Agendamento com Reconhecimento de Utilização de Cache para Multicore

Ana Flávia Freiria, Raissa Nunes, Pedro Brassi, Gustavo Andrade
Universidade Federal de Alfenas

1 Introdução

Tema: Agendamento com reconhecimento de utilização de cache para multicore.

O artigo aborda o problema recorrente em sistemas com processadores multicores que compartilham cache, ao mesmo tempo apresenta algumas formas de correção já existentes e também uma forma inovadora criada pelos autores chamada CUAS (cache utilization aware scheduling). Uma estratégia para otimizar o uso de cache compartilhado em processadores multicore, reduzindo a contenção e melhorando o desempenho.

2 Problema principal

O artigo trata de uma técnica para melhorar o desempenho em sistemas com múltiplos núcleos (*multicore*) que compartilham o mesmo cache. Nesses sistemas, múltiplas tarefas podem competir pelo uso do cache, gerando contenção (*cache contention*). Essa competição resulta em uma utilização ineficiente, aumentando a taxa de falhas no cache (*cache misses*) e prejudicando o desempenho.

Problemas destacados:

- Métodos tradicionais baseados no tamanho do conjunto de trabalho (*Working Set Size* - WSS) ou na quantidade de falhas por instrução (*Misses per Instruction* - MPI) não consideram a sensibilidade ao cache.
- Design inadequado de escalonamento resulta em subutilização ou contenção excessiva do cache.

Minimizar a contenção no cache compartilhado (LLC) é fundamental para melhorar o desempenho, especialmente em processadores multi-core. Quando vários núcleos compartilham o mesmo LLC, o espaço de cache pode se tornar um gargalo, principalmente quando aplicativos com diferentes demandas de localidade de dados competem pelo mesmo espaço. Isso pode levar a evacuações desnecessárias de dados, aumentando os erros de cache e reduzindo o desempenho do sistema.

Para resolução alternativa dessa situação, uma abordagem interessante envolve entender como os aplicativos utilizam o cache e, com base nisso, aplicar técnicas inteligentes para otimizar a distribuição e o uso do LLC: Uma dessas técnicas é o agendamento inteligente. Nela, os aplicativos são classificados de acordo com o quanto eles dependem do cache. Por exemplo, alguns aplicativos, chamados de ”LLC-friendly”, têm um uso do cache que não prejudica tanto o desempenho quando compartilham o espaço com outros, enquanto outros aplicativos precisam de uma quantidade significativa de cache para funcionar corretamente. Quando esses aplicativos ”famintos por cache”competem pelo mesmo espaço, podem causar degradação de desempenho. Ao usar um agendador que distribui os aplicativos de maneira mais inteligente, baseado nessa classificação, é possível reduzir a contenção.

Outra abordagem importante é o particionamento de cache, que implica dividir o LLC em várias ”partes”, alocando porções específicas para diferentes núcleos ou aplicativos. Isso impede que um aplicativo ”roube”a memória de outro, o que poderia acontecer em um cache compartilhado sem essa divisão. Esse método é eficaz quando diferentes cargas de trabalho têm exigências distintas de cache, permitindo que cada uma tenha a quantidade de espaço que precisa para funcionar de maneira eficiente sem prejudicar outras. Isso é especialmente útil em cenários onde certos aplicativos têm grandes conjuntos de dados e exigem mais cache, enquanto outros podem operar bem com menos espaço.

Ao melhorar a forma como o cache gerencia os dados, podemos reduzir a ”congestionamento”no LLC, o que contribui para uma performance mais consistente,especialmente quando múltiplos núcleos estão acessando os dados ao mesmo tempo.

3 Objetivo do Artigo

O artigo busca desenvolver e validar um novo método de escalonamento de tarefas em processadores multicore chamado *Cache Utilization-Aware Scheduling* (CUAS), com os seguintes objetivos específicos:

- **Validar a eficácia da abordagem CUAS:** Realizar testes experimentais em uma plataforma real, comparando o método proposto com outras estratégias populares, como o escalonamento padrão do Linux e o método CiPE.
- **Minimizar a contenção no cache compartilhado (LLC):** Reduzir os conflitos de acesso simultâneo ao cache entre tarefas executadas em diferentes núcleos.
- **Considerar as características das tarefas:** Avaliar habilidades de interferência (capacidade de uma tarefa impactar outras) e anti-interferência (resistência a impactos de outras tarefas).
- **Maximizar o desempenho do sistema:** Reduzir o tempo total de execução das tarefas e melhorar a utilização dos recursos de hardware sem sacrificar o equilíbrio da carga entre os núcleos.
- **Superar limitações dos métodos tradicionais:** Introduzir uma abordagem mais precisa e dinâmica para classificar tarefas e planejar o uso do cache, indo além de métricas simplistas como tamanho do conjunto de trabalho (WSS) ou *misses per instruction* (MPI).

4 Solução dos Autores

Os autores, com o objetivo de minimizar a contenção de cache, projetaram um novo agendamento de tarefas com reconhecimento de cache, denominado *Cache Utilization-Aware Scheduling* (CUAS), que inclui duas partes principais: o **classificador de aplicativo** e o **agendador de tarefas**.

O **classificador de aplicativo** considera a capacidade de anti-interferência e interferência de uma tarefa. Para isso, foram desenvolvidos dois microbenchmarks específicos para o experimento:

- **Attack (ATT):** possui forte capacidade de interferência. Na implementação, o ATT polui aleatoriamente e intensivamente todas as linhas de cache.
- **Defend (DEF):** possui forte capacidade anti-interferência. O DEF acessa sequencialmente cada linha de cache.

Para avaliar a capacidade anti-interferência de uma tarefa, os autores despacharam o ATT e a tarefa para núcleos diferentes, mas compartilhando o mesmo cache. Da mesma forma, co-agendaram a tarefa com o DEF para avaliar sua capacidade de interferência.

Com base nos resultados do co-agendamento com os microbenchmarks, os autores classificaram a capacidade anti-interferência e interferência de cada aplicativo. A **pontuação não saudável** de um aplicativo é definida como a soma de suas pontuações de anti-interferência e interferência. Aplicativos com pontuações não saudáveis mais altas possuem uma capacidade anti-interferência mais fraca e uma capacidade de interferência mais forte, e vice-versa.

Nos testes realizados com o CUAS, foram utilizados o *benchmark* SPEC CPU2006 e um processador Intel Core 2 Quad, comparando o desempenho do CUAS com o CiPE (*Cache Interference-aware Priority Execution*) e o escalonamento padrão do Linux.

Vale destacar que, embora diversos autores tenham realizado estudos e análises sobre a contenção de cache, empregando diferentes métodos de avaliação, nenhum deles considerou simultaneamente a contenção inter-*core* e intra-*core*, permitindo também que as tarefas não fossem limitadas. Nesse aspecto, o método CUAS se destaca, superando as abordagens existentes.

O **classificador de aplicativo** inicialmente estima a contenção de cache e a interferência entre os aplicativos. Em seguida, o **agendador de tarefas** utiliza os resultados do classificador de aplicativo como entrada para despachar adequadamente as tarefas para os núcleos, melhorando o desempenho do sistema em tempo de execução.

Nota: O CiPE é uma abordagem de escalonamento com propósito similar ao CUAS, mas considera apenas o nível de interferência entre tarefas para determinar como elas devem ser co-escalonadas.

4.1 Classificador de Aplicação

O classificador de aplicação utiliza funções matemáticas e os microbenchmarks **DEF** e **ATT** para categorizar as tarefas com base na sua capacidade de interferência e anti-interferência.

Função de Interferência

A capacidade de interferência de uma tarefa é definida pela seguinte equação:

$$I_i = T_{i,d} - A_d$$

onde:

- I_i: denota a capacidade de interferência da tarefa.
- $T_{i,d}$: é o tempo de execução do **DEF** quando ele é co-agendado com a tarefa.
- A_d: é o tempo de execução do **DEF** quando ele é executado sozinho.

O **DEF** tem uma forte capacidade anti-interferência porque acessa sequencialmente cada linha de cache, apresentando um bom atributo de localidade. Quanto maior o valor de I_i, mais agressivamente a tarefa interfere com a tarefa co-agendada.

Função de Capacidade de Anti-Interferência

A capacidade de anti-interferência de uma tarefa é definida pela equação:

$$AI_i = T_{i,a} - A_d$$

onde:

- AI_i: é a capacidade de anti-interferência da tarefa.
- $T_{i,a}$: é o tempo de execução do **ATT** quando ele é co-agendado com a tarefa.
- A_d: é o tempo de execução do **DEF** quando ele é executado sozinho.

O **ATT** tem uma forte capacidade de interferência porque polui aleatoriamente e intensivamente todas as linhas de cache. Quanto maior o valor de AI_i, mais fraca é a capacidade anti-interferência da tarefa.

Pontuação Não Saudável

A pontuação não saudável de uma tarefa H_i é definida como a soma de I_i e AI_i:

$$H_i = I_i + AI_i$$

Uma pontuação H_i mais alta indica que a tarefa tem maior impacto negativo no desempenho do sistema e maior tempo de execução quando co-agendada com outras tarefas.

4.2 Agendador de Tarefas

Para essa etapa, haviam m tarefas e um processador multicore com n caches L2. Cada dois núcleos compartilhavam um cache L2 e cada tarefa era associada a uma pontuação não saudável que apresenta seu nível de interferência. Para reduzir a contenção entre caches intra-core, o agendador agrupou tarefas com pontuações mais altas de não integridade um um core e as mais baixas em outro. O Algoritmo mostra como as tarefas são classificadas por suas pontuações não saudáveis em ordem decrescente. O agendador de tarefas primeiro determina p (número máximo de tarefas dentro de um núcleo) e Si rebaixa o i-ésimo cache compartilhado. Com base em p, o agendador de tarefas determina a tarefa executada no primeiro núcleo (C0) de cada cache compartilhado Si (linhas 3 a 9). Em seguida, determina a tarefa executada no segundo núcleo (C1) de cada cache compartilhado Si (linhas 10 a 17) considerando a contenção de cache inter-core e intra-core.

Algoritmo Agendador de Tarefas

O seguinte algoritmo descreve o funcionamento do agendador:

```
p = teto de m / (2 * n);
for i = 1 to n do
    if (m > p)
        atribuir as primeiras p tarefas não saudáveis ao C0 de Si;
        m = m - p; // remove tarefas despachadas
    else
        atribuir tarefas ao C0 de Si e sair;
    endif
end for

for i = n to 1 do
    if (m > p)
        atribuir as primeiras p tarefas não saudáveis ao C1 de Si;
        m = m - p; // remove tarefas despachadas
    else
        atribuir tarefas ao C1 de Si e sair;
    endif
end for
```

Funcionamento

O agendador de tarefas determina as tarefas a serem atribuídas ao núcleo C_0 para cada cache compartilhado S_i nas linhas 3 a 9, priorizando as tarefas com pontuações mais altas de não saúde. Em seguida, nas linhas 10 a 17, atribui as tarefas restantes ao núcleo C_1 de forma a equilibrar a carga e minimizar a contenção de cache inter-core e intra-core.

5 Resultados

Para avaliar o desempenho do método CUAS, os autores testaram cenários com diferentes quantidades de tarefas, aumentando de quatro para oito. Por exemplo: WL1: foram incluídas duas instâncias de lbm, duas de libquantum, duas de bzip2 e duas de milc.

Ao comparar o CUAS com o método CiPE e com o agendamento padrão do Linux. No CiPE, as tarefas são ordenadas com base em suas pontuações de interferência (CIS), da maior para a menor. A tarefa com a maior pontuação é emparelhada com a de menor pontuação, a segunda maior com a segunda menor, e assim sucessivamente.

Já o CUAS funciona em duas etapas. Primeiro, ele classifica as tarefas com base em sua capacidade de resistir a interferências (anti-interferência) e na tendência de causar interferências (interferência). Em seguida, as tarefas são alocadas estrategicamente nos núcleos, levando em conta o impacto tanto da contenção de cache entre núcleos (inter-core) quanto dentro de um mesmo núcleo (intra-core). Vide resultado em gráfico:

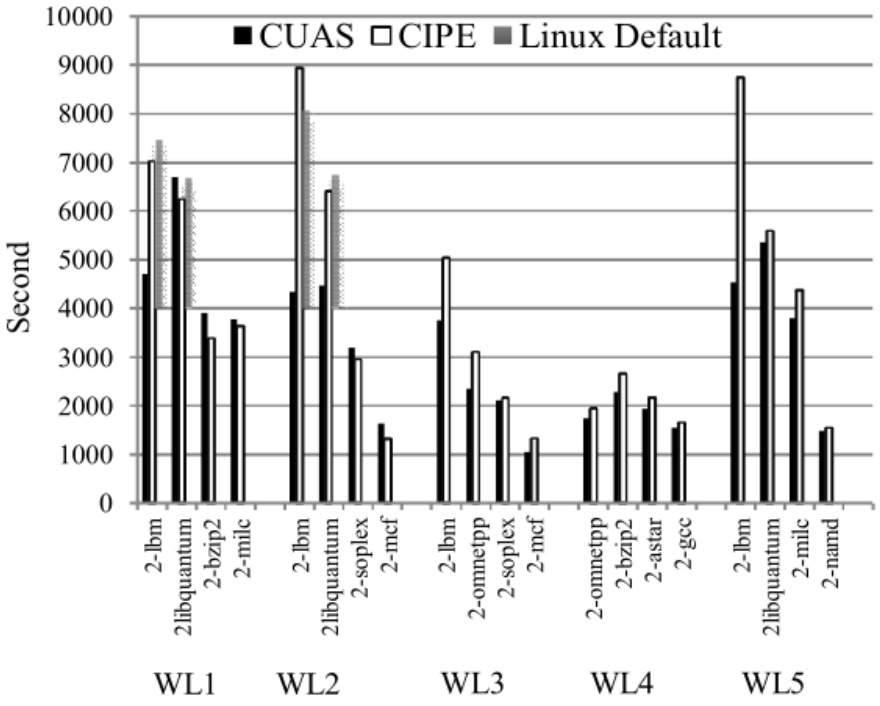

Figura 1: Imagem retirada diretamente do artigo original

Os resultados mostraram que, para os conjuntos WL1 e WL2, o uso do CiPE levou a uma degradação significativa de desempenho, causada pela alta contenção de cache entre lbm e libquantum. Em contraste, o CUAS agrupou essas tarefas de forma estratégica, reduzindo a contenção de cache e melhorando o desempenho. No caso de WL2, conseguimos reduzir o tempo total de execução em até 46% em relação ao CiPE e 43% em relação ao agendamento padrão do Linux.

Os resultados dos experimentos mostram que o CUAS é altamente eficaz, reduzindo significativamente a contenção no cache compartilhado.

6 A título de curiosidade

- **Diferentes algoritmos de escalonamento:** No contexto de agendamento com reconhecimento de utilização de cache para sistemas multicore, a importância de algoritmos de escalonamento reside na necessidade de otimizar o uso de caches locais de cada núcleo. Quando tarefas são alocadas sem considerar o cache, há um aumento no tráfego de dados entre núcleos, o que pode resultar em latência e desperdício de recursos. Algoritmos que reconhecem e priorizam a reutilização de dados no cache podem reduzir significativamente esse tráfego e melhorar o desempenho. Esse reconhecimento é especialmente crítico em sistemas multicore, onde o desempenho depende fortemente de uma boa distribuição das tarefas e do aproveitamento dos recursos de memória local, como caches, para minimizar o tempo de execução e aumentar a eficiência geral entre os algoritmos mais analisados, estão:
  - **HEFT (Heterogeneous Earliest Finish Time):** Prioriza tarefas com base no custo de computação e comunicação, alocando-as para minimizar o tempo de conclusão.
  - **CPOP (Critical Path On a Processor):** Foca nas tarefas críticas e as aloca de maneira a reduzir o tempo total de execução, especialmente em sistemas heterogêneos.
  - **PCH (Path Clustering Heuristic):** Agrupa tarefas em clusters e seleciona recursos com base no tempo de término mínimo para cada grupo.

  Estudos demonstram que a escolha do algoritmo depende de fatores como a arquitetura do sistema (homogêneo ou heterogêneo) e a natureza das tarefas. Em ambientes homogêneos, algoritmos como FIFO podem ser eficazes, enquanto em sistemas heterogêneos, estratégias mais complexas, como HEFT ou CPOP, tendem a oferecer melhores resultados em termos de tempo de

execução e uso de recursos. Torna-se viável relatar que esses algoritmos são frequentemente avaliados por meio de simulações usando ferramentas como o SimGrid, que permite comparar diferentes estratégias sem a necessidade de experimentos reais, o que ajuda a otimizar as decisões de escalonamento antes da implementação prática.

Link para visualização: IME-USP.

- **Outro artigo relacionado ao tema:** De forma relacionada, é abordado no artigo "Cache-aware static scheduling for hard real-time multicore systems based on communication affinities" um ponto de vista sobre a organização da execução de tarefas em sistemas multicore de tempo real rigoroso, onde é essencial que as tarefas sejam concluídas dentro de prazos definidos. O problema central está nos atrasos causados pela comunicação entre tarefas que utilizam o cache do processador. Quando essa comunicação não é bem planejada, ocorrem acessos desnecessários à memória principal, que é muito mais lenta, comprometendo o cumprimento dos prazos.

  Para resolver isso, numa abordagem correlata, os autores propõem um método de escalonamento estático, que decide previamente onde e quando cada tarefa será executada, considerando dois fatores importantes: as afinidades de comunicação e o uso eficiente do cache. As tarefas são representadas como nós em um grafo, e aquelas que trocam muitos dados entre si são agrupadas para reduzir o custo de comunicação. Assim, o cache é aproveitado de maneira mais eficiente, minimizando acessos demorados à memória principal.

  Esse método também garante que todas as tarefas respeitem seus prazos, mesmo levando em conta possíveis atrasos causados pela comunicação. Como resultado, o sistema ganha em desempenho, reduz o consumo de energia e melhora o cumprimento dos prazos, o que é crucial para aplicações críticas, como carros autônomos, dispositivos médicos e sistemas aeronáuticos.

Referências

Artigo principal usado como referência:
Cache Utilization Aware Scheduling for Multicore Processors

Artigos complementares de estudo: *Cache-Aware Static Scheduling for Hard Real-Time Multicore Systems* e *Avoiding Cache Thrashing Due to Private Data Placement in Last-Level Cache for Manycore Scaling*

Agradecimento à comunidade LaTex

19/11/2024