

Minimizing Cache Conflicts by Optimal Task Placement

Sebastian Altmeyer



Work in progress

Minimizing Cache Conflicts by Optimal Task Placement
is underlying ongoing research by
Gernot Gebhard and Sebastian Altmeyer.

Content

Observations and Goal

Underlying Observation

Goal

What to exploit?

Optimization

1. Collect Input Data
 2. Build Objective Function
 3. Optimize Memory Placement
 4. Link Instruction/Data
- Coming to a Real Implementation

Summary

Underlying Observation

Observation

current timing analysis methods assume tasks running to completion;

analysis of preemptive systems is considered too complex

-

obviously, different tasks placements in memory lead to different task interferences and different performances

Goal

influence memory placement of tasks to:

1. increase overall performance
(decrease number of cache-misses)
2. make cache-lines of hard-real time tasks persistent (during one run)
3. - or at least, know which cache-line will be evicted

achieve these goals by exploiting

- ▶ structure of cache
- ▶ inter-task dependencies
- ▶ structure of tasks

Used System

- ▶ embedded system with one processor
- ▶ preemptive scheduling
- ▶ LRU-cache (LRU not mandatory, but gives best results)
- ▶ no virtual memory
- ▶ task set with many inter-task dependencies
- ▶ mixed soft/hard real-time constraints
- ▶ ideas usable for data, instruction and unified cache

How to ensure cache-persistence?

A cache line is persistent, if it cannot be evicted by other data during preemption!

Remember: instructions/data not persistent in the cache all the time, but only during one run of the task

- ▶ for hard real-time tasks: no eviction is allowed
- ▶ for soft real-time tasks: eviction should be as unlikely as possible

Conflicting Tasks / Non-Conflicting Tasks

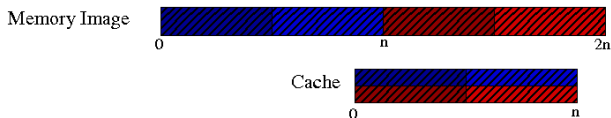
Not all tasks can evict a given cache-line of task a .

Task b might evict cache-lines of task a if:

- ▶ both tasks map to the same cache set
- ▶ task b can preempt task a

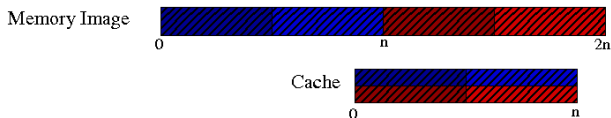
Conflicting Tasks / Non-Conflicting Tasks: Example

- ▶ 4 different tasks
- ▶ only tasks with similar colors can't preempt each other
- ▶ all tasks have size $n/2$
- ▶ cache-size of n



Conflicting Tasks / Non-Conflicting Tasks: Example

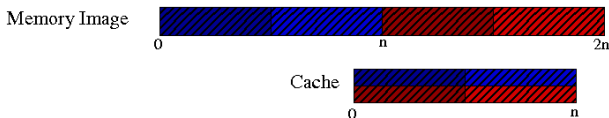
1. Case: conflicting tasks map to same cache sets
→ cache miss due to preemption possible



Conflicting Tasks / Non-Conflicting Tasks: Example

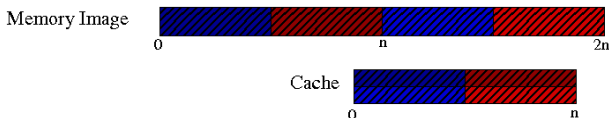
1. Case: conflicting tasks map to same cache sets

→ cache miss due to preemption possible



2. Case: conflicting tasks map to different cache set

→ cache miss due to preemption not possible



Structure of the cache

Does every conflict lead to eviction?

Within a 4-way set associative LRU cache, up to 4 accesses might map to same address.

Only a 5th access might lead to eviction of other cache-lines.

Therefore, a certain number of conflicts is allowed.

Notice: LRU is easy to analyze and therefore best choice

Splitting Tasks to Finer Granularity

Only for instruction cache:

- ▶ split tasks into its procedures
- ▶ loop code more 'important' than non-recurring code
- ▶ sequential code segments of one task cannot preempt each other

Each task might consist of several procedures. Using this granularity, more flexible solution are allowed and possible.

Optimization Steps

1. collect input data
2. build objective function
3. optimize memory allocation with respect to the objective function
4. link code and data to meet optimal memory placement

1. Collect Input Data

What details about the tasks/system influence task interference on cache?

system

- ▶ cache-size (line-size, number of sets)
- ▶ associativity

task set

- ▶ size of tasks/data
- ▶ which tasks might preempt each other
- ▶ period of tasks, execution time
- ▶ structure of tasks (loop, non-recurring code segments)
- ▶ ...

2. Build Objective Function

use collected data to build an objective function
for each possibly evicted cache-line add a certain penalty/weight

- ▶ for all hard real-time tasks, weight = ∞
- ▶ for all soft real-time tasks, weight inversely proportional to its period

objective function: $\min \sum weight$

Remark: different objective function possible

3. Optimize Memory Allocation

Use optimization method to find optimal memory allocation.

This far we used:

- ▶ ILP: very time consuming, but gives best results
- ▶ (simple form of) simulated annealing: good performance but only average results

Which method to use is still undecided, tests have to show.

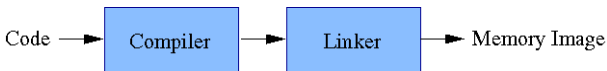
4. Link Instruction/Data

currently building prototype extension to cross-compiler for ARM9,
to link instructions/data according to computed mapping

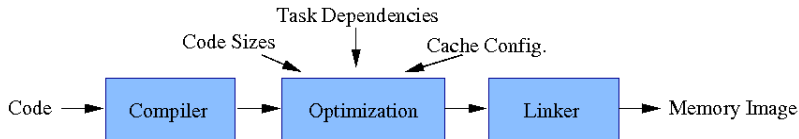
this far, mapping can be changed by hand

Extension to compiler/linker needed

former tool-chain:



new tool chain:



Summary

- ▶ (will be an) semi-automatic step
- ▶ no code changes needed (in contrast to other methods)
- ▶ improve overall performance
- ▶ we know, which cache-lines may be evicted
- ▶ (precise) timing analysis with preemption becomes possible

but:

- ▶ highly depends on the task-set

Current Status

- ▶ theoretical framework exists
- ▶ useful memory mapping can be found
- ▶ memory mapping can be done 'by hand'
- ▶ now implementating compiler-extension
- ▶ using MPARM to simulate results

Thanks for your attention!