ANALYSIS OF KERNEL EFFECTS ON OPTIMISATION MISMATCH IN CACHE RECONFIGURATION

John Shield, Peter Sutton, and Philip Machanick

School of Information Technology and Electrical Engineering
The University of Queensland
Brisbane Australia 4072
email: xue@itee.uq.edu.au, p.sutton@itee.uq.edu.au, philip@itee.uq.edu.au

ABSTRACT

The effect of kernel operations on cache optimisations in a soft-core reconfigurable system is important for dynamic cache switching design. Considering kernel operations changes the subset of cache configurations that would be chosen for dynamic cache switching and also the decisions on when to cache switch. The results show that kernel operations can skew the effectiveness of application driven cache optimisations up to 20% over the original execution time. This skew is shown by mapping the performance of the applications both with and without kernel operations. The majority of the kernel operations is due to trap events generated by system calls like memory allocation or file reading. A cache configuration analysis methodology for fast searching of the design space is also explained and was used to find relevant changes due to kernel interference.

1. INTRODUCTION

Dynamic cache switching [1] requires the performance difference between custom caches to be optimised as part of the switching algorithm. In dynamic cache switching, is it not enough to have an optimised cache that is better than the norm. The cache that is being switched to has to give a performance benefit over the currently active optimised cache. While significant work has been done on cache optimisations for applications [2-7], the optimisation difference (*optimisation mismatch*) between customised caches has not been explored.

The key difference from previous work in cache reconfigurability; is previous work was for general purpose processors, or was implemented on a softcore in a way best suited for general purpose usage. The implementations did not take into account that only a subset of applications are frequently run on an embedded system.

Dynamic cache switching means the reconfigurability can be specialised for the main applications of the system at synthesis time of the softcore processor. Instead of the full range of cache reconfigurations like in other systems, a subset is implemented. This specialisation of the cache reconfigurability means better usage of hardware resources, one of the main limiting factors on an embedded system.

Research into optimisation mismatch of customised caches is critical for synthesis tools deciding the subset of cache configurations to cater for on the embedded system.

Kernel effects on hardware cache optimisations are not present in previous research on cache reconfigurability [2-7], and there is no research on the effect of the kernel on the optimisation mismatch. These kernel effects are important in dynamic cache switching as the majority of the kernel operations are scattered throughout the process. The dispersion of kernel operations makes cache switching on every kernel event untenable. Instead kernel operations need to be considered as part of the application analysis.

It is obvious that the kernel will have an affect on cache optimisations. The question is whether the kernel operations could change the subset of cache configurations chosen and also the decisions on when to cache switch. These things are determined by optimisation mismatch analysis.

Methodology for determining optimisation mismatch and exploration of the effect of the kernel on optimisation mismatch are presented in the following sections.

2. SUPPORT TOOLS FOR ANALYSIS

The support tools consisted of a hardware test platform, a custom data gathering board, and a cache simulator. An overview of the support tools that were built for this research can be seen in Fig. 1.
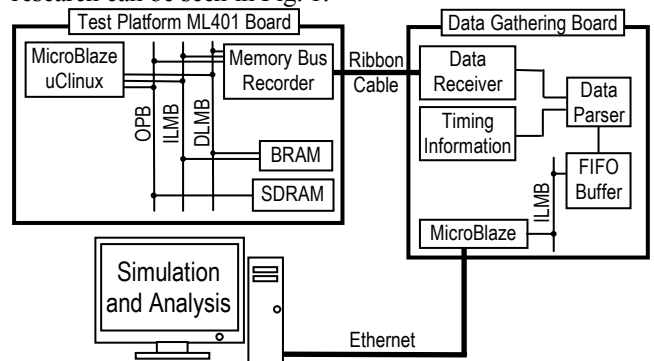

Fig. 1. Hardware for capturing cache traces

The test platform used a Xilinx ML401 board, running uClinux [8] on a MicroBlaze softcore processor [9]. A custom hardware module was built to measure the bus accesses on the board during runtime and transfer the data to a data gathering board. This is downloaded to the PC for analysis, where the data is filtered to remove memory latencies but preserve other latencies.

A simulator was built that adds the timing behaviour of the cache being explored. It returns the number of simulated clock cycles that data samples took to run with each cache. This is the *execution time* of a sample (of an application) when run on a cache.

The sample lengths vary in length so execution time for a result is first divided by the control cache's execution time, where the control cache was chosen with theoretically ideal cache settings. This gives a normalised execution time, an *execution time percentage* (**ETP**) relative to the control cache. To account for variances in the application and the operating system, multiple samples were taken so a mean ETP value is used with all the presented data although not explicitly mentioned.

ETP difference (**ΔETP**) is used to show a *performance* difference, either between two applications for the same cache, or one application between various caches.

At the moment the main performance metric for dynamic cache switching is execution time. It can be extended to consider energy at a later date.

2.1. Mapping Multi-Dimensional Cache Behaviour

Current work on cache optimisations concentrates on algorithms for cache optimisations, which cannot easily be applied to new research in analysis.

This problem led to a methodology for graphing multiple dimensions of cache attributes as a fast way for narrowing down the design space. For the cache behaviour of an application, four dimensions of cache attributes were considered in this experiment. These were Memory Size, Ways, Block Size, and Replacement Policy.

Graphing only one cache attribute at a time is problematic as the results are usually different depending on what the other cache attributes are set to. This is because the behaviour of each dimension of cache attribute is usually not independent. Instead results need to be organised in a way that can easily show application based cache behaviour modified by multiple attributes. This is useful for design work as a single optimal solution neglects data which is not explicitly placed in the algorithm, things a designer will often need to consider.

A straight forward graph everything approach does not work well. As shown in Fig. 2 the individual graphs for ways (a) are combined to create the scatter graph in (c). This makes it difficult to analyse attributes as it is not obvious what attribute the trends in the graph belong to.
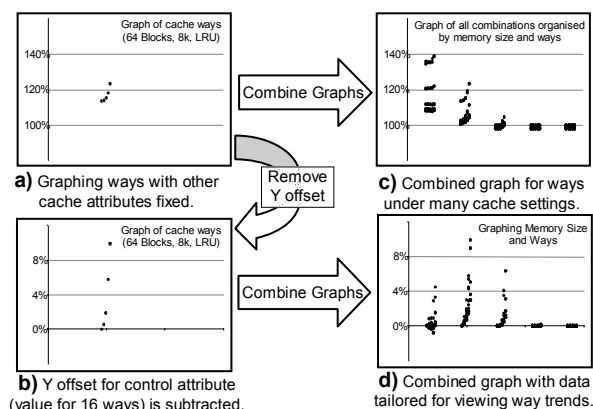

Fig. 2. Mapping Multi-Dimensional Cache Behaviour

In (c) the trends in memory size and block size confuse what is happening with the ways. This variance caused by other cache attributes can overwhelm the changes of interest for the attribute being studied.

To fix this, what is needed is the variation of the single cache attribute with how it interacts with the other cache settings, but without the unrelated differences caused by the other cache settings.

As shown in (b) and (d), this is done by subtracting each data point from the control value (of the attribute being graphed) for the same cache settings. This removes the Y axis offset of the original single dimension graph and leaves only changes that relate directly to the attribute of interest.

To further order the data, the data points are organised by two attributes on the X axis. Multiple graphs are made with the data reorganised along the X axis by the main attribute of interest and a second attribute. This helps to show how varying one cache attribute can affect how another cache attribute behaves.

The result is one set of data points tailored for each of the cache attributes, with multiple graphs used to redisplay those data points in different ways to highlight relationships between multiple cache attributes.

The analysis system used UNIX shell scripts to generate mean ETP difference (ΔETP) tables from the simulator results. These tables were then pasted into an Excel template that generated the various scatter graphs.

Finding how cache configuration affects the application is simply a matter of looking through the graphs for correlation of the data points. It will clearly show trends that depend on single and multiple cache attributes. When the data is zero there no correlation with the attributes.

This is used for finding points of interest in the design space or seeing overall trends in optimisation for various cache settings. It is part of the methodology used for analysing the kernel effects. Examples of graphs generated by this technique are shown used in previous work [1]. Future work intends to integrate this methodology with the cost of hardware overheads and possibly energy usage.

3. KERNEL IMPACT ON CACHE OPTIMISATION

Kernel operations occur on context switch, trap events (software interrupts) and IRQs (hardware interrupts). Context switching occurred when other processes were run instead of the application. Trap events occurred when the application calls the kernel to deal with hardware resources. While hardware interrupts were mostly generated by timers.

There are two direct sources of impact on cache optimisations when considering the kernel. First, the cache may flush part of the cached data when the kernel uses the same cache locations. Second, the kernel itself is using the cache for what it is doing and so optimisations will affect how fast the kernel runs.

Indirectly, the kernel also impacts on optimisations with context switches, where it is likely that a large portion of the cached data will be overwritten by another process and that data will need to be refetched.

The main question to be answered is whether kernel effects would change the optimisation mismatch analysis such that the subset of cache configurations chosen and the decisions on when to cache switch are also changed. This is the case when the good solutions of an application with kernel considered are different from the good solutions without the kernel considered.

4. RESULTS

Some benchmarks from the Mibench test suite [10] were ported to MicroBlaze and run on the uClinux. Automotive, network, security and telecomm were chosen as likely subsections of the test suite to be used in an embedded system.

Only the data bus is analysed as the benchmarks were small and an instruction cache wouldn't be heavily utilised. Direct sources of kernel operations will be covered, but not analysis of inter-application interference patterns.
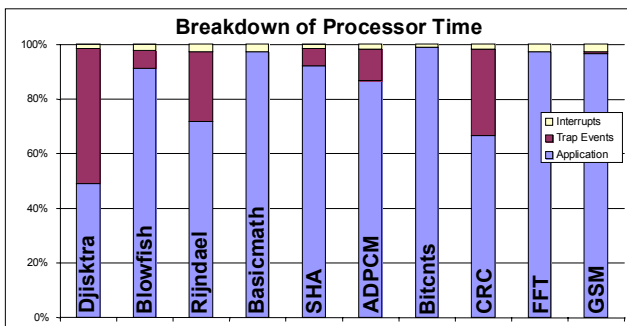
4.1. Initial Analysis

Fig. 3. Processing Time of Kernel Compared to Application

The breakdown of execution time for an application and the kernel operations that occur without a scheduled context switch is shown in Fig. 3. Background kernel processes that are called by the process scheduler are not included in the data as these are handled like another application.

There is a high variability in the impact of the kernel on applications, seen in Fig. 3. Mainly this was due to trap events that are used whenever the application uses kernel level resources. The scheduler operations, which make up the interrupts, have a minimal impact under 3% of the processing time when an application is being run.

Only the applications that have significant kernel operations are relevant for further analysis; Djisktra, Blowfish, Rijndael, SHA, ADPCM, and CRC. Using the methodology for mapping cache optimisations described in section 2.1, the design space was further narrowed down to memory sizes of 2KB and 8KB. These memory sizes were the only ones that were significantly impacted by changes in the cache settings.

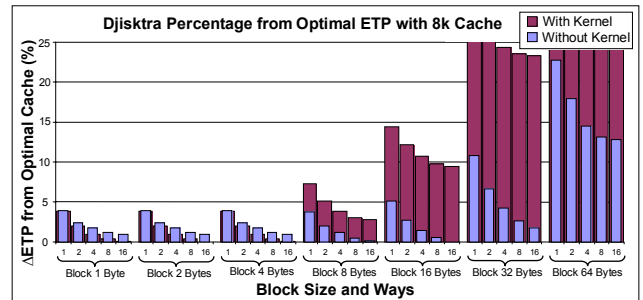### 4.2. Finding the Penalty when Disregarding Kernel

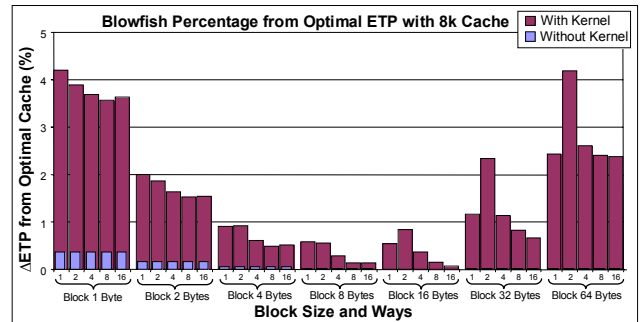**Fig. 4. Djisktra 8k Cache, Percentage from optimal ETP**

Fig. 5. CRC 2k Cache, Percentage from optimal ETP

The results were graphed for the relevant applications under 8KB and 2KB memory sizes as seen in examples Fig. 4 and Fig. 5. The graphs show the ΔETP penalty (from the optimal solution) for the various cache settings and also show how adding the kernel operations change the penalty. In many cases there is significant penalty with settings that previously had little to no penalty. However, this does not take into account the likelihood of a configuration being used.

To decide which cache configurations could realistically be chosen when the kernel is not considered. An

assumption was made that any configuration within 5% ΔETP, from the optimal value, might be used in a design for dynamic cache switching.

This cache selection tolerance factor is a realistic trade off tolerance in dynamic cache switching. Hardware overheads may make it desirable for two applications to use the same custom cache if they both run fairly well on it. Or a slight reduction in performance can be made to reduce hardware resource usage. As it is a trade off tolerance, changes smaller than the maximum trade off still matter.

With this assumption, kernel operations will have an impact on the subset of caches chosen and the cache switch decision; if the configurations meet the 5% tolerance (when kernel is not considered), but are significantly worse when the kernel is considered.

The additional execution time incurred by the kernel can add 42% (Djisktra at 8KB) over the normal execution time when considering all cases. Narrowing the possibilities down to a 5% cache selection tolerance, the Djisktra had an additional 20% execution time, when kernel operations were not considered.
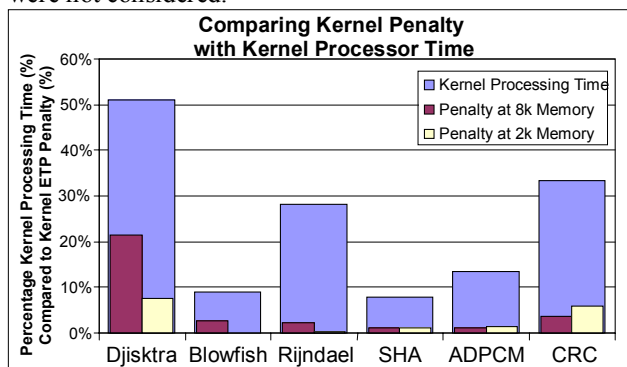

Fig. 6. Comparing Kernel Penalty with Kernel Processing Time

The amount of execution penalty that could be expected from ignoring kernel operations is shown against the amount of kernel processing time in Fig. 6. The penalty involved, depended on how different the cache behaviour of the application is compared to the kernel and how much the kernel overwrites the cache data. Analysis of the Djisktra program revealed that its trap events were generated by memory allocation kernel functions being called. While with the CRC and Blowfish the trap events were due to file reading.

5. CONCLUSIONS

Kernel effects were found to change the optimisation mismatch analysis and consequently change the subset of cache configurations that are chosen and also the decisions on when to cache switch. This happens when application-directed cache optimisations are affected by kernel operations. Kernel operations had impact of 20% additional execution time in one case while almost no impact at all

some other cases. Analysis of the program code showed that significant kernel operations in programs is mainly due to trap events generated by system calls like memory allocation or file reading.

An analysis method for discovering application related cache behaviour was also introduced, which allows multiple dimensions of cache attributes to be graphed. This was used as part of the research and was found to be a good way to narrow down the design space in cache configuration exploration.

Future work that stems from the kernel analysis includes tools that will identify which applications will call a large number of kernel functions, and also whether they will incur high cache penalties from doing so.

6. REFERENCES

[1] J. Shield, P. Sutton, P. Machanick, "Dynamic Cache Switching in Preemptive Systems", in *submitted to 17th Conference on Field Programmable Logic and Applications*, Aug. 2007.

[2] Gordon-Ross, A., Vahid, F., Dutt, N., "Automatic tuning of two-level caches to embedded applications," in *Proc. Design, Automation and Test in Europe Conference and Exhibition, ,* vol. 1, pp. 208-213, 16-20 Feb. 2004.

[3] Gordon-Ross, A., Cotterell, S., Vahid, F., "Exploiting Fixed Programs in Embedded Systems: A Loop Cache Example," *IEEE Computer Architecture Lett. ,* vol. 1, no. 1, pp. 2-2, Jan.-Feb. 2002.

[4] Zhang, C., Vahid, F., Najjar, W., "A highly configurable cache architecture for embedded systems," in *Proc. Computer Architecture, International Symposium on*, pp. 136-146, 9-11 June 2003.

[5] Chuanjun Zhang, Vahid, F., Lysecky, R., "A self-tuning cache architecture for embedded systems," in *Proc. Design, Automation and Test in Europe Conference and Exhibition*, vol. 1, pp. 142- 147, 16-20 Feb. 2004.

[6] Hu, J. S., Kandemir, M., Vijaykrishnan, N., Irwin, M. J., Saputra, H., and Zhang, W., "Compiler-directed cache polymorphism," *SIGPLAN Not.* Vol. 37, no. 7, pp. 165-174, 2002.

[7] R. Balasubramanian, D. Albonesi, A. Buyuktosunoglu and S. Dwarkadas, "Memory Hierarchy Reconfiguration for Energy and Performance in General Purpose Architectures," *Proc. of 33rd Intl. Sym. on Microarchitecture*, pp. 245-257, Dec. 2000.

[8] uClinux, http://www.uclinux.org/, January 2007.

[9] Xilinx, "MicroBlaze Processor Reference Guide", UG081 (v6.0), June 2006.

[10] 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 *Proc. IEEE 4th IEEE International Workshop on Workload Characterization*, Dec. 2001.