

Targeted Configurable Caches

John Shield, Peter Sutton, John Williams

*School of Information Technology and Electrical Engineering
The University of Queensland
Brisbane, Queensland, 4072, Australia*

{jshield,p.sutton,jwilliams}@itee.uq.edu.au

Abstract—Embedded systems are increasingly moving from running single applications to running application sets. Targeted configurable caches extend the concept of configurable caches by specially supporting the critical applications of an embedded system. Other configurable caches attempt to be completely generic but a targeted configurable cache can outperform them due to reduced overheads and better partitioning from only supporting the needed configurable options for the critical applications. A comparison between a conventional configurable cache and a targeted configurable cache demonstrates targeted performance improvement to that of a conventional configurable cache.

I. INTRODUCTION

Configurable caches [1-4] can have their settings adjusted during run time to modify various attributes like block size, way size and number of ways. These have been shown to provide significant performance benefits over a non-configurable cache when tested on single applications.

Embedded systems, including FPGAs with microprocessor cores, are increasingly moving from supporting single applications to providing multiprogramming environments for several core applications. These core applications are the important workload and it is appropriate to optimise the system for just this application set. This has been used by Viana et al. [5] with cache subsetting to improve the speed of their cache tuning algorithm. The improvement is from limiting the searched configurations for an application set to previous results from a similar benchmark, as only a subset of the cache configurations are actually needed.

While this previous work modifies the search algorithm to handle just the application set, there has been no work that considers modifying the cache architecture of the configurable cache to match an application set. All the previous configurable caches [1-4] used an architecture designed to be generic with adjustable runtime settings.

This paper proposes the idea of **targeted configurable caches**, configurable caches which are targeted for the core applications of an embedded systems. To create targeted configurable caches the application set is analysed to determine the optimal cache settings that will be needed and the cache architecture is customised to provide performance improvements for those settings. The improvements come from better partitioning of the architecture to the available resources and reduced hardware requirements as not all configurable options may be needed for an application set.

To differentiate between types of configurable caches, a **targeted configurable cache** has its hardware optimised

during the synthesis time and performs further optimisation of settings during runtime. A **conventional configurable cache** only optimises its settings at runtime.

The contributions of this paper are in presenting the benefits of targeted configurable caches for embedded systems, theory on how to create different variations of configurable caches targeted for an application set and how targeted configurable caches can be integrated into a design framework that works for a multiprogramming environment.

The remainder of the paper will explain the design considerations for creating targeted configurable cache architectures in section II, how the targeted configurable caches can fit within a design framework in section III, and in section IV, implementation results to compare the performance improvements of conventional configurable caches with targeted configurable caches.

II. TARGETED CONFIGURABLE CACHE DESIGN THEORY

The naive approach to creating a targeted configurable cache is to remove any configurable hardware that is not needed by the application set. Study of the architecture of previous configurable caches [1-4] shows that while this is possible for some of the cache adjustments, some adjustments of characteristics are integrated with others, e.g., way concatenation [6], where way size and associativity are linked together to conserve resources. Furthermore, other configurable options like replacement policy [7] were ignored due to the impracticability of implementing many different replacement policies.

To construct targeted configurable caches, further research work was needed to assist in the design. The first step was to evaluate configurable cache architecture to determine the practical limitations of configurable caches and identify areas where configurability can hinder rather than assist performance. Then a method was required for creating variations in configurable caches. Design constraints are used to set limitations for the design and these limitations can be used to define how variations of configurable cache architecture can be created.

The limitations of conventional configurable caches are studied in sub-section A below; sub-section B proposes using available memory resources as a design parameter for configurable cache architecture creation.

A. Limitations of Conventional Configurable Caches

The restrictions with a conventional configurable cache stem from the hardware that allows it to be configured. By not

customizing the configurations for an application set, it instead needs to cater for the largest selection of possibilities within the available resources.

A conventional configurable cache must use a power of 2 number of ways to allow for way concatenation [6]. Ways must be merged multiple times to provide a larger cache way size at the expense of the number of ways. Cache ways cannot be merged in an irregular non-power of 2 pattern as it results in different cache way sizes. Allowing for different sized ways also impractical as the usual cache addressing methodology does not work for different sized ways. On the other hand, a targeted configurable cache is not restricted to a power of 2 number of ways so it is easier to partition into a fixed available memory.

Another issue with a conventional configurable cache is its fixed minimum block size [8]. While it can vary between large and small block sizes by changing the fetch size, it has a fixed **physical line size** equal to the smallest block size that it intends to provide configurability for. The physical line size determines the number of cache tags needed by the system and depending on the design can also change the flushing speed. The number of cache tags also impacts on the fast memory usage which means allowing for a smaller line size will increase memory usage and can reduce the size or number of ways that can be implemented with the remaining memory.

Aiming for generic cache adjustments prevents certain kinds of cache modifications from being used due to implementation overheads. An example of this is when using different replacement policies. Each replacement policy has its own format and memory requirement for storing information about cache line usage [9]. This can make the overheads in implementing multiple replacement policies high and not worthwhile, but implementing just two can be feasible [7] for an analysed workload.

Consequently, a conventional configurable cache will sometimes require more flushing or provide less overall cache size because it was built for smaller block sizes than what was needed. Furthermore, it can not implement many of the possible configurable options demonstrated in research due to overheads from the implementation and low performance improvements when being used for generic workloads.

B. Memory Constrained Configurable Cache Design

When implementing caches, there is a limited amount of fast memory available either due to cost or FPGA platform restrictions and so memory usage can be the most important constraint in the design. The memory constraint can pose a problem when an attribute is adjusted as additional memory may be required to make the change. This has never been considered before for configurable cache design as the cache architecture was always fixed.

When memory resources are used as a constraint for creating configurable cache architecture, it becomes critically important to understand how memory usage can change from cache configuration.

Cache memory, number of tags, and tag bit-width were identified as the main sources of memory usage and Table I shows the memory resources several cache attributes impact.

Table I has block size divided into physical line size and fetch size. For most non-configurable caches line size and block size mean the same thing and fetch size is equal to the block size. However, for configurable caches the fetch size can be used as a virtual block size, which allows for a block size larger than the physical line size of the cache.

TABLE I
RESOURCES USED BY CACHE ATTRIBUTES

	Cache Memory	Number of Tags	Tag Bit-Width
Ways	✓	✓	
Way Size	✓	✓	✓
Block Size	Physical Line Size	✓	
	Fetch Size		
Replacement Policy			✓
Read and Write Buffers	✓		
Write Policy			
Cache Indexing			

The key issue is not the amount of memory resources being used, but whether the amount of memory used changes from adjusting a cache attribute. The attributes that vary in the amount of memory used when adjusted require additional memory that can only be obtained by trading-off other attributes to use less memory.

Based on these findings cache attributes can be divided into two categories. **Singular attributes** that can be adjusted by themselves, like fetch size, cache indexing [10] and sometimes replacement policy (depending on the policies), and **linked attributes** that require another attribute to be changed at the same time to balance their change in resource usage.

When designing caches that can change configuration with linked attributes, it is always a case of trading memory to the detriment of one attribute so that another can be improved. This is because the attribute adjustments that do free up memory generally reduce the performance for that attribute. For example, reducing the number of ways or way size frees up memory but reduces the performance. For block size, increasing the physical line size can free more memory and increase performance in some cases, but it is better to change the fetch size to do this instead of the physical line size.

III. TARGETED CONFIGURABLE CACHE FRAMEWORK

The architecture designs of targeted configurable caches are also dependant on the framework in which they are applied. Our previous work [11, 12] presents such a framework. Two aspects of the framework which need to be considered are described below.

Firstly, configuration overheads from runtime cache configuration are significantly more important than in the previous work [1-4] as the framework applies configurable caches to a preemptive environment where there are context switches. In conventional configurable cache research the configuration time is never used as part of the design as the configuration time was considered small compared to the time that the application runs for. Previous configurable cache research did not cover configuration overheads but it does describe cache tuning times between 28 to 3.4 seconds [13].

However, documentation on the Linux preemptive system states a context switching frequency of 200 milliseconds [14]. Consequently, when dealing with implementations on a preemptive system the runtime cache configuration overheads needs to be part of the decision making for when to change the cache.

Secondly, an automatic system is needed to create the targeted configurable cache architectures so that the creation of the caches can become part of the design flow. While the singular attributes changes can be added modularly to the design, the linked attributes require specialised cache architectures to allow for changes. Requiring the designer to hand design each targeted configurable cache is impractical. Instead, creating several targeted configurable cache templates that handle linked attributes and that can be automatically modified to suit the situation is a more viable approach.

A summary of the analysis work for configurable cache overheads are studied in sub-section A below, while the cache templates to be used in the automatic design flow are briefly covered in sub-section B.

A. Reducing Cache Configuration Overheads

Configuration overheads are more important when implementing a configurable cache on a preemptive system as the application being run will change frequently.

The overheads of a targeted configurable system may seem unpredictable due to the uncertain architecture being used. However, many of the possible architecture overheads in cache adjustment can be avoided or are small enough to be ignored. There are unavoidable overheads and are what determine the configuration overhead for a configurable cache architecture.

The unavoidable overheads stem from three main factors: loss of the data memory, loss of tag memory and incompatible tag formatting. Incompatible tag formatting or loss of tag memory will also generate a corresponding loss of data memory that it references. The losses in tag or cache memory come about as certain targeted configurable cache adjustments require some of the cache memory to be converted into tag memory or vice versa.

In many cases not all of the data memory or the tag memory was lost during a cache adjustment. There are configurable cache architecture designs that allow for reuse of tag and cache memory, but space precludes an explanation here. Other cache adjustments require a change in the tag formatting that prevents data reuse so all the tags and data are lost during the cache change.

The calculations for the configuration overheads are based on the invalidation time for the tag memory and write back time of the data memory. The amount of tag and cache memory that needs to be invalidated and flushed depends on how much of the memory is converted between tag and cache memory and can't be reused.

B. Targeted Configurable Cache Templates

Targeted configurable caches support a subset of configurations from the design space. The subset of cache configurations that are needed is based on analysis of the

resources available and the software applications being used. This in turn changes the cache hardware that is implemented.

While there are many ways to implement caches, the bulk of the hardware that is actually needed for targeted configurable caches can be reduced down to a few cache templates that support certain kinds of linked attribute cache adjustments.

Five main configurable cache architecture templates have been developed around trading cache memory and tag memory to allow for attributes changes in associativity, way size and physical line size. Cache adjustments that are singular attributes can be added onto the templates modularly.

IV. PROOF OF CONCEPT

An implementation comparison was used to provide a proof of concept for targeted configurable caches. It aims to compare the improvement that conventional configurable caches can provide over a generic cache with the improvement that targeted configurable can provide. Three different caches were created: a generic cache, a conventional configurable cache and a targeted configurable cache. These were created for the design specifications of 24kbytes available BRAM and the Mibench [15] benchmarks. The caches were built to interface with a MicroBlaze [16] softcore processor running uClinux [17] on a Xilinx ML401 board.

The generic cache used was a direct mapped cache that was 16kbytes in size and used a block size of 32 bytes. This is the default cache used by the Microblaze and NIOS softcore processors.

The conventional configurable cache used a physical line size of 16 bytes, allowing for block sizes of 16, 32, 64 and 128. It used way concatenation to provide 1 way at 16kbytes way size, 2 way at 8kbytes way size, and 4 way at 4kbytes way size. This setup is what is described in previous work on conventional configurable caches [4].

The targeted configurable cache used the dynamic cache switching framework to generate the cache architecture and find the cache configuration settings. Generation of the VHDL cache designs and integration of the configuration settings was done by hand as the automation work is still in the prototype stages. The automatic cache configuration generator was set to a maximum of 24kbytes of BRAM allowed for cache memory and tag memory and with architecture creation for the Mibench as the application set. The resulting design that was generated gave a configurable cache allowing for 5 ways, 4kbytes way size and block size variations from 16 to 128 bytes.

The experimental measurements were made using hardware timers and 40 data samples for each combination of application and configuration. The recorded samples were averaged out to reduce variations due to the operating system.

Fig. 1 shows all the benchmarks that gave performance variations due to the cache used. In all cases the targeted configurable cache performed the same or better than the conventional configurable cache. For this experimental setup the targeted configurable cache improvement was due to better partitioning of the cache into the available memory.

These results provide a proof of concept for targeted configurable caches and demonstrate the possible benefits of targeted configurable caches over conventional configurable caches. Due to different benchmarks and the performance metric instead of power metric being used in the results, the conventional configurable cache results in Table II were much lower than the improvement results reported in previous work [4, 6]. Using more suitable benchmarks where conventional configurable caches performed well will show greater improvement values for targeted configurable caches.

TABLE II
TARGETED CONFIGURABLE CACHE IMPROVEMENT

Applications	Conventional Config. Cache Improvement (%)	Targeted Cache Improvement (%)	Experimental Error +/- (%)
Rijndael De	4.20	4.32	0.08
Dijkstra	3.40	7.44	0.11
Stringsearch	3.05	3.34	0.05
Jpeg De	1.55	2.35	0.18
Tiffdither	1.53	1.62	0.26
ADPCM De	1.47	1.73	0.12
Blowfish De	0.93	0.97	0.03
Blowfish En	0.91	0.95	0.03

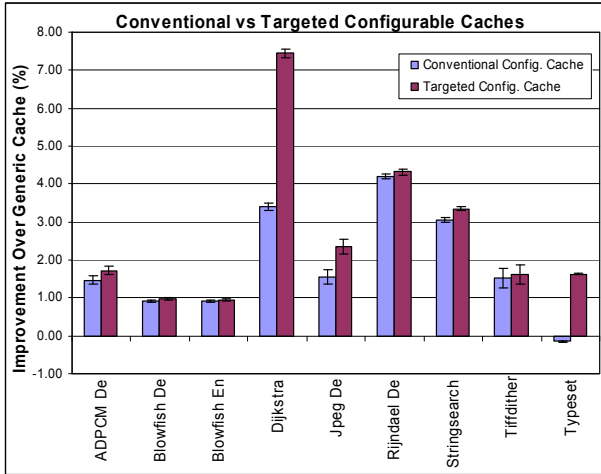


Fig. 1. Improvement of targeted configurable over conventional configurable

V. CONCLUSIONS AND FUTURE WORK

The new concept of targeted configurable caches has been introduced with an explanation of how it overcomes the flaws with a conventional configurable cache implementation. Previous work only changed the cache configuration settings, but did not use software analysis to customise the hardware implementation of the configurable cache architecture.

A summary of the design methodology for targeted configurable caches has been given, introducing the new design concepts needed for targeted configurable cache architecture design. Memory constraints are used to drive the architecture design, requiring new hardware architectures for trading available memory between cache attributes. The importance of cache configuration overheads was introduced with a summary of the discovered concepts for reducing the overheads. The methodology for design automation of targeted configurable caches was also briefly covered.

Finally a proof of concept implementation was used to measure the performance gains. It showed a targeted configurable cache is better or equivalent in all cases to a conventional configurable cache.

Future work will look into expanding the experimental setup to include more benchmarks and full automation of the hardware design for targeted cache architecture.

REFERENCES

- [1] D. Sheldon, R. Kumar, R. Lysecky, F. Vahid, and D. Tullsen, "Application-specific customization of parameterized FPGA soft-core processors," in *Proceedings of the 2006 IEEE/ACM international conference on Computer-aided design*. San Jose, California: ACM, 2006, pp. 261-268.
- [2] R. Balasubramonian, D. Albonesi, A. Buyuktosunoglu, and S. Dwarkadas, "Memory hierarchy reconfiguration for energy and performance in general-purpose processor architectures," in *Proceedings of the 33rd annual ACM/IEEE international symposium on Microarchitecture*. Monterey, California, United States: ACM, 2000, pp. 245-257.
- [3] A. Gordon-Ross, F. Vahid, and N. Dutt, "Fast configurable-cache tuning with a unified second-level cache," in *Proceedings of the 2005 international symposium on Low power electronics and design*. San Diego, CA, USA: ACM, 2005, pp. 323-326.
- [4] C. Zhang, F. Vahid, and R. Lysecky, "A self-tuning cache architecture for embedded systems," *ACM Trans. Embed. Comput. Syst.*, vol. 3, pp. 407-425, 2004.
- [5] P. Viana, A. Gordon-Ross, E. Keogh, E. Barros, and F. Vahid, "Configurable cache subsetting for fast cache tuning," in *Proceedings of the 43rd annual conference on Design automation*. San Francisco, CA, USA: ACM, 2006, pp. 695-700.
- [6] C. Zhang, F. Vahid, and W. Najjar, "A highly configurable cache architecture for embedded systems," *SIGARCH Comput. Archit. News*, vol. 31, pp. 136-146, 2003.
- [7] R. Subramanian, Y. Smaragdakis, and G. H. Loh, "Adaptive Caches: Effective Shaping of Cache Behavior to Workloads," in *Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture*: IEEE Computer Society, 2006, pp. 385-396.
- [8] A. V. Veidenbaum, W. Tang, R. Gupta, A. Nicolau, and X. Ji, "Adapting cache line size to application behavior," in *Proceedings of the 13th international conference on Supercomputing*, 1999, pp. 145-154.
- [9] H. Al-Zoubi, A. Milenkovic, and M. Milenkovic, "Performance evaluation of cache replacement policies for the SPEC CPU2000 benchmark suite," in *Proceedings of the 42nd annual Southeast regional conference*, 2004, pp. 267-272.
- [10] T. Givargis, "Improved indexing for cache miss reduction in embedded systems," in *Design Automation Conference, 2003. Proceedings, 2003*, pp. 875-880.
- [11] J. Shield, P. Sutton, and P. Machanick, "Dynamic Cache Switching in Reconfigurable Embedded Systems," in *Field Programmable Logic and Applications, 2007. FPL 2007. International Conference on, 2007*, pp. 111-116.
- [12] J. Shield, P. Sutton, and P. Machanick, "Analysis of Kernel Effects on Optimisation Mismatch in Cache Reconfiguration," in *Field Programmable Logic and Applications, 2007. FPL 2007. International Conference on, 2007*, pp. 625-628.
- [13] A. Gordon-Ross, F. Vahid, and N. Dutt, "Automatic Tuning of Two-Level Caches to Embedded Applications," in *Proceedings of the conference on Design, automation and test in Europe - Volume 1: IEEE Computer Society*, 2004, pp. 10208.
- [14] "Chapter 4 Processes," retrieved 10 August 2009 from <http://tldp.org/LDP/tlk/kernel/processes.html>
- [15] M. R. Guthaus, J. S. Ringenberg, D. Ernst, T. M. Austin, T. Mudge, and R. B. Brown, "MiBench: A free, commercially representative embedded benchmark suite," in *Workload Characterization, 2001. WWC-4. 2001 IEEE International Workshop on, 2001*, pp. 3-14.
- [16] Xilinx, "MicroBlaze Processor Reference Guide," in *UG081*, June 2006.
- [17] uClinux, "http://www.uclinux.org/," January 2007.