Finding Optimal L1 Cache Configuration for Embedded Systems

Andhi Janapsatya, Aleksandar Ignjatovic, Sri Parameswaran

School of Computer Science and Engineering

UNSW

The University of New South Wales
Sydney • Australia
Outline

- Motivation and Goal
- Existing Work
- Cache exploration algorithm
- Energy model
- Experiment and results
- Conclusion
Motivation

- Cache memories energy make up a large portion of the total processor energy consumption.
- Modern processor (i.e. ASIP) allows the configuration of its cache memory.
Motivation

Current Processor design flow typically configure the largest cache memory allows by the energy and space constraint to ensure maximum performance.

But largest cache memory does not guarantee best performance or lower energy consumption.
Performance (g721enc)

![Graph showing performance data for cache misses vs. total execution time (ms) with various cache sizes indicated. The x-axis represents cache misses ranging from 1E+02 to 1E+09, and the y-axis represents total execution time in milliseconds ranging from 100 to 10000.]
Performance (g721enc)

- **Total Execution Time (ms)**
- **Cache Misses**

Largest cache

Best Performance
Energy (g721enc)
Energy (g721enc)

- Largest cache
- Lowest energy
Goal

- Explore the effect of various cache configuration given an application trace and the computer architecture energy and performance specification.
- To configure the optimal cache parameter for minimum system energy consumption and/or best performance.
Existing Work

- Dinero IV was developed by Jan Edler and Mark Hill.
- Dinero IV is a single processor cache simulation tool.
- Its purpose is to simulate cache memory and estimate the number of cache misses given a program trace.

The inclusion property states that ‘cache C2’ is a subset of ‘cache C1’ if all content of ‘cache C2’ is contained in ‘cache C1’.
Inclusion property

- We exploit the inclusion property in two ways.
- **Inclusion 1** is when a small cache is a subset of a larger cache.
- **Inclusion 2** is when a cache with less associativity is a subset of caches with larger associativity.
For cache size = 1, all memory address will be mapped to the same cache address.
Inclusion 1 example

If an entry cause a cache hit when within the cache size = 1, then a cache hit is guarantee in the cache size = 2, due to inclusion property.
Likewise, a cache hit within the cache size = 1 or the cache size = 2 cache configurations will result in a hit within the cache size = 4.
Inclusion 1 example

And a cache hit with cache size equals 1, 2, or 4 will also result in a cache hit in cache configuration cache size = 8.
Inclusion 2

In a direct-mapped cache, all entry that maps to the current cache set can only be stored in a single location.

Cache Assoc. = 1

Most recently used element in the cache.

Top of linked list
Most recently used element in the cache.

For a 2-way associative cache, the entry is mapped to two locations. With LRU policy, a cache hit in the most recently element guarantee a cache hit.
Likewise, for larger associativity cache configurations.
Existing Work

Hill in 1989 developed the stack algorithm based on the inclusion property to simulate multiple levels of cache and investigate the effect of different cache associativity.

Sugumar in 1995 developed a variation to the stack algorithm called the binomial trees.
Existing Work

Li in 2004 extends the work by Sugumar by adding compression for the binomial trees algorithm.

Other existing cache simulation methodologies exploit parallel processing unit [Nicol, 1995] or a multiprocessor system [Heidelberger, 1990].
Contributions

We propose a custom data structure to maintain the cache content of multiple cache configurations.

The data structure is a combination of the forest and linked-list.
Contributions

The binomial trees in [10] has the complexity of:

- \( O(\log_2(N) + 1) \times A \) to find a cache entry in the binomial tree.
- and \( O(\log_2(N) + 1) \times A \) to update the content of the binomial tree.

N is the size of the cache and,
A is the associativity of the cache.
Contributions

- Our custom data structures has the following complexity:
  - \([O(\log_2(N) + 1) \times A]\) to search for the requested cache entry.
  - \([O(\log_2(N) + 1)]\) to maintain the content of the data structure.
Cache Exploration Algorithm

For each entry in the trace, the algorithm needs to calculate the cache location in all the different cache configurations and determine whether the trace entry is the same as the existing entry in the cache (i.e. cache hit) or different to the existing entry in the cache (i.e. cache miss).
Custom Data Structure

Array

Forest

Linked List
Cache Exploration Algorithm

- The memory address is used as an index into an array for inspecting the current content of that particular cache entry to determine the hit/miss.
- The size of the array is determined by the smallest sets of all the cache configurations that is to be explored.
Cache Exploration Algorithm

The figure above illustrate the used of the memory address as an index of the array.
Cache Exploration Algorithm

- Content of an array is a pointer to the top node of a tree.
- The tree contains a subset of cache configuration where the inclusion property can be exploited.
Cache Exploration Algorithm

- Each node in the tree point to a linked list.
- Each entry in the linked list contain the tag entry that is to be compared with the existing trace address.

Cache Assoc. = 1
Most recently used element in the cache.

Cache Assoc. = 2

Cache Assoc. = 4
Least Recently Used element in the cache.
System Model

System is modeled as shown below,
Energy Model

Total energy is calculated by summing the energy cost of:

- CPU energy
- I-cache access energy
- D-cache access energy
- I-cache miss * memory access energy
- D-cache miss * memory access energy
Performance Model

Total execution time is calculated by summing the time taken to:

- I-cache access
- D-cache access
- I-cache miss * memory access
- D-cache miss * memory access
Experiments

- Program trace were generated using SimpleScalar 3.0d [Burger, 1997]
- Benchmarks were taken from Mediabench suite [Lee, 1997]
- Accuracy of the cache estimation algorithm is verified against Dinero IV estimation result [Edler]
Experiments

- Accuracy of the power model is verified against wattch tools [Brooks, 2000]
- Performance of the cache estimation algorithm is compared against the time taken for Dinero IV to evaluate the same number of cache configurations.
Results

- Trace size (millions)
- Total Execution Time (min)

- **DineroIV**
- **Our Estimation Tool**
Results

cjpeg cache configuration design space.
Results

\[\text{g721dec cache configuration design space.}\]
Results

pegwitenc cache configuration design space.
Conclusion

Utilize cache inclusion property to implement an efficient cache parameter design space exploration tool.

Results were verified against Dinero IV.

System energy and performance calculation allow the selection of cache parameters for optimal energy or performance.
Thank You