Evaluation of Architectural Support for Speech Codecs Application in Large-Scale Parallel Machines

Naeem Zafar Azeemi
Christian Doppler Laboratory for Design Methodology of Signal Processing Algorithms,
Institute of Communications and Radio Frequency Engineering,
University of Technology Vienna, Gusshausstrasse 25/389, A-1040 Vienna Austria
Email: nzafar@nt.tuwien.ac.at

***Abstract*— Next generation multimedia mobile phones that use the high bandwidth 3G cellular radio network consume more power. Multimedia algorithms such as speech, video transcodecs have very large instruction foot prints and consequently stalled due to instruction cache misses. The conflicts in on-chip caches contribute a large fraction of the CPU cycle penalty and hence increase in power consumption. Many commercial tools are developed to minimize such cache misses by adequately placing the frequently called procedures in an application. Followed by profile extraction, these tools use cache line coloring and post compilation techniques for cache hit optimization. The major impediment in the optimal performance of such tools is their static layout profile, which does not consider the dynamic behavior of the application. We propose a methodology called DCP (dynamic code placement) for positioning code at run time for good instruction cache performance and have implemented in high end processors. The dynamic application profile is completely transparent to the developer's code. This technique optimizes the code footprint in memory layout of a program. It improves i-cache mapping to increase the number of cache hits and eventually reduce the CPU stalls. Our optimization is powered with static as well as detail run time profile information that extracts the relevant, temporal behavior of the applications. Moreover, while mapping code in instruction cache, the effect of inter-procedural code positioning is also considered. Improvement over the Pettis and Hansen approach (PH) is also shown in results. Though majority of multimedia applications can be optimized by our framework, application dominated with the function pointers do not work correctly. The technique incurs low overheads and enhances the cache hits architecture correlation. For a range of applications we show that instruction miss rates can be reduced by 19-68%. Using a simple model this corresponds to execution time reduction (23-85%), increase in parallelism (4-53%).**

***Keywords—Wireless applications*, embedded systems, low energy, cache optimization**

I. INTRODUCTION

The next generation wireless will allow mobile computing for applications such as high speed access to the corporate intranet and to the public internet. Fueling this trend towards high speed web browsing with mobile access are the evolution of cellular systems that will bring multi-folded increases in transmission speeds, the development of end terminal devices with human interfaces capable of displaying complex graphical presentations, and increased dependence and preference of networked information. To achieve such universal mobility and flexibility will be necessary. In the same vein, powering becomes more and more difficult as the demand on processing power increases at the order of magnitude. Battery life is now a very strategic feature for every mobile system.

In mobile devices energy dissipation is directly linked to battery size, weight, packaging, cooling and operating time. Multimedia DSP processors are the most lucrative choice to wireless application domain for their optimal performance and small form factors at low energy. The energy efficiency of these systems and subsystems depends heavily on their software design [1,2,3]. For wireless applications, the software design complexity inevitably propagates all the way to the developer. Programs are written to explicitly manage parallelism and to reorder the computation so that the instruction and data working sets fit within the cache. Cache misses make the execution speed of programs slow because they require extra cycles to transfer code or data between the cache and the main memory [2,5,7]. Cache misses consume more energy not only because the main memory is activated but also off-chip traffic increased. Parameswaran [2, 16] mentioned that energy for off-chip driving is very dominating and becoming more dominating, up to 70% of the total chip energy, along with the progress of transistor scales.

We choose to implement efficient cache management scheme based on the application run time profile steered by our energy aware framework, published in [5,9]. , whose add several implementations of energy efficient techniques for embedded DSP compiler [5,10,15] to generate high performance computing applications, they are mentioned below:

- The dynamic application profile is completely transparent to the developer's code. This technique optimizes the code footprint in memory layout of a program.
- Improvement over the Pettis and Hansen approach (PH) is also shown in results.

**This work has been funded by Christian Doppler Laboratory for Design Methodology of Signal Processing*

- Our optimization is powered with static as well as detail run time profile information that extracts the relevant, temporal behavior of the applications
- We demonstrate how compilers can be used to optimize the performance of on-chip instruction caches under compute intensive workloads with DCP.

This paper is organized in five sections: Section II presents the motivation and background of our work, Section III describes the experimental setup and bench mark applications. Case studies are discussed in Section IV, while Section V concluded the whole work.

II. MOTIVATION AND BACKGROUND

Various cache mapping scheme have been developed over the year to improve the cache performance [10,11,12,13,14]. Following two parameters have to be maximized in order to have good cache performance:

Hit Ratio: It must be increased as much as possible the likelihood of the cache containing the memory addresses that the processor wants. Otherwise, the benefit of caching will be lost, because there will be too many misses.

Search Speed: It is desirable to be as quickly as possible if scored a hit in the cache is the goal. Otherwise, small amount of time in every access will be lost during every search.

An unsuccessful attempt to read or write cache is refered as cache miss, which results in a main memory access with longer latency [14]. There are three kinds of cache misses. A cache read miss from an instruction cache generally causes the most delay, because the processor, or at least the thread of execution, has to wait (stall) until the instruction is fetched from main memory. A cache read miss from a data cache usually causes less delay, because independent cache read instructions can be issued and continue execution until the data is returned from main memory, and the dependent instructions can resume execution. A cache write miss to a data cache generally cause the least delay, because the write can be queued and there are few limitations on the execution of subsequent instructions. The processor can continue until the queue is full. In order to lower cache miss rate, a great deal of analysis has been done on cache behavior in an attempt to find the best combination of size, associativity, block size, and so on [13]. Sequences of memory references performed by benchmark programs are saved as address traces. Subsequent analyses simulate many different possible cache designs on these long address traces.

Researchers have categorized the cache miss factors in three broad classes and they are compulsary misses, conflict misses and capcity misses. Compulsory misses are those misses caused by the first reference to a datum. Cache size and associativity make no difference to the number of compulsory misses. Prefetching can help here, as can larger cache block sizes (which are a form of prefetching). Capacity misses are those misses that occur regardless of associativity or block size, solely due to the finite size of the cache. The curve of capacity miss rate versus cache size gives some measure of the temporal locality of a particular reference stream [12, 13, 14]. Note that there is no useful notion of a cache being "full" or "empty" or "near capacity": CPU caches almost always have nearly every line filled with a copy of some line in main memory, and nearly every allocation of a new line requires the eviction of an old line. Conflict misses are those misses that could have been avoided, had the cache not evicted an entry earlier.

In VLIW processors, many of the components in the CPU are not completely utilized during the program execution. Primary reason for such slack is the poor architecture-application correlation [16,17]. This indicate that for an energy efficient application binary there is a need to gather more detailed profiles, containing information about system behavior on various levels. The goal of profiling is to find cause-effect relations between performance phenomena and finally generating an architecture efficient code. Our energy-aware framework [9] embodies a series of profiling stages that enable the optimization process.

III. EXPERIMENTAL SETUP

This section describes the hardware and software used to collects our data and the workload selected. The experiment was conducted on a parallel machine, but outcomes are also applicable to a uni-processor machine with similar processing elements. We use profile monitor [9] to capture a large range of system activity, including dynamic activity.

A. Hardware System

We use a hardware performance monitor to gather uninterrupted reference traces of applications in real-time without introducing perturbation. The performance monitors has one probe connected to the sensing resistor at the core input current of the processor. The probe collects all the references issued by the processor except the slave processors. The trace buffer for each probe can hold up to 10 millions of references. For each reference, the information stored includes:

- The address accessed (32 bits)
- Sample time stamp (24 bits)
- Memory access stamp (6 bits)
- Clock shift stamp (1 bit)
- DMA access stamp (3 bits)
- Bus arbiter control stamps (2 bits)

This can record a trace of an unbounded continuous stretch of the underlying multimedia application.

B. Software System

Table 1, presents our benchmark suite, which consists of a modified version of the MediaBench suite. Our set of applications contains computer-intensive DSP kernels as well

as applications composed of more complex algorithms and data structures.

Table 1. Workload Applications for Profile Monitoring

Applications	Description
m100	Matrix 100x100 multiplications
m200	Matrix 200x200 multiplications
nlivq	Non linear interpolative vector quantization
MPEG-1	MPEG video transcodec
G.721	ADPCM Speech codec
H.264L	Mpeg-4 H.264L Video compression codec

IV. Low Energy Code Placement (LECP) Methodology

Code placement can be employed to minimize mutual instruction replacement and resulting cache misses. We propose a new code mapping scheme which rearranges functions of a signal processing application in memory based on run-time profile so that the mapping of frequently accessed code parts to same cache lines. In contrast to existing code placement methods that rely on purely statistical data such as call graphs; our method exploits the address sequence information inherent in instruction trace data.

The scheme is integrated in an energy aware framework, which have been already discussed in a former publication [9]. Briefly, in this framework we proposed an energy-cycle cost model together with a source-to-source transformation methodology, suitable for embedded systems based on VLIW (very long instruction word) cores. The system level methodology includes generalized energy models for each module, composing the system architecture (processing unit, on-chip/ off-chip memory units, address/ data highway etc.) and SW application parameters. During methodology flow, 'C' source code is successively restructured for a given energy and cycle performance. Further details can be found in [9].

In this work we focus explicitly on energy-cycle cost guarded cache performance improvement. The proposed code mapping approach consists of two steps: trace data processing and function allocation. Trace data has to be collected while executing the target application with typical input test vectors. Address sequence information, however, is essential for cache behavior optimization since occurrence of cache misses is related to the sequence of memory accesses rather than to their mere number. The new method evaluates the address sequence of memory accesses in order to extract only those accesses which can actually lead to cache misses during code execution.

V. Results and Discussion

In this section we evaluate the merits of the proposed dynamic code placement strategy. We embed this strategy in our energy aware framework. The cache miss contribution of individual address block in G-721 codec is depicted in Figure 5.1. G.721 is an ADPCM speech codec. The individual share of each address block shows the percentage of cumulative cache misses in that block. The address range corresponds to the basic code block. The address range 0x001d, 0x032 and 0x029 reflect the major cache misses. A careful look inside the code reveals the deep branching inside the code. A conditional or unconditional branching in a code leads to CPU stall.

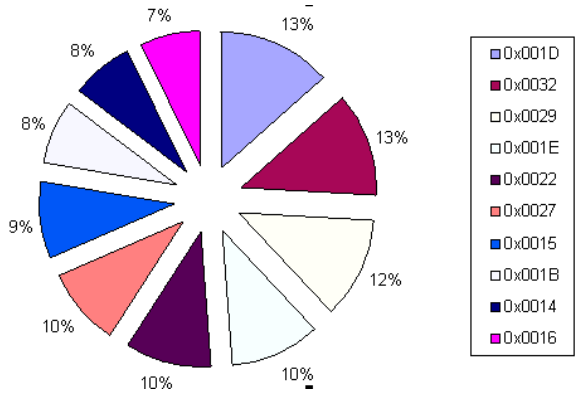

Figure 5.1. Proportional Cumulative Cache Misses in G.721 ADPCM Speech Codec

The improvement offered by DCP over a full address range of nlivq (non-linear interpolative vector quantization) application is more modest for the parallel architecture: 400 averages CPU stalls Figure 5.2. It shown four graphs, top two graphs show the aggregate CPU stalls before and after the application of methodology, while two graphs at the bottom are corresponding power consumption in milliwatts.

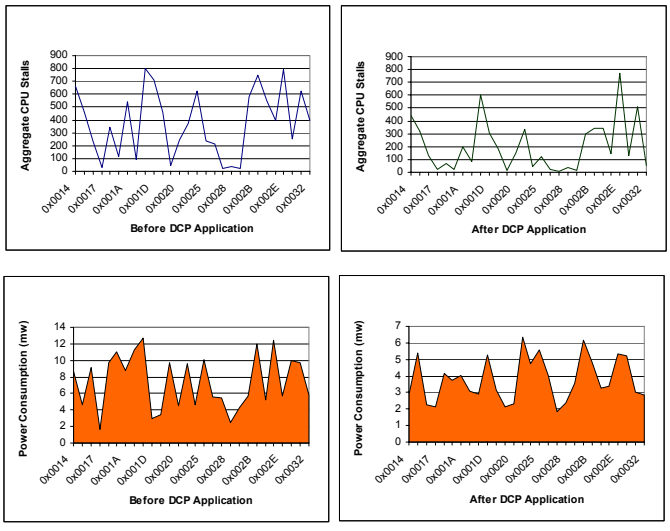

Figure 5.2, Comparative Power Performance Evaluation of nlivq application for DCP strategy.

To better understand the overall impact of DCP, we assess the individual code block cache performance at the address scale. The top-left illustrate the cache trace during the

execution of part of address range. A highest peak at address range 0x001c show the 802 stalls, it give rise to power peak up to 12.67 mW. Next three higher power consumption peaks can be observed at 0x002E, 0x002B, 0x001D, their corresponding stalls and power consumptions are (790, 5.64 mW), (748, 11.9 mW) and (706, 2.94mW) respectively.

The discrepancy between the number of stalls and magnitude of power consumption is quite obvious, where 790 stalls lead to 5.64 mW, while 748 stalls lead to 11.9 mW. The reason is, these results should be observed in conjunction with the area under the power consumption curve. Hardware needs time to settle down the input power, the switching capacitance caused by the instruction activity incurs charge/ discharge time delays and elongate the area under the power consumption curve, add an offset to total power eventually.

DCP reduce down the energy consumption of the wireless speech codec G.721 with a factor of 50% and reduce CPU stalls 65%.

VI. CONCLUSIONS

This paper addresses the problem of power saving in 3G wireless systems. Though next generation hardware designers are using advanced power saving technique to help minimize integrated circuit and system power consumption. Yet these techniques do not yield significant power savings without intelligent energy conservation software to exploit them effectively. We demonstrate how compilers can be used to optimize the performance of on-chip instruction caches under compute intensive workloads with DCP. This paper makes three contributions.

Firstly, it characterized the spatial locality pattern in the multimedia applications. It is shown that substantial loop locality can open the opportunities to save CPU stall cycles.

Secondly, application expression profiling at all levels is considered, for capturing and correlating performance problems across multiple execution layers.

Thirdly, our scheme consistently outperform existing PH algorithm by a significant amount.

In addition, our framework offers an out-of-the-box solution, which is a simple and faster design environment, because designers do not have to engineer so aggressively for reduced power consumption.

ACKNOWLEDGMENT

This work is supported by ÖAD-Pakistan scholarship program initiated by Prof. Dr. Atta-ur-Rahman chairman HEC and Federal Minister Pakistan.

The author would like to thank Prof. Dr. Markus Rupp, Prof. Dr. Arpad Scholtz and Christian Doppler Laboratory at Institute of Communication and Radio-Frequency Engineering, Vienna University of Technology for their support and kind input during this work.

REFERENCES

[1] J. B. Chen and B. D. D. Leupen. “Improving Instruction Locality with Just-In-Time Code Layout,” Proc. of the *USENIX Windows NT Workshop*, Aug. 1997.

[2] N. Zafar Azeemi, “A Framework for Architecture Based Energy-Aware Code Transformations in VLIW Processors," Proc. of the IEEE International Symposium on Telecommunications (IST 2005) pp.393-398. Sep. 2005.

[3] R. Cohn, D. Goodwin, P. G. Lowney, and N. Rubin, “Spike: An Optimizer for Alpha/NT Executables,” Proc. of the *USENIX Windows NT Workshop*, Aug. 1997.

[4] N. Gloy, T. Blackwell, M. D. Smith, and B. Calder, “Procedure Placement Using Temporal Ordering Information,” Proc. of the *30th International Symposium on Microarchitecture*, pages 303–313, Dec. 1997.

[5] N. Zafar Azeemi, “Power Aware Framework for Dense Matrix Operations in Multimedia Processors,” Proc. of the IEEE 9th International Multi-topic Conference, Dec. 2005.

[6] E.A. Lee, "Embedded Software," *Advances in Computers,* E. Zelkowitz, ed., Academic Press, 2002.

[7] A. Hashemi, D. Kaeli, and B. Calder , “Efficient Procedure Mapping Using Cache Line Coloring,” Proc. of the *ACM SIGPLAN'97 Conference on Programming Language Design and Implementation*, pages 171–182, June 1997.

[8] J. Kalamatianos and D. Kaeli, “Temporal-Based Procedure Reordering for Improved Instruction Cache Performance,” Proc. of the *Fourth International Symposium on High-Performance Computer Architecture*, pages 244–253, Feb. 1998.

[9] N. Zafar, M. Rupp, “Energy-aware source-to-source transformations for a VLIW DSP processor,” Proc. of the IEEE 17th ICM 2005, pp. 133-138, Dec. 2005.

[10] S. McFarling. Program, “Optimization for Instruction Caches,” Proc. of the *Third International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 183–191, 1989.

[11] K. Pettis and R. Hansen, “Profile Guided Code Positioning,” Proc. of the *ACM SIGPLAN'90 Conference on Programming Language Design and Implementation*, pages 16– 27, June 1990.

[12] M. Rosenblum, E. Bugnion, S. A. Herrod, and S. Devine, “Using the SimOS Machine Simulator to Study Complex Computer Systems,” *ACM Transactions on Modeling and Computer Simulation*, 7(1):78–103, Jan. 1997.

[13] A. D. Samples and P. N. Hilfinger, “Code Reorganization for Instruction Caches,” Technical Report UCB/CSD 88/447 University of California, Berkeley, Oct. 1988.

[14] S. C. Woo, M. Ohara, E. Torrie, J. P. Singh, and A. Gupta, “The SPLASH-2 Programs: Characterization and Methodological Considerations,” Proc. of the *22nd International Symposium on Computer Architecture*, pages 24–36, June 1995.

[15] X. Zhang, Z. Wang, N. Gloy, J. B. Chen, and M. D. Smith. , “System Support for Automatic Profiling and Optimization,” Proc. of the *16th ACM Symposium on Operating Systems Principles*, Oct. 1997.

[16] Parameswaran, S. “Code placement in hardware/software co-synthesis to improve performance and reduce cost,” Proc. of the Conference on Design, Automation and Test. pp 626-632, 2001.

[17] N. Z. Azeemi, M. Rupp, “Muticriteria Low Energy Source Level Optimization of Embedded Programs,” Proc. of the IEEE Informationstagung Mikroelektronik 2006, pp. 150-158, Oct. 2006.