Execução concorrente e determinista de transações

Tiago M. Vale[1], Ricardo J. Dias[1,2], João A. Silva[1], and João M. Lourenço[1]

[1] NOVA LINCS, Universidade Nova de Lisboa
[2] INESC-ID

Resumo Neste artigo apresentamos um protocolo de controlo de concorrência que garante que a execução concorrente de transações é equivalente à sua execução sequencial por uma ordem predefinida. Isto permite executar programas que usam transações de forma determinista. O protocolo (1) permite, pela primeira vez, a execução determinista de programas que usam memória transacional por *hardware*; e (2) garante a execução determinista de programas que usam memória transacional por *software* com um desempenho claramente superior ao estado da arte.

1 Introdução

Devido à massificação dos sistemas computacionais com múltiplos núcleos e/ou processadores, a programação concorrente com múltiplos fios de execução deixou de ser uma tarefa para especialistas e é agora uma tarefa essencial no processo de desenvolvimento de software. Nos programas sequenciais as operações são executada por um único fio de execução e pela ordem especificada no código fonte. Já no caso do programas concorrentes, a ordem pela qual operações de diferentes fios de execução executam é imprevisível e irá provavelmente variar entre execuções. Embora esta imprevisibilidade possa ser vista como um inevitável preço a pagar pelos benefícios da execução de múltiplos fios de execução, a imprevisibilidade da ordem de execução das operações dificulta a construção de programas confiáveis. Por exemplo, a imprevisibilidade da execução concorrente dificulta a reprodução de execuções e, portanto, o entendimento de comportamentos anómalos e o diagnóstico das sua causa; muitos destes erros apenas se manifestam quando certas operações de diferentes fios de execução executam numa determinada ordem [11].

A investigação em execução determinista de programas com múltiplos fios de execução tem sido focada em programas que usam trincos [1,4,10,12,14]. Embora os trincos sejam a técnica convencional utilizada para controlar acessos concorrentes ao estado partilhado por vários fios de execução, a Memória Transacional (MT) [8,18] é cada vez mais uma alternativa viável devido a um bom compromisso entre desempenho e complexidade de desenvolvimento. Com MT os programadores apenas precisam de especificar quais as sequências de operações que devem executar de forma indivisível (transações) sem se preocuparem em *como* garantir essa indivisibilidade. Esta é garantida em tempo de execução

através de um protocolo de controlo de concorrência implementado em *software* (MTS), *hardware* (MTH), ou numa mistura de ambos. A MT está a chegar ao programador comum: os processadores mais recentes da Intel e IBM já dispõem de MTH [3,23], o compilador GCC já possuí funcionalidades experimentais desde a versão 4.7 que permitem escrever programas com MT (utilizando MTS ou MTH) [6], e existem também esforços em curso para integrar construções de MT diretamente na linguagem C/C++ [2].

Os sistemas para execução determinista de programas que usam trincos podem, em princípio, ser usados para garantir a execução determinista de programas que usam MTS se, e só se, o protocolo de controlo de concorrência da MTS for implementado com trincos (deterministas). Contudo, esta abordagem tem alguns inconvenientes, nomeadamente: (a) não pode ser aplicada a MTH, porque o protocolo de controlo de concorrência está implementado no *hardware*; (b) perde a oportunidade de tirar partido da semântica transacional para reduzir os custos de garantir determinismo, porque este está a ser garantido pelo uso de trincos, que estão num nível de abstração inferior ao das transações; e (c) muitas MTS práticas usam primitivas atómicas de baixo nível, ex. *compare-and-swap*, em vez de trincos. Este artigo discute como garantir, eficientemente, a execução determinista de programas que usam MT. Nós advogamos a utilização de transações preordenadas como uma abordagem fundamental: estas fornecem a ilusão de que executam uma de cada vez, sempre numa ordem predefinida. Assim, propomos um novo protocolo de controlo de concorrência, aplicável tanto a MTS como MTH[3], que garante que o resultado da execução concorrente de um conjunto de transações é sempre equivalente a uma execução sequencial dessas transações numa ordem predeterminada. Este novo protocolo de controlo de concorrência, a que chamamos TPO, utiliza duas técnicas para garantir equivalência a uma ordem predefinida: *confirmação ordenada* e *modos de execução*. A confirmação ordenada força as transações a serem confirmadas de acordo com a ordem predefinida. Infelizmente a confirmação ordenada pode induzir desperdício de tempo porque uma transação pronta a ser confirmada tem de esperar que outra que a precede termine e seja confirmada primeiro. Para mitigar este problema tiramos partido da observação que, a todo o momento, existe sempre uma transação que é a próxima a ser confirmada. O protocolo TPO executa essa transação o mais rapidamente possível evitando tanto quanto possível os custos associados a controlo de concorrência (modo de execução *garantido*), enquanto que as restantes transações executam de acordo com um protocolo de controlo concorrência tradicional que garante a correção dessas transações na presença da transação rápida (modo de execução *especulativo*).

Implementámos dois protótipos do TPO: um que usa MTS e outro que usa MTH. A versão que usa MTH baseia-se num sistema de MTH disponível comercialmente e sem qualquer modificação. Avaliámos ambos os protótipos utilizando o popular banco de testes STAMP [13] e concluímos que o protocolo TPO avança o estado da arte tanto em *software* como em *hardware*. A versão *software* tem um desempenho claramente superior ao estado da arte (DeSTM [16]) e per-

[3] E sistemas transacionais em geral.

mite executar transações deterministicamente com baixo custo, demonstrando que usar simultaneamente transações e execução determinista é uma aproximação viável para facilitar o desenvolvimento de programas concorrentes. A versão *hardware* do protocolo TPO é, tanto quanto sabemos a primeira a permitir a execução determinista de transações utilizando um mecanismo de MTH disponível comercialmente, conseguindo-o com um custo modesto.

2 Visão geral

A opacidade [7] é o critério de correção utilizado em Memória Transacional (MT). Brevemente, a opacidade dá as mesmas garantias que *serializability* [15] com a restrição adicional que nenhuma transação pode observar estado inconsistente, nem mesmo as transações que venham a ser abortadas. As transações opacas são executadas concorrentemente mas aparentam executar uma de cada vez. Apesar disso a opacidade permite flexibilidade ao nível do protocolo de controlo de concorrência porque as transações podem aparentar executar por uma ordem *qualquer*. Os protocolos de controlo de concorrência tradicionais que implementam opacidade abraçam a flexibilidade dada por esta e realizam duas tarefas simultaneamente com a execução de transações: (a) computam a ordem na qual as transações aparentam executar (ordenação); e (b) controlam a execução concorrente das transações para respeitar essa ordem (controlo de concorrência). Como o processo de ordenação está entrelaçado com o controlo de concorrência, a ordem na qual as transações aparentam executar depende das intercalações imprevisíveis que ocorrem durante a execução concorrente de transações. Como tal, a ordem das transações pode, e costuma, mudar entre diferentes execuções do mesmo programa. Neste artigo apelidamos este modelo de execução de *transações tradicionais*.

Com *transações preordenadas* a ordem de execução é independente das intercalações que possam ocorrer em tempo de execução porque, ao contrário das transações tradicionais, as transações preordenadas, como o nome indica, já estão ordenadas *antes* de serem executadas. Conceptualmente as transações preordenadas passam por duas fases durante a sua execução: (1) a fase de *ordenação*, que define qual é a posição de todas as transações na ordem de execução; e (2) a fase de *execução*, onde as transações são executadas concorrentemente de tal maneira que o resultado final é equivalente à execução sequencial das transações na ordem predefinida. Os protocolos de controlo de concorrência tradicionais *não* podem ser usados na fase de execução porque implementam simultaneamente a ordenação e o controlo de concorrência. Neste artigo propomos um novo protocolo que pode ser usado na fase de execução.

2.1 Fase de ordenação

Uma consequência da separação da ordenação do controlo de concorrência é que a fase de ordenação e a fase de execução (onde o controlo de concorrência é feito) podem ser realizadas separadamente por dois componentes diferentes. A fase de

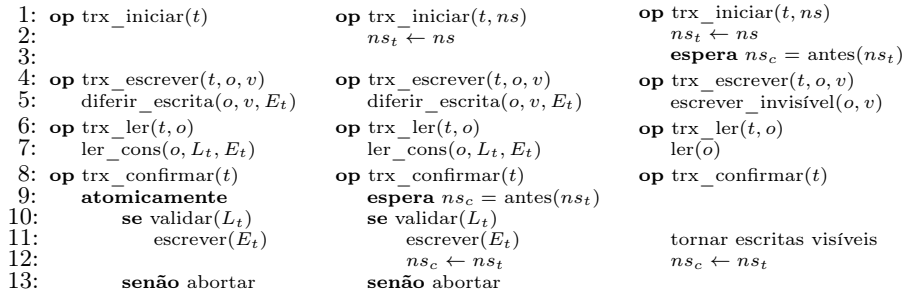
ordenação é efetuada por um *ordenador* que calcula uma ordem total sobre o conjunto de todas as transações. O ordenador é livre de computar qualquer ordem deterministicamente de acordo com critérios dependentes da aplicação. Na prática, para garantir a execução determinista de uma aplicação, o ordenador pode ordenar as transações à medida que estas estão a ser geradas pela aplicação, de uma forma determinista, para garantir que todas as execuções da aplicação resultam na mesma ordenação das transações. Por exemplo, o ordenador pode ordenar por turnos as transações geradas pelos vários fios de execução. Dado um programa com dois fios de execução t_1 e t_2, tal que o fio de execução t_1 executa as transações a_1 e b_1, e o fio de execução t_2 executa as transações c_2 e d_2, a ordenação seria $\langle a_1, c_2, b_1, d_2 \rangle$. Outra hipótese possível para o ordenador é usá-lo para reproduzir uma execução (não-determinista) anterior. Neste caso é necessário gravar a ordem pela qual os fios de execução confirmam cada transação na execução que se pretende reproduzir. Para reproduzir a execução o ordenador ordena as transações de acordo com a informação gravada.

2.2 Fase de execução

Uma vez ordenadas, as transações podem ser executadas. No coração da fase de execução está um protocolo de controlo de concorrência que garante equivalência à ordem definida na fase de ordenação. A implementação trivial desse protocolo é simplesmente executar as transações sequencialmente pela ordem indicada. Contudo essa estratégia é claramente insuficiente porque não tira partido do paralelismo inerente às arquiteturas com múltiplos processadores. Na próxima secção apresentamos o cerne deste artigo, um protocolo para ser usado na fase de execução que permite a execução concorrente de transações preordenadas.

3 Protocolo TPO

Nesta Secção descrevemos o TPO, um novo protocolo de controlo de concorrência que, embora execute transações concorrentemente, garante equivalência a uma execução sequencial dessas transações numa ordem predefinda. O TPO está desenhado como uma metodologia que pode ser aplicada a protocolos de controlo de concorrência otimistas [9], que funcionam da seguinte maneira. Uma transação otimista consiste numa, ou mais, execuções especulativas. Cada execução especulativa está dividida em três fases: (1) leitura; (2) validação; e (3) escrita. Durante a fase de leitura a transação regista no seu conjunto de leituras todos os objetos do estado partilhado que lê. As operações de escrita não modificam o estado partilhado; em vez disso a transação regista o novo valor no seu conjunto de escritas, diferindo as modificações para quando for confirmada. (Objetos que sejam lidos e escritos estão contidos tanto no conjunto de leituras como no de escritas.) Após a fase de leitura a transação passa por uma fase de validação onde verifica se algum objeto do seu conjunto de leituras foi modificado por uma transação concorrente. Se sim, então a transação é abortada para garantir opacidade, e pode ser reexecutada; caso contrário a transação passa para a fase de



(a) Otimista. (b) TPO: especulativa. (c) TPO: garantida.

Figura 1: Metodologia para transformar o protocolo otimista no TPO. A Figura 1a modela uma transação otimista típica. As Figuras 1b e 1c modelam o modo de execução especulativo e garantido do TPO, respectivamente. ns_c representa o número de sequência da última transação que foi confirmada. ns_t, L_t e E_t representam o número de sequência, conjunto de leituras, e conjunto de escritas da transação t, respectivamente.

escrita. Na fase de escrita a transação modifica todos os objetos pendentes no seu conjunto de escritas de forma indivisível.

Escolhemos um protocolo otimista como base para o TPO porque é apropriado para transações dinâmicas, isto é, transações para as quais é muito difícil (ou até impossível) identificar os seus conjuntos de leitura e escrita sem as executar. Transações dinâmicas são bastante comuns em programas que usam MT em linguagens de propósito geral devido ao *aliasing* e à natureza não-estruturada da *heap*. De facto, os protocolos de controlo de concorrência da MT, tanto por *software* como por *hardware*, são normalmente otimistas.

No resto desta Secção apresentamos o TPO de forma incremental. Primeiro descrevemos o protocolo otimista base na Secção 3.1, seguido da nossa metodologia para transformar o protocolo base no TPO através da aplicação de duas técnicas-chave: confirmação ordenada, na Secção 3.2, e modos de execução, na Secção 3.3.

3.1 Protocolo base

Consideremos o protocolo na Figura 1a que modela o esquema otimista. A fase de leitura ocorre depois da operação $trx\_iniciar$ e antes da $trx\_confirmar$, e consiste em invocações da trx_ler e/ou $trx\_escrever$. Tanto a fase de validação como a de escrita ocorrem durante a operação $trx\_confirmar$.

Fase de leitura. Operações de escrita com o objetivo de modificar o valor do objeto o para v registam essa informação no conjunto de escritas E_t (Figura 1a, linha 5). Uma operação de leitura no objeto o regista esse acesso no conjunto de leituras L_t e retorna o valor da última escrita pendente para o em E_t; caso não exista então retorna um valor de o no estado partilhado que seja consistente com o resto do conjunto de leituras (linha 7). Se não for possível retornar um valor de o consistente então a transação aborta. Por exemplo, imaginemos dois

objetos x e y, ambos inicialmente com o valor 0. A transação t observa $x = 0$. Entretanto, outra transação é confirmada e modifica tanto o valor de x como de y para 1. Se a transação t tentar ler y deve observar o valor 0 ou abortar, mas nunca poderá observar 1 porque $x = 0$ e $y = 1$ não é um estado possível de acordo com a opacidade.

Fase de validação. Na fase de validação a transação verifica se todos os objetos que observou ainda estão coerentes, isto é, se entretanto nenhuma transação os alterou (linha 10).

Fase de escrita. Se a validação for bem sucedida então a transação executa todas as modificações que tem pendentes no seu conjunto de escritas (linha 11).

Correção. Este protocolo garante opacidade devido à restrição de que as transações abortam se as leituras não puderem devolver um valor consistente e à indivisibilidade da execução das fases de validação e escrita (linhas 9–13). Se a fase de validação for bem sucedida então nenhum dos objetos lidos foi modificado desde a fase de leitura. Isto significa que a fase de leitura acontece logicamente no mesmo instante que a fase de validação. Como as fases de validação e de escrita executam de forma indivisível, a fase de escrita também acontece logicamente no mesmo instante que a fase de leitura. Como tal, a transação t aparenta ter executado sozinha no sistema, depois das transações que escreveram os valores que t observou e antes das transações que observarão os valores escritos por t.

3.2 Confirmação ordenada

O protocolo base (otimista) descrito na secção anterior dá a ilusão de que as transações executam uma de cada vez. Contudo, a ordem pela qual as transações aparentam executar é imprevisível porque depende da intercalação das operações das transações que ocorra durante a execução.

Para respeitar a ordem definida pela fase de ordenação, fazemos duas observações: (a) as transações otimistas só modificam o estado partilhado durante a sua fase de escrita; e (b) a posição de cada transação na ordem de execução depende da ordem relativa pela qual cada transação (atomicamente) executa a sua fase de validação e escrita. Portanto se restringirmos as transações a executarem as suas fases de validação e escrita na ordem definida pela fase de ordenação, garantimos que o resultado é equivalente à respectiva execução sequencial.

Para transformar o protocolo otimista descrito na secção anterior no TPO, começamos por alterar a operação $trx\_iniciar$ para ter um parâmetro adicional, um número de sequência ns, que reflete a posição dessa transação na ordem de execução definida pelo ordenador na fase de ordenação (Figura 1b, linha 1). A transação t está preordenada depois da transação com o número de sequência $antes(ns_t)$ e depois da transação com o número de sequência $depois(ns_t)$. Para forçarmos as transações a serem confirmadas de acordo com a ordem predefinida, inserimos uma espera condicional na operação $trx\_confirmar$. Quando a transação t quer ser confirmada, esta espera até a transação com o número de sequência $antes(ns_t)$ seja confirmada (linha 9). As transações comunicam entre elas através de um objeto ns_c cujo valor é o número de sequência da última transação a ser confirmada (linha 12).

Correção. No protocolo otimista original um dos principais fatores que garantem a sua correção é a execução indivisível da fase de validação e escrita das transações. Contudo a ordem pela qual as transações executam esse bloco indivisível depende da execução concorrente e imprevisível das transações. Para respeitar a ordem predefinida, no protocolo TPO esse bloco indivisível é substituído por uma espera condicional que restringe a ordem pela qual as transações são confirmadas. Concretamente, uma transação que acabe a sua fase de leitura só executa a sua fase de validação e escrita *após* a transação que a precede diretamente na ordem predefinida ter sido confirmada. Como as transações estão totalmente ordenadas, a condição de espera é verdade apenas para *uma transação de cada vez*. O protocolo continua a estar correto porque a espera condicional continua a garantir a indivisibilidade da fase de validação e escrita (o escopo de atomicidade é entre a espera condicional e a atualização de ns_c).

3.3 Modos de execução

O controlo de concorrência otimista usa uma série de técnicas para garantir a correção, nomeadamente conjuntos de leitura e escrita, validação, e escritas diferidas. Todas as transações são executadas utilizando essas técnicas porque *qualquer* transação *pode* vir a ser a próxima transação a ser confirmada. Contudo, a utilização dessas técnicas impõe computação extra quando comparado com uma execução sequencial sem qualquer controlo de concorrência. Com o TPO, ao contrário do protocolo base, a ordem de execução está predefinida. Como o TPO restringe a ordem pela qual as transação são confirmadas, estas podem agora ter que esperar pela sua vez, o que leva a uma perda de paralelismo em comparação com o protocolo base. Para mitigar esta perda de paralelismo, tiramos partido da observação que a qualquer momento existe sempre uma única transação, à qual chamados *garantida*, que é a próxima transação a ser confirmada. A transação garantida pode executar sem utilizar a maior parte das técnicas do protocolo base, evitando assim computação extra (e desnecessária) e acelerando a sua execução. Como tal, o TPO distinge entre dois tipos de transações: garantidas e especulativas.

Transação garantida. A transação garantida é a *única* transação ativa cujos antecessores já foram todos confirmados. Esta é a próxima, e única, transação que pode ser confirmada imediatamente. Esta transação pode ser executada de forma mais eficiente se fundirmos a fase de leitura com a fase de escrita e removermos completamente a fase de validação, eliminando assim a maior parte das técnicas do controlo de concorrência otimista e os seus custos associados. As transações garantidas executam de acordo com o protocolo da Figura 1c.

Fase de leitura. As operações de escrita já não são diferidas; em vez disso elas modificam o estado partilhado diretamente (linha 5). (Notamos que apesar disso a escrita tem que se manter invisível para *outras* transações até à confirmação da transação garantida, de forma a garantir que quaisquer outras transações vejam *todas* as escritas da transação garantida, ou nenhuma.) Como as modificações são feitas imediatamente durante a (agora combinada) fase de leitura e escrita, as operações de leitura são reduzidas a simplesmente ler o valor atual do

objeto sem efetuar quaisquer testes de consistência nem registo no conjunto de leituras (linha 7).
Fase de validação. As transações garantidas executam até ao fim sem qualquer interferência de outras transações, portanto a fase de validação é desnecessária.
Fase de escrita. A fase de escrita é feita implicitamente durante a fase de leitura devido à estratégia de escrever diretamente no estado partilhado, portanto esta fase também é eliminada. É apenas necessário tornar todas as escritas efetuadas visíveis atomicamente para as outras transações (linha 11).
Correção. O nosso argumento para a correção é o mesmo que para a técnica da confirmação ordenada. Contudo, uma transação garantida não executa a sua fase de leitura de forma especulativa, esperando pelo seu turno para transitar para fase de validação e escrita. Em vez disso, a transação executa a (agora combinada) fase de leitura e escrita quando já for a sua vez de ser confirmada. Visto que a transação tem, efetivamente, acesso exclusivo de escrita no estado partilhado até ao fim da sua execução, fundir as fases de leitura e escrita (substituindo as escritas diferidas por escritas diretas) e remover a fase de validação não afeta a correção do protocolo.
Transação especulativa. Uma transação cuja vez para ser confirmada ainda não chegou é uma transação especulativa. Estas executam de acordo com o protocolo com confirmação ordenada descrito na Secção 3.2, onde já discutimos a correção. Estas mantém-se corretas na presença da transação garantida porque as escritas desta última são feitos atomicamente.
Promoção durante a execução. Como as transações garantidas não pagam a maior parte dos custos associados ao controlo de concorrência, uma transação especulativa t que esteja a executar a sua fase de leitura pode mudar imediatamente para o modo garantido assim que $ns_c = ns_t$ seja verdade, i.e., seja a sua vez de ser confirmada. Para ser promovida de especulativa a garantida durante a execução, a transação t valida imediatamente a parte da fase de leitura que já executou. Se a validação for bem sucedida então a transação t aplica no estado partilhado todas as suas escritas que estejam diferidas, mas *não* atualiza o objeto ns_c. A partir deste ponto a transação t pode executar o resto da sua lógica como uma transação garantida. Caso contrário, ela aborta e pode reexecutar em modo garantido desde o início.
Múltiplas transações garantidas simultaneamente. Se soubermos mais informação sobre as transações podemos executar múltiplas transações garantidas em paralelo. Transações consecutivas que não leiam, ou escrevam, objetos escritos uma pela outra podem todas executar simultaneamente em modo garantido, porque o resultado final é independente da ordem pela qual elas são confirmadas.

4 Avaliação experimental

Todas as experiências foram executadas num IBM POWER 8 com 20 núcleos divididos por 2 *sockets* (10 núcleos em cada um) e um total de 128 GB de memória primária. As experiências executam com *simultaneous multithreading* desativado. Realçamos que a máquina tem uma arquitetura com acesso não uniforme

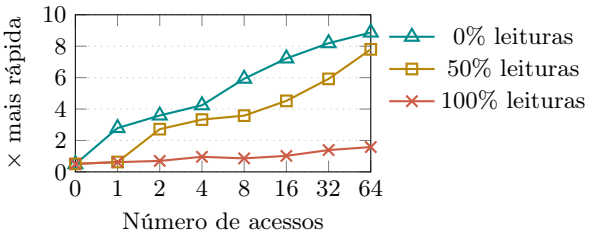


Figura 2: Quão mais rápida é uma transação TPO garantida que uma transação tradicional (em *software*).

à memória (NUMA). Concretamente, a latência de acesso à memória nas experiências é a seguinte: com 1–4 fios de execução a latência é uniforme; com 8 fios a latência aumenta até 2×, e com 16 fios aumenta até 4×.[4]

4.1 Memória transacional por *software*

Nesta secção apresentamos a avaliação do nosso protótipo que garante a execução determinista de transações em memória por *software*. Este protótipo foi implementado sobre o protocolo TL2 [5], ao qual aplicámos as transformações propostas na Figura 1. Pretendemos com esta avaliação responder às seguintes questões: (1) Quão eficazes são as transações garantidas? E (2) qual é o custo adicional no desempenho introduzido pelo protocolo TPO para garantir execução determinista? E como é que este se compara com o estado da arte?

Eficácia das transações garantidas. O objetivo das transações garantidas é reduzir o custo do controlo de concorrência na sua execução para mitigar a potencial perda de paralelismo induzida pela confirmação ordenada. Para medir quão eficaz é o modo garantido, executámos um banco de dados que consiste numa estrutura de dados chave-valor implementada com um vector de contadores. Executámos o banco de dados com um único fio de execução e fizemos variar (1) o número de acessos efetuado pelas transações; e (2) o rácio de leituras e escritas. A Figura 2 mostra quão mais rápida é a execução de uma transação TPO garantida comparativamente com a execução da mesma transação usando o protocolo de controlo de concorrência base. Uma transação com 0 acessos consiste na invocação de $trx\_iniciar$ imediatamente seguida de $trx\_confirmar$. Isto permite-nos medir o custo adicional imposto no desempenho pela nossa implementação da confirmação ordenada e modos de execução, que é de cerca de 2×. Aumentando o número de acessos feito pelas transação observamos que, como esperado, o modo garantido executa cada vez mais rápido que o protocolo base. Também observamos que as operações de escrita contribuem mais para o ganho do modo garantido: isto deve-se ao facto de que no protocolo base as operações de escrita impõem custos nas operações de leitura porque estas precisam de pesquisar no conjunto de escritas. As operações de escrita também contribuem

[4] Informação recolhida com o comando `numactl`.

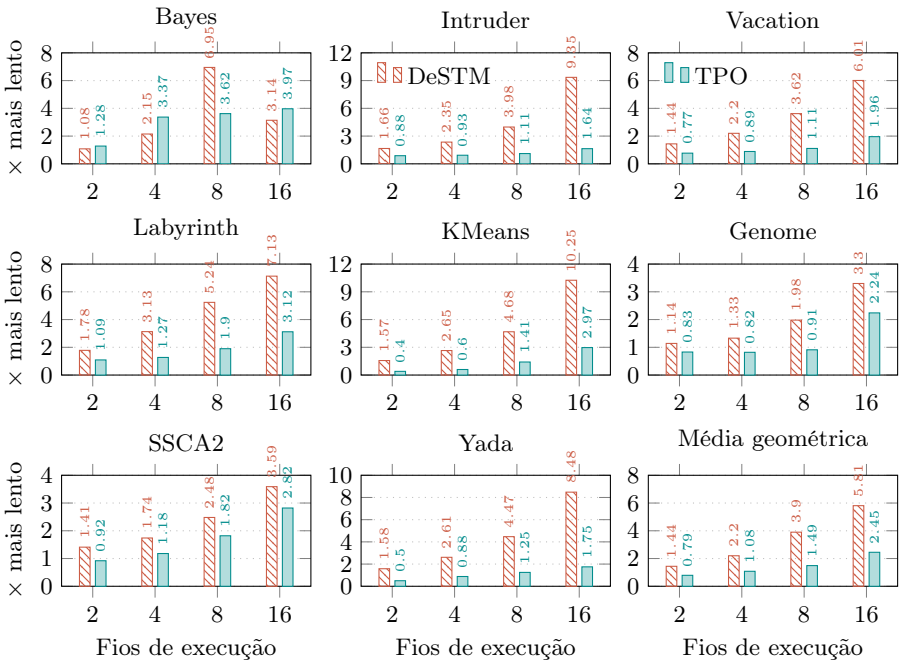


Figura 3: Custo de garantir execução determinista (em *software*) com o DeSTM e o TPO nos bancos de dados do STAMP. O eixo das ordenadas (yy) mede o tempo de execução de cada banco de dados normalizado para a execução não-determinista utilizando o protocolo base (menor é melhor). O canto inferir direito mostra a média geométrica, ou seja, a tendência geral no STAMP.

mais para o custo da confirmação da transação devido à necessidade de adquirir trincos para o conjunto de escritas antes de realizar as modificações pendentes e de posteriormente libertar todos os trincos adquiridos. As transações em modo garantido evitam todas estas fontes de computação adicional. Contudo, notamos que não existem ganhos observáveis quando as transações são apenas de leitura. A razão é simples: no TL2 as transações apenas de leitura não precisam de validar o seu conjunto de leituras para serem confirmadas. No geral, consideramos que o modo garantido é eficaz a minimizar os custos associados ao controlo de concorrência, mesmo em transações que efetuam poucos acessos: uma transação que efetua apenas uma leitura e uma escrita executa cerca de 3× mais rápido em modo garantido que em modo normal.

Comparação de desempenho com o estado da arte. Nesta secção avaliamos a execução determinista utilizando o TPO no popular banco de dados STAMP [13]. Este consiste em 8 aplicações representativas de diferentes domínios.Todos os resultados foram obtidos utilizando as configurações recomendadas e resultam da média de 5 execuções. Também comparamos contra o DeSTM [16], que representa o estado da arte de execução determinista com MTS.

DeSTM. Para realizar uma comparação honesta e significativa, implementámos o DeSTM no nosso próprio protótipo.[5] Tanto o DeSTM como o TPO são baseados no mesmo protocolo base e usam exatamente o mesmo ordenador. Uma das principais diferenças entre o DeSTM e o TPO é que neste último o ordenador estabelece uma ordem de serialização determinista das transações que o TPO garante, isto é, o resultado final é equivalente à execução sequencial das transações pela ordem definida pelo ordenador. Já o DeSTM usa o ordenador para definir uma ordem determinista entre os fios de execução na qual estes passam um testemunho que lhes permite tentar confirmar transações. Como tal, o resultado final é sempre equivalente a uma mesma ordem de serialização das transações, embora essa ordem seja desconhecida inicialmente. Como consequência deste desenho, o DeSTM ordena tanto as operações de aborto como de confirmação e requer que os conflitos entre transações sejam deterministas. Nós argumentamos que a necessidade de conflitos/abortos deterministas é um grande inconveniente, visto que a maior parte das implementações de MTS e todos os processadores com suporte para MTH padecem de falsos conflitos.

Desempenho. A Figura 3 quantifica o custo de garantir execução determinista utilizando o DeSTM e o TPO. O eixo das ordenadas (yy) mostra o tempo de execução dos bancos de dados normalizado para a execução não-determinista utilizando o protocolo base, dependendo do número de fios de execução (eixo das abcissas / xx). Nestes gráficos quanto menor o valor, melhor, e valores abaixo de 1 significam que a execução determinista foi *mais rápida* que a execução não-determinista tradicional. Realçamos quatro observações: (a) o custo de garantir determinismo aumenta com o número de fios de execução; (b) o TPO é melhor que o DeSTM em todos os bancos de dados independentemente do número de fios de execução, com a exceção do Bayes; (c) o TPO é no máximo $\approx 4\times$ mais lento que o protocolo (não-determinista) base enquanto que o DeSTM é até $\approx 10\times$ mais lento; e (d) o TPO chega a ser mais rápido que a execução não-determinista para algumas combinações banco de dados/fios de execução.

Não é surpreendente que o custo de garantir determinismo aumente com o número de fios de execução; a probabilidade de uma transação t querer ser confirmada antes do seu turno aumenta com o número de fios de execução, particularmente se existirem transações ordenadas antes de t que demorem mais tempo a executar que t. Ao contrário do DeSTM, o TPO minimiza estas situações utilizando transações garantidas para aumentar a probabilidade das transações não terem que esperar pela sua vez quando querem ser confirmadas. A Tabela 1 sustenta esta afirmação. Esta mostra, para cada combinação banco de dados/fios de execução, a percentagem do tempo de execução que as transações em média "desperdiçam" à espera que chegue a sua vez de confirmar. Observamos que em geral as transações TPO ficam menos tempo à espera da sua vez que as transações DeSTM, i.e., é mais frequente, com o TPO do que com o DeSTM, já ser

[5] O artigo do DeSTM aponta para um *URL* onde supostamente seriam disponibilizados os ficheiros executáveis da sua implementação, mas tal não se verificou até à data da submissão deste artigo. O nosso contacto com os autores do DeSTM no sentido de nos facultarem o acesso ao seu código não obteve qualquer resposta.

Tabela 1: Percentagem do seu tempo de execução que as transações passam à espera da sua vez para garantir determinismo, em média, utilizando o DeSTM e o TPO nos bancos de dados do STAMP.

Banco de dados	2 fios exec.		4 fios exec.		8 fios exec.		16 fios exec.	
	DeSTM	TPO	DeSTM	TPO	DeSTM	TPO	DeSTM	TPO
Bayes	42%	35%	66%	50–86%	80%	70–83%	92%	64–94%
Genome	19%	3%	37%	11%	66%	38%	85%	82%
Intruder	45%	21%	65%	45%	85%	73%	95%	89%
KMeans	40%	24%	67%	28%	84%	71%	94%	93%
Labyrinth	43%	8–45%	75%	50%	99%	48–69%	99%	49–84%
SSCA2	50%	10%	70%	61%	85%	88%	93%	96%
Vacation	23%	5%	46%	12%	68%	24%	82%	68%
Yada	45%	33%	70%	62%	84%	79%	96%	84%

a vez das transações serem confirmadas quando estas invocam $trx\_confirmar$. (O tempo gasto aumenta com o número de fios de execução, como esperado.)

Os resultados do Bayes são diferentes porque este é relativamente diferente dos restantes bancos de dados. No Bayes a ordem pela qual as transações são confirmadas no início da execução afeta dramaticamente o tempo de execução [17]. Portanto o comportamento é imprevisível, particularmente com o DeSTM dado que a ordem de serialização das transações é desconhecida. Tanto o DeSTM como o TPO usam sempre o mesmo ordenador[6]; não tentámos otimizar a ordenação para cada banco de dados.

Em conclusão, os excelentes resultados do TPO em comparação com o DeSTM devem-se à aceleração permitida pelo modo garantido, como vemos na Figura 2, e pela redução do tempo "perdido" à espera, como vemos na Tabela 1. O TPO marca assim um avanço significativo no estado da arte em termos de desempenho e apresenta evidências promissoras de que usar simultaneamente transações e determinismo para facilitar a programação concorrente é viável e poderá ser prático.

4.2 Memória transacional por *hardware*

Também avaliamos o nosso protótipo de MTH utilizando o STAMP. O suporte para MTH existente nos dias de hoje tem a particularidade de apenas garantir "o melhor que for possível" (*best effort*). Os processadores com MTH mantêm os conjuntos de leitura e escrita das transações apenas na *cache*; consequentemente, o número de operações que uma transação em *hardware* consegue efetuar está limitado pelo tamanho da *cache*. Transações por hardware que excedam os limites físicos do *hardware* nunca conseguirão executar com sucesso. É necessário fornecer *sempre* um fluxo de execução alternativo em *software* que garante a execução dessas transações. No nosso caso implementámos a solução mais comum: quando uma transação não consegue completar em *hardware* é executada em *software* com recurso a um trinco global que garante a execução em exclusão

[6] O ordenador ordena as transações dos fios de execução por turnos, tal como no artigo do DeSTM, p. ex., com 4 fios temos 1-2-3-4-1-2-3-4-1...

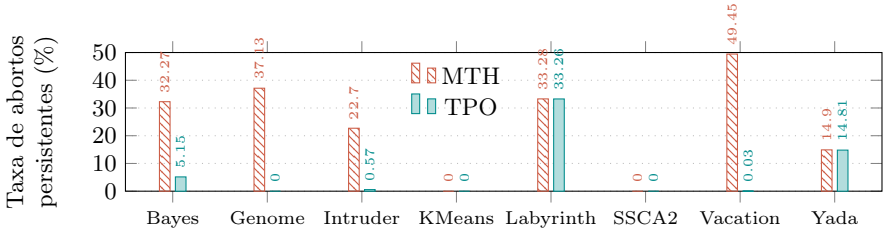


Figura 4: Taxa de abortos persistentes das transações tradicionais e transações TPO garantidas nos bancos de dados do STAMP (em *hardware*).

mútua com todas as outras transações, estejam essas a executar em *hardware* ou também em *software*.

Eficácia das transações garantidas. Enquanto que as transações garantidas em MTS permitem reduzir os custos associados ao controlo de concorrência, uma implementação em MTH que queira atingir o mesmo objetivo necessita de suporte do próprio *hardware*, coisa que não existe nos processadores atuais. Contudo, no nosso protótipo tiramos partido de um tipo específico de transações que a IBM fornece chamadas *rollback-only transactions* (ROT). As ROT são transações que apenas garantem que as suas escritas são indivisíveis, isto é, apenas mantêm o conjunto de escritas. Todas as leituras acedem diretamente à memória sem qualquer controlo de concorrência. Como as ROT só mantêm o conjunto de escritas, estas conseguem executar um maior número de acessos que uma transação normal. Contudo uma ROT pode ser abortada devido a conflitos de escrita-escrita na *cache*. No nosso protótipo as transações garantidas são executadas como ROT, o que lhes permite uma maior chance de executar completamente em *hardware*, não sendo necessário reverter para uma execução sequencial em *software*. Note-se que a transação garantida não pode executar em modo não-transacional porque isso não garantiria a atomicidade das suas escritas, o que violaria a opacidade.

Executámos cada banco de dados do STAMP com um único fio de execução, utilizando MTH tradicional e TPO e medimos o número de transações que excedem a capacidade do *hardware* e necessitam de reverter para o trinco global. A Figura 4 mostra que podemos dividir os bancos de dados em 3 classes. Na primeira classe temos o Labyrinth e o Yada, que possuem transações que não conseguem ser executadas em *hardware*. Na segunda classe temos exatamente a situação oposta: no KMeans e no SSCA2 todas as transações conseguem ser executadas em *hardware*. Finalmente temos a terceira classe com os restantes bancos de dados. Nesta classe existem transações que não conseguem executar em *hardware* utilizando o modo normal, mas conseguem executar como ROTs. Conseguimos ver claramente o benefício das transações garantidas: com o TPO a taxa de abortos é de cerca de 5% no Bayes e praticamente 0% nos restantes, enquanto que com MTH tradicional a taxa está acima de 20% em todos os bancos de dados. Em conclusão, as transações garantidas conseguem recuperar algum

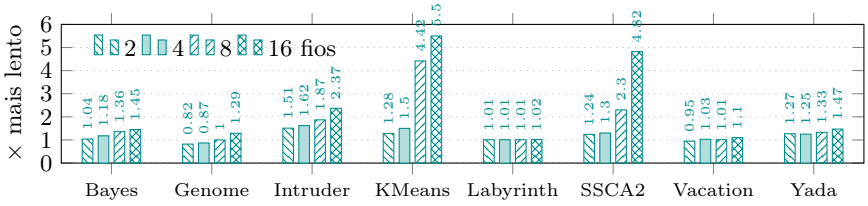


Figura 5: Custo de garantir execução determinista (em *hardware*) com o TPO nos bancos de dados do STAMP. O eixo das ordenadas (yy) mede o tempo de execução de cada banco de dados normalizado para a execução não-determinista utilizando o protocolo base (menor é melhor).

do paralelismo perdido devido à confirmação ordenada quando o protocolo base reverte para *software*.

Desempenho. A Figura 5 mostra o custo de garantir execução determinista utilizando o protótipo TPO em MTH. O desempenho do TPO é praticamente igual nos bancos de dados nos quais a execução tradicional reverte para *software* muitas vezes (Bayes, Genome, Labyrinth, e Vacation). Os resultados mais interessantes são os dos bancos de dados onde o protocolo base não necessita de reverter para *software* muitas vezes (Intruder, KMeans, SSCA2, e Yada). No Intruder e no Yada o TPO é apenas até $\approx 2\times$ mais lento que a execução não-determinista. O KMeans e o SSCA2, com transações pequenas e poucos conflitos, são ótimos para o protocolo base. Estas características tornam difícil mascarar os custos de garantir determinismo. Notamos que, como as transações garantidas não são aceleradas em MTH, e podem até abortar, existe uma descida significativa no desempenho de 4 para 8 fios de execução e ainda maior de 8 para 16, devido ao aumento da latência no acesso à memória da arquitetura NUMA.

Em conclusão, tanto quanto sabemos, o TPO avança o estado da arte porque permite pela primeira vez a execução determinista de programas que usam MTH. Em geral, o TPO garante execução determinista com um custo modesto, mas o aumento da latência no acesso à memória leva a uma perda de desempenho relativamente à execução não-determinista. No nosso protótipo as transações garantidas são implementadas como *ROT*s. Os resultados das transações garantidas no protótipo em *software* sugerem que suporte do próprio *hardware* para transações que *não* abortam permitirá diminuir os custos de garantir determinismo com MTH.

5 Trabalho relacionado

Transações preordenadas. A ideia de preordenar transações também é usada na comunidade de bases de dados com o objetivo de reduzir os custos das transações distribuídas [20]. Neste trabalho nós advogamos o seu uso para garantir execução determinista de programas baseados em MT. Os trabalhos referidos também descrevem um protocolo de controlo de concorrência para garantir que

uma ordem predefinida é respeitada. Contudo esse protocolo não pode ser usado no contexto do nosso trabalho, particularmente em MTH, porque utiliza uma estratégia pessimista e está desenhado para transações cujos conjuntos de leituras e escritas são determinados *a priori*. O TPO é otimista e mais geral porque funciona tanto com transações que tenham conjuntos de escritas e leituras estáticos como dinâmicos.

Execução determinista. Foram propostos variados sistemas para execução determinista de programas que usam trincos [1,10,14,4,12]. Se as transações forem implementados com trincos, executá-las deterministicamente pode ser feito utilizando esses sistemas como base. Contudo essa abordagem tem algumas limitações, nomeadamente: (a) não pode ser aplicada a MTH; (b) não tira partido da semântica transacional para reduzir os custos de garantir determinismo; e (c) muitas MTS práticas usam primitivas atómicas como *compare-and-swap* em vez de trincos. O DeSTM [16] adapta a técnica da dupla-barreira, usada por muitos sistemas que garantem a execução determinista de programas com trincos, para o contexto da MTS. O TPO estabelece uma ordem de serialização determinista e garante que a execução a respeita. O DeSTM define uma ordem determinista entre fios de execução pela qual estes trocam um testemunho que lhes permite confirmar transações. Como consequência, o DeSTM requer que os conflitos entre transações sejam deterministas (o que impossibilita que seja usado em MTH), enquanto que o TPO funciona independentemente disso.

Modos de execução. Executar uma transação com a garantia que ela não aborta é usado para permitir a utilização de *I/O* dentro de transações [19,22], e para melhorar o desempenho de MTS com poucos fios de execução [21]. As transações garantidas do TPO são semelhantes em espírito a estes trabalhos, contudo o seu objetivo é minimizar os custos de garantir determinismo. Ao contrário dos trabalhos referidos, múltiplas transações garantidas podem executar simultaneamente em paralelo tirando partido da existência de uma ordem predefinda, p. ex., uma sequência de transações que não tenham conflitos leitura-escrita e escrita-escrita entre si podem todas executar simultaneamente em modo garantido.

6 Conclusão

Neste artigo apresentámos o TPO, um protocolo de controlo de concorrência que faz uso do conceito de transações preordenadas para garantir a execução determinista de programas que usam múltiplos fios de execução e MT. A execução determinista é garantida utilizando duas técnicas: confirmação ordenada e modos de execução. Este artigo avança o estado da arte porque: (1) garante a execução determinista de programas que usam MTS com um desempenho claramente superior ao estado da arte, evidenciando que utilizar MTS e execução determinista para simplificar a programação concorrente é uma aproximação viável e poderá ser prático; e (2) permite, pela primeira vez, a execução determinista de programas que usam MTH, mas sugere que é desejável um melhor suporte do *hardware* para as transações garantidas.

Referências

1. T. Bergan, O. Anderson, J. Devietti, L. Ceze, and D. Grossman. CoreDet: A compiler and runtime system for deterministic multithreaded execution. In *ASPLOS*, 2010.
2. H. Boehm, J. Gottschlich, V. Luchangco, M. Michael, M. Moir, C. Nelson, T. Riegel, T. Shpeisman, and M. Wong. Transactional language constructs for C++. http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2012/n3341.pdf, 2012.
3. H. Cain, M. Michael, B. Frey, C. May, D. Williams, and H. Le. Robust architectural support for transactional memory in the POWER architecture. In *ISCA*, 2013.
4. H. Cui, J. Simsa, Y. Lin, H. Li, B. Blum, X. Xu, J. Yang, G. Gibson, and R. Bryant. Parrot: A practical runtime for deterministic, stable, and reliable threads. In *SOSP*, 2013.
5. D. Dice, O. Shalev, and N. Shavit. Transactional locking II. In *DISC*, 2006.
6. GCC. Changes in GCC 4.7. https://gcc.gnu.org/gcc-4.7/changes.html, 2012.
7. R. Guerraoui and M. Kapalka. On the correctness of transactional memory. In *PPoPP*, 2008.
8. M. Herlihy and J. Moss. Transactional memory: Architectural support for lock-free data structures. In *ISCA*, 1993.
9. H. Kung and J. Robinson. On optimistic methods for concurrency control. *ACM TODS*, 6(2), 1981.
10. T. Liu, C. Curtsinger, and E. Berger. Dthreads: Efficient deterministic multithreading. In *SOSP*, 2011.
11. S. Lu, S. Park, E. Seo, and Y. Zhou. Learning from mistakes: A comprehensive study on real world concurrency bug characteristics. In *ASPLOS*, 2008.
12. T. Merrifield and J. Eriksson. Conversion: Multi-version concurrency control for main memory segments. In *EuroSys*, 2013.
13. C. Minh, J. Chung, C. Kozyrakis, and K. Olukotun. STAMP: Stanford transactional applications for multi-processing. In *IISWC*, 2008.
14. M. Olszewski, J. Ansel, and S. Amarasinghe. Kendo: Efficient deterministic multithreading in software. In *ASPLOS*, 2009.
15. C. Papadimitriou. The serializability of concurrent database updates. *JACM*, 26(4), 1979.
16. K. Ravichandran, A. Gavrilovska, and S. Pande. DeSTM: Harnessing determinism in STMs for application development. In *PACT*, 2014.
17. W. Ruan, Y. Liu, and M. Spear. STAMP need not be considered harmful. In *TRANSACT*, 2014.
18. N. Shavit and D. Touitou. Software transactional memory. *Dist. Comp.*, 10(2), 1997.
19. M. Spear, M. Silverman, L. Dalessandro, M. Michael, and M. Scott. Implementing and exploiting inevitability in software transactional memory. In *ICPP*, 2008.
20. A. Thomson, T. Diamond, S. Weng, K. Ren, P. Shao, and D. Abadi. Calvin: Fast distributed transactions for partitioned database systems. In *SIGMOD*, 2012.
21. J. Wamhoff, C. Fetzer, P. Felber, E. Rivière, and G. Muller. FastLane: Improving performance of software transactional memory for low thread counts. In *PPoPP*, 2013.
22. A. Welc, B. Saha, and A. Adl-Tabatabai. Irrevocable transactions and their applications. In *SPAA*, 2008.
23. R. Yoo, C. Hughes, K. Lai, and R. Rajwar. Performance evaluation of Intel® transactional synchronization extensions for high-performance computing. In *SC*, 2013.