Cache Utilization-Aware Scheduling for Multicore Processors

Grupo: Ana Flávia Freiria, Raissa Nunes,
Pedro Brassi, Gustavo Andrade

Professor: Eliseu César Miguel

Introdução

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

- Problemas em processadores multicore compartilhando cache(LLC - last level cache).
- Técnicas atuais limitadas (WSS e MPI).
- Proposta inovadora: **Cuas (Cache Utilizaations Awware Scheduling.**

Problema Principal

- Contenção no cache compartilhado:
  - Reduz desempenho.
  - Aumenta falhas no cache.
- Soluções existentes não lidam bem com interferências inter-core e intra-core.
- 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.

Objetivos do Artigo

- Validar eficácia do método **CUAS.**
- Minimizar contenção no LLC (last level cache).
- Considerar as características das tarefas.
- Maximizar o desempenho do sistema.
- Superar limitações de metódos tradicionais

Solução Proposta: CUAS

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.

- Componentes principais:
  - Classificador de aplicação;
  - Agendador de Tarefas.
- Avalialiação de tarefas:
  - **Interfência**(tarefa atrapalha outras).
  - **Anti-interferência**(resistência à interferência).

Classificador de Aplicação

O classificador de aplicação 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.

Função de Interferência

A capacidade de interferência de uma tarefa é definida por:

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

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

Função de Capacidade de Anti-interferência

A capacidade de anti-interferência de uma tarefa é definida por:

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

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

Pontuação não Saudável

A pontuação não saudável de uma tarefa H_i *é definida como* :

$$H_i = I_i + AI_i$$

Uma pontuação 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

Agendador de Tarefas

- Baseado na pontuação das tarefas.
- Estratégias:
  - Reduzir contenção entre caches intra-core e inter-core.
  - Tarefas menos "saudáveis" agrupadas estrategicamente.

Algoritmo

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.

Resultados

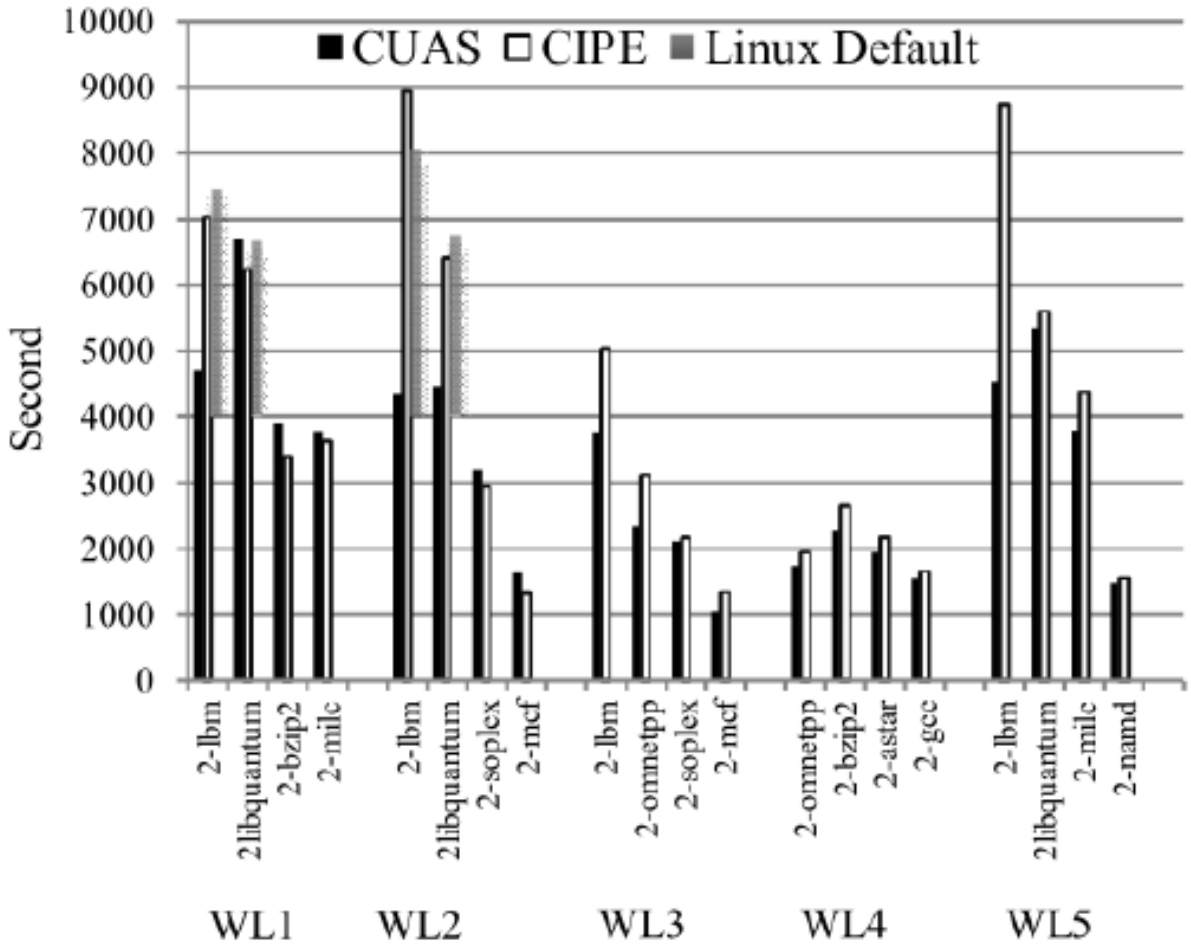

Figure: Performace de diferentes agendamentos

Resultados

- Cenários de teste:
  - Processador: Intel Core2 Quad Q8400 (4 cores–2.66GHZ).
    - Os cores foram divididos em dois grupos, e cada grupo dividia uma memória cache L2 de 2MB com linhas de 64 bytes.
  - Benchmarks: SPEC CPU2006.
  - Comparação com **CiPE** e agendador padrão do Linux.
- Melhorias:
  - WL2: redução de até **46%** em relaçao ao **CiPE**, e de **43%** em relação ao **Linux**.

Curiosidades

- Algoritmos semelhantes:
  - 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.
- Artigos Relacionado: **"Cache-aware static scheduling for hard real-time multicore systems"**.

Conclusão

- **CUAS** se mostrou eficaz na resução da contenção de cache.
- Melhor desempenho em comparação a métodos tradicionais.
- Impacto positivo em sistemas multicore

Referência

Artigo principal usado como referência:

Cache Utilization-Aware Scheduling for Multicore Processors Edward T.-H. Chu, Wen-wei Lu edwardchu, g9917722 @yuntech.edu.tw Department of Computer Science and Information Engineering, National Yunlin University of Science Technology, Taiwan

Artigo complementar 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

The End