Fractal Coherence: Scalably Verifiable Cache Coherence

Meng Zhang, Alvin R. Lebeck e Daniel J. Sorin

(43rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO), dec. 2010)

Resumo por Carla Négri Lintzmayer – RA 134042

Um protocolo de coerência de cache deve ser bem projetado e inspecionado para que não haja falhas no sistema que for utilizá-lo. Protocolos atuais permitem requisições pendentes e operações concorrentes, levando a vários estados de transição e um comportamento não-determinístico, tornando difícil a verificação do projeto para garantir sua confiabilidade. O uso de simulações não verifica todos os estados possíveis; a verificação formal automatizada é possível e pode superar o problema da simulação, mas possui ainda alguns problemas como o "problema da explosão de estado", que é causado pelo fato do método ser exaustivo e "explodir" o espaço de estados exponencialmente com o aumento do número de *cores*.

O artigo propõe uma metodologia de projeto que permite que os protocolos sejam verificados com o uso de ferramentas formais automatizadas existentes. A base está na teoria dos fractais, pelo fato de um fractal ser uma forma que pode ser dividida em partes onde cada parte é uma cópia reduzida do todo. Os autores criaram uma classe chamada *Coerência Fractal* de protocolos de referência que podem ser verificados por indução: se um sistema grande B tem comportamento fractal e um sistema menor A (que pode ter sua coerência comprovada) é cópia reduzida de B, então pode-se provar a coerência de B com base na de A.

A coerência de cache de um sistema fractal é verificada formalmente garantindo dois passos[1]. Os autores apresentam, além da teoria, a implementação de um protocolo de coerência fractal específico denominado *TreeFractal* para mostrar que a metodologia de projeto fractal é viável e explicam como utilizar duas ferramentas automatizadas de verificação para realizar esses dois passos:

1. **O sistema mínimo deve ser coerente em cache.** Um sistema mínimo inclui todos os tipos diferentes de componentes usados em sistemas maiores. Sua modelagem deve capturar o comportamento do protocolo de coerência de cache e garantir que ele fique dentro da capacidade de verificação das ferramentas atuais para evitar o problema da explosão de estado.

   O sistema mínimo implementado tem formato de árvore binária e possui dois nós básicos e uma Tag de Topo. Cada nó básico tem um *core*, duas caches privadas L1 e L2, uma porção de memória compartilhada e um controlador de coerência. A Tag de Topo é o "pai" dos dois nós básicos. Os autores utilizaram o verificador chamado *Murphi* para verificar a coerência de cache desse sistema mínimo. *Murphi* levou três horas para verificar o sistema e foram explorados 12.031.400 estados.

2. **O sistema inteiro deve possuir comportamento fractal.** Isso garante que a coerência será mantida quando o sistema for escalado. Como o sistema é construído por iteração, basta verificar a equivalência entre os níveis 1 e 2, mostrando que o "mundo exterior" não vê diferenças entre eles.

   O sistema mínimo implementado pode ser escalado para qualquer sistema com N nós adicionando o que os autores chamaram de Tags Internas entre a Tag do Topo e os nós básicos. Essa estrutura é uma árvore binária. As Tags são interfaces que dão suporte ao comportamento fractal mantendo cópias das tags das caches e dos estados de coerência dos seus filhos. No sistema como um todo, os nós básicos são as folhas da árvore, os nós internos são as Tags Internas e a raiz é a Tag do Topo.

   O comportamento fractal do sistema implementado foi verificado utilizando um verificador chamado *Bisimulator*, que retornou `true` indicando que os sistemas comparados eram de fato equivalentes.

Por fim, foram realizados experimentos para mostrar que *TreeFractal* não apresenta uma degradação de desempenho. Ele foi comparado com os protocolos *Snooping* e *Directory*. Os três utilizaram os mesmos parâmetros arquiteturais e vários *benchmarks* foram testados. De modo geral, *TreeFractal* tem um desempenho comparável ao *Snooping* e ao *Directory*, superando-os em alguns *benchmarks* em determinadas situações (por exemplo, com 2 ou 4 *cores*, *TreeFractal* se saiu melhor em praticamente todos os *benchmarks*). O mais importante é que o desempenho de *TreeFractal* é comparável com os protocolos utilizados e ainda garante a corretude do sistema por meio de métodos formais de verificação.

[1] É apresentada uma prova por indução do porque esses dois passos são suficientes.