Code Transformations for Low Power Caching in Embedded Multimedia Processors

C. Kulkarni F. Catthoor† H. De Man †
IMEC, Kapeldreef 75, B-3001 Leuven, Belgium
†Professor at the Katholieke Universiteit Leuven
E-mail: {kulkarni,catthoor,deman}@imec.be

Abstract

In this paper, we present several novel strategies to improve software controlled cache utilization, so as to achieve lower power requirements for multi-media and signal processing applications. Our methodology is targeted towards embedded multi-media and DSP processors. This methodology takes into account many program parameters like the locality of data, size of data structures, access structures of large array variables, regularity of loop nests and the size and type of cache with the objective of improving the cache performance for lower power. We also take into account the potential overhead due to the different transformations on the instruction count and the number of execution cycles to meet the real time constraints and code size limitations. Experiments on a real life demonstrator illustrate the fact that our methodology is able to achieve significant gain in power requirements while meeting all other system constraints.

1. Introduction and Related Work

Rapid progress in the field of embedded processors has given a new impetus to the domain of real-time multi-media processing (RMP) applications. Embedded processors can be broadly classified into two main categories: (1) domain specific processors, such as those in the DSP domain (e.g. Trimedia from Philips, Oak core from LSI logic etc), and (2) general-purpose processors (e.g. ARM core from Advanced RISC Machines). In embedded processors the following main differences can be identified compared to the traditional ones: (1) we now have a single application running on the processor when the product is shipped, so more compile-time analysis is feasible (2) we are permitted longer analysis and compilation times for the application, and (3) cost issues like data and program memory size and especially power consumption play a much bigger role.

Embedded multi-media applications are usually memory intensive and most of the power consumption is due to the memory accesses for data transfers. Power management and reduction is becoming a major issue in such applications [7, 8]. Many of the current algorithm standards (including the ones which are still under development like MPEG-x and object-oriented coders [6]) are based on a hierarchy of submodules. The main data-dependencies and irregularities are situated in the higher layers. However, from there several submodules are called, such as multi-mode motion estimation/compensation, wavelet schemes, DCT. These lend themselves for extensive compile-time analysis enabling efficient use of software-controlled caches, as illustrated below. These observations are the main motivations for us to come up with a compile-time code transformation strategy to utilize the cache efficiently for low power.

Our target architecture is based on (parallel) programmable video or multi-media processors (TMS320C6x, TriMedia, etc.), as this is a current trend to use in RMP [12].

Recently, a study has been presented on the effect of power requirements on the type of cache used in an architecture [13]. But this work does not address the basic issue of improving the cache utilization by means of program transformations. Many software compilers try to come up with the best array layout in memory for optimal cache performance (see e.g. [1] for a recent approach based on compile-time analysis). Most of the work related to efficient utilization of caches has been directed towards optimization of the throughput by means of (local) loop nest transformations to improve data locality [2, 18] and loop blocking transformations to improve the cache utilisation by reducing conflict misses [14, 17, 19]. Some work [22] has been reported on the data organization for improved cache performance in embedded processors but they do not take into account a power oriented model. None of the previous work tries to directly reduce the storage requirements by partially overlapping data, as memory is allocated based on the available variable declarations. In our work we will use global locality improving transformations [10], and aggressive in-

place mapping of the data [11] to improve on this. They were originally introduced by us to reduce the size of custom memory organisations. These transformations are part of our ATOMIUM [20] global data transfer and storage exploration methodology. This work has focussed on (customized) uni-processors. Some complementary aspects of our newer work on system-level memory management for parallel processor mapping has been discussed in [9].

The paper is organised as follows. In section 2, the correlation between caching and power is presented. In section 3, a formalized methodology is proposed to optimize the program code step by step for effective cache utilization. A discussion of the results obtained using a real life voice coder application is presented in section 4. In section 5 we provide some concluding remarks and a scope for future work in this field.

2. Problem Exploration

The critical issues that determine the cache performance during low power exploration in compiling real time signal processing and multi-media programs on embedded processors are: (1) number of accesses to cache and main memory, which directly determine the power budget for the program (2) size of the data structures (mostly array variables) which indirectly influence the power per access (3) the cycle budget and timing constraints which govern the real-time implementation requirements.

2.1. Caching and Power Correlation

The relation between the extent of data caching and power is explored in this section. Consider the simple but representative architecture model that we are using in this paper as shown in figure 1. The mapping of data to the cache is performed in a fully associative style which provides more freedom. But this is not a principle limitation of our methodology. The power model used in our work

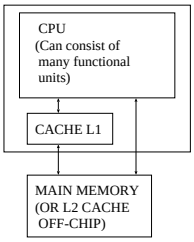

Figure 1. The target architecture used.

comprises a power function which is dependent upon access frequency, size of memory, number of read/write ports and number of bits that can be accessed in every access. The power is directly proportional to the number of memory accesses. Let P_t be the total power, N_t be the total number of accesses to a memory (can be SRAM or DRAM or any other type of memory) and S_t be the size of the memory. The total power is then given by:

$$P_t = N_t.F(S_t) \tag{1}$$

where F is a polynomial function with the different coefficients representing a vendor specific energy model per access. F is completely technology dependent.

For an architecture with no cache at all i.e. only a CPU and off chip memory, power can be approximated as follows:

$$P_{tm} = N_t.F(S_m) \tag{2}$$

where, $N_t = N_m$; N_m= number of accesses to main memory. Let us introduce a level one cache as in figure 1, then we observe that the total number of original accesses in the code (N_{tc}) is now distributed into a number of cache accesses (N_c) and a number of main memory accesses (N_{mm}):

$$P_{tc} = P_m + P_c \tag{3}$$

where, $P_m = (N_{mc} + N_{mm}).F(S_m)$
and $P_c = (N_{mc} + N_c).F(S_c)$.
Here the term N_{mc} represents the amount of additional traffic between the two levels of memory. It is a useful indicator of the power overhead due to various characteristics like block size, associativity, replacement policy etc. for a architecture with hierarchy. It is worth to note that N_c is dependent upon the size of the cache (S) and also on the locality of data (η) i.e. $N_c = G(S, \eta)$. The locality of data can be estimated in many ways as proposed in literature [16, 25].

Let the factor of gain for an on-chip cache access compared to that of an off-chip main memory access be α. Here α is technology dependent. Then we have:

$$F(S_c) = \alpha * F(S_m) \quad where \ \ \alpha < 1. \tag{4}$$

It is obvious that better usage of the cache improves the power cost. In general though, the optimal code for power (global notion) is very different from the optimal code for performance (which is related only to the critical path).

2.2. Algorithm optimization strategies

The size of array variables in most RMP applications contributes quite significantly to efficient cache utilization. In this work we are using the aggressive in-place mapping between signals, described in [11]. The caching decisions are partially correlated to the in-place optimization. If in-place is not considered then for many data structures the criterion for transferring data from main memory to cache is not fulfilled and this results in an inefficient solution.

In literature many types of loop and data flow transformations are proposed [16, 25] but here we are using only those transformations that result in the reduction of the number of accesses or in the improvement of locality and regularity of the program. These include loop fusion, loop interchange and index set splitting in conjunction with window-based intra-array in-place mapping [11] to improve locality. Some negative effects of loop transformations on performance are presented in [4]. We also use some transformations to offset the overhead in instruction and cycle count and to obtain a better energy-delay product.

3. Methodology

The steps comprising our methodology are shown in figure 2. We mainly focus on the cache related issues here since other steps in our global approach have been addressed in [9, 10].

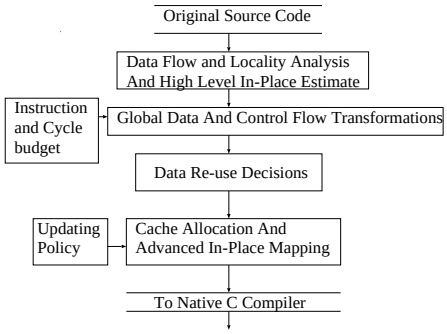

Figure 2. Proposed methodology

1. **Data Flow and Locality Analysis**: This first step is necessary to identify the parts and data structures in the program where there is some potential gain by means of the compile-time optimizations. Apart from this, parts of the program which are performance bottlenecks are identified.

2. **High-Level In-place Estimate**: Once the data structures that have the potential for optimization are identified, the gain due to the in-place optimization is estimated. In this step only the intra-signal in-place is explored.

3. **Global Data-flow and Control flow/Loop transformations**: The next step is to apply loop transformations over the scope of the entire program so as to improve the regularity and locality. This also results in a reduced number of cache and main memory accesses because many accesses to temporary data are moved to the foreground registers. Data-flow transformations are included to break the dependence bottlenecks and to remove redundant accesses. The constraints on applying these transformations are the given cycle budget and the code size limitations.

4. **Cache Allocation**: The last step in this code transformation methodology is to define the data layout in main memory and caches. An advanced in-place mapping approach which consists of both inter-signal in-place and final intra-signal in-place mapping is applied to obtain maximum cache utilization. The cache allocation is dependent upon the type of updating policy for main memory.

4. Demonstration on realistic test-vehicle

We illustrate the validity and effectiveness of the above methodology on a real-life voice coder application.

4.1. Voice Coder Algorithm

We have used a Linear Predictive Coding (LPC) vocoder which provides an analysis-synthesis approach for speech signals [24]. In our algorithm, speech characteristics (energy, transfer function and pitch) are extracted from a frame of 240 consecutive samples of speech. Consecutive frames share 60 overlapping samples. All information to reproduce the speech signal with an acceptable quality is coded in 54 bits. The general characteristics of the algorithm are presented in figure 3. It contains 16 pages of complex C code.

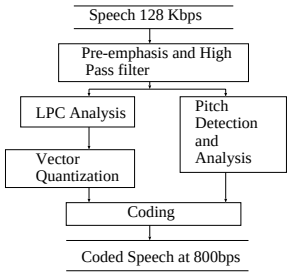

Figure 3. The different parts of the voice coder algorithm and the global data flow

4.2. Low power Caching strategies

We have performed experiments on our test vehicle in a step by step manner for different sizes of the cache to provide a global understanding of the relation with power. Accordingly, we present three different caching strategies [15] based on the above discussion.

1. **Conventional caching strategy**:In this strategy only the first step i.e. data flow and locality analysis and part of the last step of our caching methodology i.e. conventional cache allocation are performed in addition to the traditional loop blocking step.

2. **Advanced In-placed caching strategy**: In this type of caching strategy the following sequence of steps are followed (a) Data flow and locality analysis and high level in-place mapping and (b) Cache allocation including advanced in-place mapping.

3. **Combined In-placed and global Loop/Control flow transformations caching strategy (Combined strategy)**: The third caching strategy follows the complete methodology presented in the section 3.

The target architecture used in this study is shown in figure 1. The main feature in this architecture is the presence of software control over data transfer in and out of the cache, and the possibility to bypass the cache. The results and discussion of the experiment are presented in the next subsection.

4.3. Discussion of Experimental Results

Figure 5 shows the number of accesses to/from main memory and figure 6 for the cache. The following observations can be made from the figures. Conventional caching strategy requires the largest number of accesses to main memory of all the three strategies. Similarly, the cache is maximally exploited by the combined strategy and least by the conventional strategy. This illustrates the effectiveness of in-place mapping and the use of the above methodology. It is interesting to note here that loop blocking improves the transfer of accesses from main memory to cache but is unable to reduce the extra transfers from cache to main memory that result due to the large size of data structures. This reduction due to the size of data structures comes only with the use of aggressive in-place mapping. This concept is shown in figure 4. Here arrow 1 represents the minimum number of transfers required from main memory to the cache. Arrow 2 represents the transfers needed for storage of data (in main memory) referenced later. Arrow 3 represents extra transfers required due to the size of the data structure, which can be eliminated using aggressive in-place mapping. Also due to the algebraic manipulations in the test-vehicle, many non-rectangular (trapezoidal) access structures were present for which in-place mapping performs much better than just using loop blocking. Figure 7, shows the power cost of the three caching strategies. We observe that the power requirements of the conventional caching strategy are the highest in general. An important observation at this stage is that the general idea of the larger

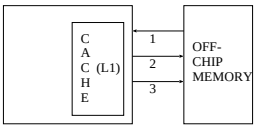

Figure 4. Effect of in-place mapping on the accesses between cache and main memory.

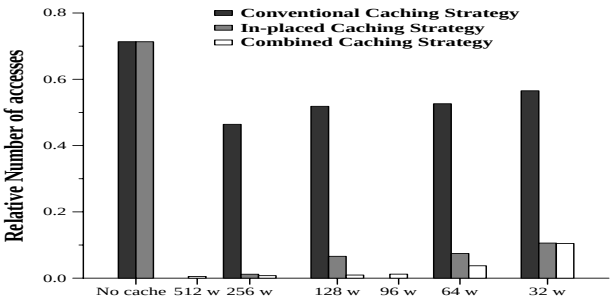

Figure 5. Number of main memory accesses.

the cache size the better the performance (for speed), does not hold for power as a cost measure. This is another illustration of the difference in approach required for embedded processors. We obtain upto a factor 24 of power improvement in the memory system, which is very large compared to the performance effect of conventional approaches in the (parallel) compiler community. This demonstrates the large impact of high level decisions on power requirements. We also see that the power requirements for a 128-word cache are smaller compared to a 256-word and a 512-word cache for the combined strategy. Here we assume that the cache is partitioned into banks and each bank can be activated individually by the designer. Hence the information obtained about the optimal size of the cache helps to selectively store the data and use the cache more efficiently. Figure 8, gives the instruction count values and also code sizes for the voice coder application. These values were obtained using the

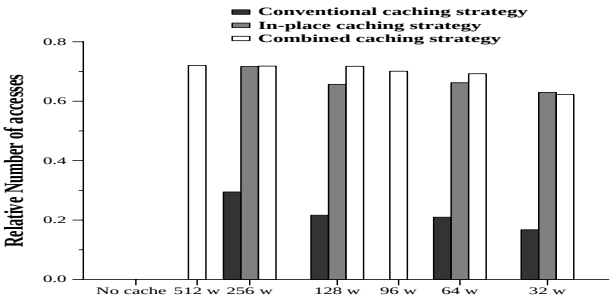

Figure 6. Number of cache accesses.

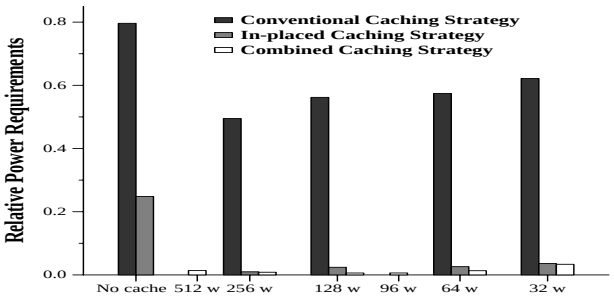

Figure 7. Power requirements for all cases.

ARMulator tool[1] [3]. We observe that the instruction count and the code size show a very gradual increase. This confirms that use of real-time constraint oriented transformations do result in a balanced solution. Figure 9, shows the

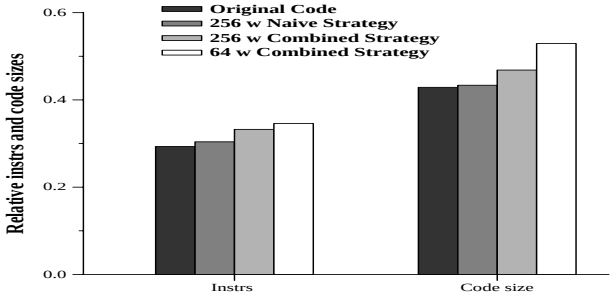

Figure 8. Instruction count and code size for a few cases.

number of accesses for a write through policy for the best cases obtained earlier. For the write through updating policy every change in the cache content needs to be written back to cache, hence the effectiveness of in-place mapping is largely lost. However, due to the other steps in our approach, we still have a gain of almost 67% compared to the original algorithm, which is quite significant. To conclude the experimental analysis, we have performed some experiments on three workstations to observe the cycle count characteristics. The execution times were obtained on three different machines (HP-UX platform 9.05 (which is a design station), HP-UX platform 10.20 (which is also a compute server with four processors and 5 times faster compared to HP-UX 9.05) and SUN station (platform 5.4)). We observe that the transformed code performs much better than the original code. This is due to (a) the better access locality due to global nature of our loop transformation methodology, which leads to heavily improved register allocation, and (b) the native compiler of these systems is able to utilize the

[1]which is an ARM processor emulator generating much profiling and performance data

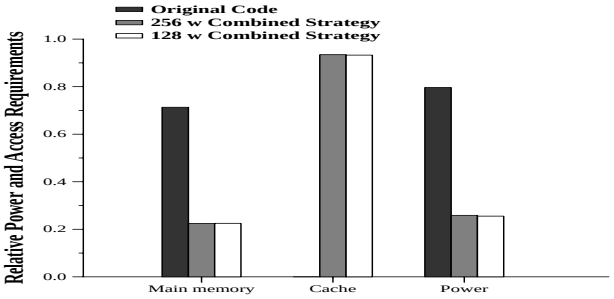

Figure 9. Power and access analysis for write through udating policy.

source level transformations better. Also we do not intend to compare performance of different workstations since the conditions used for computation were different for different machines. This analysis shows that our methodology is indeed able to significantly reduce power requirements and still meet all the other design constraints. Even if power is not an issue, a significant improvement in speed is achieved still.

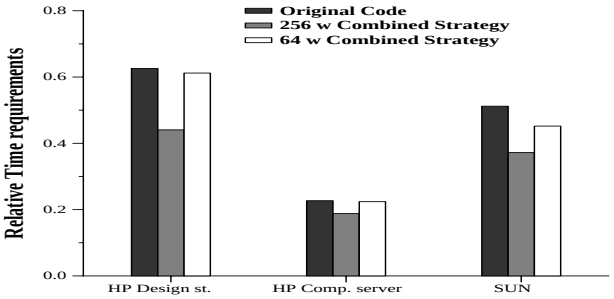

Figure 10. Timing analysis of above code on different workstations.

5. Conclusion and Future work

Low power realisation of embedded real-time multimedia applications is crucial for many future systems. The main results of this paper are: (1) a formalized methodology for power-oriented source-level transformations to improve software controlled cache utilization has been proposed, and (2) this approach is very effective as demonstrated on a real-life voice coder test-vehicle where upto a factor of 24 gain in data storage related power is obtained. The results presented are worked out for a 1-level cache but this methodology can be extended to multi-level caches too. The above methodology can be extended also for compilation (or specifically source level pre-compilation) of programs for parallel multi-media processors. Both these issues are

topics of current research. Another important conclusion is that the availability of more software control over the cache behaviour is very useful in achieving efficient cache usage and this feature needs to be supported in all power conscious multi-media oriented processors.

References

[1] C.Ancourt, D.Barthou, C.Guettier, F.Irigoin, B.Jeannet, J.Jourdan, J.Mattioli, "Automatic data mapping of signal processing applications", *Proc. Intnl. Conf. on Applic.-Spec. Array Processors*, Zurich, Switzerland, pp.350-362, July 1997.

[2] J.Anderson, S.Amarasinghe, M.Lam, "Data and computation transformations for multiprocessors", in *5th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, pp.39-50, August 1995.

[3] "ARM Software Development toolkit manual", Version 2.0, Advanced RISC Machines Inc., Cambridge, U.K., 1995.

[4] P.Baglietto, M.Maresca and M.Migliardi, "Image processing on high-performance RISC sytems", *Proc. of the IEEE*, invited paper, Vol.84, No.7, pp.917-930, July 1996.

[5] U.Banerjee, "Dependence analysis for supercomputing", Kluwer, Boston, 1988.

[6] V.Bhaskaran and K.Konstantinides, "Image and video compression standards: Algorithms and Architectures", *Kluwer Academic publishers, Boston*, 1995.

[7] R.W.Broderson, "The network computer and its future", *Proc. IEEE Int. Solid-State Circuits Conf.*, San Francisco CA, pp.32-36, Feb. 1997.

[8] P.Chatterjee (President Personal Productivity Products, Texas Instruments), "Gigachips: deliver affordable digital multi-media for work and play via broadband network and set-top box", Plenary paper in *Proc. IEEE Int. Solid-State Circ. Conf.*, San Francisco CA, pp.26-30, Feb. 1995.

[9] K.Danckaert, F.Catthoor, H.De Man, "System-level memory management for weakly parallel image processing", *Proc. EuroPar Conference*, Lyon, France, August 1996. "Lecture notes in computer science" series, Vol.1124, Springer Verlag, pp.217-225, 1996.

[10] E.De Greef, F.Catthoor, H.De Man, "Program transformation strategies for reduced power and memory size in pseudo-regular multimedia applications", accepted for publication in *IEEE Trans. on Circuits and Systems for Video Technology*, 1998.

[11] E.De Greef, F.Catthoor, H.De Man, "Memory Size Reduction through Storage Order Optimization for Embedded Parallel Multimedia Applications", *Intnl. Parallel Proc. Symp.(IPPS)* in Proc. Workshop on "Parallel Processing and Multimedia", Geneva, Switzerland, pp.84-98, April 1997.

[12] T.Halfhill and J.Montgomery, "Chip fashion: multimedia chips", *Byte Magazine*, pp.171-178, Nov. 1995.

[13] P.Hicks, M.Walnock and R.M.Owens, "Analysis of power consumption in memory hierarchies", *Proc. IEEE Intnl. Symp. on Low Power Design*, Monterey CA, pp.239-242, Aug. 1997.

[14] M.Jimenez, J.M.Liaberia, A.Fernandez and E.Morancho, "A unified transformation technique for multi-level blocking" *Proc. EuroPar Conference*, Lyon, France, August 1996. "Lecture notes in computer science" series, Springer Verlag, pp.402-405, 1996.

[15] C.Kulkarni, "Caching strategies for voice coder application on embedded processors", *Master's thesis*, Dept. of Electrical Engineering, K.U.Leuven, June 1997.

[16] D.Kulkarni and M.Stumm, "Linear loop transformations in optimizing compilers for parallel machines", *The Australian Computer Journal*, pp.41-50, May 1995.

[17] M.Lam, E.Rothberg and M.Wolf, " The cache performance and optimizations of blocked algorithms", *In Proc. ASPLOS-IV*, pp.63-74, Santa Clara, Ca., 1991.

[18] N.Manjiakian, T.Abdelrahman, "Fusion of loops for parallelism and locality", Technical report CSRI-315, Comp. Systems Res. Inst. Univ. of Toronto, Canada, Feb. 1995.

[19] N.Manjikian and T.Abdelrahman, "Array data layout for reduction of cache conflicts", *Int. Conf. on Parallel and Distributed Computing Systems*, 1995.

[20] L.Nachtergaele, F.Catthoor, F.Balasa, F.Franssen, E.De Greef, H.Samsom and H.De Man, "Optimisation of memory organisation and hierarchy for decreased size and power in video and image processing systems", *Proc. Intnl. Workshop on Memory Technology, Design and Testing*, San Jose CA, pp.82-87, Aug. 1995.

[21] D.A.Patterson and J.L.Hennessy, "Computer Architecture: A quantitative approach", *Morgan kaufmann publishers, Inc.*, San Francisce, Ca., 1996.

[22] P.R.Panda, N.D.Dutt, A.Nicolau, " Memory data organization for improved cache performance in embedded processor applications", *Proc. 9th ACM/IEEE Intnl. Symp. on System-Level Synthesis*, La Jolla CA, pp.90-95, Nov. 1996.

[23] J.P.Diguet, S.Wuytack, F.Catthoor, H.De Man, "Formalized methodology for data reuse exploration in hierarchical memory mappings", *Proc. IEEE Intnl. Symp. on Low Power Design*, Monterey, pp.30-35, Aug. 1997.

[24] L.R.Rabiner and R.W.Schafer, "Digital signal processing of speech signals", *Prentice hall Int'l Inc.*, Englewood cliffs, NJ., 1988.

[25] M.Wolf, M.Lam, "A loop transformation theory and an algorithm to maximize parallelism", *IEEE Trans. on Parallel and Distributed Systems*, Vol.2, No.4, pp.452-471, Oct. 1991.