, se espera que pueda tenerse esa ventaja particular de la realización en una implementación de hilo de procesador.

Ejecución de transacción en realizaciones basadas en Intel®

En "Intel® Architecture Instruction Set Extensions Programming Reference" 319433-012A, febrero de 2012, capítulo 8 enseña, en parte, que las aplicaciones de múltiples hilos pueden aprovecharse de los números crecientes de núcleos de CPU para conseguir rendimiento superior. Sin embargo, la escritura de aplicaciones de múltiples hilos requiere que los programadores entiendan y tengan en cuenta la compartición de datos entre los múltiples hilos. El acceso a datos compartidos típicamente requiere mecanismos de sincronización. Estos mecanismos de sincronización se usan para asegurar que múltiples hilos actualicen datos compartidos serializando operaciones que se aplican a los datos compartidos, a menudo a través del uso de una sección crítica que se protege por un bloqueo. Puesto que la serialización limita la concurrencia, los programadores intentan limitar la sobrecarga debido a la sincronización.

Extensiones de Sincronización Transaccionales de Intel® (Intel® TSX) permite que un procesador determine dinámicamente si los hilos necesitan serializarse a través de secciones críticas protegidas por bloqueo, y realicen esa serialización únicamente cuando se requiera. Esto permite que el procesador exponga y aproveche la concurrencia que está oculta en una aplicación debido a la sincronización dinámicamente innecesaria.

Con Intel TSX, las regiones de código especificadas por el programador (también denominadas como "regiones transaccionales" o solamente "transacciones") se ejecutan de manera transaccional. Si la ejecución transaccional se completa satisfactoriamente, entonces todas las operaciones de memoria realizadas en la región transaccional parecerán que han tenido lugar de manera instantánea cuando se observan desde otros procesadores. Un procesador hace las operaciones de memoria de la transacción ejecutada, realizadas en la región transaccional, visibles a otros procesadores únicamente cuando tiene lugar una confirmación satisfactoria, es decir, cuando la transacción completa satisfactoriamente la ejecución. Este proceso a menudo se denomina como una confirmación atómica.

Intel TSX proporciona dos interfaces de software para especificar regiones de código para su ejecución transaccional. Supresión de Bloqueo de Hardware (HLE) es una extensión de conjunto de instrucciones compatible heredada (que comprende los prefijos XACQUIRE y XRELEASE) para especificar regiones transaccionales. Memoria Transaccional Restringida (RTM) es una nueva interfaz de conjunto de instrucciones (que comprende las instrucciones XBEGIN, XEND, y XABORT) para que los programadores definan regiones transaccionales de una manera más flexible que lo que era posible con HLE. HLE es para programadores que prefieren la compatibilidad hacia atrás del modelo de programación de exclusión mutua convencional y que probablemente ejecutarían software habilitado para HLE en hardware heredado pero que también desearían aprovecharse de las nuevas capacidades de supresión de bloqueo en hardware con soporte de HLE. RTM es para programadores que prefieren una interfaz flexible para el hardware de ejecución transaccional. Además, Intel TSX también proporciona una instrucción XTEST. Esta instrucción permite que el software consulte si el procesador lógico se está ejecutando de manera transaccional en una región transaccional identificada por cualquiera de HLE o RTM.

Puesto que una ejecución transaccional satisfactoria asegura una confirmación atómica, el procesador ejecuta la región de código de manera optimista sin sincronización explícita. Si la sincronización era innecesaria para esa ejecución específica, la ejecución puede confirmar sin ninguna serialización de hilo cruzado. Si el procesador no puede confirmar de manera atómica, entonces la ejecución optimista falla. Cuando esto ocurre, el procesador restaurará la ejecución, un proceso denominado como un aborto transaccional. En un aborto transaccional, el procesador descartará todas las actualizaciones realizadas en la región de memoria usada por la transacción, restaurará el estado de arquitectura para que parezca como si la ejecución optimista nunca hubiera tenido lugar, y reanudará la ejecución de manera no transaccional.

Un procesador puede realizar un aborto transaccional por numerosas razones. Una razón principal para abortar una transacción es debido a accesos de memoria en conflicto entre el procesador lógico que ejecuta de manera transaccional y otro procesador lógico. Tales accesos de memoria que entra en conflicto pueden evitar una ejecución transaccional satisfactoria. Las direcciones de memoria leídas de una región transaccional constituyen el ajuste de lectura de la región transaccional y las direcciones escritas en la región transaccional constituyen el ajuste de escritura de la región transaccional. Intel TSX mantiene los conjuntos de lectura y escritura a la granularidad de una línea de caché. Un acceso de memoria que entra en conflicto tiene lugar si otro procesador lógico lee una localización que es parte del ajuste de escritura de la región transaccional o escribe una localización que es una parte de cualquiera del ajuste de lectura o escritura de la región transaccional. Un acceso que entra en conflicto típicamente significa que se requiere serialización para esta región de código. Puesto que Intel TSX detecta conflictos de datos a la granularidad de una línea de caché, las localizaciones de datos no relacionadas en la misma línea de caché se detectarán como conflictos que dan como resultado abortos transaccionales. Los abortos transaccionales pueden tener lugar también debido a recursos transaccionales limitados. Por ejemplo, la cantidad de datos accedida en la región puede superar una capacidad específica de implementación. Adicionalmente, algunas instrucciones y eventos de sistema pueden provocar abortos transaccionales. Los abortos transaccionales frecuentes dan como resultado ciclos desperdiciados e ineficacia aumentada.

Supresión de bloqueo de hardware

La Supresión de Bloqueo de Hardware (HLE) proporciona una interfaz de conjunto de instrucciones compatible heredada para que los programadores usen la ejecución transaccional. HLE proporciona dos nuevas sugerencias de prefijo de instrucción: XACQUIRE y XRELEASE.

Con HLE, un programador añade el prefijo XACQUIRE a la parte delantera de la instrucción que se usa para obtener el bloqueo que está protegiendo la sección crítica. El procesador trata el prefijo como una sugerencia para evitar la escritura asociada con la operación de obtención de bloqueo. Incluso aunque el bloqueo obtenga una operación de escritura asociada al bloqueo, el procesador no añade la dirección del bloqueo al ajuste de escritura de la región transaccional ni emite ninguna solicitud de escritura al bloqueo. En su lugar, la dirección del bloqueo se añade al ajuste de lectura. El procesador lógico entra en la ejecución transaccional. Si el bloqueo estaba disponible antes de la instrucción prefijada XACQUIRE, entonces todos los demás procesadores continuarán para observar el bloqueo como si estuviera disponible posteriormente. Puesto que el procesador lógico que ejecuta de manera transaccional ni añade la dirección del bloqueo a su ajuste de escritura ni realiza operaciones de escritura externamente visibles al bloqueo, otros procesadores lógicos pueden leer el bloqueo sin provocar un conflicto de datos. Esto permite que otros procesadores lógicos también entren en y ejecuten concurrentemente la sección crítica protegida por el bloqueo. El procesador detecta automáticamente cualesquiera conflictos de datos que tienen lugar durante la ejecución transaccional y realizará un aborto transaccional si fuera necesario.

Incluso aunque el procesador de supresión no realizara ninguna operación de escritura externa al bloqueo, el hardware asegura el orden de programa de las operaciones en el bloqueo. Si el mismo procesador de supresión lee el valor del bloqueo en la sección crítica, aparecerá como si el procesador hubiera obtenido el bloqueo, es decir la lectura devolverá el valor no suprimido. Este comportamiento permite que una ejecución de HLE sea funcionalmente equivalente a una ejecución sin los prefijos HLE.

Un prefijo XRELEASE puede añadirse en la parte delantera de una instrucción que se usa para liberar el bloqueo que protege una sección crítica. Liberar el bloqueo implica una escritura al bloqueo. Si la instrucción es para restaurar el valor del bloqueo al valor del bloqueo que tuviera anteriormente a la operación de obtención de bloqueo prefijada XACQUIRE en el mismo bloqueo, a continuación el procesador suprime la solicitud de escritura externa asociada con la liberación del bloqueo y no añade la dirección del bloqueo al ajuste de escritura. El procesador a continuación intenta confirmar la ejecución transaccional.

Con HLE, si múltiples hilos ejecutan secciones críticas protegidas por el mismo bloqueo pero no realizan ninguna operación que entre en conflicto en cada uno de los otros datos, a continuación los hilos pueden ejecutarse de manera concurrente sin serialización. Incluso aunque el software use operaciones de obtención de bloqueo en un bloqueo común, el hardware reconoce esto, suprime el bloqueo, y ejecuta las secciones críticas en los dos hilos sin requerir ninguna comunicación a través del bloqueo - si tal comunicación fuera dinámicamente innecesaria.

Si el procesador no puede ejecutar la región de manera transaccional, a continuación el procesador ejecutará la región

de manera no transaccional sin supresión. El software habilitado para HLE tiene las mismas garantías de progreso hacia adelante que la ejecución basada en bloqueo no de HLE subyacente. Para la ejecución de HLE satisfactoria, el bloqueo y el código de la sección crítica deberán seguir ciertas directrices. Estas directrices únicamente afectan al rendimiento; y el fallo al seguir estas directrices no dará como resultado un fallo funcional. El hardware sin soporte de HLE ignorará las sugerencias de prefijo XACQUIRE y XRELEASE y no realizará ninguna supresión puesto que estos prefijos corresponden a los prefijos REPNE/REPE IA-32 que se ignoran en las instrucciones donde son válidos XACQUIRE y XRELEASE. De manera importante, HLE es compatible con el modelo de programación basado en bloqueo existente. El uso inapropiado de las sugerencias no provocará fallos de programación funcionales aunque puede exponer fallos de programación latentes ya en el código.

La Memoria Transaccional Restringida (RTM) proporciona una interfaz de software flexible para su ejecución transaccional. RTM proporciona tres nuevas instrucciones XBEGIN, XEND, y XABORT para que los programadores inicien, confirmen, y aborten una ejecución transaccional.

El programador usa la instrucción XBEGIN pasa especificar el inicio de una región de código transaccional y la instrucción XEND para especificar el fin de la región de código transaccional. Si la región RTM no pudiera ejecutarse satisfactoriamente de manera transaccional, entonces la instrucción XBEGIN toma un operando que proporciona un desplazamiento relativo a la dirección de instrucción de repliegue.

Un procesador puede abortar la ejecución transaccional de RTM por muchas razones. En muchos casos, el hardware detecta automáticamente condiciones de aborto transaccionales y reinicia la ejecución de la dirección de instrucción de repliegue con el estado de arquitectura que corresponde a la presente en el inicio de la instrucción XBEGIN y el registro de EAX actualizado para describir el estado de aborto.

La instrucción XABORT permite que los programadores aborten la ejecución de una región RTM explícitamente. La instrucción XABORT toma un argumento inmediato de 8 bits que se carga en el registro de EAX y por lo tanto estará disponible para software que sigue un aborto de RTM. Las instrucciones RTM no tienen ninguna localización de memoria de datos asociada con ellas. Mientras que el hardware no proporciona garantías en cuanto a si una región RTM se confirmará alguna vez satisfactoriamente de manera transaccional, la mayoría de las transacciones que siguen las directrices recomendadas se espera que se confirmen satisfactoriamente de manera transaccional. Sin embargo, los programadores deben siempre proporcionar una secuencia de código alternativa en la ruta de repliegue para garantizar el progreso hacia adelante. Esto puede ser tan sencillo como un bloqueo y ejecución de la región de código especificada de manera no transaccional. Además, una transacción que siempre se aborta en una implementación dada puede completarse de manera transaccional en una implementación futura. Por lo tanto, los programadores deben asegurar las rutas de código para la región transaccional y la secuencia de código alternativa se prueban de manera funcional.

Detección de soporte de HLE

Un procesador soporta ejecución de HLE si CPUID.07H.EBX.HLE [bit 4] = 1. Sin embargo, una aplicación puede usar los prefijos HLE (XACQUIRE y XRELEASE) sin comprobar si el procesador soporta HLE. Los procesadores sin HLE soportan ignorar estos prefijos y ejecutarán el código sin entrar en ejecución transaccional.

Detección de soporte de RTM

Un procesador soporta ejecución de RTM si CPUID.07H.EBX.RTM [bit 11] = 1. Una aplicación debe comprobar si el procesador soporta RTM antes de que use las instrucciones RTM (XBEGIN, XEND, XABORT). Estas instrucciones generarán una excepción #UD cuando se usan en un procesador que no soportan RTM.

Detección de instrucción XTEST

Un procesador soporta la instrucción XTEST si soporta cualquiera de HLE o RTM. Una aplicación debe comprobar cualquiera de estas banderas características antes de usar la instrucción XTEST. Esta instrucción generará una excepción #UD cuando se usa en un procesador que no soporta cualquiera de HLE o RTM.

Consultar estado de ejecución transaccional

La instrucción XTEST puede usarse para determinar el estado transaccional de una región transaccional especificada por HLE o RTM. Obsérvese que mientras que los prefijos HLE se ignoran en procesadores que no soportan HLE, la instrucción XTEST generará una excepción #UD cuando se usa en procesadores que no soportan cualquiera de HLE o RTM.

Requisitos de bloqueos de HLE

Para ejecución de HLE para confirmar satisfactoriamente de manera transaccional, el bloqueo debe satisfacer ciertas propiedades y el acceso al bloqueo que debe seguir ciertas directrices.

Una instrucción prefijada XRELEASE debe restaurar el valor del bloqueo suprimido al valor que tenía antes de la obtención del bloqueo. Esto permite que el hardware suprima de manera segura bloqueos no añadiéndolos al ajuste de escritura. El tamaño de datos y la dirección de datos de la instrucción de liberación de bloqueo (XRELEASE prefijada) debe coincidir con los de la obtención de bloqueo (XACQUIRE prefijada) y el bloqueo no debe cruzar un límite de línea de caché.

El software no debería escribir el bloqueo suprimido dentro de una región de HLE transaccional con ninguna instrucción distinta de una instrucción prefijada XRELEASE, de lo contrario una escritura de este tipo puede provocar un aborto transaccional. Además, los bloqueos recursivos (donde un hilo obtiene el mismo bloqueo múltiples veces sin liberar en primer lugar el bloqueo) pueden también provocar un aborto transaccional. Obsérvese que el software puede observar el resultado del bloqueo suprimido dentro de la sección crítica. Una operación de lectura de este tipo devolverá el valor de la escritura al bloqueo.

El procesador detecta automáticamente violaciones a estas directrices, y pasa de manera segura a una ejecución no transaccional sin supresión. Puesto que Intel TSX detecta conflictos a la resolución de una línea de caché, escribe en datos colocados en la misma línea de caché ya que el bloqueo suprimido puede detectarse como conflictos de datos por otros procesadores lógicos que suprimen el mismo bloqueo.

Anidación transaccional

Tanto HLE como RTM soportan regiones transaccionales anidadas. Sin embargo, un aborto transaccional restaura el estado a la operación que inició la ejecución transaccional: ya sea la instrucción elegible de HLE prefijada de XACQUIRE o la instrucción XBEGIN más exterior. El procesador trata todas las transacciones anidadas como una transacción.

Anidación y supresión de HLE

Los programadores pueden anidar regiones HLE hasta una profundidad específica de implementación de MAX_HLE_NEST_COUNT. Cada procesador lógico rastrea el contador de anidación internamente pero este contador no está disponible para software. Una instrucción elegible de HLE prefijada de XACQUIRE incrementa el contador de anidación y una instrucción elegible de HLE prefijada de XRELEASE lo decrementa. El procesador lógico entra en ejecución transaccional cuando el contador de anidación va de cero a uno. El procesador lógico intenta confirmar únicamente cuando el contador de anidación se hace cero. Un aborto transaccional puede tener lugar si el contador de anidación supera MAX_HLE_NEST_COUNT.

Además de soportar regiones de HLE anidadas, el procesador puede suprimir también múltiples bloqueos anidados. El procesador rastrea un bloqueo para supresión que comienza con la instrucción elegible de HLE prefijada de XACQUIRE para ese bloqueo y que finaliza con la instrucción elegible de HLE prefijada de XRELEASE para ese mismo bloqueo. El procesador puede rastrear, en un momento cualquiera, hasta un número MAX_HLE_ELIDED_LOCKS de bloqueos. Por ejemplo, si la implementación soporta un valor MAX_HLE_ELIDED LOCKS de dos y si el programador anida tres secciones críticas identificadas de HLE (realizando instrucciones elegibles de HLE prefijadas de XACQUIRE en tres bloqueos distintos que realizan una instrucción elegible de HLE prefijada de XRELEASE intermedia en uno cualquiera de los bloqueos), a continuación se suprimirán los dos primeros bloqueos, pero el tercero no se suprime (sino que se añadirá al ajuste de escritura de la transacción). Sin embargo, la ejecución continuará aún de manera transaccional. Una vez que se encuentra una XRELEASE para uno de los dos bloqueos suprimidos, se suprimirá un bloqueo posterior obtenido a través de la instrucción elegible de HLE prefijada de XACQUIRE.

El procesador intenta confirmar la ejecución de HLE cuando se han hecho coincidir todos los pares XACQUIRE y XRELEASE suprimidos, el contador de anidación va a cero, y los bloqueos tienen requisitos satisfechos. Si la ejecución no puede confirmarse de manera atómica, a continuación la ejecución pasa a una ejecución no transaccional sin supresión como si la primera instrucción no tuviera un prefijo XACQUIRE.

Anidación de RTM

Los programadores pueden anidar regiones RTM hasta un MAX_RTM_NEST_COUNT específico de la implementación. El procesador lógico rastrea el contador de anidación internamente pero este contador no está disponible para software. Una instrucción XBEGIN incrementa el contador de anidación, y una instrucción XEND decrementa el contador de anidación. El procesador lógico intenta confirmar únicamente si el contador de anidación se vuelve cero. Un aborto transaccional tiene lugar si el contador de anidación supera MAX_RTM_NEST_COUNT.

Anidar HLE y RTM

HLE y RTM proporcionan dos interfaces de software alternativas para una capacidad de ejecución transaccional común. El comportamiento de procesamiento transaccional es específico de la implementación cuando se anidan

juntos HLE y RTM, por ejemplo, HLE está dentro de RTM o RTM está dentro de HLE. Sin embargo, en todos los casos, la implementación mantendrá las semánticas de HLE y RTM. Una implementación puede elegir ignorar sugerencias de HLE cuando se usan dentro de regiones RTM, y puede provocar un aborto transaccional cuando se usan instrucciones de RTM dentro de regiones de HLE. En el último caso, la transacción de ejecución transaccional a no transaccional tiene lugar sin interrupciones puesto que el procesador re-ejecutará la región de HLE sin realmente hacer supresión, y a continuación ejecutará las instrucciones de RTM.

Definición de estado de aborto

RTM usa el registro de EAX para comunicar el estado de aborto al software. Después de un aborto de RTM el registro de EAX tiene la siguiente definición.

TABLA 1

	Definición de estado de aborto de RTM
Posición de bit de registro de EAX	Significado
0	Establecer si aborto provocado por instrucción XABORT
1	Si está establecido, la transacción puede tener éxito en el reintento, este bit siempre está limpio si se establece el bit 0
2	Establecer si otro procesador lógico en conflicto con una dirección de memoria que era parte de la transacción que abortó
3	Establecer si una memoria intermedia interna está sobrecargada
4	Establecer si se alcanzó un punto de ruptura de depuración
5	Establecer si tuvo lugar un aborto durante la ejecución de una transacción anidada
Posición de bit de registro de EAX	Significado
23:6	Reservado
31-24	argumento XABORT (únicamente válido si el bit 0 está establecido, de lo contrario reservado)

El estado de aborto de *EAX* para RTM únicamente proporciona causas para abortos. No se codificará por sí mismo si tuvo lugar un aborto o confirmación para la región RTM. El valor de EAX puede ser 0 que sigue un aborto de RTM. Por ejemplo, una instrucción de CPUID cuando se usa dentro de una región RTM provoca un aborto transaccional y puede no satisfacer los requisitos para establecer cualesquiera de los bits de EAX. Esto puede dar como resultado un valor de EAX de 0.

Ordenación de memoria de RTM

Una confirmación de RTM satisfactoria provoca que todas las operaciones de memoria en la región RTM parezca que se ejecutan de manera atómica. Una región de RTM confirmada de manera satisfactoria que consiste en un XBEGIN seguido por un XEND, incluso sin operaciones de memoria en la región RTM, tiene las mismas semánticas de ordenación como una instrucción prefijada LOCK.

La instrucción XBEGIN no tiene semántica de cercado. Sin embargo, si se aborta una ejecución de RTM, a continuación todas las actualizaciones de memoria dentro de la región RTM se descartan y no se hacen visibles a ningún otro procesador lógico.

Soporte de depurador activado para RTM

Por defecto, cualquier excepción de depuración dentro de una región RTM provocará un aborto transaccional y redirigirá el flujo de control a la dirección de instrucción de repliegue con estado de arquitectura recuperado y bit 4 en cada ajuste de EAX. Sin embargo, para permitir que los depuradores de software intercepten la ejecución en excepciones de depuración, la arquitectura RTM proporciona capacidad adicional.

Si el bit 11 de DR7 y el bit 15 de IA32_DEBUGCTL_MSR son ambos 1, cualquier aborto de RTM debido a una excepción de depuración (#DB) o excepción de punto de ruptura (#BP) provoca la ejecución para restaurar y reiniciar desde la instrucción XBEGIN en lugar de la dirección de repliegue. En este escenario, el registro de EAX también se restaurará de vuelta al punto de la instrucción XBEGIN.

Consideraciones de programación

Las regiones identificadas de programador típicas se espera que se ejecuten de manera transaccional y se confirmen de manera satisfactoria. Sin embargo, Intel TSX no proporciona ninguna garantía de este tipo. Una ejecución

transaccional puede abortarse por muchas razones. Para aprovechar al máximo las capacidades transaccionales, los programadores deberían seguir ciertas directrices para aumentar la probabilidad de su confirmación de ejecución transaccional de manera satisfactoria.

Esta sección analiza diversos eventos que pueden provocar abortos transaccionales. La arquitectura asegura que las actualizaciones realizadas dentro de una transacción que aborta posteriormente la ejecución nunca será visible. Únicamente las ejecuciones transaccionales confirmadas inician una actualización al estado de arquitectura. Los abortos transaccionales nunca provocan fallos funcionales y únicamente afectan al rendimiento.

Consideraciones basadas en instrucción

Los programadores pueden usar cualquier instrucción de manera segura dentro de una transacción (HLE o RTM) y pueden usar transacciones en cualquier nivel de privilegio. Sin embargo, algunas instrucciones siempre abortarán la ejecución transaccional y provocan la ejecución para pasar de manera sin interrupción y segura a una ruta no transaccional.

Intel TSX permite que se use la mayoría de las instrucciones comunes dentro de transacciones sin provocar abortos. Las siguientes operaciones dentro de una transacción típicamente no provocan un aborto:

- Las operaciones en el registro de puntero de instrucción, registros de fin general (GPR) y las banderas de estado (CF, OF, SF, PF, AF, y ZF); y
- Las operaciones en registros XMM y YMM y el registro MXCSR.

Sin embargo, los programadores deben ser cuidadosos cuando se entremezclan operaciones SSE y AVX dentro de una región transaccional. Entremezclar instrucciones de SSE que acceden a registros XMM e instrucciones de AVX que acceden a registros YMM puede provocar que se aborten las transacciones. Los programadores pueden usar operaciones de cadena prefijadas REP/REPNE dentro de transacciones. Sin embargo, las cadenas largas pueden provocar abortos. Además, el uso de instrucciones CLD y STD pueden provocar abortos si cambian el valor de la bandera DF. Sin embargo, si DF es 1, la instrucción STD no provocará un aborto. De manera similar, si DF es 0, entonces la instrucción CLD no provocará un aborto.

Las instrucciones no enumeradas en este punto como que provocan un aborto cuando se usan dentro de una transacción típicamente no provocarán que aborte una transacción (ejemplos incluyen, pero sin limitación, MFENCE, LFENCE, SFENCE, RDTSC, RDTSCP, etc.).

Las siguientes instrucciones abortarán la ejecución transaccional en cualquier implementación:

- XABORT
- CPUID
- PAUSE

Además, en algunas implementaciones, las siguientes instrucciones pueden provocar siempre abortos transaccionales. Estas instrucciones no se espera que se usen de manera común dentro de regiones transaccionales típicas. Sin embargo, los programadores no deben basarse en estas instrucciones para forzar un aborto transaccional, puesto que si provocan abortos transaccionales depende de la implementación.

- Las operaciones en el estado de arquitectura X87 y MMX. Esto incluye todas las instrucciones MMX y X87, que incluyen las instrucciones FXRSTOR y FXSAVE.
- Actualizar porción no de estado de EFLAGS: CLI, STI, POPFD, POPFQ, CLTS.
- Instrucciones que actualizan registros de segmento, registros de depuración y/o registros de control: MOV a DS/ES/FS/GS/SS, POP DS/ES/FS/GS/SS, LDS, LES, LFS, LGS, LSS, SWAPGS, WRFSBASE, WRGSBASE, LGDT, SGDT, LIDT, SIDT, LLDT, SLDT, LTR, STR, Far CALL, Far JMP, Far RET, IRET, MOV a DRx, MOV a CR0/CR2/CR3/CR4/CR8 y LMSW.
- Transiciones de anillo: SYSENTER, SYSCALL, SYSEXIT, y SYSRET.
- TLB y control de capacidad de almacenamiento en caché: CLFLUSH, INVD, WBINVD, INVLPG, INVPCID, e instrucciones de memoria con una sugerencia no temporal (MOVNTDQA, MOVNTDQ, MOVNTI, MOVNTPD, MOVNTPS, y MOVNTQ).
- Grabación de estado de procesador: XSAVE, XSAVEOPT, y XRSTOR.

- Interrupciones: INTn, INTO.
- IO: IN, INS, REP INS, OUT, OUTS, REP OUTS y sus variantes.
- VMX: VMPTRLD, VMPTRST, VMCLEAR, VMREAD, VMWRITE, VMCALL, VMLAUNCH, VMRESUME, VMXOFF, VMXON, INVEPT, e INVVPID.
- SMX: GETSEC.
- UD2, RSM, RDMSR, WRMSR, HLT, MONITOR, MWAIT, XSETBV, VZEROUPPER, MASKMOVQ, y V/MASK-MOVDQU.

Consideraciones de tiempo de ejecución

Además de las consideraciones basadas en instrucción, los eventos de tiempo de ejecución pueden provocar que aborte la ejecución transaccional. Estos pueden ser debido a patrones de acceso de datos o características de implementación de micro-arquitectura. La siguiente lista es un análisis no comprensivo de todas las causas de aborto.

Se suprimirá cualquier fallo o trampa en una transacción que debe exponerse a software. La ejecución transaccional abortará y la ejecución pasará a una ejecución no transaccional, como si nunca hubiera tenido lugar el fallo o trampa. Si no se enmascara una excepción, a continuación una excepción no enmascarada dará como resultado un aborto transaccional y el estado aparecerá como si la excepción nunca hubiera tenido lugar.

Los eventos de excepción síncronos (#DE, #OF, #NP, #SS, #GP, #BR, #UD, #AC, #XF, #PF, #NM, #TS, #MF, #DB, #BP/INT3) que tienen lugar durante la ejecución transaccional pueden provocar que una ejecución no se confirme de manera transaccional, y requerir una ejecución no transaccional. Estos eventos se suprimen como si nunca hubieran tenido lugar. Con HLE, puesto que el código no transaccional es idéntico a la ruta de código transaccional, estos eventos típicamente re-aparecerán cuando la instrucción que provocó la excepción se re-ejecuta de manera no transaccional, provocando que se entreguen de manera apropiada eventos síncronos asociados en la ejecución no transaccional. Los eventos asíncronos (NMI, SMI, INTR, IPI, PMI, etc.) que tienen lugar durante la ejecución transaccional pueden provocar que se aborte la ejecución transaccional y pasar a una ejecución no transaccional. Los eventos asíncronos quedarán pendientes y se manejarán después de que se procese el aborto transaccional.

Las transacciones únicamente soportan operaciones de tipo de memoria que pueden almacenarse en caché de escritura de vuelta. Una transacción puede abortar siempre si la transacción incluye operaciones en cualquier tipo de memoria. Esto incluye capturas de instrucción a tipo de memoria de UC.

Los accesos de memoria en una región transaccional pueden requerir que el procesador establezca las banderas accedidas y sucias de la entrada de la tabla de página referenciada. El comportamiento de cómo el procesador maneja esto es específico de la implementación. Algunas implementaciones pueden permitir las actualizaciones a estas banderas para que se hagan externamente visibles incluso si la región transaccional se aborta posteriormente. Algunas implementaciones de Intel TSX pueden elegir abortar la ejecución transaccional si estas banderas necesitan actualizarse. Además, un paseo por la tabla de página del procesador puede generar accesos para su propio estado no conformado pero transaccionalmente escrito. Algunas implementaciones de Intel TSX pueden elegir abortar la ejecución de una región transaccional en tales situaciones. Independientemente, la arquitectura asegura que, si la región transaccional se aborta, entonces el estado transaccionalmente escrito no se hará visible a nivel de arquitectura a través del comportamiento de estructuras tales como TLB.

Ejecutar el código de auto-modificación de manera transaccional puede provocar también abortos transaccionales. Los programadores deben continuar para seguir las directrices recomendadas de Intel para escribir código de auto-modificación y modificación-cruzada incluso cuando se emplea HLE y RTM. Aunque una implementación de RTM y HLE proporcionará típicamente suficientes recursos para ejecutar regiones transaccionales comunes, las restricciones de implementación y tamaños excesivos para regiones transaccionales pueden provocar que se aborte una ejecución transaccional y pase a una ejecución no transaccional. La arquitectura no garantiza la cantidad de recursos disponibles para hacer ejecución transaccional y no garantiza que una ejecución transaccional tendrá éxito siempre.

Las solicitudes en conflicto a una línea de caché accedida en una región transaccional pueden evitar que la transacción se ejecute de manera satisfactoria. Por ejemplo, si el procesador lógico P0 lee la línea A en una región transaccional y otro procesador lógico P1 escribe la línea A (ya sea dentro o fuera de una región transaccional) a continuación el procesador lógico P0 puede abortar si la escritura del procesador lógico P1 interfiere con la capacidad del procesador P0 para ejecutar de manera transaccional.

De manera similar, si P0 escribe la línea A en una región transaccional y P1 lee o escribe la línea A (ya sea dentro o fuera de una región transaccional), a continuación P0 puede abortar si el acceso de P1 a la línea A interfiere con la capacidad de P0 de ejecutar de manera transicional. Además, otro tráfico de coherencia puede aparecer en ocasiones

como solicitudes en conflicto y puede provocar abortos. Aunque pueden ocurrir estos conflictos falsos, se espera que sean poco comunes. La política de resolución de conflictos para determinar si P0 o P1 abortan en los escenarios anteriores es específica de la implementación.

Realizaciones de ejecución de transacción genéricas:

De acuerdo con las "ARQUITECTURAS PARA MEMORIA TRANSACCIONAL", una disertación enviada al Departamento de Ciencia Informática y al Comité sobre Estudios de Graduado de la Universidad de Stanford en cumplimiento parcial de los requisitos para el grado de Doctor de Filosofía, por Austen McDonald, junio de 2009, fundamentalmente, hay tres mecanismos necesarios para implementar una región atómica y aislada: generación de versiones, detección de conflicto y gestión de contención.

Para hacer que una región de código transaccional parezca atómica, todas las modificaciones realizadas por esa región de código transaccional deben almacenarse y mantenerse aisladas de otras transacciones hasta el tiempo de confirmación. El sistema hace esto implementando una política de generación de versiones. Existen dos paradigmas de generación de versiones: voraz y perezoso. Un sistema de generación de versiones voraz almacena valores transaccionales nuevamente generados en su lugar y almacena valores de memoria anteriores en el lateral, en lo que se denomina un registro de deshacer. Un sistema de generación de versiones perezoso almacena nuevos valores de manera temporal en lo que se denomina una memoria intermedia de escritura, copiándolos a memoria únicamente en la confirmación. En cualquier sistema, se usa la caché para optimizar el almacenamiento de nuevas versiones.

Para asegurar que las transacciones parezca que se realizan de manera atómica, deben detectarse y resolverse conflictos. Los dos sistemas, es decir, los sistemas de generación de versiones voraz y perezoso, detectan conflictos implementando una política de detección de conflicto, ya sea optimista o pesimista. Un sistema optimista ejecuta transacciones en paralelo, comprobando conflictos únicamente cuando se confirma una transacción. Un sistema pesimista comprueba conflictos en cada carga y almacén. De manera similar a la generación de versiones, la detección de conflicto también usa la caché, marcando cada línea como cualquier parte del ajuste de lectura, parte del ajuste de escritura o ambos. Los dos sistemas resuelven conflictos implementando una política de gestión de contención. Existen muchas políticas de gestión de contención, algunas son más apropiadas para detección de conflicto optimista y algunas son más apropiadas para el pesimista. Se describen a continuación algunas políticas de ejemplo.

Puesto que cada sistema de memoria transaccional (TM) necesita tanto detección de generación de versión como detección de conflicto, estas opciones dan lugar a cuatro diseños de TM distintos: voraz-pesimista (EP), voraz-optimista (EO), perezoso-pesimista (LP), y perezoso-optimista (LO). La tabla 2 describe brevemente todos los cuatro diseños de TM distintos.

Las Figuras 1 y 2 representan un ejemplo de un entorno de TM de múltiples núcleos. La Figura 1 muestra muchas CPU habilitadas para TM (CPU1 114a, CPU2 114b, etc.) en un molde 100, conectado con una interconexión 122, bajo la gestión de un control 120a, 120b de interconexión. Cada CPU 114a, 114b (también conocida como un procesador) puede tener una caché de división que consiste en una caché 116a, 166b de instrucciones para almacenar en caché instrucciones de la memoria a ejecutarse y una caché 118a de datos, 118b con soporte de TM para almacenar en caché datos (operandos) de localizaciones de memoria a operarse en la CPU 114a, 114b (en la Figura 1, cada CPU 114a, 114b y sus cachés asociadas se denominan como 112a, 112b). En una implementación, las cachés de múltiples moldes 100 están interconectadas para soportar coherencia de caché entre las cachés de los múltiples moldes 100. En una implementación, se emplea una única caché, en lugar de la caché de división, que maneja tanto instrucciones como datos. En implementaciones, las cachés de CPU están en un nivel de almacenamiento de caché en una estructura de caché jerárquica. Por ejemplo, cada molde 100 puede emplear una caché 124 compartida para compartirse entre todas las CPU en el molde 100. En otra implementación, cada molde puede tener acceso a una caché 124 compartida, compartida entre todos los procesadores de todos los moldes 100.

La Figura 2 muestra los detalles de una CPU 114 transaccional de ejemplo, que incluye adiciones para soportar TM. La CPU (procesador) 114 transaccional puede incluir hardware para soportar puntos de comprobación 126 de registro y registros 128 de TM especiales. La caché de CPU transaccional puede tener los bits 130 de MESI, etiquetas 140 y datos 142 de una caché convencional pero también, por ejemplo, R bits 132 que muestran una línea que se ha leído por la CPU 114 mientras se ejecuta una transacción y W bits 138 que muestran una línea que se ha escrito por la CPU 114 mientras se ejecuta una transacción.

Un detalle clave para los programadores en cualquier sistema de TM es cómo los accesos no transaccionales interactúan con las transacciones. Por diseño, los accesos transaccionales se exploran entre sí usando los mecanismos anteriores. Sin embargo, la interacción entre una carga no transaccional regular con una transacción que contiene un nuevo valor para esa dirección debe considerare aún. Además, la interacción entre un almacén no transaccional con una transacción que ha leído esa dirección debe explorarse también. Estos son problemas del aislamiento del concepto de base de datos.

Un sistema de TM se dice que implementa aislamiento fuerte, en ocasiones denominado atomicidad fuerte, cuando cada carga y almacén no transaccional actúa como una transacción atómica. Por lo tanto, las cargas no

transaccionales no pueden ver datos no confirmados y los almacenes no transaccionales provocan violaciones de atomicidad en cualesquiera transacciones que hayan leído esa dirección. Un sistema donde este no es el caso se dice que implementa aislamiento débil, en ocasiones denominado atomicidad débil.

El aislamiento fuerte a menudo es más deseable que el aislamiento débil debido a la facilidad relativa de conceptualización de aislamiento fuerte. Adicionalmente, si un programador ha olvidado rodear algunas referencias de memoria compartida con transiciones, que provoca errores de programación, entonces con aislamiento fuerte, el programador a menudo detectará esa inadvertencia usando una interfaz de depuración sencilla puesto que el programador observará una región no transaccional que provoca violaciones de atomicidad. También, los programas escritos en un modelo pueden funcionar de manera diferente en otro modelo.

Además, el aislamiento fuerte a menudo es más fácil de soportar en TM de hardware que el aislamiento débil. Con aislamiento fuerte, puesto que el protocolo de coherencia ya gestiona comunicación de carga y almacén entre procesadores, las transacciones pueden detectar cargas y almacenes no transaccionales y actuar de manera apropiada. Para implementar aislamiento fuerte en memoria transaccional de software (TM), debe modificarse código no transaccional para incluir barreras de lectura y escritura; paralizando potencialmente el rendimiento. Aunque se ha gastado un gran esfuerzo en eliminar muchas barreras innecesarias, tales técnicas son a menudo complejas y el rendimiento es típicamente bastante más bajo de el de HTM.

TABLA 2

Espacio de diseño de memoria transaccional

		GENERACIÓN DE VERSIONES	
		perezoso	voraz
DETECCIÓN DE CONFLICTO	optimista	almacenar actualizaciones en una memoria intermedia de escritura; detectar conflictos en tiempo de confirmación.	no práctico: esperar actualizar memoria hasta el tiempo de confirmación pero detectar conflictos en tiempo de acceso garantiza trabajo desperdiciado y no proporciona ventajas
	pesimista	almacenar actualizaciones en una memoria intermedia de escritura; detectar conflictos en tiempo de acceso.	actualizar memoria, mantener valores antiguos en registro de deshacer; detectar conflictos en tiempo de acceso.

La Tabla 2 ilustra el espacio de diseño fundamental de memoria transaccional (generación de versiones y detección de conflicto).

Voraz-pesimista (EP)

Este primer diseño de TM descrito a continuación es conocido como voraz-pesimista. Un sistema EP almacena su ajuste de escritura "en su lugar" (por lo tanto el nombre "voraz") y, para soportar restauración, almacena los valores antiguos de líneas sobrescritas en un "registro de deshacer". Los procesadores usan los bits de caché W 138 y R 132 para rastrear los ajustes de lectura y escritura y detectan conflictos cuando se reciben solicitudes de carga entrometidas. Tal vez los ejemplos más notables de sistemas EP en la biblioteca conocida son LogTM y UTM.

Comenzar una transacción en un sistema EP es al igual que comenzar una transacción en otros sistemas: tm_begin() toma un punto de comprobación de registro, e inicializa cualesquiera registros de estado. Un sistema EP también requiere inicializar el registro de deshacer, los detalles del cual son dependientes del formato de registro, pero a menudo implican inicializar un puntero de base de registro a una región de memoria de hilo privada pre asignada y limpiar un registro de límites de registro.

Generación de versiones: En EP, debido la manera en que está diseñada para funcionar la generación de versiones voraz, las transiciones de estado de MESI 130 (indicadores de línea de caché que corresponden a estados de código modificado, exclusivo, compartido e inválido) se dejan en su mayoría sin cambiar. Fuera de una transacción, las transiciones de estado de MESI 130 se dejan completamente sin cambiar. Cuando se lee una línea dentro de una transacción, las transiciones de coherencia convencionales se aplican (S (compartido) →S, I (inválido) →S, o I →E (exclusivo)), que emite una carga perdida según sea necesaria, pero el bit R 132 también se establece. Análogamente,

escribir una línea se aplica a las transiciones convencionales (S →M, E →I, I →M), que emiten una perdida según sea necesario, pero también establecen el bit W 138 (escrito). La primera vez que se escribe una línea, la versión antigua de la línea se carga, después se escribe en el registro de deshacer para conservarla en el caso de que se aborte la transacción actual. Los datos nuevamente escritos se almacenan a continuación "en su lugar", sobre los datos antiguos.

Detección de conflicto: la detección de conflicto pesimista usa mensajes de coherencia intercambiados en pérdidas, o mejoras, para buscar conflictos entre transacciones. Cuando tiene lugar una pérdida de lectura en una transacción, otros procesadores reciben una solicitud de carga; pero ignoran la solicitud si no tienen la línea necesaria. Si los otros procesadores tienen la línea necesaria de manera no especulativa o tienen la línea R 132 (lectura), degradan esa línea a S, y en ciertos casos emiten una transferencia de caché a caché si tienen la línea en el estado de MESI 130 M o E. Sin embargo, si la caché tiene la línea W 138, entonces se detecta un conflicto entre las dos transacciones y deben tomarse la acción o acciones adicionales.

De manera similar, cuando una transacción busca mejorar una línea de compartida a modificada (en una primera escritura), la transacción emite una solicitud de carga exclusiva, que también se usa para detectar conflictos. Si una caché de recepción tiene la línea de manera no especulativa, a continuación la línea se invalida, y en ciertos casos se emite una transferencia de caché a caché (estados M o E). Pero, si la línea es R 132 o W 138, se detecta un conflicto.

Validación: puesto que se realiza detección de conflicto en cada carga, una transacción siempre tiene acceso exclusivo a su propio ajuste de escritura. Por lo tanto, la validación no requiere ningún trabajo adicional.

Confirmación: Puesto que la generación de versiones voraz almacena la nueva versión de elementos de datos en su lugar, el proceso de confirmación simplemente limpia los bits W 138 y R 132 y descarga el registro de deshacer.

Abortar: Cuando se restaura una transacción, la versión original de cada línea de caché en el registro de deshacer debe restaurarse, un proceso denominado "desenrollado" o "aplicación" del registro. Esto se hace durante tm_discard() y debe ser atómico con respecto a otras transacciones. Específicamente, el ajuste de escritura debe aún usarse para detectar conflictos: esta transacción tiene la única versión correcta de líneas en su registro de deshacer, y la solicitud de transacciones debe esperar la versión correcta para que se restaure desde ese registro. Un registro de este tipo puede aplicarse usando una máquina de estado de hardware o manejador de aborto de software.

Voraz-pesimista tiene las características de: la confirmación es sencilla y puesto que está en su lugar, muy rápida. De manera similar, la validación es una no-op. La detección de conflicto pesimista detecta conflictos de manera temprana, reduciendo de esta manera el número de transacciones "condenadas". Por ejemplo, si están implicadas dos transacciones en una dependencia escribir-después-leer, entonces esa dependencia se detecta inmediatamente en la detección de conflicto pesimista. Sin embargo, en detección de conflicto optimista tales conflictos no se detectan hasta que se confirme la escritura.

Voraz-pesimista también tiene las características de: como se ha descrito anteriormente, la primera vez que se escribe una línea de caché, el valor antiguo debe escribirse en el registro, incurriendo en accesos de caché adicionales. Los abortos son costosos ya que requieren deshacer el registro. Para cada línea de caché en el registro, debe emitirse una carga, tal vez yendo tan lejos de la memoria principal antes de continuar a la siguiente línea. La detección de conflicto pesimista también evita que existan ciertas planificaciones que pueden serializarse.

Adicionalmente, puesto que los conflictos se manejan a medida que tienen lugar, hay un potencial para mecanismos de gestión de bloqueo activo y contención cuidadosa que deben emplearse para garantizar el progreso hacia adelante.

Perezoso-optimista (LO)

Otro diseño de TM conocido es perezoso-optimista (LO), que almacena su ajuste de escritura en una "memoria intermedia de escritura" o "registro de rehacer" y detecta conflictos en tiempo de confirmación (usando aún los bits R 132 y W 138).

Generación de versiones: Como en el sistema EP, el protocolo MESI del diseño LO se hace aplicar fuera de las transacciones. Una vez dentro de una transacción, leer una línea incurre en las transiciones de MESI convencionales pero también establece el bit R 132. Análogamente, escribir una línea establece el bit W 138 de la línea, pero manejar las transiciones de MESI del diseño de LO es diferente de el del diseño de EP. En primer lugar, con generación de versiones perezosa, las nuevas versiones de datos escritos se almacenan en la caché jerárquicamente hasta confirmación mientras que otras transacciones tienen acceso a versiones antiguas disponibles en memoria o en otras cachés. Para hacer disponibles las versiones antiguas, deben suprimirse líneas sucias (líneas M) cuando se escriben en primer lugar por una transacción. En segundo lugar, no son necesarias pérdidas de mejora debido a la característica de detección de conflicto optimista: si una transacción tiene una línea en el estado S, puede simplemente escribir en ella y mejorar esa línea a un estado M sin comunicar los cambios con otras transacciones puesto que se hace la detección de conflicto en tiempo de confirmación.

Detección y validación de conflicto: para validar una transacción y detectar conflictos, LO comunica las direcciones de líneas modificadas de manera especulativa a otras transacciones únicamente cuando está preparando confirmar. En la validación, el procesador envía un paquete de red grande, potencialmente grande, que contiene todas las direcciones en el ajuste de escritura. Los datos no se envían, sino que se dejan en la caché del confirmador y se marcan sucios (M). Para crear este paquete sin buscar en la caché líneas marcadas W, se usa un vector de bit sencillo, denominado una "memoria intermedia de almacenamiento", con un bit por línea de caché para rastrear estas líneas modificadas de manera especulativa. Otras transacciones usan este paquete de dirección para detectar conflictos: si se halla una dirección en la caché y se establecen los bits R 132 y/o W 138 bits, entonces se inicia un conflicto. Si se halla la línea pero no se establece ni R 132 ni W 138, entonces la línea simplemente se invalida, que es similar a procesar una carga exclusiva.

Para soportar atomicidad de transacción, estos paquetes de dirección deben manejarse atómicamente, es decir, no pueden existir dos paquetes de dirección a la vez con las mismas direcciones. En un sistema LO, esto puede conseguirse obteniendo simplemente un testigo de confirmación global antes de enviar el paquete de dirección. Sin embargo, podría emplearse un esquema de confirmación de dos fases empleado enviando en primer lugar el paquete de dirección, recogiendo respuestas, aplicando un protocolo de ordenación (tal vez la transacción más antigua en primer lugar), y confirmando una vez que todas las respuestas son satisfactorias.

Confirmación: Una vez que ha tenido lugar la validación, la confirmación no necesita tratamiento especial: simplemente limpiar W 138 y R 132 bits y la memoria intermedia de almacenamiento. Las escrituras de la transacción ya están marcadas sucias en la caché y otras copias de cachés de estas líneas se han invalidado mediante el paquete de dirección. Otros procesadores pueden a continuación acceder a los datos confirmados a través del protocolo de coherencia regular.

Abortar: la restauración es igualmente fácil: puesto que el ajuste de escritura está contenido en las cachés locales, estas líneas pueden invalidarse, entonces limpiar los bits W 138 y R 132 y la memoria intermedia de almacenamiento. La memoria intermedia de almacenamiento permite que se hallen W líneas para invalidar sin la necesidad de buscar la caché.

Perezoso-optimista tiene las características de: los abortos son muy rápidos, no requiriendo cargas o almacenes adicionales y haciendo únicamente cambios locales. Pueden existir planificaciones más serializables que las halladas en EP, que permite que un sistema LO especule de manera más agresiva esas transacciones que son independientes, que puede producir rendimiento superior. Finalmente, la detección tardía de conflictos puede aumentar la probabilidad de progreso hacia delante.

Perezoso-optimista también tiene las características de: la validación toma el tiempo de comunicación global proporcional al tamaño de ajuste de escritura. Las transacciones condenadas pueden desperdiciar el trabajo puesto que se detectan conflictos únicamente en tiempo de confirmación.

Perezoso-pesimista (LP)

Perezoso-pesimista (LP) representa una tercera opción de diseño de TM, que se sitúa en algún lugar entre EP y LO: almacenando líneas nuevamente escritas en una memoria intermedia de escritura pero detectando conflictos en una base por acceso.

Generación de versiones: la generación de versiones es similar pero no idéntica a la de LO: leer una línea establece su R bit 132, escribir una línea establece su W bit 138, y se usa una memoria intermedia de almacenamiento para rastrear líneas W en la caché. También, deben suprimirse líneas sucias (M) cuando se escriben en primer lugar por una transacción, justo como en LO. Sin embargo, puesto que la detección de conflicto es pesimista, deben realizarse cargas exclusivas cuando se mejora a una línea transaccional de I, S→M, que es a diferencia de LO.

Detección de conflicto: la detección de conflicto de LP opera igual que en EP: usando mensajes de coherencia para buscar conflictos entre transacciones.

Validación: como en EP, la detección de conflicto pesimista asegura que en cualquier punto, una transacción en ejecución no tiene conflictos con ninguna otra transacción en ejecución, por lo que la validación es una no-op.

Confirmación: la confirmación no necesita tratamiento especial: simplemente limpiar los bits W 138 y R 132 y la memoria intermedia de almacenamiento, como en LO.

Abortar: la restauración también es como la de LO: simplemente invalidar el ajuste de escritura usando la memoria intermedia de almacenamiento y limpiar los bits W y R y la memoria intermedia de almacenamiento.

Voraz-optimista (EO)

El LP tiene las características de: como LO, los abortos son muy rápidos. Como EP, el uso de detección de conflicto

pesimista reduce el número de transacciones "condenadas". Como EP, algunas planificaciones serializables no están permitidas y debe realizarse detección de conflicto en cada pérdida de caché.

La combinación final de generación de versiones y detección de conflicto es voraz-optimista (EO). EO puede ser una elección menos óptima para sistemas HTM: puesto que las nuevas versiones transaccionales se escriben en su lugar, otras transacciones no tiene elección pero notifican conflictos a medida que tienen lugar (es decir, a medida que tienen lugar pérdidas de caché). Pero puesto que EO espera hasta el tiempo de confirmación para detectar conflictos, aquellas transacciones se vuelven "zombis", continuando ejecutándose, desperdiciando recursos, aún "condenadas" a abortar.

EO ha probado ser útil en STM y se implementa por Bartok-STM y McRT. Una STM de generación de versiones perezosa necesita comprobar su memoria intermedia de escritura en cada lectura para asegurar que está leyendo el valor más reciente. Puesto que la memoria intermedia de escritura no es una estructura de hardware, esta es costosa, de esta manera la preferencia para generación de versiones voraz de escritura en su lugar. Adicionalmente, puesto que la comprobación de conflictos es también costosa en una STM, la detección de conflicto optimista ofrece la ventaja de realizar esta operación sin procesar.

Gestión de contención

Cómo una transacción se restaura una vez que el sistema ha decidido abortar esa transacción se ha descrito anteriormente, pero, puesto que un conflicto implica dos transacciones, necesitan explorarse los asuntos de qué transacción debería abortarse, cómo debería iniciarse ese aborto, y cuándo debería reintentarse la transacción abortada. Estos son asuntos que se tratan por la gestión de contenido (CM), un componente clave de memoria transaccional. Se describen a continuación políticas, con respecto a cómo los sistemas inician abortos y los diversos métodos establecidos de gestión que las transacciones deberían abortar en un conflicto.

Políticas de gestión de contenido

Una política de gestión de contenido (CM) es un mecanismo que determina qué transacción implicada en un conflicto debería abortar y cuándo debería reintentarse la transacción abortada. Por ejemplo, a menudo se encuentra el caso de que reintentar una transacción abortada inmediatamente no conduce al mejor rendimiento. A la inversa, emplear un mecanismo de repliegue, que retarda el reintento de una transacción abortada, puede producir mejor rendimiento. STM en primer lugar lidió con el hallazgo de las mejores políticas de gestión de contención y muchas de las políticas señaladas a continuación se desarrollaron originalmente para STM.

Las políticas CM se basan en un número de medidas para realizar decisiones, incluyendo antigüedades de las transacciones, tamaño de ajustes de lectura y escritura, el número de abortos anteriores, etc. Las combinaciones de medidas para hacer tales decisiones son sin fin, pero se describen a continuación ciertas combinaciones, aproximadamente en orden de complejidad creciente.

Para establecer alguna nomenclatura, obsérvese en primer lugar que en un conflicto hay dos lados: el atacante y el defensor. El atacante es la transacción que solicita acceso a una localización de memoria compartida. En detección de conflicto pesimista, el atacante es la transacción que emite la carga o carga exclusiva. En optimista, el atacante es la transacción que intenta validar. El defensor en ambos casos es la transacción que recibe la solicitud del atacante.

Una política de CM agresiva reintenta inmediatamente y siempre cualquiera del atacante o el defensor. En LO, agresivo significa que el atacante siempre gana, y de esta manera agresivo en ocasiones a menudo se denomina gana el que confirma. Una política de este tipo se usó para los sistemas LO más anteriores. En el caso de EP, agresivo puede ser cualquiera del defensor gana o el atacante gana.

Reiniciar una transacción en conflicto que experimentará inmediatamente otro conflicto está destinada a desperdiciar el trabajo, es decir, las pérdidas de caché de relleno de ancho de banda de interconexión. Una política de CM política emplea repliegue exponencial (aunque también podría usarse lineal) antes de reiniciar conflictos. Para evitar la falta, una situación donde un proceso no tiene recursos asignados a él por el planificador, el repliegue exponencial aumenta enormemente los impares de éxito de transacción después de algunos n reintentos.

Otro enfoque para resolución de conflictos es abortar aleatoriamente el atacante o defensor (una política denominada aleatorizada). Una política de este tipo puede combinarse con un esquema de repliegue aleatorizado para evitar contención innecesaria.

Sin embargo, hacer elecciones aleatorias, cuando se selecciona una transacción para abortar, puede dar como resultado abortar transacciones que tienen "una gran cantidad de trabajo" completada, que puede desperdiciar recursos. Para evitar tal desperdicio, la cantidad de trabajo completado en la transacción puede tenerse en cuenta cuando se determina qué transacción abortar. Una medida de trabajo podría ser una antigüedad de la transacción. Otros métodos incluyen el más antiguo, TM en bruto, el tamaño importa, Karma, y Polka. El más antiguo es un método de indicación de tiempo sencillo que aborta la transacción más reciente en un conflicto. TM en bruto usa este esquema.

El tamaño importa es como el más antiguo pero en lugar de antigüedad de transacción, se usa el número de palabras escritas/leídas como la prioridad, revirtiendo al más antiguo después de un número fijo de abortos. Karma es similar, usando el tamaño del ajuste de escritura como prioridad. Restauración a continuación continúa después de replegar una cantidad de tiempo fija. Las transacciones abortadas mantienen sus prioridades después de abortarse (de esta manera el nombre Karma). Polka funciona como Karma pero en lugar de replegar una cantidad predefinida de tiempo, repliega exponencialmente más cada vez.

Puesto que el aborto de desperdicios funciona, es lógico argumentar que detener un atacante hasta que el defensor haya finalizado su transacción conduciría a mejor rendimiento. Desafortunadamente, un esquema sencillo de este tipo conduce fácilmente a bloqueo inactivo.

Las técnicas de evitación de bloqueo inactivo pueden usarse para resolver este problema. El codicioso usa dos reglas para evitar bloqueo inactivo. La primera regla es, si una primera transacción, T1, tiene prioridad inferior que una segunda transacción, T0, o si T1 está esperando otra transacción, entonces T1 se aborta cuando entra en conflicto con T0. La segunda regla es, si T1 tiene prioridad inferior que T0 y no está esperando, entonces T0 espera hasta que se confirme T1, aborta, o empieza a esperar (caso en el que se aplica la primera regla). Codicioso proporciona algunas garantías acerca de los límites de tiempo para ejecutar un conjunto de transacciones. Un diseño EP (LogTM) usa una política de CM similar a codicioso para conseguir detención con evitación de bloqueo inactivo conservador.

Las reglas de coherencia de MESI de ejemplo proporcionan cuatro posibles estados en los que puede residir una línea de caché de un sistema de caché de múltiples procesadores, M, E, S, y I, definida como sigue:

Modificado (M): la línea de caché está presente únicamente en la caché actual, y está sucia; se ha modificado a partir del valor en memoria principal. La caché se requiere que escriba los datos de vuelta a memoria principal en algún tiempo en el futuro, antes de permitir otra lectura del estado de memoria principal (ya no es más válida). La escritura de vuelta cambia la línea al estado exclusivo.

Exclusivo (E): la línea de caché está presente únicamente en la caché actual, pero está limpia; coincide con la memoria principal. Puede cambiarse al estado compartido en cualquier momento, en respuesta a una solicitud de lectura. Como alternativa, puede cambiarse al estado modificado cuando se está escribiendo en ella.

Compartido (S): indica que esta línea de caché puede almacenarse en otras cachés de la máquina y está "limpia"; coincide con la memoria principal. La línea puede descartarse (cambiarse al estado inválido) en cualquier momento.

Inválido (I): indica que esta línea de caché es inválida (sin uso).

Pueden proporcionarse indicadores de estado de coherencia TM (R 132, W 138) para cada línea de caché, además de, o codificados en los bits de coherencia de MESI. Un indicador R 132 indica que la transacción actual ha leído de los datos de la línea de caché, y un indicador W 138 indica que la transacción actual se ha escrito en los datos de la línea de caché.

En otro aspecto del diseño TM, un sistema está diseñado usando memorias intermedias de almacén transaccional. La patente de Estados Unidos N. º 6.349.361 titulada "Methods and Apparatus for Reordering and Renaming Memory References in a Multiprocessor Computer System" presentada el 31 de marzo de 2000, enseña un método para reordenar y renombrar referencias de memoria en un sistema informático de múltiples procesadores que tiene al menos un primer y un segundo procesador. El primer procesador tiene una primera caché privada y una primera memoria intermedia, y el segundo procesador tiene una segunda caché privada y una segunda memoria intermedia. El método incluye las operaciones de, para cada una de una pluralidad de solicitudes de almacén cerradas recibidas por el primer procesador para almacenar un dato, obtener exclusivamente una línea de caché que contiene el dato por la primera caché privada, y almacenar el dato en la primera memoria intermedia. Tras recibir la primera memoria intermedia una solicitud de carga del primer procesador para cargar un dato particular, el dato particular se proporciona al primer procesador de entre los datos almacenados en la primera memoria intermedia basándose en una secuencia en orden de operaciones de carga y almacén. Tras recibir la primera caché una solicitud de carga de la segunda caché para un dato dado, se indica una condición de error y un estado actual de al menos uno de los procesadores se resetea a un estado anterior cuando la solicitud de carga para el dato dado corresponde a los datos almacenados en la primera memoria intermedia.

Los componentes de implementación principal de una instalación de memoria transacción de este tipo son un fichero de registro de respaldo de transacción para mantener contenido de GR (registro general) de pre-transacción, un directorio de caché para rastrear las líneas de caché accedidas durante la transacción, una caché de almacén para almacenar en memoria intermedia almacenes hasta que la transacción finalice, y rutinas de firmware para realizar diversas funciones complejas. En esta sección se describe una implementación detallada.

Realización de servidor empresarial IBM zEnterprise EC12

El servidor empresarial IBM zEnterprise EC12 introduce ejecución transaccional (TX) en memoria transaccional, y se

describe en parte en un artículo, "Transactional Memory Architecture and Implementation for IBM System z" of Proceedings páginas 25-36 presentado en MICRO-45, 1-5 de diciembre de 2012, Vancouver, British Columbia, Canadá, disponible de IEEE Computer Society Conference Publishing Services (CPS).

La Tabla 3 muestra una transacción de ejemplo. Las transacciones iniciadas con TBEGIN no se garantiza que se complete satisfactoriamente siempre con TEND, puesto que pueden experimentar una condición de aborto en cada ejecución intentada, por ejemplo, debido a repetir conflictos con otras CPU. Esto requiere que el programa soporte una ruta de repliegue para realizar la misma operación de manera no transaccional, por ejemplo, usando esquemas de bloqueo tradicionales. Esto impone una carga significativa en los equipos de programación y verificación de software, especialmente donde la ruta de repliegue no se genera automáticamente por un compilador fiable.

TABLA 3

Código de transacción de ejemplo			
	LHI	R0,0	*inicializar contador de reintento=0
bucle	TBEGIN		*comenzar transacción
	JNZ	abortar	*ir a código de aborto si CC1=0
	LT	R1, bloquear	*cargar y probar el bloqueo de repliegue
	JNZ	lckbzy	*bifurcar si bloqueo ocupado
	... realizar operación ...		
	TEND … … … …		*finalizar transacción
lckbzy	TABORT		*abortar si bloqueo ocupado; esto *se reanuda después de TBEGIN
abortar	JO	replegar	*no reintentar si CC=3
	AHI	R0, 1	*incrementar contador de reintento
	CIJNL	R0,6, replegar	*renunciar después de 6 intentos
	PPA	R0, TX	*retardo aleatorio basándose en contador de reintento
	... potencialmente	esperar a que bloqueo se quede libre ...	
	J	bucle	*saltar de vuelta a repliegue de reintento
	OBTAIN	bloquear	*usar comparar e intercambiar
	... realizar operación ...		
	RELEASE … …	bloquear … …	

El requisito de proporcionar una ruta de repliegue para las transacciones de Ejecución de Transacción (TX) abortadas puede ser oneroso. Muchas transacciones que operan en estructuras de datos compartidas se espera que sean cortas, tocando únicamente unas pocas localizaciones de memoria distintas, y usen instrucciones sencillas únicamente. Para estas transacciones, el IBM zEnterprise EC12 introduce el concepto de transacciones restringidas; bajo condiciones normales, la CPU 114 asegura que las transacciones restringidas eventualmente finalizan satisfactoriamente, aunque sin proporciona un límite estricto en el número de reintentos necesarios. Una transacción restringida se inicia con una instrucción TBEGINC y finalizan con un TEND regular. Implementar una tarea como una transacción restringida o no restringida típicamente da como resultado rendimiento muy comparable, pero las transacciones restringidas simplifican el desarrollo de software eliminado la necesidad de una ruta de repliegue. La arquitectura de Ejecución Transaccional de IBM se describe adicionalmente en z/Architecture, Principles of Operation, décima edición, SA22-7832-09 publicada en septiembre de 2012 de IBM.

Una transacción restringida se inicia con la instrucción TBEGINC. Una transacción iniciada con TBEGINC debe seguir una lista de restricciones de programación; de lo contrario el programa toma una interrupción de violación de restricción que no puede filtrarse. Las restricciones ejemplares pueden incluir, pero sin limitación: la transacción puede ejecutar un máximo de 32 instrucciones, todo el texto de la instrucción debe estar dentro de 256 bytes consecutivos de memoria; la transacción contiene únicamente ramales relativos que apuntan hacia delante (es decir, no bucles o llamadas de subrutina); la transacción puede acceder a un máximo de 4 octopalabras alineadas (una octopalabra son 32 bytes) de memoria; y la restricción del conjunto de instrucciones para excluir instrucciones complejas como operaciones decimales o de punto flotante. Las restricciones se eligen de manera que pueden realizarse muchas operaciones comunes como operaciones de insertar/borrar lista doblemente enlazada, incluyendo el concepto muy potente de comparar e intercambiar atómico que tiene como objeto hasta 4 octopalabras alineadas. Al mismo tiempo,

las restricciones se eligieron de manera conservadora de manera que futuras implementaciones de CPU puedan asegurar el éxito de la transacción sin necesitar ajustar las restricciones, puesto que de lo contrario conducirían a incompatibilidad de software.

TBEGINC se comporta en su mayoría como XBEGIN en TSX o TBEGIN en los servidores zEC12 de IBM, excepto que los campos de control de registro de punto flotante (FPR) y de la filtración de interrupción de programa no existen y los controles se considera que son cero. En un aborto de transacción, la dirección de instrucción se establece atrás directamente al TBEGINC en lugar de a la instrucción después, que refleja el reintento inmediato y la ausencia de ruta de aborto para transacciones restringidas.

Las transacciones anidadas no están permitidas con transacciones restringidas, pero si tiene lugar un TBEGINC dentro de una transacción no restringida se trata como la apertura de un nivel de anidación no restringido como lo haría TBEGIN. Esto puede ocurrir, por ejemplo, si una transacción no restringida llama a una subrutina que usa una transacción restringida internamente.

Puesto que la filtración de interrupción está implícitamente desconectada, todas las excepciones durante una transacción restringida conducen a una interrupción en el sistema operativo (SO). La finalización satisfactoria eventual de la transacción se basa la capacidad del SO para paginar de entrada como máximo 4 páginas tocadas por cualquier transacción restringida. El SO también debe asegurar intervalos de tiempo suficientemente largos para permitir que se complete la transacción.

TABLA 4

Ejemplo de código de transacción	
TBEGINC	*comenzar transacción restringida
... realizar operación ...	
TEND	*finalizar transacción

La Tabla 4 muestra la implementación transaccional restringida del código en la Tabla 3, suponiendo que las transacciones restringidas no interactúan con otro código basado en bloqueo. No se muestra prueba de bloqueo por lo tanto, pero podría añadirse si se mezclaran transacciones restringidas y código basado en bloqueo.

Cuando tiene lugar un fallo de manera repetida, se realiza emulación de software usando milicode como parte del firmware de sistema. Ventajosamente, las transacciones restringidas tienen propiedades deseables debido a la carga eliminada de los programadores.

Con referencia a la Figura 3, el procesador IBM zEnterprise EC12 introdujo la instalación de ejecución transaccional. El procesador puede decodificar 3 instrucciones por ciclo de reloj; las instrucciones sencillas se despachan como micro-op sencillas, y las instrucciones más complejas se rompen en múltiples micro-op. Las micro-op (Uops 232b) se escriben en una cola 216 de emisión unificada, a partir de las que pueden emitirse fuera de orden. Pueden ejecutarse cada ciclo hasta dos instrucciones de punto fijo, una de punto flotante y dos de carga/almacén, y dos de ramal. Una Tabla de Finalización Global (GCT) 232 mantiene cada micro-op y una profundidad de anidación de transacción (TND) 232a. La GCT 232 se escribe en orden en tiempo de decodificación, rastrea el estado de ejecución de cada micro-op 232b, y completa instrucciones cuando se han ejecutado satisfactoriamente todas las micro-op 232b del grupo de instrucciones más antiguo.

La caché 240 de datos de nivel 1 (LI) es una caché asociativa de 6 sentidos de 96 KB (kilo-bytes) con líneas de caché de 256 bytes y latencia de uso de 4 ciclos, acoplada a una caché 268 de datos de 2º nivel (L2) asociativa de 8 sentidos de 1 MB (megabyte) privada con penalización de latencia de uso de 7 ciclos durante 240 perdidas de L1. La caché L1 240 es la caché más cercana a un procesador y la caché Ln es una caché en el enésimo nivel de almacenamiento en caché. Ambas cachés L1 240 y L2 268 son de almacén-a través. Seis núcleos en cada chip de procesador central (CP) comparten una caché de almacenamiento de entrada de 3er nivel de 48 MB, y seis chips CP están conectados a una caché de 4º nivel de 384 MB fuera de chip, empaquetados juntos en un módulo de múltiples chips de cerámica de vidrio (MCM). Pueden conectarse hasta 4 módulos de múltiples chips (MCM) a un sistema de múltiples procesadores simétrico (SMP) coherente con hasta 144 núcleos (no todos los núcleos están disponibles para ejecutar carga de trabajo de cliente).

La coherencia se gestiona con una variante del protocolo MESI. Las líneas de caché pueden tener propiedad de solo lectura (compartida) o ser exclusivas; las L1 240 y L2 268 son de almacén-a través y por lo tanto no contienen líneas sucias. Las cachés L3 272 y L4 (no mostradas) son de estados de almacenamiento de entrada y de rastreo de suciedad. Cada caché es inclusiva de todas sus cachés de nivel inferior conectadas.

Las solicitudes de coherencia se denominan "interrogaciones cruzadas" (XI) y se envían jerárquicamente desde cachés de nivel inferior y entre las L4. Cuando un núcleo pierde las L1 240 y L2 268 y solicita la línea de caché desde su L3 272 local, la L3 272 comprueba si posee la línea, y si fuera necesario envía una XI a la L2 268/l1 240 que

actualmente está en posesión bajo la de L3 272 para asegurar coherencia, antes devuelve la línea de caché al solicitante. Si la solicitud también pierde la L3 272, la L3 272 envía una solicitud a la L4 (no mostrada), que aplica coherencia enviando XI a todas las L3 necesarias bajo esa L4, y a las L4 vecinas. A continuación la L4 responde a la L3 solicitante que reenvía la respuesta a la L2 268/I1 240.

Obsérvese que debido al papel de inclusividad de la jerarquía de caché, en ocasiones las líneas están XI desde cachés de nivel inferior debido a supresiones en cachés de nivel superior provocadas por desbordamientos de asociación de solicitudes a otras líneas de cache. Estas XI pueden denominarse "XI de LRU", donde LRU significa menos recientemente usada.

Haciendo referencia a otro tipo de solicitudes de XI más, la propiedad de caché de transición de XI de degradación del estado exclusivo a solo lectura y la propiedad de caché de paso de exclusivo-XI en el estado inválido. Las XI de degradación y las XI exclusivas necesitan una respuesta de vuelta al emisor de XI. La caché objetivo puede "aceptar" la XI, o enviar una respuesta de "rechazo" si en primer lugar necesita suprimir datos sucios antes de aceptar la XI. Las cachés L1 240/l2 268 son de almacén a través, pero pueden rechazar las XI de degradación y XI exclusivas si tienen almacenes en sus colas de almacén que necesitan enviarse a L3 antes de degradar el estado exclusivo. Una XI rechazada se repetirá por el emisor. Las XI de solo lectura se envían a cachés que poseen la línea de solo lectura; no es necesaria respuesta para tales XI puesto que no pueden rechazarse. Los detalles del protocolo SMP son similares a aquellos descritos para IBM z10 por P. Mak, C. Walters, y G. Strait, en "IBM System z10 processor cache subsystem microarchitecture", IBM Journal of Research y Development, Vol 53:1, 2009.

Ejecución de instrucción transaccional

La Figura 3 representa componentes de un entorno 112 de CPU de ejemplo, que incluyen una CPU 114 y cachés/componentes con los que interactúa (tales como aquellos representados en las Figuras 1 y 2). La unidad 208 de decodificación de instrucción (IDU) mantiene el registro de la profundidad 212 de anidación de transacción (TND) actual. Cuando la IDU 208 recibe una instrucción TBEGIN, la profundidad 212 de anidación se incrementa, y a la inversa se decrementa en instrucciones TEND. La profundidad 212 de anidación se escribe en la GCT 232 para cada instrucción despachada. Cuando se decodifica una TBEGIN o TEND en una ruta especulativa que más tarde se vaciará, la profundidad 212 de anidación de la IDU 208 se referencia desde la entrada de GCT 232 más reciente que no se ha vaciado. El estado transaccional también se escribe en la cola 216 de emisión para su consumo por las unidades de ejecución, en su mayoría por la unidad de carga/almacén (LSU) 280, que también tiene un calculador 236 de dirección efectiva que está incluido en la LSU 280. La instrucción TBEGIN puede especificar un bloque de diagnóstico de transacción (TDB) para registrar información de estado, si abortara la transacción antes de alcanzar una instrucción TEND.

Similar a la profundidad de anidación, la IDU 208/GCT 232 rastrea de manera colaborativa las máscaras de modificación de registro de acceso / registro de punto flotante (AR/FPR) a través de la anidación de transacción; la IDU 208 puede colocar una solicitud de aborto en la GCT 232 cuando se decodifica una instrucción de AR/FPR-modificación y la máscara de modificación la bloquea. Cuando la instrucción se vuelve próxima a finalizarse, se bloquea la finalización y la transacción se aborta. Otras instrucciones restringidas se manejan de manera similar, incluyendo TBEGIN si se decodifica mientras está en una transacción restringida, o supera la profundidad de anidación máxima.

Una TBEGIN más exterior se rompe en múltiples micro-op dependiendo de la máscara de grabación de GR; cada micro-op 232b (que incluye, por ejemplo uop 0, uop 1, y uop2) se ejecutará por una de dos unidades de punto fijo (FXU) 220 para grabar un par de GR 228 en un fichero 224 de registro de reserva de transacción especial, que se usa para restaurar más tarde el contenido de GR 228 en caso de un aborto de transacción. También la TBEGIN abarca las micro-op 232b para realizar una prueba de accesibilidad para la TDB si se especifica una; la dirección se graba en un registro de fin especial para uso posterior en caso de aborto. En la decodificación de una TBEGIN más exterior, la dirección de instrucción y el texto de instrucción de la TBEGIN también se graban en registros de fin especial para un procesamiento de aborto potencial más tarde.

TEND y NTSTG son instrucciones de micro-op 232b únicas; NTSTG (almacén no transaccional) se maneja como un almacén normal excepto que se marca como no transaccional en la cola 216 de emisión de modo que la LSU 280 pueda tratarlo de manera apropiada. TEND es una no-op en tiempo de ejecución, la finalización de la transacción se realiza cuando se completa TEND.

Como se ha mencionado, las instrucciones que están dentro de una transacción se marcan como tal en la cola 216 de emisión, pero de lo contrario se ejecutan en su mayoría sin cambiar; la LSU 280 realiza rastreo de aislamiento como se describe en la siguiente sección.

Puesto que la decodificación está en orden, y puesto que la IDU 208 mantiene el registro del estado transaccional y lo escribe en la cola 216 de emisión junto con cada instrucción de la transacción, la ejecución de TBEGIN, TEND, e instrucciones antes, durante y después de la transacción pueden realizarse fuera de orden. Incluso si es posible (aunque poco probable) que se ejecute en primer lugar TEND, entonces se ejecuta la transacción completa y

finalmente la TBEGIN. El orden de programa se restaura a través de la GCT 232 en tiempo de finalización. La longitud de las transacciones no está limitada por el tamaño de la GCT 232, puesto que pueden restaurarse registros de fin general (GR) 228 del fichero 224 de registro de reserva.

Durante la ejecución, los eventos de grabación de evento de programa (PER) se filtran basándose en el Control de Supresión de Evento, y se detecta un evento PER TEND si está activado. De manera similar, mientras se está en el modo transaccional, un generador pseudo-aleatorio puede estar causando los abortos aleatorios como se posibilita por el Control de Diagnóstico de Transacción.

Rastreo de aislamiento transaccional

La unidad 280 de carga/almacén rastrea líneas de caché que se accedieron durante la ejecución transaccional, y desencadena un aborto si una XI de otra CPU (o una LRU-XI) entra en conflicto con el espacio ocupado. Si la XI en conflicto es una XI exclusiva o degradada, la LSU 280 rechaza la XI de vuelta a la L3 272 en la esperanza de finalizar la transacción antes de que la L3 272 repita la XI. Esta "rigidez" es muy eficaz en transacciones altamente contenidas. Para evitar cuelgues cuando dos CPU se hacen rígidas entre sí, se implementa un contador de rechazo de XI, que desencadena un aborto de transacción cuando se cumple un umbral.

El directorio 240 de caché L1 se implementa tradicionalmente con memorias de acceso aleatorio estáticas (SRAM). Para la implementación de memoria transaccional, los bits válidos 244 (64 filas x 6 sentidos) de los directorios se han movido en pestañas lógicas normales, y se complementan con dos bits más por línea de caché: los bits de lectura de TX 248 y sucios de TX 252.

Los bits de lectura de TX 248 se resetean cuando se decodifica una nueva TBEGIN más exterior (que está interbloqueada contra una transacción aún pendiente anterior). Los bits de lectura de TX 248 se establecen en tiempo de ejecución por cada instrucción de carga que se marca "transaccional" en la cola de emisión. Obsérvese que esto puede conducir a sobre-marcaje si se ejecutan las cargas especulativas, por ejemplo, en una ruta de ramal prevista de manera incorrecta. La alternativa del ajuste del bit de lectura de TX 248 en tiempo de finalización de carga era demasiado costosa para el área del silicio, puesto que múltiples cargas pueden completarse al mismo tiempo, requiriendo muchos puertos de lectura en la cola de carga.

Los almacenes se ejecutan de la misma manera que en modo no transaccional, pero se coloca una marca de transacción en la entrada de cola de almacén (STQ) 260 de la instrucción de almacén. En tiempo de escritura de vuelta, cuando se escriben los datos de la STQ 260 en la L1 240, el bit sucio de TX 252 en el directorio Li 256 se establece para la línea de caché escrita. La escritura de vuelta del almacén en la L1 240 tiene lugar únicamente después de que se ha completado la instrucción de almacén, y como máximo se escribe de vuelta un almacén por ciclo. Antes de la finalización y escritura de vuelta, las cargas pueden acceder a los datos de la STQ 260 por medio de almacén-reenvío; después de escritura de vuelta, la CPU 114 (Figura 2) puede acceder a los datos actualizados de manera especulativa en la L1 240. Si la transacción finaliza satisfactoriamente, los bits sucios de TX 252 de todas las líneas de caché se limpian, y también las marcas de Tx de almacenes aún no escritos se limpian en la STQ 260, volviendo de manera eficaz los almacenes pendientes en almacenes normales.

En un aborto de transacción, todos los almacenes transaccionales pendientes están invalidados de la STQ 260, incluso aquellos ya completados. Todas las líneas de caché que se modificaron por la transacción en la L1 240, es decir, tienen el bit sucio de 252 activado, tienen sus bits válidos apagados, eliminándolos de manera eficaz de la caché L1 240 de manera instantánea.

La arquitectura requiere que antes de finalizar una nueva instrucción, se mantenga el aislamiento del ajuste de lectura y escritura de la transacción. El aislamiento se asegura deteniendo la finalización de la instrucción a tiempos apropiados cuando están pendientes las XI; está permitida la ejecución fuera de orden especulativa, suponiendo de manera optimista que las XI pendientes son para diferentes direcciones y no provocan realmente un conflicto de transacción. Este diseño se adapta de manera muy natural con los interbloqueos de XI frente a finalización que se implementa en sistemas del estado de la técnica para asegurar la ordenación de memoria fuerte que requiere la arquitectura.

Cuando la L1 240 recibe una XI, L1 240 accede al directorio para comprobar la validez de la dirección que se ha realizado XI en la L1 240, y si el bit de lectura de TX 248 está activo en la línea que se ha realizado XI y la XI no se rechaza, la LSU 280 desencadena un aborto. Cuando una línea de caché con el bit 248 de lectura de TX se realiza LRU desde la L1 240, un vector de extensión de LRU especial recuerda para cada una de las 64 filas de la L1 240 que existió una línea de lectura de TX en esa fila. Puesto que no existe rastreo de dirección precisa para las extensiones LRU, cualesquiera XI no rechazadas que sugieran una fila de extensión válida de la LSU 280 desencadenan un aborto. Proporcionar la extensión de LRU de manera eficaz aumenta la capacidad de espacio ocupado de lectura del tamaño de la L1 al tamaño de la L2 y la capacidad de asociación, no proporcionando conflictos con otras CPU 114 (Figura 1) contra el rastreo de extensión de LRU no preciso que provoca abortos.

El espacio ocupado de almacén está limitado por el tamaño de caché de almacenamiento (la caché de almacenamiento

se analiza en más detalle a continuación) y por lo tanto implícitamente por el tamaño de L2 268 y la capacidad de asociación. No necesita realizarse acción de extensión de LRU cuando una línea de caché sucia de TX 252 se realiza LRU desde la L1 240.

Caché de almacén

En sistemas del estado de la técnica, puesto que la L1 240 y L2 268 son cachés de almacén-a través, cada instrucción de almacén provoca un acceso de almacén de L3 272; con 6 núcleos ahora por L3 272 y rendimiento mejorado adicional de cada núcleo, la tasa de almacén para la L3 272 (y hasta una extensión menor para la L2 268) se vuelve problemática para ciertas cargas de trabajo. Para evitar retardos de puesta en cola de almacén, tuvo que añadirse una caché 264 de almacén de recogida, que combina almacenes a direcciones vecinas antes de enviarlas a la L3 272.

Para rendimiento de memoria transaccional, es aceptable invalidar cada línea de caché sucia de TX 252 de la L1 240 en abortos de transacción, puesto que la caché L2 268 está muy cerca (penalización de pérdida de 7 ciclos de L1 240) para proporcionar de vuelta las líneas limpias. Sin embargo, sería inaceptable para el rendimiento (y área de silicio para rastreo) tener los almacenes transaccionales que escribir la L2 268 antes de que la transacción finalice y a continuación invalidar todas las líneas de caché de L2 sucias 268 al abortar (o incluso peor en la L3 compartida 272).

Los dos problemas de ancho de banda de almacén y manejo de almacén de memoria transaccional pueden ambos tratarse con la caché 264 de almacén de recogida. La caché 232 es una cola circular de 64 entradas, manteniendo cada una 128 bytes de datos con bits válidos de bytes precisos. En operación no transaccional, cuando se recibe un almacén de la LSU 280, la caché de almacenamiento comprueba si existe una entrada para la misma dirección, y en caso afirmativo recoge el nuevo almacén en la entrada existente. Si no existe entrada, se escribe una nueva entrada en la cola, y si el número de entradas libres cae por debajo de un umbral, las entradas más antiguas se escriben de vuelta a las cachés L2 268 y L3 272.

Cuando comienza una nueva transacción más exterior, todas las entradas existentes en la caché de almacenamiento se marcan cerradas de modo que no pueden recogerse nuevos almacenes en ellas, y se inicia la supresión de estas entradas en L2 268 y L3 272. A partir de ese momento, los almacenes transaccionales que van fuera de la LSU 280 STQ 260 asignan nuevas entradas, o se recogen en entradas transaccionales existentes. La escritura de vuelta de estos almacenes en L2 268 y L3 272 se bloquea, hasta que la transacción finalice satisfactoriamente; en ese punto posterior (pos-transacción) los almacenes pueden continuar recogiendo en entradas existentes, hasta que la siguiente transacción cierre estas entradas de nuevo.

La caché de almacenamiento se consulta en cada XI exclusiva o degradada, y provoca un rechazo de XI si la XI se compara a cualquier entrada activa. Si el núcleo no está completando instrucciones adicionales mientras está rechazando de manera continua XI, la transacción se aborta en un cierto umbral para evitar cuelgues.

La LSU 280 solicita un aborto de transacción cuando la caché de almacenamiento se desborda. La LSU 280 detecta esta condición cuando intenta enviar un nuevo almacén que no puede fusionarse en una entrada existente, y toda la caché de almacén se rellena con almacenes de la transacción actual. La caché de almacenamiento se gestiona como un subconjunto de la L2 268: mientras que pueden suprimirse líneas sucias de manera transaccional de la L1 240, tienen que permanecer residentes en la L2 268 a través de la transacción. El espacio ocupado de almacén máximo por lo tanto está limitado al tamaño de caché de almacenamiento de 64 x 128 bytes, y también está limitado por la capacidad de asociación de la L2 268. Puesto que la L2 268 es asociativa en 8 sentidos y tiene 512 filas, es típicamente lo suficientemente larga para no provocar abortos de transacción.

Si una transacción se aborta, se notifica a la caché de almacenamiento y todas las entradas que mantienen datos transaccionales se invalidan. La caché de almacenamiento también tiene una marca por palabra doble (8 bytes) si la entrada se escribió por una instrucción NTSTG - estas dobles palabras permanecen válidas a través de abortos de transacción.

Funciones implementadas por milicode

De manera tradicional, los procesadores de servidor de ordenadores centrales de IBM contienen una capa de firmware denominada milicode que realiza funciones complejas como ciertas ejecuciones de instrucción de CISC, manejo de interrupciones, sincronización de sistema, y RAS. Milicode incluye instrucciones dependientes de máquina así como instrucciones de la arquitectura de conjunto de instrucciones (ISA) que se recuperan y ejecutan de memoria de manera similar a instrucciones de programas de aplicación y el sistema operativo (SO). El firmware reside en un área restringida de memoria principal que los programas de cliente no pueden acceder. Cuando el hardware detecta una situación que necesita invocar milicode, la unidad 204 de captura de instrucción conmuta en "modo milicode" y empieza a capturar en la localización apropiada en el área de memoria de milicode. Milicode puede capturarse y ejecutarse de la misma manera que instrucciones de la arquitectura de conjunto de instrucciones (ISA), y puede incluir instrucciones ISA.

Para memoria transaccional, milicode está implicado en diversas situaciones complejas. Cada aborto de transacción

invoca una subrutina de milicode especializada para realizar las operaciones de aborto necesarias. El milicode de aborto de transacción empieza leyendo registros de fin especial (SPR) que mantienen la razón de aborto interna de hardware, razones de excepción potenciales, y la dirección de instrucción abortada, que milicode a continuación usa para almacenar una TDB si se especificara una. El texto de la instrucción TBEGIN se carga desde un SPR para obtener la máscara de grabación de GR, que es necesaria para que milicode conozca qué GR 238 restaurar.

La CPU 114 (Figura 2) soporta una instrucción de milicode únicamente especial para leer los GR de respaldo 224 y copiarlos en los GR 228. La dirección de instrucción de TBEGIN también se carga desde un SPR para establecer la nueva dirección de instrucción en la PSW para continuar la ejecución después de la TBEGIN una vez que finaliza la subrutina de aborto de milicode. Esa PSW puede grabarse más tarde como PSW de programa antiguo en el caso en el que el aborto se provoque por una interrupción de programa no filtrado.

La instrucción TABORT puede implementarse por milicode; cuando la IDU 208 decodifica TABORT, da órdenes a la unidad de recuperación de instrucción para bifurcarse en milicode de TABORT, a partir de lo cual milicode se bifurca en la subrutina de aborto común.

La instrucción Extraer Profundidad de Anidación de Transacción (ETND) puede también estar en milicode, puesto que no es crítica para el rendimiento; milicode carga la profundidad de anidación actual de un registro de hardware especial y la coloca en un GR 228. La instrucción de PPA está en milicode; realiza el retardo óptimo basándose en el contador de aborto actual proporcionado por software como un operando a PPA, y también basándose en otro estado interno de hardware.

Para transacciones restringidas, milicode puede mantener el registro del número de abortos. El contador se resetea a 0 en la finalización de TEND satisfactoria, o si tiene lugar una interrupción en el SO (puesto que no es conocido si o cuándo el SO retornará al programa). Dependiendo del contador de aborto actual, milicode puede invocar ciertos mecanismos para mejorar la posibilidad de éxito para el reintento de transacción posterior. Los mecanismos implican, por ejemplo, aumentar de manera sucesiva retardos aleatorios entre reintentos, y reducir la cantidad de ejecución especulativa para evitar encontrar abortos provocados por accesos especulativos a datos que la transacción no está realmente usando. Como un último recurso, milicode puede difundir a otras CPU 114 (Figura 2) para detener todo el trabajo que entre en conflicto, recuperar la transacción local, antes de liberar las otras CPU 114 para continuar el procesamiento normal. Múltiples CPU 114 deben coordinarse para no provocar bloqueos inactivos, por lo que se requiere alguna serialización entre instancias de milicode en diferentes CPU 114.

La Figura 4 ilustra un sistema 300 informático de acuerdo con una realización. La Figura 4 está configurada para implementar características analizadas en las Figuras 1-3 y 5-18. El ordenador 300 comprende unos múltiples procesadores designados como el procesador 112a (CPU 1) y 112b (CPU 2) (junto con procesadores adicionales no mostrados) en comunicación con una memoria 310 por medio de un subsistema de caché jerárquico, en el que las cargas transaccionales, por el procesador, se monitorizan en una cache de un subsistema de caché jerárquico. El sistema 300 informático mostrado en la Figura 4 tiene los mismos elementos que el sistema informático 100 mostrado en la Figura 1 y los mismos números de referencia, pero cada elemento en la Figura 1 no se muestra en la Figura 4.

El sistema 300 informático puede gestionar una solicitud, por ejemplo, una interrupción, presentada a uno o más procesadores que están disponibles para, o que procesan actualmente transacciones. En un ejemplo, un procesador solicitante (por ejemplo, CPU 1 (112a)) puede seleccionar el procesador de recepción/remoto (por ejemplo, CPU 2 (112b)) y enviar una solicitud al procesador remoto seleccionado. En un ejemplo, el sistema informático es un sistema o entorno de ejecución transaccional (TX), por ejemplo, que incluye una CPU o procesador que puede ejecutar una transacción. Cada transacción se muestra respectivamente como instrucciones 320a y 320b de transacción que se ejecutan respectivamente en los procesadores 112a y 112b. Cada procesador 112a y 112b tiene su propio registro 334a y 334b, respectivamente.

Cada caché 118a y 118b de datos puede incluir respectivamente sus propia caché de L1 y L2. La memoria informática se representa genéricamente por la memoria 310, que puede incluir memoria caché de nivel superior en la CPU designada como la CPU de TX, es decir, los procesadores 112a y 112b. Cada procesador 112a y 112b tiene su propia tabla de rastreo de interferencia de transacción local designada como las tablas 1350a y 1350b, respectivamente. Las tablas 1350a y 1350b pueden almacenarse respectivamente en la caché 118a, 118b de datos, registros 334a, 334b, y/o en memoria 310. La memoria 310 puede incluir también un bloque 350 de diagnóstico de transacción para almacenar información de diagnóstico de transacciones, que puede incluir la información de interferencia de transacción (junto con estadísticas) almacenada en las Tablas 1350a y 1350b, como se analiza adicionalmente en el presente documento.

El sistema 300 informático es una representación de un entorno de transacción (TX) que envía, recibe, y procesa tanto solicitudes como respuestas de acuerdo con las realizaciones. Obsérvese que se proporcionan diversos ejemplos en los que el procesador 112a (CPU 1) se ilustra como el procesador solicitante que genera y envía una solicitud al procesador 112b (CPU 2) que se ilustra como el procesador de recepción/remoto que recibe la solicitud. Se entiende que esta designación es para fines de explicación, ya que cualquier procesador puede enviar y recibir solicitudes para datos como se entiende por un experto en la materia.

La Figura 5 ilustra un ejemplo de solicitud y respuesta de protocolo. La solicitud de protocolo puede denominarse como protocolo de múltiples procesadores de ordenador principal, y el protocolo puede ser mediante una interconexión basada en bus o basada en conmutador. El protocolo puede ser señalización toda paralela (intromisión de bus), paquetes en serie o una combinación de los mismos.

Como un ejemplo, puede enviarse una solicitud 505 para datos desde la CPU 1 (112a) a la CPU 2 (112b). La solicitud 505 incluye un campo 506 de tipo que indica qué tipo de solicitud se está enviando (por ejemplo, lectura compartida o una lectura exclusiva - o, lectura de propiedad - solicitud de acuerdo con el protocolo de coherencia MESI conocido, o solicitudes de protocolo de acuerdo con otros protocolos de este tipo), y un campo 507 de etiqueta que identifica el procesador particular (por ejemplo, CPU 1) que envía la solicitud y opcionalmente el procesador de recepción, por ejemplo, CPU 2 al que se está enviando la solicitud y, opcionalmente, si pueden procesarse de manera concurrente múltiples solicitudes, una ID de solicitud específica para identificar de manera única cada solicitud. La solicitud 505 también incluye un campo 508 de acceso que identifica el tipo de acceso que se está solicitando por el procesador solicitante (CPU 1), y un campo 509 de dirección. El campo 509 de dirección identifica la dirección de memoria de la línea de caché o palabra de memoria que se está solicitando. El protocolo de solicitud 505 puede incluir un campo 510 de corrección de errores que contiene código de detección y/o de corrección de errores usado, por ejemplo, comprobación de redundancia cíclica (CRC), bits de paridad, o ECC.

Puede enviarse una respuesta 515 de vuelta desde el procesador de recepción (CPU 2) al procesador solicitante (CPU 1). La respuesta 515 incluye el campo 516 de tipo que indica el tipo de respuesta, tal como una respuesta de lectura (RESPUESTA DE LECTURA), y un campo 517 de etiqueta. El campo 517 de etiqueta puede ser la misma etiqueta que el campo 507 de etiqueta de la solicitud 505 original y/o el campo 517 de etiqueta puede incluir la dirección de memoria solicitada de la línea de caché. La respuesta 515 incluye un campo 518 de datos que son los datos solicitados que se solicitaron por el procesador solicitante (CPU 1). Algunas respuestas de protocolo pueden no incluir transferencias de datos (por ejemplo, solicitudes de protocolo para cambiar de escala la propiedad de compartida a propiedad exclusiva para una línea) y pueden incluir únicamente un acuse de recibo que se ha realizado el procesamiento. Un campo 519 de corrección de errores está incluido en la respuesta 515.

En algunas realizaciones, la señalización para las solicitudes de protocolo puede tener lugar en paralelo a través de una pluralidad de líneas de bits, y cualesquiera campos sin uso pueden corresponder a líneas que no tienen valor de protocolo definido, se establecen a un valor por defecto, o de otra manera no se consideran una parte de un mensaje de protocolo. En algunas realizaciones, la solicitud de protocolo puede transmitirse en múltiples "latidos", por ejemplo, grupos de bits sucesivos que en su totalidad representan un mensaje de protocolo. En otras realizaciones más, pueden transmitirse solicitudes de protocolo en bits en serie. En protocolos transmitidos en múltiples latidos o en serie, algunos mensajes pueden consistir en más ciclos de señalización de bus que otras solicitudes de protocolo.

La Figura 6 ilustra otro ejemplo de solicitud de protocolo. Muchos protocolos obtienen acceso exclusivo mediante RFO (lectura de propiedad) o solicitudes de lectura exclusiva para operaciones de escritura, mientras que algunos otros protocolos pueden escribir directamente (limitado). Como un ejemplo, una solicitud 605 de ejemplo para escribir datos puede enviarse de la CPU 1 (112a) a la CPU 2 (112b). La solicitud 605 incluye el campo 506 de tipo que identifica el tipo de solicitud que se está enviando (por ejemplo, escritura), y el campo 507 de etiqueta que identifica el procesador particular (por ejemplo, CPU 1) que envía la solicitud de escritura, y opcionalmente el procesador de recepción, por ejemplo, CPU 2 que está enviando la solicitud y una solicitud particular. La solicitud 605 también incluye un campo 504 de datos que transmite los datos que se están escribiendo por el procesador solicitante (CPU 1), y el campo 509 de dirección. El campo 509 de dirección identifica la dirección de memoria de la línea de caché o línea de dirección en la que se está escribiendo. El protocolo de solicitud 605 puede incluir el campo 510 de corrección de errores que contiene código de detección y/o de corrección de errores usado, por ejemplo, comprobación de redundancia cíclica (CRC), bits de paridad, o ECC.

En respuesta a una solicitud de escritura, típicamente no hay respuesta puesto que no es necesario obtener datos para acceso exclusivo antes de realizar una escritura.

La Figura 7 es un diagrama 700 de flujo de la generación de solicitud de protocolo por el procesador (por ejemplo, CPU 1 (112a)) que realiza una solicitud de datos de acuerdo con una realización. Un procesador (por ejemplo, CPU 1) tiene una solicitud para datos de memoria en el bloque 705. El procesador (por ejemplo, CPU 1 (112a)) comprueba si los datos solicitados están en su propia caché local (por ejemplo, caché L1 en la caché 118a de datos) en el bloque 710. Cuando los datos están disponibles en la propia caché local del procesador, el flujo continúa al bloque 735. Cuando los datos no están disponibles en las cachés locales del procesador (CPU 1), el procesador genera la solicitud XI (interrogación cruzada) para solicitar los datos deseados de otros procesadores (tales como la CPU 2 (112b)) en el bloque 715. El procesador solicitante (CPU 1) envía la solicitud XI para los datos al procesador de recepción (CPU 2) mediante la interconexión 122 en el bloque 720, y el procesador solicitante (CPU 1) recibe la respuesta XI con los datos (solicitados) del procesador de recepción (CPU 2) en el bloque 725. El procesador solicitante (CPU 1) coloca los datos en sus cachés locales (por ejemplo, caché LI, L2) en la caché 118a de datos en el bloque 730. El procesador solicitante (CPU 1) obtiene los datos desde su caché 118a local mediante la caché 116a de instrucción en el bloque 735. La caché 116a de instrucción del procesador solicitante proporciona los datos a los circuitos de la CPU 1 para su

procesamiento en el bloque 740.

En una realización, y de acuerdo con un protocolo de caché común (por ejemplo, el protocolo MESI conocido), cuando el procesador accede a los datos para lectura, y los datos no están disponibles, se genera una XI para lectura-compartición en la que los datos se obtienen de una manera compartida de manera que múltiples CPU 112a, 112b pueden tener una copia de los datos en una caché y en el que cada CPU puede procesar un acceso de lectura de memoria que corresponde a los datos. Los datos recibidos se colocan en la caché y se marcan para acceso compartido, y el procesador puede realizar accesos de lectura de la copia en respuesta a unas operaciones de lectura de memoria. Cuando el procesador accede a datos para escritura, y los datos no están disponibles en estado exclusivo, se genera una XI para lectura exclusiva en la que se obtienen datos de una manera exclusiva, de manera que únicamente una única CPU (por ejemplo, la CPU 112a) puede tener una copia de los datos en una caché. Los datos recibidos se colocan en la caché y se marcan para acceso exclusivo, y el procesador puede actualizar la copia en respuesta a operaciones de escritura de memoria. En una realización, cuando están presentes datos en modo compartido, y se recibe un acceso de escritura, se genera una XI de lectura exclusiva. En al menos una realización, esto se indica como una solicitud de lectura exclusiva distinta en la que no se reciben datos como parte de la respuesta. En una realización, cuando se ha recibido una respuesta los datos de caché se marcan para acceso exclusivo.

La Figura 8 ilustra un diagrama 800 de flujo de ejemplo de manejo de solicitud por el procesador de recepción/remoto (por ejemplo, CPU 2 (112b)) que recibe la solicitud y envía una respuesta de acuerdo con una realización.

El procesador remoto (CPU 2) recibe la solicitud XI para datos desde el procesador solicitante (CPU 1) en el bloque 805. En el bloque 810, el procesador remoto (CPU 2) comprueba si se detecta una interferencia comprobando si el procesador remoto está procesando una transacción que necesita actualmente los datos solicitados (en la caché local del procesador remoto). Cuando el procesador remoto (CPU 2 (112b)) determina que el procesador remoto está usando actualmente los datos solicitados por el procesador solicitante (CPU 1 (112a)) en el bloque 810, el procesador remoto determina (SÍ) que se detecta interferencia y el procesador remoto (CPU 2) aborta la transacción local que tiene lugar en el procesador remoto (CPU 2) en el bloque 815. Una vez que se aborta la transacción local en el procesador remoto (CPU 2), el procesador remoto transmite los datos con la respuesta de lectura (RESPUESTA DE LECTURA) al procesador solicitante (CPU 1) en el bloque 820. El procesador remoto (CPU 2) cambia el estado de datos y opcionalmente purga datos de sus cachés locales si fuera necesario en el bloque 825. En una realización, un cambio de estado de caché puede incluir liberar datos de al menos uno de un ajuste de lectura o escritura cuando se ha abortado una transacción. En una realización, un cambio de estado de caché puede incluir cambiar el estado de una línea de caché en el directorio de caché, por ejemplo, establecer el estado a uno de compartido, exclusivo, inválido y así sucesivamente, de acuerdo con un protocolo de caché tal como el protocolo de MESI conocido. El procesador remoto (CPU 1) inicia el procesamiento de fallo de transacción en el bloque 830 y el flujo finaliza. Cuando el procesador remoto (CPU 2 (112b)) determina que el procesador remoto no está usando actualmente los datos solicitados por el procesador solicitante (CPU 1 (112a)) en el bloque 810, el procesador remoto determina que NO se detecta interferencia y el procesador remoto (CPU 2) transmite datos con la respuesta de lectura (RESPUESTA DE LECTURA) en el bloque 835. El procesador remoto (CPU 2) cambia el estado de datos y opcionalmente purga datos de cachés locales si fuera necesario en el bloque 840, y el flujo finaliza.

La Figura 9 es un diagrama 900 de flujo que ilustra manejo de transacción por un procesador de acuerdo con una realización. En el bloque 905, la transacción comienza la ejecución en el procesador (por ejemplo, CPU 1 o CPU 2). En la Figura 9, obsérvese que cada procesador (CPU 1 y CPU 2) puede estar realizando estas acciones, es decir, pueden procesarse las transacciones tanto por 112a, 112b y así sucesivamente. El procesador realiza las instrucciones dentro de la transacción en el bloque 910 (por ejemplo, como se analiza en la Figura 7). El procesador realiza acciones de protocolo en respuesta a instrucciones transaccionales en el bloque 915. El procesador comprueba si hay interferencia (con el uso de los datos) que requiere que el procesador aborte su transacción en 920. Cuando el procesador determina (SÍ) que se ha detectado interferencia, el procesador aborta su propia transacción (en los datos) en el bloque 925 y el flujo continúa al bloque 935. Cuando el procesador determina que NO se detecta interferencia, el procesador completa (las instrucciones de) su transacción en el bloque 930. El procesador escribe información de transacción (tal como el bloque de diagnóstico de transacción (TBD)) en un registro en el bloque 935.

De acuerdo con las realizaciones, el protocolo de coherencia se amplía para incluir información adicional acerca del estado de transacción. La Figura 10 ilustra la solicitud 505 de protocolo y una nueva respuesta 1005 de protocolo de acuerdo con las realizaciones. Como se muestra en la Figura 10, algunos campos de la solicitud 505 son idénticos a los campos analizados en la Figura 5. Como se ha indicado anteriormente, la solicitud 505 incluye el campo 506 de tipo que identifica qué tipo de solicitud se está enviando (por ejemplo, LECTURA), y el campo 507 de etiqueta que identifica el procesador particular (por ejemplo, CPU 1) que envía la solicitud (y opcionalmente el procesador de recepción/remoto, por ejemplo, CPU 2 que la solicitud se está enviando y un número de solicitud). La solicitud 505 también incluye el campo 508 de acceso que indica el tipo de acceso que se está solicitando por el procesador solicitante (CPU 1), y el campo 509 de dirección que indica la dirección de memoria de la línea de caché o línea de dirección que se está solicitando. El protocolo de solicitud 505 incluye el campo 510 de corrección de errores que indica el tipo de código de error usado tal como comprobación de redundancia cíclica (CRC).

La nueva respuesta 1005 incluye los campos de la respuesta 515 (en la Figura 5) junto con un campo 1010 de estado

de aborto de transacción adicional. La respuesta 1005 puede enviarse de vuelta desde el procesador de recepción/remoto (CPU 2) al procesador solicitante (CPU 1). La respuesta 515 incluye el campo de tipo 516 que indica el tipo de respuesta, tal como una respuesta de lectura (RESPUESTA DE LECTURA), y el campo 517 de etiqueta que indica (cuál puede ser la misma etiqueta que el campo 507 de etiqueta de la solicitud 505 original) la dirección de memoria solicitada de la línea de caché. La respuesta 515 incluye el campo 518 de datos que son los datos solicitados que se solicitaron por el procesador solicitante (CPU 1). Si no se transmiten datos con una respuesta de protocolo, entonces el campo 518 de datos está vacío o no presente. El campo 519 de detección/corrección de errores está incluido en la respuesta 1005.

Adicionalmente, la respuesta 1005 de nuevo protocolo (enviada por el procesador remoto CPU 2 al procesador solicitante CPU 1) tiene el campo 1010 de estado de aborto de transacción. El campo 1010 de estado de aborto de transacción incluye una o más de la siguiente información acerca de la transacción que se ha ejecutado anteriormente en el procesador remoto/de recepción (CPU 2) antes de que se aborte:

1) Si la solicitud 505 (del procesador solicitante (CPU 1) provocó y/o no provocó restauración (es decir, aborto);

2) Prioridad de esta transacción que se estaba ejecutando en el procesador remoto/de recepción (CPU 2) antes de que se abortara;

3) Cuántas instrucciones, operaciones de memoria y/u otra medida de trabajo se han realizado por la transacción que se ha ejecutado previamente en el procesador remoto/de recepción (CPU 2) antes de que se abortara la transacción;

4) Identificación de la transacción (por ejemplo, testigo, dirección de TBEGIN, y/u otros medios de identificación de la transacción abortada) que se ha ejecutado previamente en el procesador remoto (CPU 2).

Adicionalmente, el campo 1010 de estado de aborto de transacción indica qué datos se obtienen por la transacción (que se ha ejecutado previamente en el procesador remoto/de recepción) que tuvo que abortar, indica la dirección de la transacción, e indica el coste de abortar la transacción (que pueden ser 3 ciclos de reloj o 20.000 ciclos de reloj de trabajo).

Cuando un procesador (por ejemplo, el procesador de recepción, CPU 2 en escenarios de ejemplo) está en ejecución transaccional, una solicitud de coherencia puede provocar la ejecución transaccional para abortar, por ejemplo, puesto que los datos son parte del ajuste de lectura o escritura de la transacción, y se detecta un conflicto (es decir, interferencia).

De acuerdo con una realización, la Figura 11 ilustra la solicitud 605 (escritura) en la Figura 6 para escribir datos (enviados del procesador solicitante CPU 1 (112a) al procesador de recepción/remoto CPU 2 (112b)) y se envía de vuelta una nueva respuesta 1105 al procesador solicitante CPU 1 (112a) desde el procesador remoto CPU 2 (112b). La Figura 11 es un ejemplo de solicitud y respuesta de protocolo que añade información de transacción en la respuesta de acuerdo con una realización introduciendo una nueva respuesta de protocolo a una transacción que previamente no requiere una respuesta de protocolo para transmitir información de interferencia/aborto de transacción al originador de la solicitud. En al menos una realización, la respuesta de protocolo que se transmite para el único fin de proporcionar estado de aborto de transacción es opcional y puede suprimirse en algunos modos, en respuesta a bits de configuración, en respuesta a congestión de bus, y/o por otras razones. En un escenario de este tipo cuando no se recibe respuesta, no se informa estado de aborto de transacción que corresponde a una solicitud para la que no se ha recibido respuesta. En al menos una realización, puede informarse una ausencia de una o más respuestas. Como se ha indicado anteriormente, la solicitud 605 incluye el campo 506 de tipo que indica qué tipo de solicitud se está enviando (por ejemplo, escritura), y el campo 507 de etiqueta que identifica el procesador particular (por ejemplo, CPU 1) que envió la solicitud de escritura (y el procesador de recepción, por ejemplo, CPU 2 al que se está enviando la solicitud). La solicitud 605 también incluye el campo 508 de acceso que indica el tipo de acceso que se está solicitando por el procesador solicitante (CPU 1), y el campo 509 de dirección. El campo 509 de dirección indica la dirección de memoria de la línea de caché o línea de dirección a la que se está solicitando para escribir. El protocolo de solicitud 605 puede incluir el campo 510 de detección y/o corrección de errores con un código de corrección/detección de errores, tal como bits de paridad, ECC o código de comprobación de redundancia cíclica (CRC).

En respuesta a la solicitud 605 de escritura, la nueva respuesta 1105 (a la solicitud de escritura) incluye ahora el campo 1010 de estado de aborto de transacción analizado en el presente documento. La nueva respuesta 1105 incluye los campos de la respuesta 1005 (en la Figura 10). La respuesta 1105 puede enviarse de vuelta desde el procesador de recepción/remoto (CPU 2) al procesador solicitante (CPU 1). La respuesta 1105 incluye el campo de tipo 516 que indica el tipo de respuesta, tal como una respuesta de escritura (RESPUESTA DE ESCRITURA), y el campo 517 de etiqueta que indica (que puede ser la misma etiqueta que el campo 507 de etiqueta de la solicitud 505 original para identificar la solicitud a la que corresponde la respuesta) la dirección de memoria solicitada de la línea de caché. La respuesta 515 puede incluir un campo 518 de datos que son los datos solicitados que se solicitaron por el procesador solicitante (CPU 1). Puesto que no hay datos desde la caché (local) del procesador de recepción/remoto (CPU 2), entonces el campo 518 de datos está vacío. El campo 519 de corrección de errores está incluido en la respuesta 1005.

Como se analiza en el presente documento, el campo 1010 de estado de aborto de transacción proporciona el estado de la transacción (que se ha ejecutado previamente en el procesador de recepción/remoto (CPU 2)) que se requirió para abortar debido a la solicitud 605 de escritura (del procesador solicitante (CPU 1)).

Aunque se han descrito mejoras de protocolo de acuerdo con la presente divulgación en una realización ejemplar en conjunto con la adición de un campo de protocolo que corresponde a un estado de aborto de transacción e información asociada a una respuesta de lectura existente a una solicitud de lectura, y en otra realización ejemplar en conjunto con la adición de una respuesta de protocolo que corresponde a una solicitud de escritura de protocolo que no tiene una respuesta de protocolo de acuerdo con el estado de la técnica para transmitir al menos un campo de protocolo que corresponde a un estado de aborto de transacción e información asociada e identificación de una correspondiente solicitud de escritura, los expertos en la materia podrán aplicar las enseñanzas contenidas en el presente documento a otros formatos de XI, formatos de protocolo, tipos de solicitudes, protocolos de coherencia y así sucesivamente.

De acuerdo con una realización, la Figura 12 es un diagrama 1200 de flujo que ilustra el manejo de solicitud de coherencia, por ejemplo, por la CPU de procesador de recepción/remoto 2 (112b) que recibe la solicitud desde el procesador solicitante CPU 1 (112a). La Figura 12 incluye los bloques de la Figura 8, junto con nuevos bloques 1205 y 1210 (modificados) que comprenden transmitir el estado de aborto de transacción (en el campo 1010 de aborto de transacción) como parte de respuestas de protocolo.

En la Figura 12, el manejo de la solicitud por el procesador de recepción/remoto (por ejemplo, CPU 2 (112b)) recibe la solicitud de acuerdo con una realización. El procesador remoto (CPU 2) recibe la solicitud XI (por ejemplo, la solicitud 505 y/o solicitud 605) para datos desde el procesador solicitante (CPU 1) en el bloque 805. En el bloque 810, el procesador remoto (CPU 2) comprueba si se detecta una interferencia comprobando si el procesador remoto está procesando una transacción que hace referencia a datos de una manera incompatible con la solicitud recibida, por ejemplo, la solicitud 505 y 605. Por ejemplo, en una realización ejemplar, una solicitud compartida de lectura es compatible con una referencia a datos en un ajuste de lectura de transacciones, pero no una solicitud de escritura. Además, una solicitud de lectura exclusiva, cambia de escala de lectura (es decir, cambio de compartida a exclusiva) o de escritura no es compatible con los mismos datos referenciados en cualquiera del ajuste de lectura o escritura de la transacción. Cuando el procesador remoto (CPU 2 (112b)) determina que el procesador (él mismo) remoto está usando actualmente los datos solicitados por el procesador solicitante (CPU 1 (112a)) en el bloque 810, el procesador remoto determina (SÍ) que se detecta interferencia y el procesador remoto (CPU 2) aborta la transacción local que tiene lugar en el procesador remoto (CPU 2) en el bloque 815. Una vez que se aborta la transacción local en el procesador remoto (CPU 2), el procesador remoto transmite los datos con la respuesta de lectura (RESPUESTA DE LECTURA) al procesador solicitante (CPU 1) junto con el campo 1010 de estado de transacción (que notifica la solicitud que provocó la transacción (que se ha ejecutado previamente) para abortar en el procesador remoto/de recepción (CPU 2) para satisfacer la solicitud) en el bloque 1205. En otra realización, puede realizar acuse de recibo de una solicitud de escritura recibida con una respuesta 1105 de escritura. En los sistemas del estado de la técnica, no se habría incluido el campo 1010 de estado de transacción en la respuesta (RESPUESTA) enviado desde la CPU 2 remota (que estaba ejecutando previamente su propia transacción ahora abortada) de vuelta a la CPU 1 solicitante (que envió la solicitud 505 y/o 605). El procesador remoto (CPU 2) cambia el estado de datos y opcionalmente purga datos de sus cachés locales si fuera necesario en el bloque 825. El procesador remoto (CPU 1) inicia el procesamiento de fallo de transacción en el bloque 830 y el flujo finaliza. Cuando el procesador remoto (CPU 2 (112b)) determina que el procesador remoto no está usando actualmente los datos solicitados por el procesador solicitante (CPU 1 (112a)) en el bloque 810, el procesador remoto determina que NO se detecta interferencia y el procesador remoto (CPU 2) transmite los datos con la respuesta de lectura (RESPUESTA DE LECTURA) junto con el campo 1010 de estado de transacción (que indica que la solicitud no provocó que una transacción se abortara) en el bloque 1210. El procesador remoto (CPU 2) cambia el estado de datos y opcionalmente purga datos de cachés locales si fuera necesario en el bloque 840, y el flujo finaliza.

La Figura 13 es un diagrama 1300 de flujo que ilustra el origen de solicitud de protocolo y procesamiento por el procesador solicitante (CPU 1 (112a)) de acuerdo con una realización. La Figura 13 incluye los bloques anteriormente analizados en la Figura 7, y el análisis de la Figura 7 no se repite. Adicionalmente, el diagrama 1300 de flujo incluye nuevos bloques 1305 y 1310. Por ejemplo, el procesador solicitante (CPU 1) recibe la respuesta XI con datos del procesador remoto (CPU 2) (bloque 725) junto con el nuevo campo 1010 de estado de aborto de transacción en el bloque 1305. El procesador solicitante (CPU 1) coloca los datos en cachés locales (tales como las cachés L1 y/o L2) en el bloque 730. En el bloque 1310, el procesador solicitante (CPU 1) añade el estado recibido (por ejemplo, información en el campo 1010 de estado de transacción) a una tabla 1350a de rastreo de interferencia de transacción local en la caché 118a de datos (y/o la memoria 310). La tabla 1350a de rastreo de interferencia de transacción local puede ser una localización de almacenamiento que mantiene el registro de la interferencia cuando el procesador solicitante (CPU 1) provoca la interferencia (que da como resultado aborto de transacción en el procesador remoto (CPU 2)), y la tabla 1350a de rastreo de interferencia de transacción local puede incluir un registro de esta información. La tabla 1350a de interferencia (almacenamiento) de transacción de rastreo local incluye estadísticas de incremento y estado de transacción actual. Las estadísticas de incremento aumentan para cada aborto de transacción (informado de vuelta al procesador de recepción (CPU 1), y las estadísticas de incremento pueden separarse en solicitudes compartidas/exclusivas (R/W) y solicitudes agregadas. La tabla 1350a de almacenamiento de interferencia de

transacción de rastreo local puede incluir un contador que se incrementa para cada aborto de transacción provocado solicitando los procesadores (CPU 1) y para cada vez que no se aborta la transacción en los procesadores solicitantes (CPU 1). En algunas realizaciones, el estado de aborto de transacción recibido en conjunto con el bloque 1310 puede también agregarse en una unidad de monitorización de rendimiento, una unidad de instrumentación de tiempo de ejecución, y/u otra lógica de rastreo de rendimiento para hacer la información disponible a optimizadores dinámicos, compiladores justo a tiempo (JIT), o para ajuste de rendimiento por los programadores. La información puede grabarse mediante registro en almacenamiento, en estructuras dirigidas a la monitorización de rendimiento y/o elevando notificaciones, por ejemplo, ramales basados en eventos de PMU de acuerdo con Power ISA™, o excepciones. La generación de informes puede incluir la naturaleza de interferencia, identificación de la transacción interferente y/o interferencia, los ID de procesador, direcciones que son el objeto de tal interferencia y así sucesivamente.

Como se analiza en el presente documento, los protocolos de coherencia usan un número de bits, bits de dirección, datos, y de estado y control. Estos datos se usan para emitir una solicitud de datos, y para indicar su propiedad (por ejemplo, compartida, exclusiva) y estado (por ejemplo, sucio, limpio). Uno o más bits adicionales (para el campo 1010 de estado de aborto de transacción) se añaden para indicar que una solicitud provocó que una transacción se abortara y/o podría provocar que la transacción se abortara. Por ejemplo, un bit de estado (campo 1010 de estado de aborto de transacción) indica que los datos solicitados provocan un conflicto para la ejecución transaccional, cuando los datos se proporcionaban por el procesador remoto (CPU 1). Como se indica, una respuesta a la solicitud puede provenir con una indicación de que una transacción está en progreso, y que la solicitud provocó que una transacción se abortara.

En una realización, se transmiten métricas adicionales, tales como número de instrucciones en una transacción abortada en el campo 1010 de estado de aborto de transacción. En una realización, se usan indicaciones para determinar un retraso, por ejemplo, con respecto al reinicio de una transacción en caso de que la transacción ganadora más tarde se aborte y se reinicie, para evitar escenarios de bloqueo activo. El procesador solicitante (CPU 1) puede también regular (reducir) su tasa de solicitud en respuesta a abortar demasiadas transacciones remotas para aumentar el rendimiento de sistema global, cuando el procesador solicitante (CPU 1) determina que el procesador solicitante ha provocado demasiados (por ejemplo, una cantidad predefinida) abortos de transacción en el procesador remoto/de recepción (CPU 2) comprobando el campo 1010 de estado de aborto de transacción local.

En otra realización más, tales notificaciones (en el campo 1010 de estado de aborto de transacción) se recogen en registros y/o notifican mediante un mecanismo de notificación (por ejemplo, una excepción) a un componente de reoptimización dinámico; el componente de reoptimización dinámico puede reoptimizar dinámicamente código para reducir la interferencia y/o generar código alternativo para usar bloqueos en lugar de transacciones.

Ahora, los bloques de la Figura 9 están incluidos en la Figura 14 pero su análisis no se repite. La Figura 14 también incluye una modificación a la Figura 9. La Figura 14 es un diagrama 1400 de flujo que ilustra manejo de transacción por un procesador de acuerdo con una realización. Puede suponerse que el procesador es el procesador solicitante (CPU 1). La Figura 14 incluye los bloques de la Figura 9, junto con el bloque 1405. En el bloque 1405, el procesador solicitante (CPU 1) escribe información de transacción (tal como el bloque de diagnóstico de transacción (TBD)) que incluye las estadísticas de interferencia de transacción (obtenidas del nuevo campo 1010 de estado de aborto de transacción y almacenadas en la tabla 1350a de almacenamiento de interferencia de transacción de rastreo local) en el registro 334, caché 118a, y/o memoria 310 en el bloque 1405. En una realización, cuando se completa una transacción, la transacción puede incluir una indicación de si ha provocado que se aborte una o más transacciones como parte de un código de resultado, por ejemplo, un registro de resultado, registro de condición de resultado y así sucesivamente (por ejemplo, en los registros 334a, 334b). En una realización, cuando la transacción establece un estado de resultado en uno o más registros (por ejemplo, los registros 334a, 334b) y/o localizaciones de memoria, el uno o más registros (por ejemplo, en los registros 334a, 334b) y/o localizaciones de memoria (por ejemplo, en la memoria 310, cachés 118a, 118b) puede incluir el número de transacciones interferidas con las que se abortaron para proporcionar datos a la transacción, la naturaleza de la interferencia (solicitud de lectura con ajuste de escritura, o solicitud de escritura con cualquiera de lectura o escritura y así sucesivamente). En una realización, la información así almacenada puede incluir información específica acerca de cada una de las transacciones, y que incluye una manera para identificar un procesador en el que ha tenido lugar la interferencia, una transacción que se ha abortado (por ejemplo, especificando el comienzo de la dirección de la transacción XBEGIN o TBEGIN, un testigo, un ID de transacción y así sucesivamente), una medida de trabajo que se ha realizado por esa transacción antes de que se abortara y así sucesivamente. En una realización, cuando un bloque de diagnóstico de transacción del estado de la técnica se amplía de acuerdo con la presente divulgación para que tenga campos adicionales que corresponden a estado de interferencia y aborto de transacción, la información almacenada en una o más localizaciones de memoria corresponde a localizaciones de memoria de un bloque de diagnóstico de transacción (TDB) de acuerdo, por ejemplo, con la arquitectura z/ y a campos de un bloque de diagnóstico de transacción de acuerdo con la presente divulgación, correspondiendo los campos a información transmitida individualmente por campos de protocolo nuevamente introducidos, y/o por información agregada de una pluralidad de tales campos. En otra realización, la información almacenada en registros y/o localizaciones de memoria se proporciona como distinta, separada e independiente de un estado de la técnica TDB.

En el bloque 1405, el procesador solicitante (CPU 1) está configurado para proporcionar las estadísticas de interferencia mediante la unidad de monitorización de rendimiento y/o unidad de instrumentación de tiempo de

ejecución para el programador, y proporcionar información de interferencia a firmware, milicode, etc. En milicode, el código milicode puede usar la información de estadística de interferencia para evitar el bloqueo activo y/o usar la información de estadística de interferencia para optimizar el reinicio de la transacción. En una aplicación, la aplicación puede usar información de estadísticas de interferencia para evitar el bloqueo activo y optimizar el reinicio de la transacción.

De acuerdo con una realización, se proporciona código de ejemplo a continuación para fines de explicación.

El siguiente pseudocódigo (ejecutado en los procesadores 112a, 112b) proporciona una forma de realización de la optimización de reinicio de transacción (ya sea de manera transparente en milicode y/o por código integrado en una aplicación que se ejecuta en el procesador) para reiniciar una transacción cuando se ha abortado la transacción, y optimizarse en particular para evitar escenarios de bloqueo activo. El código ejemplar usa información almacenada en la TDB de una manera ejemplar, sin embargo, los expertos en la materia podrán usar información obtenida de, incluyendo, pero sin limitación, localizaciones de memoria (por ejemplo, la memoria 310), registros 334a, 334b, códigos de estado, y/o una unidad de monitorización de rendimiento, unidad de instrumentación de tiempo de ejecución, y así sucesivamente:

```
IF (esta transacción se aborta) {
   ACCESS TDB information
   IF (esta transacción aborta otra transacción) {
      mutual_shootdown detected <= TRUE;
      IF (mutual_shootdown_detected)
         avoid_livelock(); // tomar acciones para evitar bloqueo activo,
                           //  por  ejemplo,  reducir  prioridad  de  transacción
                           actual;
                           // esperar periodo de repliegue;
                           // sincronizar usando bloqueos;
                           // y así sucesivamente
   }
}
```

De acuerdo con el código ejemplar, el código de reinicio de transacción incluye comprobaciones de si se ha abortado a sí mismo, y a su vez ha abortado otra transacción. En una realización, un posible bloqueo activo se diagnostica inmediatamente. En otra realización, se realiza una prueba (no mostrada) si la transacción actual y la transacción interferida son parte de un gráfico de interferencia cíclica (es decir, cada transacción está abortando directa o indirectamente la otra transacción). En una realización, cuando se diagnostica una desconexión mutua, se toman acciones de evitación de interferencia. En una realización, cuando se diagnostica una desconexión mutua, se toman acciones de evitación de bloqueo activo. En una realización, las acciones de evitación de interferencia se invocan llamando una función avoid_livelock(). En algunas realizaciones, estas acciones pueden comprender, pero sin limitación, una o más de reducir prioridad de la transacción actual; esperar periodo de repliegue; sincronizar usando bloqueos; y así sucesivamente.

En otra realización ejemplar, el siguiente pseudocódigo proporciona una forma de realización de optimización de reinicio de transacción (ya sea de manera transparente en milicode o por código integrado en una aplicación que se ejecuta en el procesador) para reiniciar una transacción cuando se ha abortado, y optimizarse, en particular, para evitar escenarios de bloqueo activo. El código ejemplar usa información almacenada en la TDB de una manera ejemplar; sin embargo, los expertos en la materia podrán usar información obtenida de, incluyendo, pero sin limitación, localizaciones de memoria, registros, códigos de estado, o una unidad de monitorización de rendimiento, unidad de instrumentación de tiempo de ejecución, y así sucesivamente. En el código ejemplar, la transacción toma operaciones de prevención de bloqueo activo cuando corresponde a una transacción que ha superado un umbral de (posiblemente) desconexiones mutuas:

```
IF (esta transacción se aborta) {
   ACCESS TDB information
   IF (esta transacción aborta otra transacción) {
      mutual_shootdown++;
      IF (mutual_shootdown > threshold)
         avoid_livelock(); // tomar acciones para evitar bloqueo activo,
                           //  por  ejemplo,  reducir  prioridad  de  transacción
                           actual;
                           // esperar periodo de repliegue;
                           // sincronizar usando bloqueos;
                           // y así sucesivamente
   }
}
```

De acuerdo con el código ejemplar, el código de reinicio de transacción incluye comprobaciones de si la transacción se ha abortado a sí misma, y a su vez ha abortado otra transacción. En una realización, un posible bloqueo activo se diagnostica inmediatamente. En otra realización, se realiza una prueba (no mostrada) si la transacción actual y la transacción interferida son parte de un gráfico de interferencia cíclica (es decir, cada transacción está abortando directa o indirectamente la otra transacción). En una realización, cuando se han diagnosticado más de un número umbral de desconexiones mutuas (mutual_shootdown > threshold), se toman acciones de evitación de interferencia). En una realización, cuando se han diagnosticado más de un número umbral de desconexiones mutuas (mutual_shootdown > threshold), se toman acciones de evitación de bloqueo activo. En una realización, las acciones de evitación de interferencia se invocan llamando una función avoid_livelock(). En algunas realizaciones, estas acciones pueden comprender, pero sin limitación, una o más de reducir prioridad de la transacción actual; esperar periodo de repliegue; sincronizar usando bloqueos; y así sucesivamente.

Optimización de reinicio (ya sea de manera separada en milicode o en aplicación):

```
IF (esta transacción se aborta) {
   ACCESS TDB information
   IF (esta transacción aborta otra transacción) {
      Mutual_shootdown++;
      If ((mutual_shootdown > threshold) && live_lock_loop_detected())
         avoid_livelock(); // tomar acciones para evitar bloqueo activo,
                           //  por  ejemplo,  reducir  prioridad  de  transacción
                           actual;
                           // esperar periodo de repliegue;
                           // sincronizar usando bloqueos;
                           // y así sucesivamente
   }
}
```

De acuerdo con el código ejemplar, el código de reinicio de transacción incluye comprobar de si la transacción se ha abortado a sí misma, y a su vez ha abortado otra transacción. En una realización, un posible bloqueo activo se diagnostica inmediatamente. En una realización, cuando se ha diagnosticado más de un número umbral de desconexiones mutuas (desconexión mutua > umbral), se realiza una prueba live_lock_loop_detected() que puede acceder a información adicional recibida mediante mensajes de estado de aborto de respuesta opcionalmente en conjunto con otros medios, si la transacción actual y la transacción interferida son - o, en otra realización, pueden ser - parte de un gráfico de interferencia cíclica (es decir, cada transacción está abortando directa o indirectamente la otra transacción). En caso afirmativo, a continuación se toman acciones de evitación de bloqueo activo. En una realización, las acciones de evitación de bloqueo activo se invocan llamando una función avoid_livelock(). En algunas realizaciones, estas acciones pueden comprender, pero sin limitación, una o más de reducir prioridad de la transacción actual; esperar periodo de repliegue; sincronizar usando bloqueos; y así sucesivamente.

De acuerdo con una realización, la Figura 15 es un diagrama 1500 de flujo que ilustra cómo el procesador (por ejemplo, el procesador solicitante CPU 1) responde a la indicación de interferencia, por ejemplo, almacenada (e incrementada/rastreada) en la tabla 1350a de almacenamiento de interferencia de transacción de rastreo local de la información en el campo 1010 de estado de aborto de transacción. El procesador solicitante (CPU 1) informa las estadísticas de interferencia en el bloque 1505. La generación de informe de las estadísticas de interferencia puede incluir escribir la información de transacción como se analiza en el bloque 1405. En el bloque 1510, el procesador solicitante (CPU 1) determina si es posible sincronización adicional para evitar interferencia repetida determinado cuándo el coste de la interferencia es demasiado alto. El coste de interferencia es demasiado alto cuando el procesador solicitante (CPU 1) aborta de manera repetitiva la (misma) transacción que se ejecuta en el procesador de recepción/remoto (CPU 2) un número predefinido de veces (por ejemplo, 4) y/o la transacción ha completado un número predefinido de ciclos de reloj (por ejemplo, 10.000 ciclos de reloj) de instrucciones antes de que se aborte.

Cuando el procesador solicitante (CPU 1) determina que la sincronización adicional no es posible para evitar la interferencia repetida (la transacción repetida se aborta en el procesador remoto (CPU 2) provocado por la solicitud de datos del procesador solicitante (CPU 1) en el bloque 1510, el procesador solicitante (CPU 1) comprueba si es posible la reoptimización de código en el bloque 1515. Cuando el procesador solicitante (CPU 1) determina que es posible la reoptimización, el procesador solicitante reoptimiza el código en el bloque 1520. Cuando el procesador solicitante (CPU 1) determina que no es posible la reoptimización, el procesador solicitante continúa con el código actual (que incluye medidas de toleración tales como repliegue) en el bloque 1525. El repliegue es cuando el procesador solicitante determina esperar una cantidad predefinida de tiempo antes de hacer la solicitud de datos, para que el procesador de recepción/remoto (CPU 2) tenga tiempo para que la transacción del procesador de recepción/remoto complete la ejecución (sin tener que abortar).

Cuando el procesador solicitante (CPU 1) determina que es posible la sincronización adicional para evitar interferencia repetida (abortos de transacción repetidos en el procesador remoto (CPU 2) provocados por la solicitud de datos del procesador solicitante (CPU 1) en el bloque 1510, el procesador solicitante (CPU 1) utiliza sincronización alternativa

en el bloque 1530. El procesador solicitante (CPU 1) comprueba, por ejemplo, el registro en la tabla 1350a de almacenamiento de interferencia de transacción de rastreo local para determinar cuándo la interferencia repetida (de la misma transacción (es decir, que tiene la misma dirección de caché/memoria) ha tenido lugar en el procesador remoto/de recepción (CPU 2) en el bloque 1535. Cuando no hay interferencia repetida, el flujo continúa al bloque 1525. Cuando el procesador solicitante (CPU 1) determina que hay interferencia repetida (para la misma transacción abortada en el procesador remoto/de recepción (CPU 2)), el procesador solicitante comprueba si es posible reoptimización de código en el bloque 1540. Cuando es posible la reoptimización, el procesador solicitante (CPU 1) reoptimiza el código en el bloque 1545.

Cuando el procesador solicitante (CPU 1) determina que no es posible la reoptimización de código, el procesador solicitante (CPU 1) está configurado para elegir permanentemente (y/o durante un periodo definido) el método de sincronización alternativa para esta transacción particular (que se ejecuta en el procesador solicitante (CPU 1)) que solicita datos del procesador de recepción/remoto (CPU 2) en el bloque 1550.

De acuerdo con una realización, la Figura 16 es un diagrama 1600 de flujo que ilustra cómo el procesador (por ejemplo, el procesador solicitante CPU 1) responde a la indicación de interferencia, por ejemplo, almacenada (e incrementada/rastreada) en la tabla 1350a de almacenamiento de interferencia de transacción de rastreo local de la información en el campo 1010 de estado de aborto de transacción. La Figura 16 incluye los bloques de la Figura 15, excepto que el bloque 1605 sustituye el bloque 1540. Los bloques de la Figura 15 no están repetidos.

Cuando el bloque 1535 es SÍ, el flujo continúa al bloque 1605 en la Figura 16. En el bloque 1605, el procesador solicitante (CPU 1) comprueba si se prefiere reoptimización o sincronización alternativa. Cuando la determinación en el bloque 1605 es SÍ, el flujo continúa al bloque 1545. Cuando la determinación en el bloque 1605 es NO, el flujo continúa al bloque 1550. En el bloque 1540 de la Figura 15, cuando es posible reoptimización de código, siempre se realiza reoptimización de código. En 1605, se calcula una métrica para determinar si se desea reoptimización de código o un método de sincronización alternativo (por ejemplo, bloqueos), y se selecciona uno o el otro. Esto puede estar basado, por ejemplo, en una prueba que compara el coste agregado de uso de unos métodos de sincronización alternativos con el coste de uso de coste reoptimizado más el coste de realizar reoptimización, para determinar cuál se desea. Son posibles otras métricas de coste y se contemplan por la presente divulgación, tales como minimizar el coste de carga de reoptimización, por ejemplo, comparando el coste de reoptimización con un umbral.

La Figura 17 es un diagrama 1700 de flujo de un método (ejecutado por los procesadores 112a, 112b) para implementar un protocolo de coherencia de acuerdo con una realización.

En el bloque 1705, el procesador 112a solicitante (CPU 1) envía una solicitud (tal como las solicitudes 505, 605) de datos al procesador 112b remoto (CPU2) mediante la interconexión 122. En el bloque 1710, el procesador 112a solicitante (CPU 1) recibe una respuesta (tal como las respuestas 1005, 1105) del procesador 112b remoto, en el que la respuesta incluye el estado de transacción (por ejemplo, estado 1010 de aborto de transacción) de una transacción remota (por ejemplo, la transacción 320b) en el procesador 112b remoto. En el bloque 1715, el procesador 112a solicitante añade el estado de transacción (información del campo 1010) de la transacción remota en el procesador remoto en una tabla de rastreo de interferencia de transacción local (por ejemplo, la tabla 1350a).

Además de una o más de las características anteriormente descritas, o como una alternativa, realizaciones adicionales pueden incluir que se añada el estado de transacción de la transacción a un bloque de diagnóstico de transacción (TBD) (por el procesador 112a solicitante). La transacción remota se ejecuta en el procesador 112b remoto y aborta la ejecución basándose en el envío de la solicitud de datos al procesador remoto. La solicitud es mediante una transacción solicitante (por ejemplo, la transacción 320a) que se ejecuta en el procesador 112a solicitante que envía la solicitud.

Además de una o más de las características anteriormente descritas, o como una alternativa, realizaciones adicionales pueden incluir que estén basadas en la solicitud por la transacción solicitante que provoca que la transacción remota (por ejemplo, la transacción 320b) se aborte en el procesador 112b remoto, el procesador 112a solicitante añade el estado de transacción de la transacción remota (obtenido del campo 1010) a la tabla 1350a de rastreo de interferencia de transacción local e incrementa un contador de abortos de transacción que han tenido lugar para la transacción remota (cada transacción 320b particular que se aborta).

Además de una o más de las características anteriormente descritas, o como una alternativa, realizaciones adicionales pueden incluir que el estado de transacción de la transacción remota, recibida por el procesador 112a solicitante en la respuesta 1005, 1105 del procesador 112b remoto, indique que la transacción remota (la transacción 320b) tenía que haberse abortado basándose en la recepción de la solicitud 505, 605 del procesador 112a solicitante.

Además de una o más de las características anteriormente descritas, o como una alternativa, realizaciones adicionales pueden incluir que en la tabla 1350a de rastreo de interferencia de transacción local se mantenga un número de transacciones que han sido interferidas con y abortadas por la transacción 320a solicitante que se ejecuta en el procesador 112a solicitante. La tabla 1350a de rastreo de interferencia de transacción local mantiene información que describe transacciones 320b remotas en procesadores 112b remotos (y otros procesadores). La información que

describe las transacciones 320b remotas en los procesadores 112a remotos incluye al menos una de: un tipo de interferencia provocada por la transacción solicitante que se ejecuta en el procesador, una identificación o dirección de cada una de las transacciones remotas que se abortaron por la transacción solicitante, una identificación de cada uno de los procesadores remotos en los que tuvo lugar la interferencia, una dirección de cada una de las transacciones remotas que se han abortado, y/o una medida de trabajo que se ha realizado por cada una de las transacciones remotas antes de que se abortaran.

Los efectos y beneficios técnicos incluyen un protocolo de coherencia extendido para incluir información adicional acerca del estado de transacción. Cuando un procesador está en ejecución transaccional, una solicitud de coherencia puede provocar que se aborte la ejecución de transacción del procesador de recepción debido a que se detecta un conflicto. La solicitud de protocolo de coherencia se amplía con información adicional que el procesador de recepción ha abortado su ejecución transaccional de acuerdo con las realizaciones.

La terminología usada en el presente documento es para el fin de describir realizaciones particulares únicamente y no se pretende que sea para limitar la invención. Como se usa en el presente documento, las formas singulares "un", "una" y "el", "la" se pretende que incluyan las formas plurales también, a menos que el contexto lo indique claramente de otra manera. Se entenderá adicionalmente que los términos "comprende" y/o "que comprende" cuando se usan en esta memoria descriptiva, especifican la presencia de características indicadas, enteros, etapas, operaciones, elementos, y/o componentes, pero no excluyen la presencia o adición de una o más otras características, elemento integrantes, etapas, operaciones, elemento, componentes, y/o grupos de los mismos.

La descripción de la presente invención se ha presentado para fines de ilustración y descripción, pero no se pretende que sea exhaustiva o esté limitada a la invención en la forma desvelada. Serán evidentes muchas modificaciones y variaciones para los expertos en la materia sin alejarse del alcance de la invención. La realización se eligió y describió para explicar mejor los principios de la invención y la aplicación práctica, y para posibilitar a los expertos en la materia entender la invención como se define en las reivindicaciones adjuntas.

Haciendo referencia ahora a la Figura 18, se muestra en general un producto 1800 de programa informático de acuerdo con una realización que incluye un medio 1802 de almacenamiento legible por ordenador e instrucciones 1804 de programa.

La presente invención puede ser un sistema, un método, y/o un producto de programa informático. El producto de programa informático puede incluir un medio de almacenamiento legible por ordenador (o medios) que tiene instrucciones de programa legibles por ordenador en el mismo para provocar que un procesador lleve a cabo aspectos de la presente invención.

El medio de almacenamiento legible por ordenador puede ser un dispositivo tangible que puede mantener y almacenar instrucciones para su uso por un dispositivo de ejecución de instrucciones. El medio de almacenamiento legible por ordenador puede ser, por ejemplo, pero sin limitación, un dispositivo de almacenamiento electrónico, un dispositivo de almacenamiento magnético, un dispositivo de almacenamiento óptico, un dispositivo de almacenamiento electromagnético, un dispositivo de almacenamiento de semiconductores, o cualquier combinación adecuada de lo anterior. Una lista no exhaustiva de ejemplos más específicos del medio de almacenamiento legible por ordenador incluye lo siguiente: un disquete de ordenador portátil, un disco duro, una memoria de acceso aleatorio (RAM), una memoria de solo lectura (ROM), una memoria de solo lectura programable borrable (EPROM o memoria Flash), una memoria de acceso aleatorio estática (SRAM), una memoria de solo lectura de disco compacto portátil (CD-ROM), un disco versátil digital (DVD), un lápiz de memoria, un disco flexible, un dispositivo codificado mecánicamente tal como tarjetas de perforación o estructuras elevadas en un surco que tienen instrucciones grabadas en las mismas, y cualquier combinación adecuada de lo anterior. Un medio de almacenamiento legible por ordenador, como se usa en el presente documento, no ha de interpretarse como que son señales transitorias *per se,* tales como ondas de radio u otras ondas electromagnéticas que se propagan libremente, ondas electromagnéticas que se propagan a través de una guía de onda u otro medio de transmisión (por ejemplo, pulsos de luz que pasan a través de un cable de fibra óptica), o señales eléctricas transmitidas a través de un alambre.

Las instrucciones de programa legibles por ordenador descritas en el presente documento pueden descargarse a respectivos dispositivos informáticos/de procesamiento desde un medio de almacenamiento legible por ordenador o a un ordenador externo o dispositivo de almacenamiento externo mediante una red, por ejemplo, Internet, una red de área local, una red de área extensa y/o una red inalámbrica. La red puede comprender cables de transmisión de cobre, fibras de transmisión óptica, transmisión inalámbrica, encaminadores, cortafuegos, conmutadores, ordenadores de pasarela y/o servidores de borde. Una tarjeta adaptadora de red o interfaz de red en cada dispositivo informático/de procesamiento recibe instrucciones de programa legibles por ordenador de la red y reenvía las instrucciones de programa legibles por ordenador para su almacenamiento en un medio de almacenamiento legible por ordenador dentro del respectivo dispositivo informático/de procesamiento.

Las instrucciones de programa legibles por ordenador para llevar a cabo las operaciones de la presente invención pueden ser instrucciones de ensamblador, instrucciones de la arquitectura de conjunto de instrucciones (ISA), instrucciones de máquina, instrucciones dependientes de máquina, microcódigo, instrucciones de firmware, datos de

ajuste de estado, o cualquier código fuente o código objeto escrito en cualquier combinación de uno o más lenguajes de programación, incluyendo un lenguaje de programación orientado a objetos tal como Smalltalk, C++ o similares, y lenguajes de programación procedurales convencionales, tal como el lenguaje de programación "C" o lenguajes de programación similares. Las instrucciones de programa legibles por ordenador pueden ejecutarse completamente en el ordenador del usuario, parcialmente en el ordenador del usuario, como un paquete de software independiente, parcialmente en el ordenador del usuario y parcialmente en un ordenador remoto o completamente en el ordenador remoto o servidor. En el último escenario, el ordenador remoto puede estar conectado al ordenador del usuario a través de un tipo de red, que incluye una red de área local (LAN) o una red de área extensa (WAN), o la conexión puede hacerse a un ordenador externo (por ejemplo, a través de Internet usando un Proveedor de Servicio de Internet). En algunas realizaciones, la circuitería electrónica que incluye, por ejemplo, circuitería de lógica programable, campos de matrices de puertas programables (FPGA), o matrices de lógica programable (PLA) puede ejecutar las instrucciones de programa legibles por ordenador utilizando información de estado de las instrucciones de programa legibles por ordenador para personalizar la circuitería electrónica, para realizar aspectos de la presente invención.

Los aspectos de la presente invención se describen en el presente documento con referencia a ilustraciones de diagrama de flujo y/o diagramas de bloques de los métodos, aparato (sistemas), y productos de programa informático de acuerdo con las realizaciones de la invención. Se entenderá que cada bloque de las ilustraciones de diagrama de flujo y/o diagramas de bloques, y combinaciones de bloques en las ilustraciones de diagrama de flujo y/o diagramas de bloques, pueden implementarse por instrucciones de programa legibles por ordenador.

Estas instrucciones de programa legibles por ordenador pueden proporcionarse a un procesador de un ordenador de fin general, ordenador de fin especial, u otro aparato de procesamiento de datos programable para producir una máquina, de manera que las instrucciones, que se ejecutan mediante el procesador del ordenador u otro aparato de procesamiento de datos programable, crean medios para implementar las funciones/actos especificados en el diagrama de flujo y/o bloque o bloques del diagrama de bloques. Estas instrucciones de programa legibles por ordenador pueden almacenarse también en un medio de almacenamiento legible por ordenador que puede dirigir un ordenador, un aparato de procesamiento de datos programable, y/u otros dispositivos para funcionar de una manera particular, de manera que el medio de almacenamiento legible por ordenador que tiene instrucciones almacenadas en el mismo comprende un artículo de fabricación que incluye instrucciones que implementan aspectos de la función/acto especificado en el diagrama de flujo y/o bloque o bloques del diagrama de bloques.

Las instrucciones de programa legibles por ordenador pueden cargarse también en un ordenador, otro aparato de procesamiento de datos programable, u otro dispositivo para provocar que se realice una serie de etapas operacionales en el ordenador, otro aparato programable u otro dispositivo para producir un proceso implementado por ordenador, de manera que las instrucciones que se ejecutan en el ordenador, otro aparato programable, u otro dispositivo implementan las funciones/actos especificados en el diagrama de flujo y/o bloque o bloques de diagrama de bloques.

El diagrama de flujo y diagramas de bloques en las figuras ilustran la arquitectura, funcionalidad, y operación de posibles implementaciones de sistemas, métodos, y productos de programa informático de acuerdo con diversas realizaciones de la presente invención. En este sentido, cada bloque en el diagrama de flujo o diagramas de bloques puede representar un módulo, segmento, o porción de instrucciones, que comprende una o más instrucciones ejecutables para implementar la función o funciones lógicas especificadas. En algunas implementaciones alternativas, las funciones indicadas en el bloque pueden tener lugar fuera del orden indicado en las figuras. Por ejemplo, dos bloques mostrados en serie, de hecho, pueden ejecutarse sustancialmente de manera concurrente, o los bloques pueden ejecutarse en ocasiones en el orden inverso, dependiendo de la funcionalidad implicada. Se observará también que cada bloque de los diagramas de bloques y/o ilustración de diagrama de flujo y combinaciones de bloques en los diagramas de bloques y/o ilustración de diagrama de flujo, puede implementarse por sistemas basados en hardware de fin especial que realizan las funciones especificadas o actos o llevan a cabo combinaciones de hardware de fin especial e instrucciones informáticas.

REIVINDICACIONES

1. Un método implementado por ordenador para implementar un protocolo de coherencia, comprendiendo el método:

enviar (1705), por un procesador (112a) solicitante, una solicitud de datos a un procesador remoto, siendo dicha solicitud por una transacción solicitante que se ejecuta en el procesador (112a) solicitante que envía la solicitud;
recibir (1710), por el procesador solicitante, una respuesta del procesador remoto, incluyendo la respuesta un estado de transacción de una transacción remota en el procesador remoto, en el que el estado de transacción recibido en la respuesta del procesador remoto incluye: un tipo de interferencia en el procesador remoto provocada por la transacción solicitante que se ejecuta en el procesador solicitante, un número de ciclos de reloj de trabajo que se han realizado por la transacción remota antes de que se aborte en el procesador remoto, o una indicación de si se provocó una restauración en el procesador remoto enviando la solicitud al procesador remoto; y
añadir (1715), por el procesador solicitante, el estado de transacción de la transacción remota en el procesador remoto en una tabla (1350a) de rastreo de interferencia de transacción;
en el que el procesador solicitante es un procesador separado del procesador remoto.

2. El método de la reivindicación 1, en el que el estado de transacción de la transacción remota se añade a un bloque de diagnóstico de transacción.

3. El método de la reivindicación 1, en el que la transacción remota se ejecuta en el procesador remoto y aborta la ejecución basándose en el envío de la solicitud de datos al procesador remoto.

4. El método de la reivindicación 1, en el que la solicitud es mediante una transacción solicitante que se ejecuta en el procesador solicitante que envía la solicitud.

5. El método de la reivindicación 1, en el que basándose en la solicitud por la transacción solicitante que provoca que se aborte la transacción remota en el procesador remoto, el procesador solicitante añade el estado de transacción de la transacción remota a la tabla de rastreo de interferencia de transacción e incrementa un contador de abortos de transacción que han tenido lugar para la transacción remota.

6. El método de la reivindicación 1, en el que el estado de transacción de la transacción remota, recibido por el procesador solicitante en la respuesta del procesador remoto, indica que la transacción remota tuvo que abortarse basándose en recibir la solicitud del procesador solicitante.

7. El método de la reivindicación 1, en el que la tabla de rastreo de interferencia de transacción local mantiene un número de transacciones que se han interferido con y abortado por la transacción solicitante que se ejecuta en el procesador solicitante.

8. El método de la reivindicación 1, en el que la tabla de rastreo de interferencia de transacción mantiene información que describe transacciones remotas en procesadores remotos; y
en el que la información que describe las transacciones remotas en los procesadores remotos incluye: una identificación de cada uno de los procesadores remotos en los que ha tenido lugar la interferencia; y una dirección de cada una de las transacciones remotas que se han abortado.

9. Un programa informático que comprende medios de código informático adaptados para realizar un método de acuerdo con cualquier reivindicación anterior cuando dicho programa se ejecuta en un ordenador.

10. Un sistema informático para implementar un protocolo de coherencia, comprendiendo el sistema:

una memoria; y
un procesador, acoplado de manera comunicativa a dicha memoria, el sistema informático configurado para realizar un método de acuerdo con cualquiera de las reivindicaciones 1 a 8.

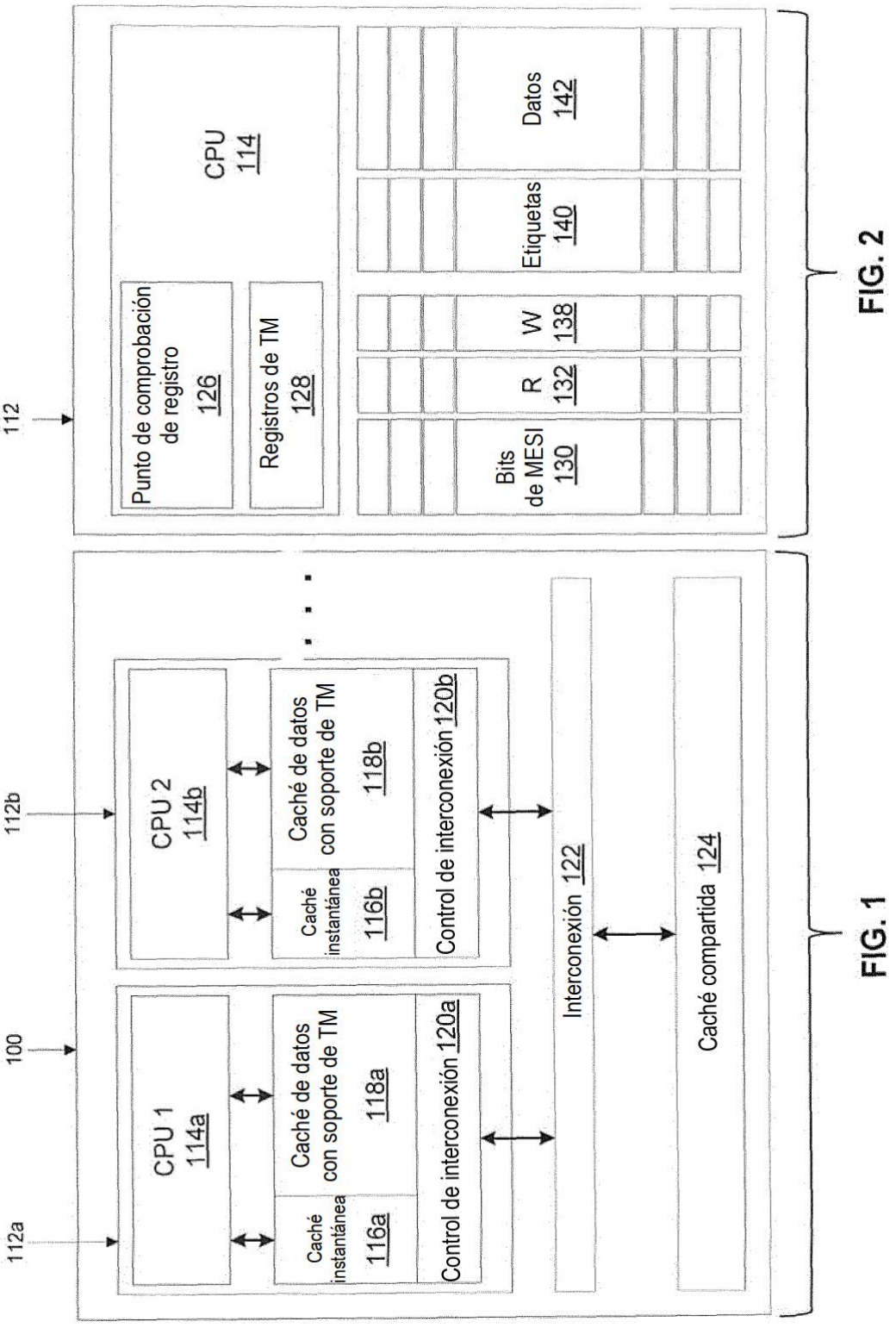

112
Punto de comprobación de registro 126
Registros de TM 128
CPU 114
Bits de MESI 130
R 132
W 138
Etiquetas 140
Datos 142
100
112a
CPU 1 114a
Caché instantánea 116a
Caché de datos con soporte de TM 118a
Control de interconexión 120a
112b
CPU 2 114b
Caché instantánea 116b
Caché de datos con soporte de TM 118b
Control de interconexión 120b
Interconexión 122
Caché compartida 124

FIG. 2

FIG. 1

FIG. 3

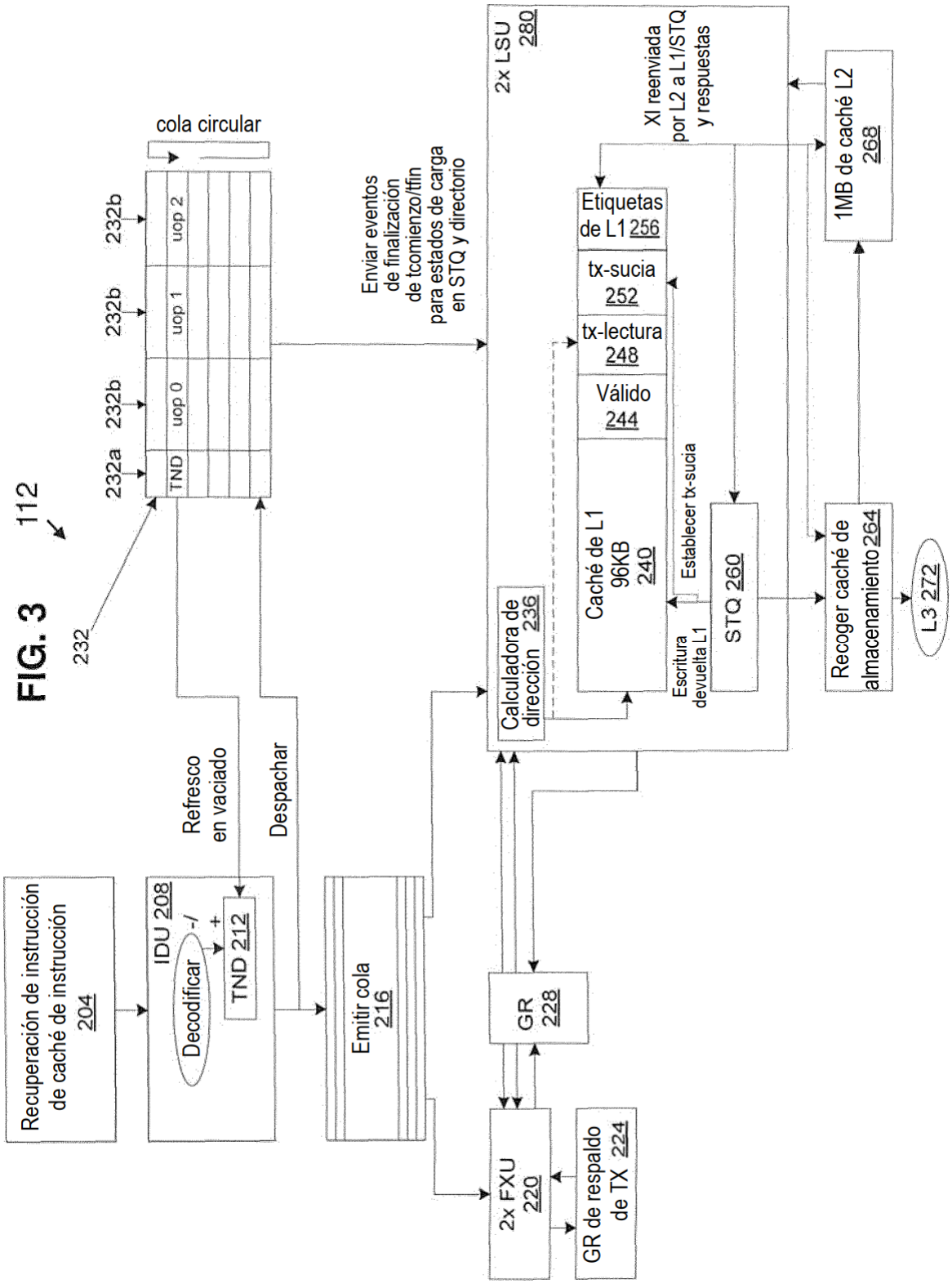

112
232
cola circular
232a
232b
232b
232b
TND
uop 0
uop 1
uop 2
Enviar eventos de finalización de tcomienzo/tfin para estados de carga en STQ y directorio
2x LSU 280
XI reenviada por L2 a L1/STQ y respuestas
1MB de caché L2 268
Etiquetas de L1 256
tx-sucia 252
tx-lectura 248
Válido 244
Caché de L1 96KB 240
Establecer tx-sucia
Escritura devuelta L1
STQ 260
Recoger caché de almacenamiento 264
L3 272
Calculadora de dirección 236
Refresco en vaciado
Despachar
Recuperación de instrucción de caché de instrucción 204
IDU 208
Decodificar
-/+
TND 212
Emitir cola 216
GR 228
2x FXU 220
GR de respaldo de TX 224

FIG. 4

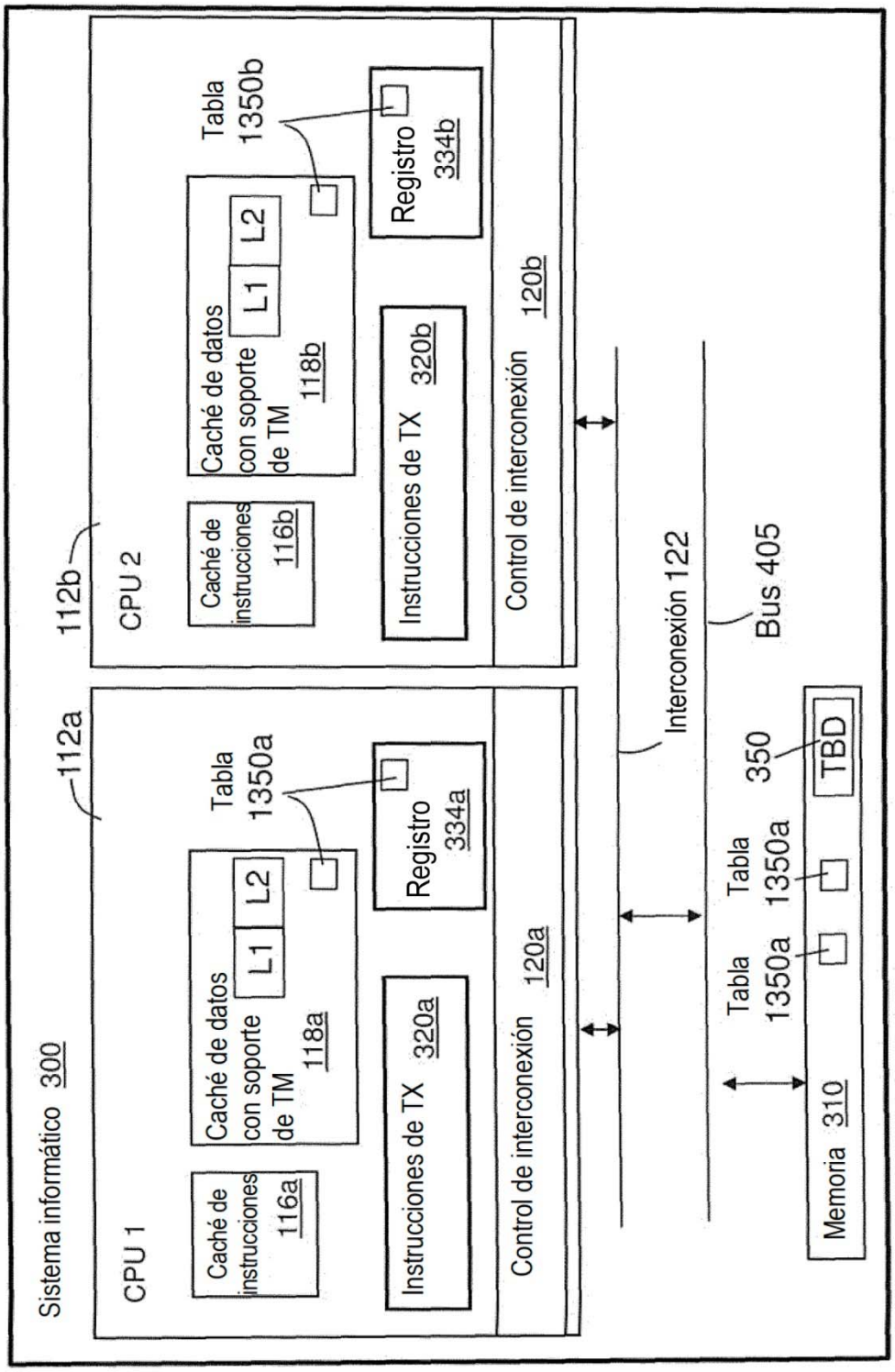

Sistema informático 300
CPU 1
112a
Caché de instrucciones 116a
Caché de datos con soporte de TM 118a
L1
L2
Tabla 1350a
Registro 334a
Instrucciones de TX 320a
Control de interconexión 120a
CPU 2
112b
Caché de instrucciones 116b
Caché de datos con soporte de TM 118b
L1
L2
Tabla 1350b
Registro 334b
Instrucciones de TX 320b
Control de interconexión 120b
Interconexión 122
Bus 405
350
Tabla 1350a
Tabla 1350a
TBD
Memoria 310

FIG. 5

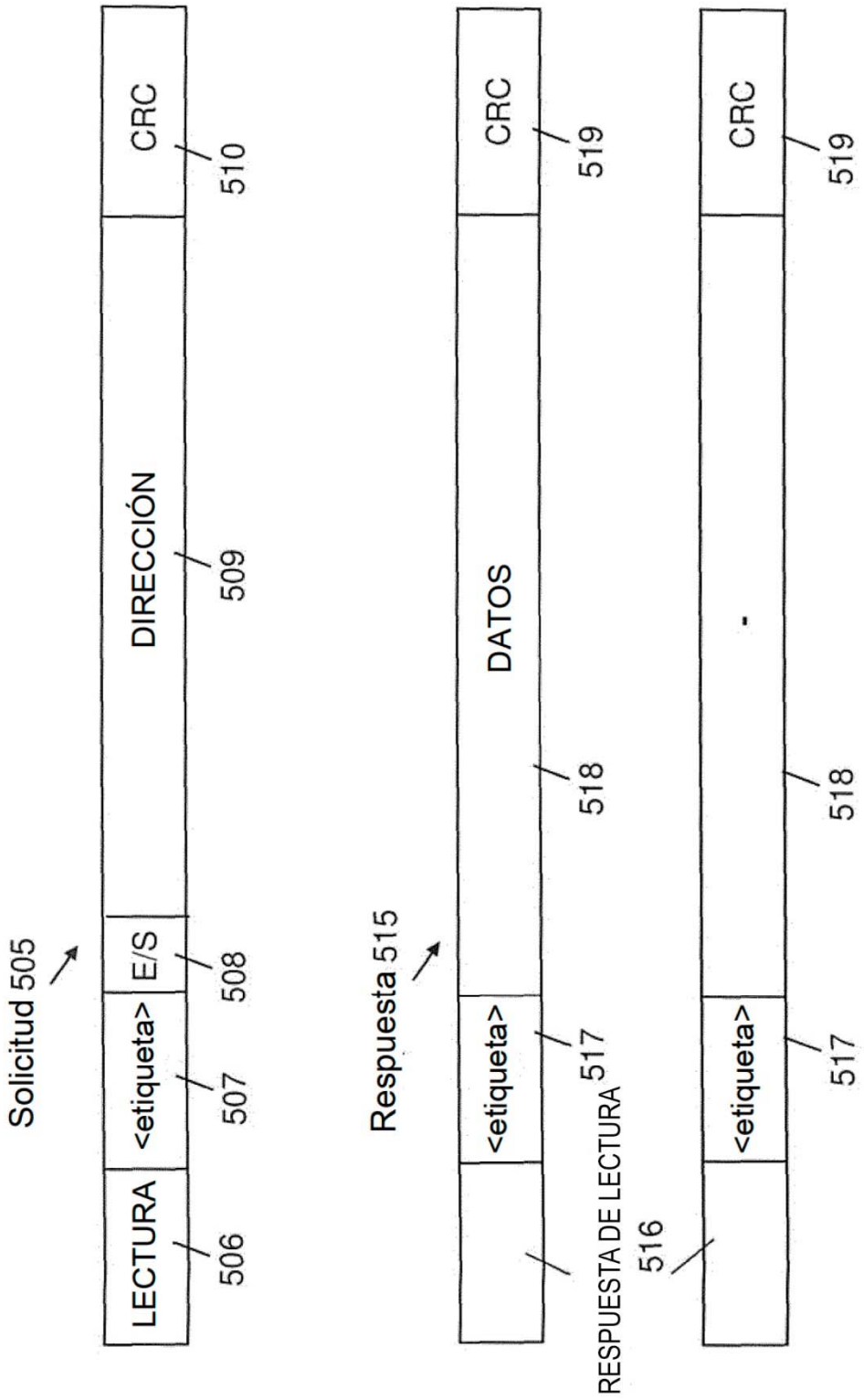

Solicitud 505
LECTURA
<etiqueta>
E/S
DIRECCIÓN
CRC
506
507
508
509
510
Respuesta 515
<etiqueta>
DATOS
CRC
517
518
519
RESPUESTA DE LECTURA 516
<etiqueta>
-
CRC
517
518
519

FIG. 6

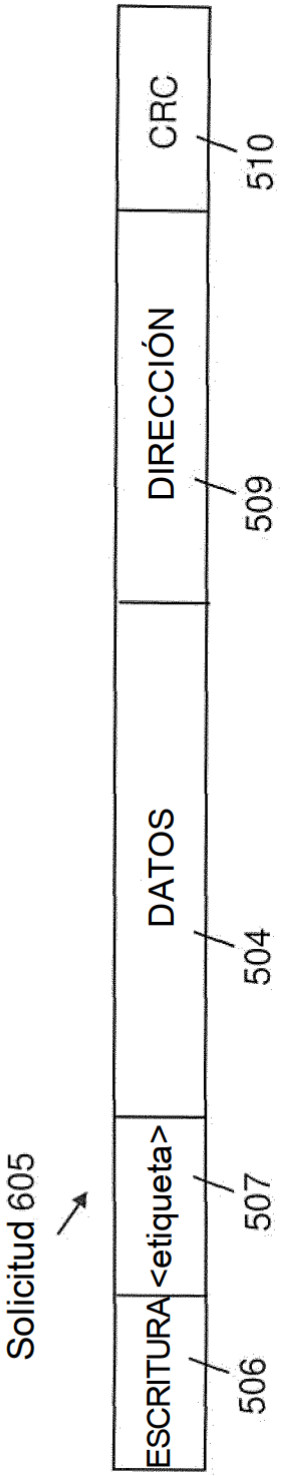

Solicitud 605
ESCRITURA
<etiqueta>
DATOS
DIRECCIÓN
CRC
506
507
504
509
510

FIG. 7

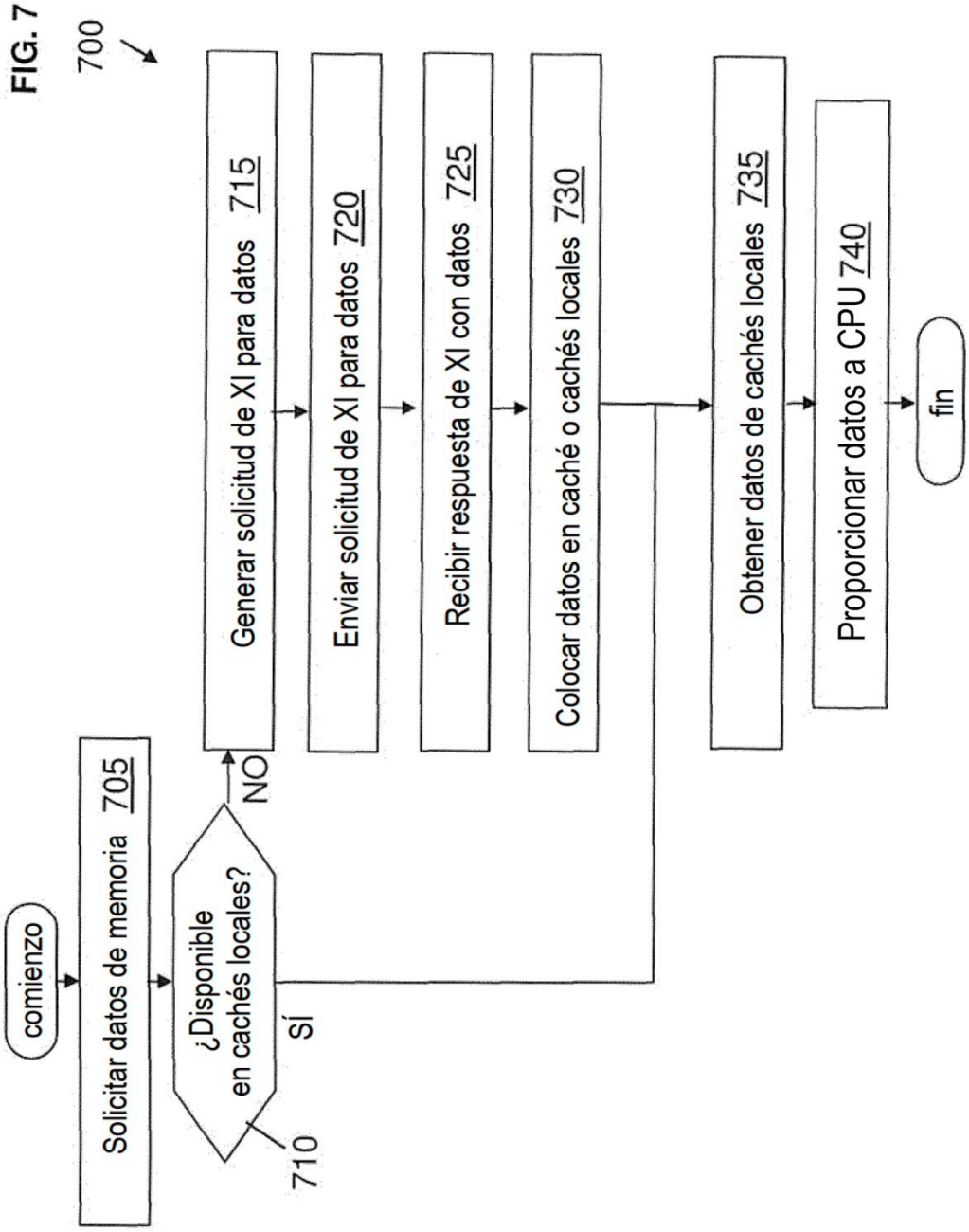

700
comienzo
Solicitar datos de memoria 705
¿Disponible en cachés locales? 710
SÍ
NO
Generar solicitud de XI para datos 715
Enviar solicitud de XI para datos 720
Recibir respuesta de XI con datos 725
Colocar datos en caché o cachés locales 730
Obtener datos de cachés locales 735
Proporcionar datos a CPU 740
fin

FIG. 8

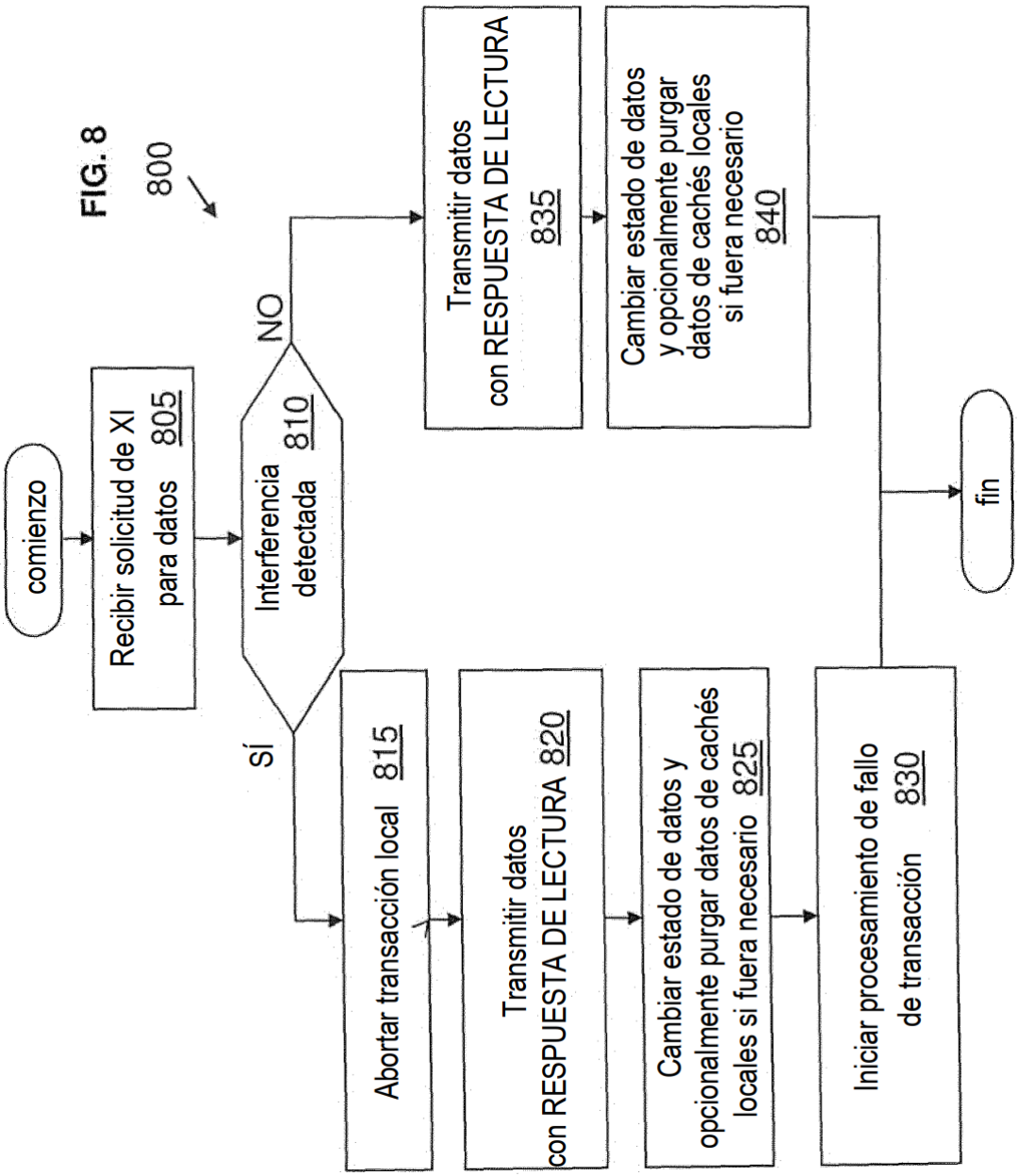

800
comienzo
Recibir solicitud de XI para datos 805
Interferencia detectada 810
NO
SÍ
Transmitir datos con RESPUESTA DE LECTURA 835
Cambiar estado de datos y opcionalmente purgar datos de cachés locales si fuera necesario 840
Abortar transacción local 815
Transmitir datos con RESPUESTA DE LECTURA 820
Cambiar estado de datos y opcionalmente purgar datos de cachés locales si fuera necesario 825
Iniciar procesamiento de fallo de transacción 830
fin

FIG. 9

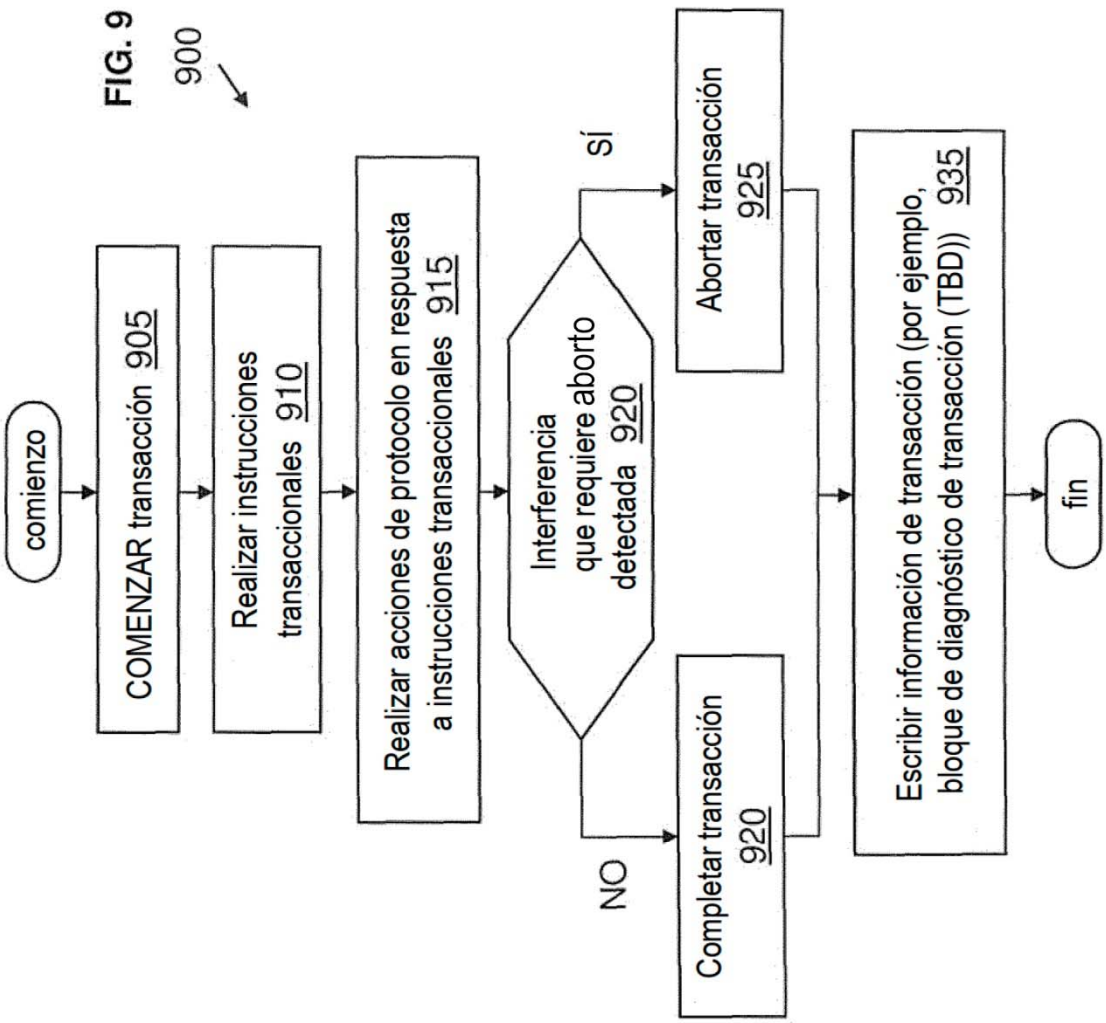

900
comienzo
COMENZAR transacción 905
Realizar instrucciones transaccionales 910
Realizar acciones de protocolo en respuesta a instrucciones transaccionales 915
Interferencia que requiere aborto detectada 920
SÍ
Abortar transacción 925
NO
Completar transacción 920
Escribir información de transacción (por ejemplo, bloque de diagnóstico de transacción (TBD)) 935
fin

FIG. 10

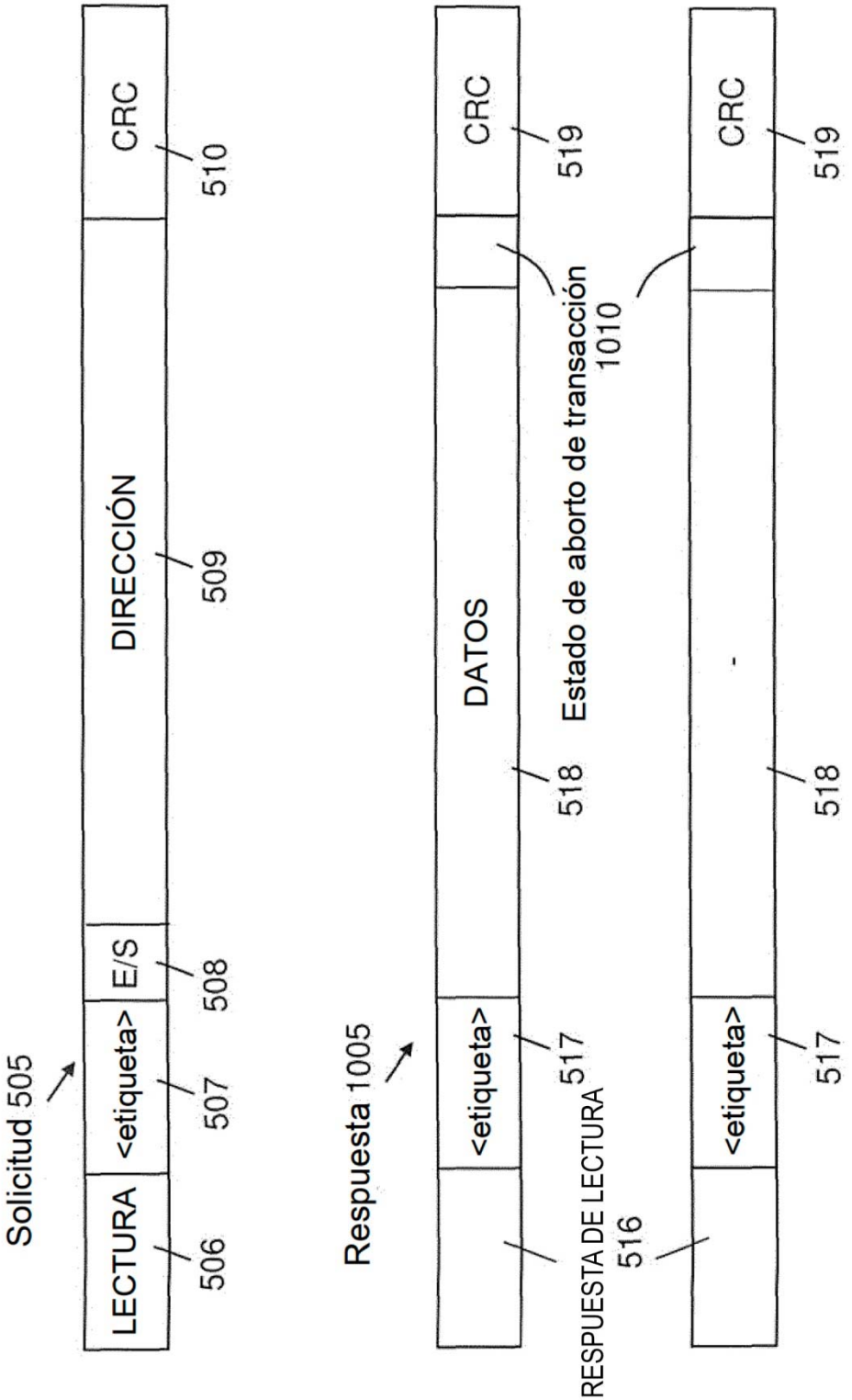

Solicitud 505
LECTURA
<etiqueta>
E/S
DIRECCIÓN
CRC
506
507
508
509
510
Respuesta 1005
RESPUESTA DE LECTURA
516
<etiqueta>
DATOS
Estado de aborto de transacción
1010
CRC
517
518
519
<etiqueta>
-
CRC
517
518
519

FIG. 11

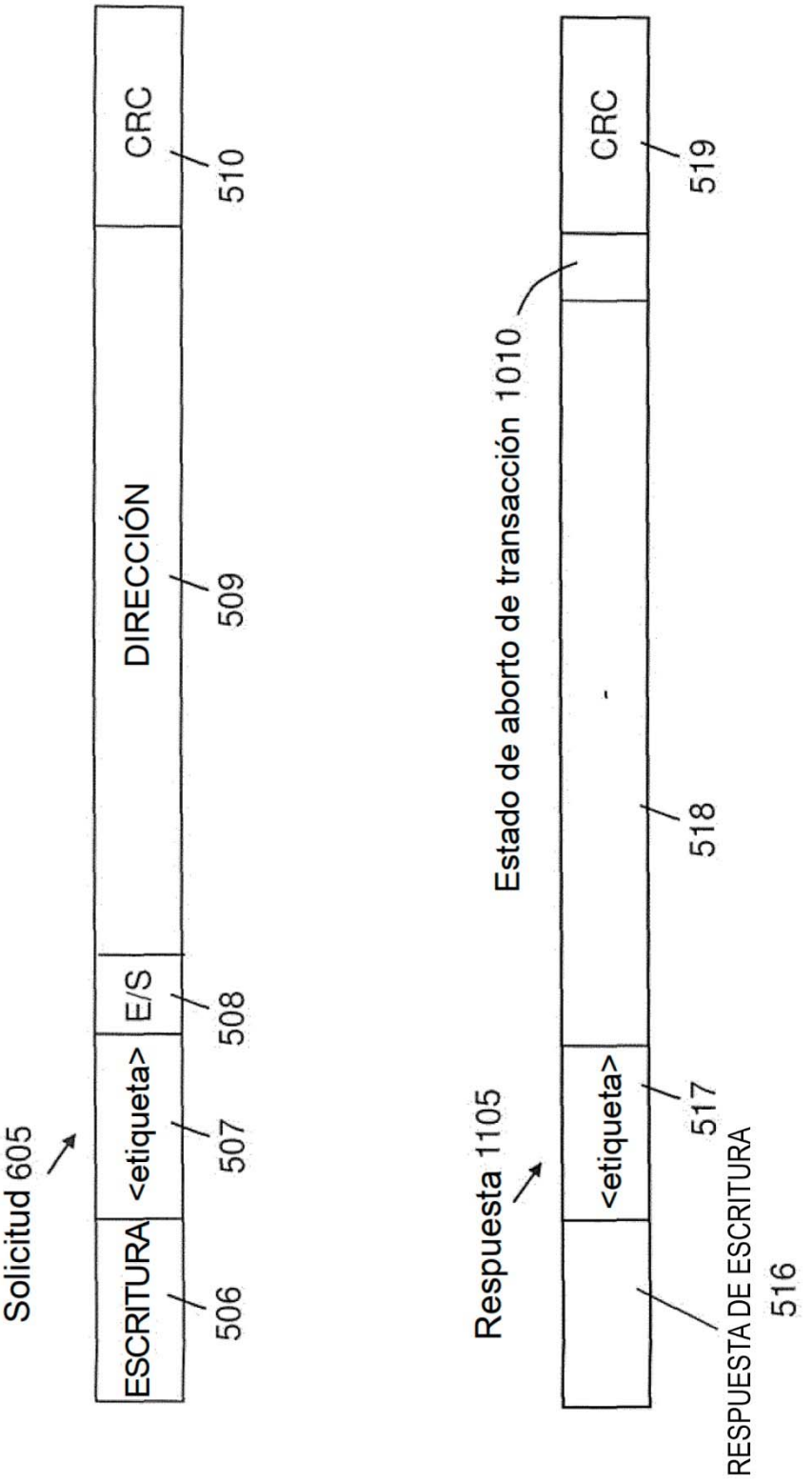

Solicitud 605
ESCRITURA
<etiqueta>
E/S
DIRECCIÓN
CRC
506
507
508
509
510
Respuesta 1105
<etiqueta>
Estado de aborto de transacción 1010
-
CRC
517
518
519
RESPUESTA DE ESCRITURA
516

FIG. 12

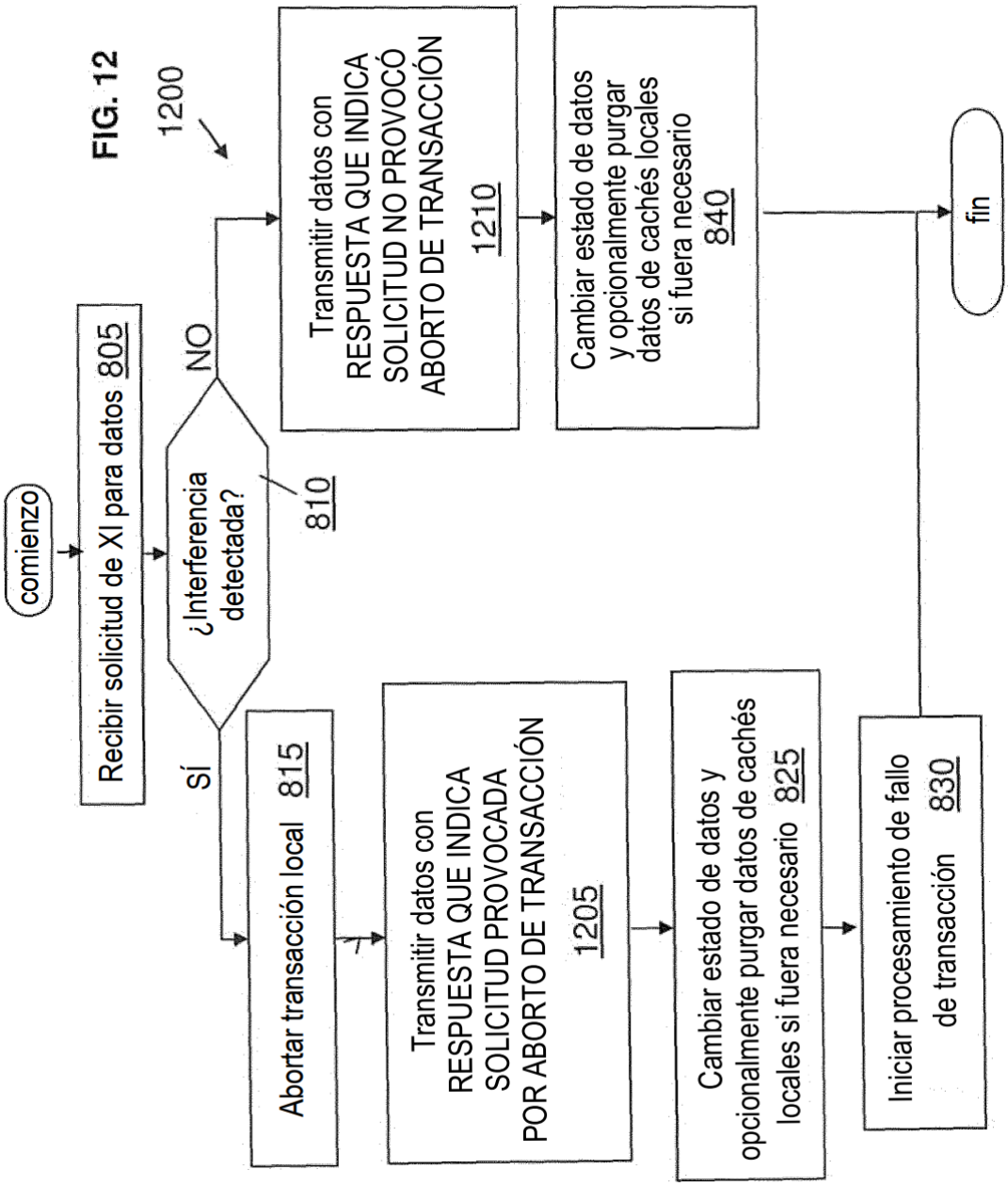

1200
comienzo
Recibir solicitud de XI para datos 805
¿Interferencia detectada? 810
NO
SÍ
Transmitir datos con RESPUESTA QUE INDICA SOLICITUD NO PROVOCÓ ABORTO DE TRANSACCIÓN 1210
Cambiar estado de datos y opcionalmente purgar datos de cachés locales si fuera necesario 840
fin
Abortar transacción local 815
Transmitir datos con RESPUESTA QUE INDICA SOLICITUD PROVOCADA POR ABORTO DE TRANSACCIÓN 1205
Cambiar estado de datos y opcionalmente purgar datos de cachés locales si fuera necesario 825
Iniciar procesamiento de fallo de transacción 830

FIG. 13

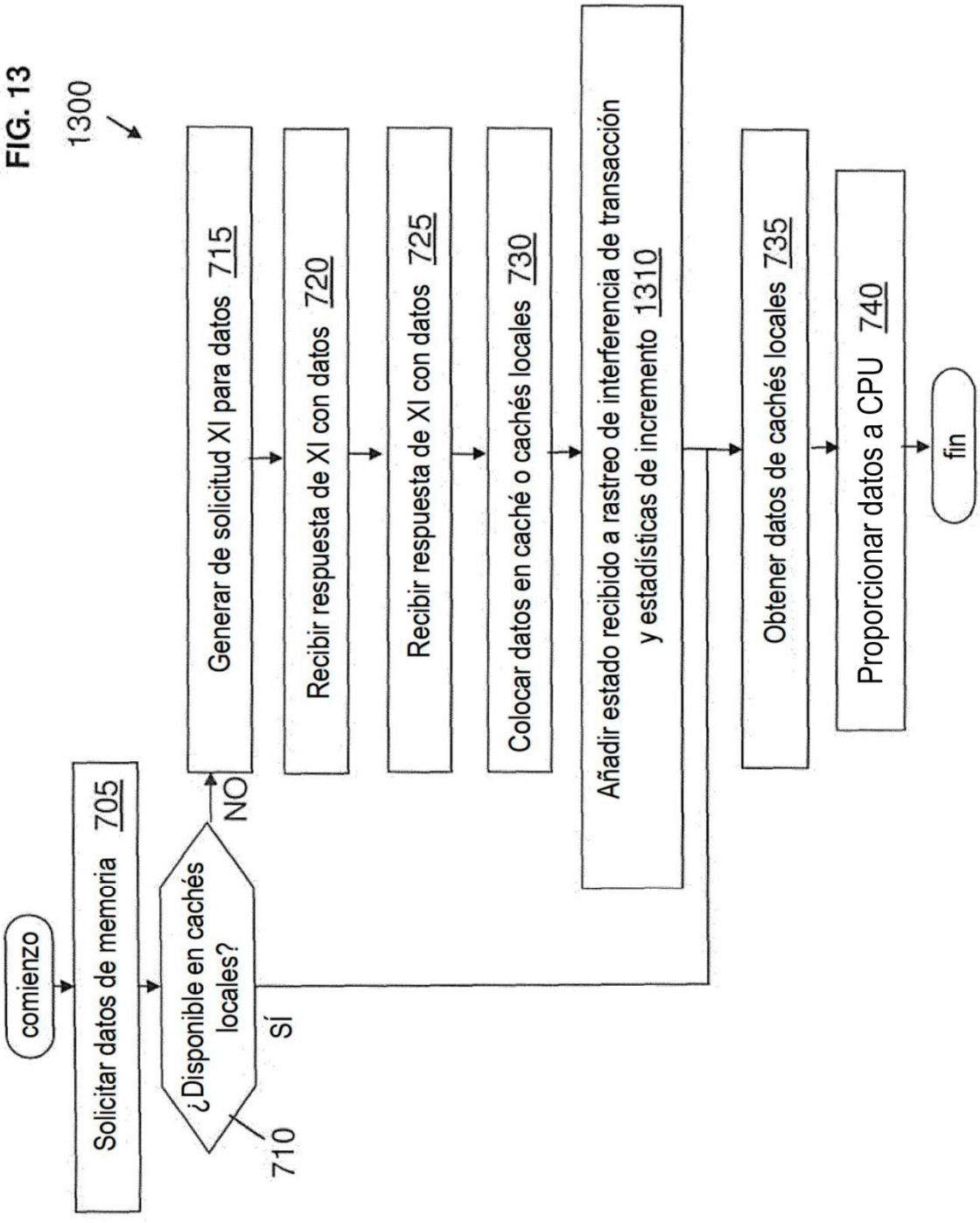
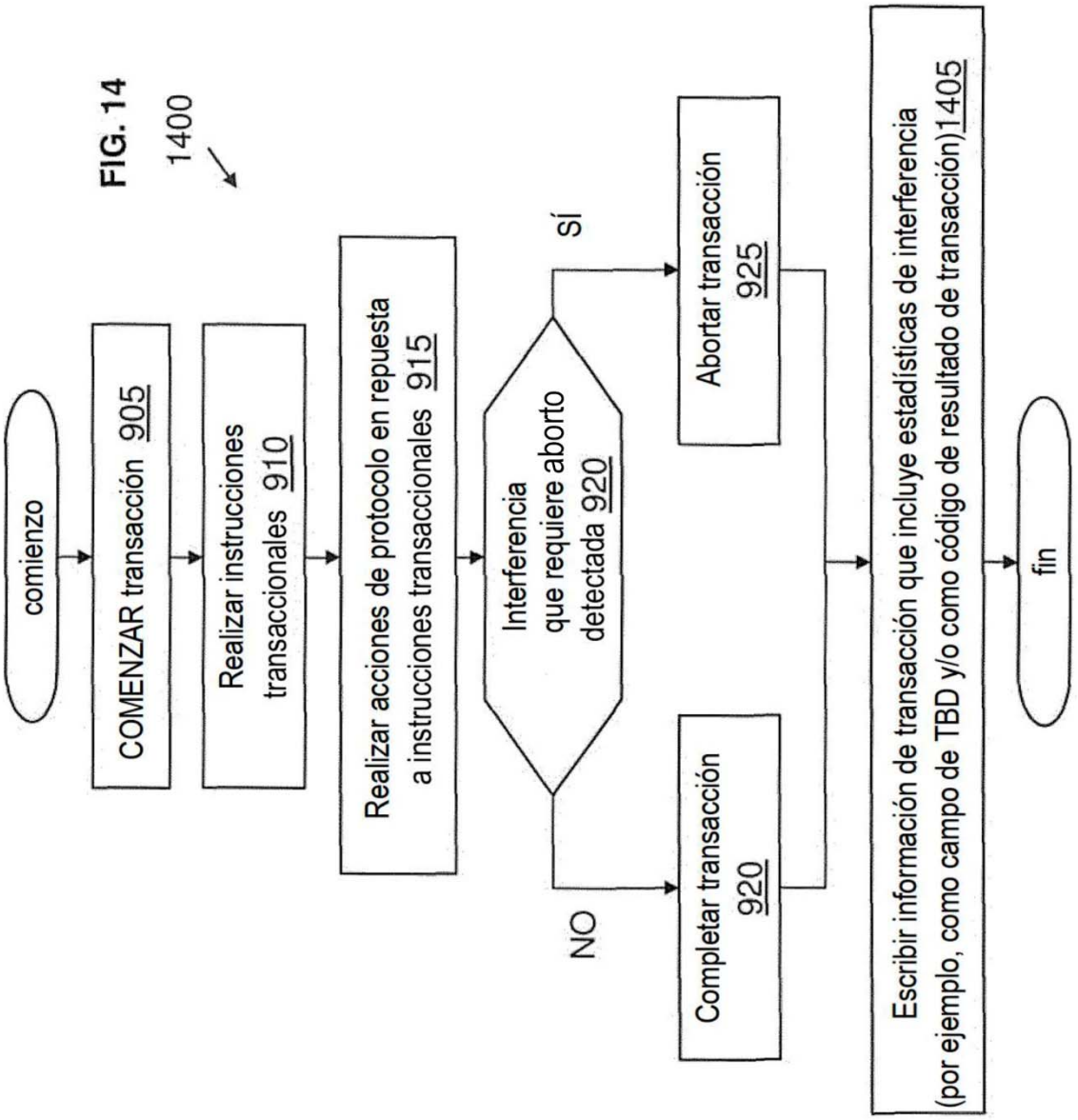

1300
comienzo
Solicitar datos de memoria 705
¿Disponible en cachés locales?
710
NO
SÍ
Generar de solicitud XI para datos 715
Recibir respuesta de XI con datos 720
Recibir respuesta de XI con datos 725
Colocar datos en caché o cachés locales 730
Añadir estado recibido a rastreo de interferencia de transacción y estadísticas de incremento 1310
Obtener datos de cachés locales 735
Proporcionar datos a CPU 740
fin

FIG. 14


1400
comienzo
COMENZAR transacción 905
Realizar instrucciones transaccionales 910
Realizar acciones de protocolo en repuesta a instrucciones transaccionales 915
Interferencia que requiere aborto detectada 920
SÍ
NO
Abortar transacción 925
Completar transacción 920
Escribir información de transacción que incluye estadísticas de interferencia (por ejemplo, como campo de TBD y/o como código de resultado de transacción)1405
fin

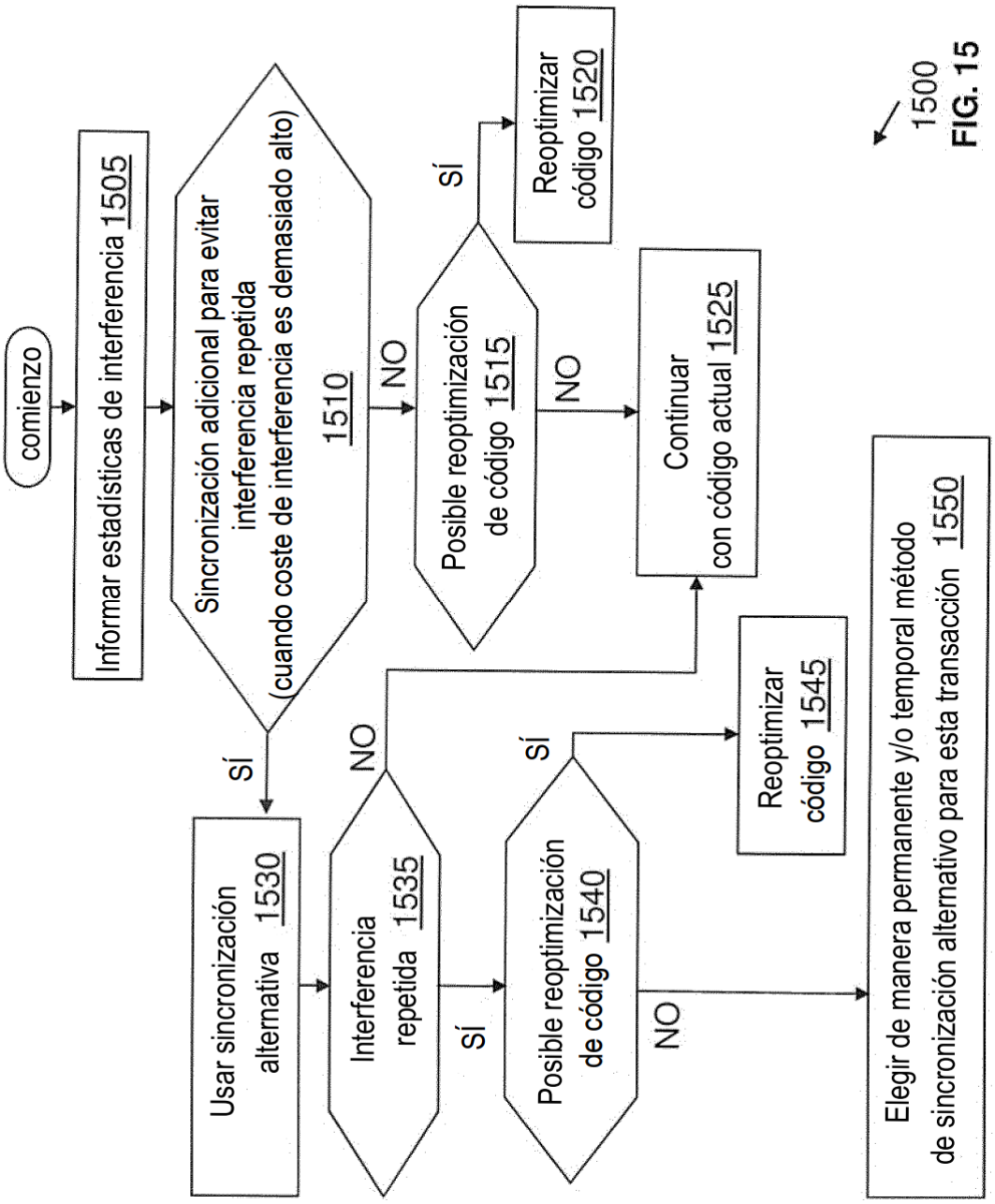

comienzo
Informar estadísticas de interferencia 1505
Sincronización adicional para evitar interferencia repetida (cuando coste de interferencia es demasiado alto) 1510
SÍ
NO
Posible reoptimización de código 1515
SÍ
NO
Reoptimizar código 1520
Continuar con código actual 1525
Usar sincronización alternativa 1530
Interferencia repetida 1535
SÍ
NO
Posible reoptimización de código 1540
SÍ
NO
Reoptimizar código 1545
Elegir de manera permanente y/o temporal método de sincronización alternativo para esta transacción 1550
1500

FIG. 15

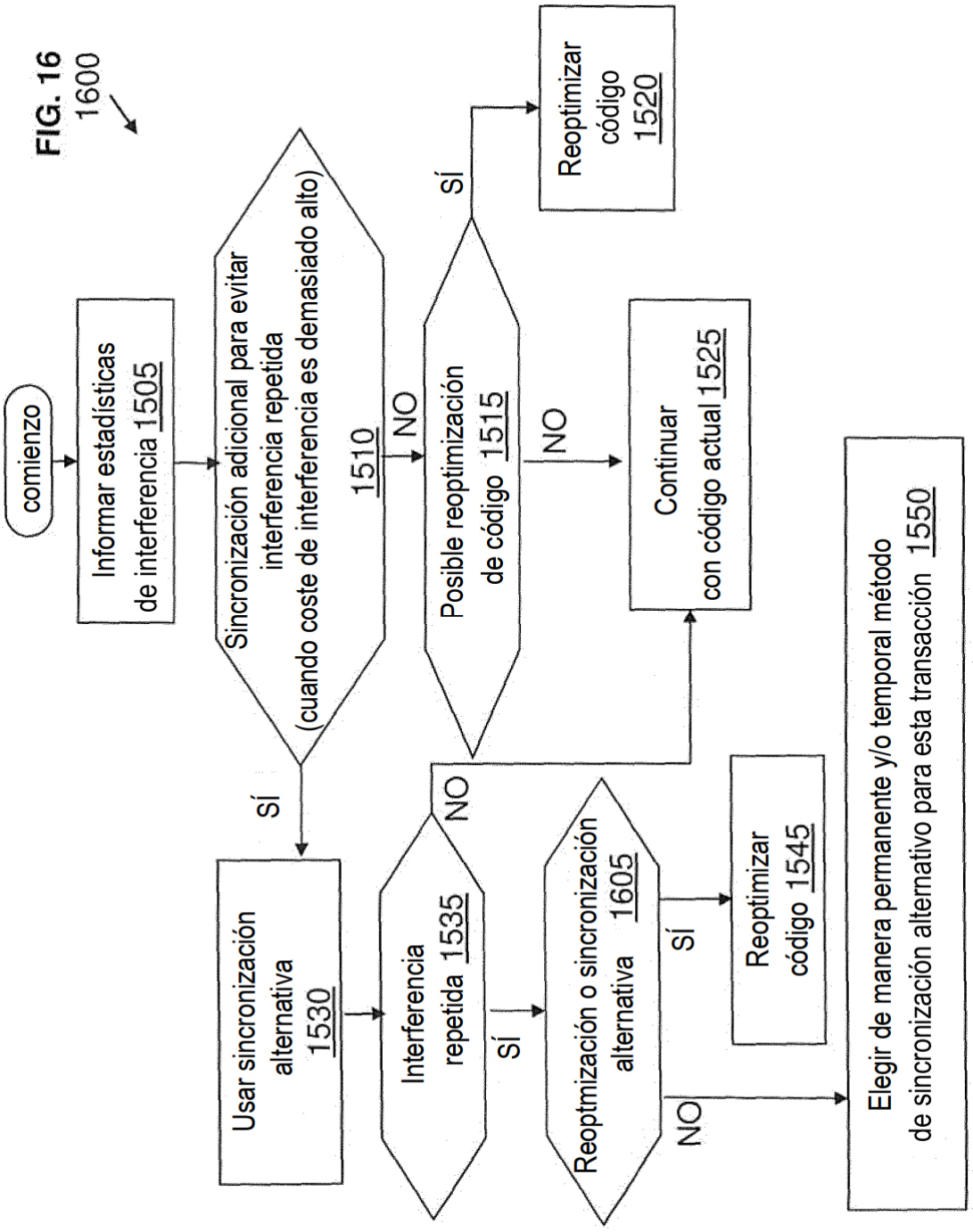


FIG. 16
1600
comienzo
Informar estadísticas de interferencia 1505
Sincronización adicional para evitar interferencia repetida (cuando coste de interferencia es demasiado alto) 1510
SÍ
NO
Reoptimizar código 1520
Posible reoptimización de código 1515
SÍ
NO
Continuar con código actual 1525
Usar sincronización alternativa 1530
Interferencia repetida 1535
SÍ
NO
Reoptmización o sincronización alternativa 1605
SÍ
NO
Reoptimizar código 1545
Elegir de manera permanente y/o temporal método de sincronización alternativo para esta transacción 1550

FIG. 17

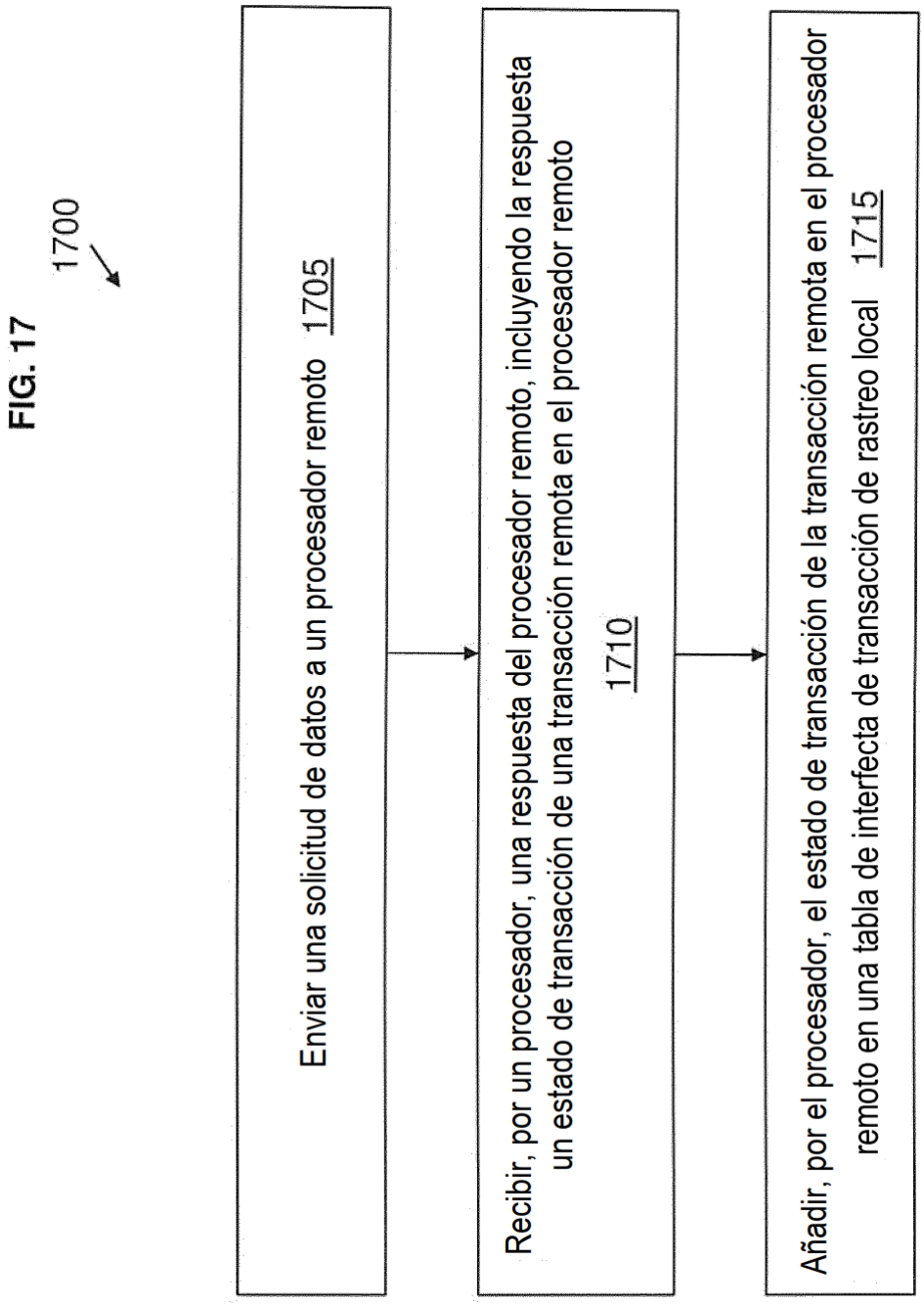

1700
Enviar una solicitud de datos a un procesador remoto 1705
Recibir, por un procesador, una respuesta del procesador remoto, incluyendo la respuesta un estado de transacción de una transacción remota en el procesador remoto 1710
Añadir, por el procesador, el estado de transacción de la transacción remota en el procesador remoto en una tabla de interfecta de transacción de rastreo local 1715

FIG. 18


Producto de programa informático 1800
Instrucciones de programa 1804
Medio de almacenamiento usable/legible por ordenador 1802