Systematic Design Flow for Dynamic Data Management in Visual Texture Decoder of MPEG-4

Alexandros Bartzas*, Miguel Peon†, Stylianos Mamagkakis*, David Atienza†, Francky Catthoor‡, Dimitrios Soudris* and Manuel Mendias†

* Department of Electrical and Computer Engineering, Democritus University of Thrace, Xanthi, Greece 67100
Email: {ampartza,smamagka,dsoudris}@ee.duth.gr

† Departamento Arquitectura de Computadores y Automatica, Universidad Computense de Madrid, Spain 28040
Email: mikepeon@fdi.ucm.es,{datienza,mendias}@dacya.ucm.es

‡ IMEC vzw, Kapeldreef 75, 3001 Leuven, Belgium. Also Professor at Katholieke Universiteit Leuven
Email: catthoor@imec.be

***Abstract*— There is a clear trend of future embedded systems in moving toward wireless, multimedia, multi-functional and ubiquitous applications. This emerges new challenges in the existing solutions on performance, power, flexibility and costs, calling for innovations in both architecture and design methodology. In this paper we propose a design flow consisting of three stages to handle dynamic data, allowing the designer to create highly customized dynamic memory managers, make them bank-aware and create a design-time schedule of the different tasks of the application. We evaluated the proposed flow using the Visual Texture Coding (VTC) application, mapping it on a dual processor embedded platform achieving 5.5% reduction in memory footprint and 10% gains in execution time.**

I. Introduction

Nowadays the semiconductor industry is facing several technological challenges to build media-rich mobile wireless terminals in a profitable way. The applications realized by these devices require an enormous computational performance at a sufficiently low-power consumption. Industry strongly believes that multiprocessor platforms (heterogeneous or homogeneous) are a promising way to meet the aforementioned challenges [1]. To limit power consumption, platforms for mobile systems usually consist of multiple processors. Employing multiprocessor systems make easier for the embedded devices to support more services and additionally to cope with the advanced complexity and dynamism of modern applications.

However, there is the need to exploit the dynamic and multi-threaded character of the targeted application domain. One of the most critical bottlenecks is the very dynamic and concurrent behavior of many current multimedia applications. In order to deal with these new dynamic applications where tasks and complex data types are created and deleted at run-time based on non-deterministic events a new design flow is needed.

Additionally, among the different multiprocessor platforms available, the designer can find different memory hierarchy organizations. These hierarchies consist of combinations of caches, scratchpad memories and SDRAMs. Thus, the designer not only has to map efficiently the application on the platform, but to perform an efficient exploitation of the underlying memory hierarchy. The efficient use of memory can avoid conflicts and page misses. The number of page misses heavily affects the dynamic energy cost and the execution time. Techniques have been proposed for avoiding page misses by optimizing assignment of data to banks [2]. An SDRAM-aware data assignment stores frequently accessed data structures in separate banks as much as possible. Nevertheless, the data assignment itself is not always sufficient to reduce the number of page misses. When several tasks assign all their data to a limited number of banks, finding an optimal data assignment is a very difficult task, sometimes even impossible. In that case, the designer has little freedom in order to separate frequently accessed data with high locality from other data. In such a situation, task scheduling can be used in order to expand data assignment freedom.

In this paper, the goal is to provide a complete framework for mapping dynamic multimedia applications onto embedded multi-processor platforms. We present the first attempt to unify all the design stages, creating a design flow handling the dynamic data of multimedia applications. Initially, the methodologies were developed separately from each other [3]–[5]. Now, that they have reached in a quite mature level, is needed to make them work in a unified design flow, assessing how they cooperate with each other and eliminate possible conflicts among them. With the development of the adequate automation tools, the proposed design flow will demand minimum intervention by the designer, leading to an increase of design productivity.

The remainder of the paper is organized as follows. In Section II, we provide an overview of the related work. In Section III, we present the set of methodologies consisting our design flow. In Section IV, are described the targeted application, the execution platform and the experimental results. Finally, in Section V we draw our conclusions.

II. Related Work

Traditionally, embedded systems designers have been very reluctant to the use of dynamic memory allocation in their applications. The reason is that the management of the data structures needed to keep track of dynamic memory was too expensive. Relying on the dynamic memory subsystem can ease the design of the rest of the system, but can also in-

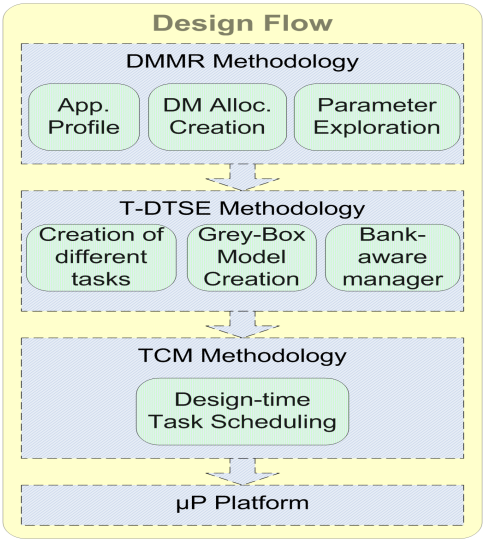

Fig. 1. The proposed design flow.

troduce new problems: memory fragmentation and processing overhead. The optimization in the management of dynamically allocated data is always handled by general purpose solutions in the Operating Systems, wherever it is available, or it is done manually for specific applications [6]–[9].

Current memory-hierarchy aware data assignment methodologies mostly focus on the design-time assignment of the data and overlook the run-time assignment on the various levels of the memory hierarchy due to its complexity. Also, they focus on shared memory architectures and favor either the use of on-chip software-controlled scratchpad memory or hardware-controlled cache memories [2], [10]–[14]. Wherever the run-time layer is available, it is based on very generic libraries or hardware control (such as MMUs).

The work presented in this paper is related to [3]–[5]. The major contribution of the work presented there is the building of a design framework combining three complementary methods namely, Dynamic Memory Management Refinement (DMMR), Task-level Data Transfer and Storage Exploration (T-DTSE) and Task Concurrency Management (TCM). Furthermore, it is the first time that a Dynamic Memory Allocator produced systematically by the DMMR Methodology is enhanced with bank-aware capabilities, combining DMMR and T-DTSE methodologies. Next we combine the T-DTSE and TCM methodologies by creating design-time schedules.

III. Design Flow

The design flow (depicted in Fig. 1) consists of three stages, with each stage trying to effectively handle the dynamic data of applications from its own perspective. In order to maximize the results of these stages, they should be efficiently combined in a unified flow. The first method is the Dynamic Memory Management Refinement (DMMR), consisting the first stage. The second one is the Task-level Data Transfer and Storage Exploration (T-DTSE), consisting the second stage. Finally, the third one is the Task Concurrency Management (TCM). Next paragraphs describe the main features of each design stage.

A. Dynamic Memory Management Refinement Stage

The goal of the DMMR methodology is to optimize the dynamic allocation and deallocation of data by optimizing/altering the Dynamic Memory Managers provided by the underlying Operating System (for example operator overloading of `malloc/new` and `delete/free`). The work in the DMMR stage of the design flow consists on analyzing the dynamic memory allocation and deallocation of the embedded applications under study and finding the best solutions for these particular cases. This work is possible because of the existence of the DMMR library [3], [6], a mixin-based library that eases the composition of dynamic memory managers and thus allows a semi-automatic exploration of the design space for each particular application.

As the first step of our DMMR methodology dictates, the VTC kernel is profiled to determine its dynamic behavior. As we have seen in several dynamic multimedia software, the VTC application [15] uses only a few distinct Dynamic Memory (DM) block-sizes ranging from 32 Bytes to 2 MBytes in size (14 in total). While the blocks spanned quite a large size-range, they showed one distinct feature, most of them were powers of two. The maximum number of allocated blocks for a representative input was 1841 blocks with a total memory requirement of 6.8 MBytes. Then, the most allocated block-size was the one 1024 Bytes and the most accessed block-size was 2048 Bytes.

Next, using the profiling information and according to the second step of our methodology, a custom DM allocator is designed for this application. First, separated memory pools are defined for the most accessed block sizes (i.e. 5 out of 14), using FIFO single linked lists [7]. Single linked lists suffice for these pools as no coalescing or splitting is used (i.e. less memory overhead in the DM block headers), thus accessing previous DM blocks in the lists are not needed. Next, one or two different abstract memory heaps are created. In the case where we use multi-bank memories, only the 2048 byte blocks are allocated onto a separate bank, since they are the most accessed. So we allow them to take up the whole space in a separate memory bank for all the sizes studied in our explorations (i.e. 4-128 KBytes). Hence, it did not require coalescing or splitting.

In the last step of the exploration methodology, different configuration parameters are explored. For the free-lists, we explore between 0-6 separated free-lists for the most allocated block sizes (i.e. 1024, 2048, 512, 256 and 128 Bytes in order). Then, *exact fit* was chosen for the *fit algorithm* due to the fact that the block-sizes were very different [3]. As for the coalescing and splitting only the memory pool in abstract memory heap 1 needs to support coalescing and splitting due to the fact that only blocks of 2048 Bytes are allocated from the abstract memory heap 2 memory.

Experiments were made to see the effect of how much coalescing and splitting would affect Dynamic Memory (DM) footprint of the main memory pool (four different values of maximum coalesced sizes and four different values of

minimum split sizes), but the effects were minimal (variations of less than 5%). Experiments for different heap sizes (i.e. 4-128 KBytes) indicate that the effect of different abstract memory heap sizes are minimal.

B. Task-level Data Transfer and Storage Exploration Stage

In the T-DTSE stage the application is being split into task that may be executed concurrently. The goal is to study the relationships between tasks and the way the data are accessed, so to make an optimal scheduling of their memory accesses for the underlying memory hierarchy of the targeted platform. As first step, the designer specify an embedded application at a Multi-Task Graph (MTG) model combined with high-level features of a Control-Data Flow Graph (CDFG) model [16]. That allows the designer to identify the memory conflicts that occur very often between the concurrent tasks.

Having the tasks being formulated and in order to eliminate those conflicts, T-DTSE methodology offers the means for dynamic memory pool assignment. The methodology provides a framework that making feasible the optimization of the clustering of blocks of dynamic data in order to be placed later in physical memories. In order to achieve that, the designer uses the DM allocators built in the previous stage with the addition of a few extra features (e.g. bank awareness, scheduling of data). Thus, the DM allocators take into account the conflicts between the different dynamic data (i.e. coherency of data accessed at the same time). Using the new DM allocators and by using different data assignment in the memories the application is executed. Next using the profiling information extracted by the execution, the optimal data assignment is defined.

C. Task Concurrency Management Stage

TCM is the methodology to find the energy-efficient way to map dynamic real-time applications with concurrent tasks/subtasks onto multiprocessor platforms.

The methodology takes as input a high-level description of the application. That application has already implemented inside it the optimized memory allocators originated by the two previous steps. The purpose of the methodology is to determine a cost-optimal (e.g. energy consumption, deadline miss rate) constraint-driven (e.g. throughput or latency) scheduling of the various tasks and subtasks on a set of homogeneous or heterogeneous processors. The output of the methodology is Pareto curves with each Pareto point indicating a schedule. Then, the designer must choose that schedule that meets, in the best available way, the design constraints.

IV. Experimental Results

The application that was used in order to apply the design flow was the Visual Texture Coding (VTC), which is used in MPEG-4 standard [15] in order to compress the texture information in photo-realistic 3D models. As the texture in a 3D model is similar to a still picture, the application can also be used for compression of still images. It is based on the discrete wavelet transform (DWT), scalar quantization, zero-tree coding and arithmetic coding. Its software realization requires around 5K lines of C++ code.

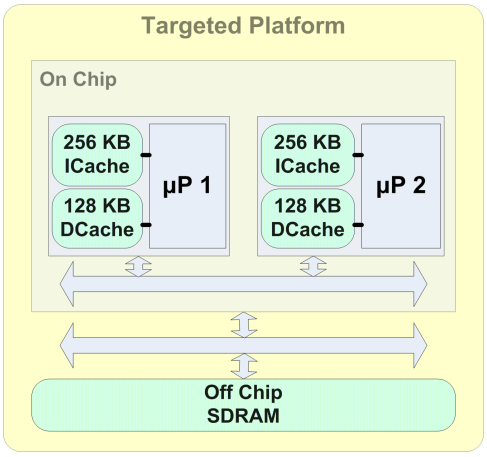

Fig. 2. The targeted multiprocessor platform.

The embedded platform (depicted in Fig. 2) that was used in our experiments consisted of two Texas Instruments (TI) C6202 [17] running at 250 MHz each and a two-bank SDRAM memory of 32 MB in total. The platform was running the VIRTUOSO real-time operating system. Virtuoso (now known as VSPWorks from Windriver) is a commercial RTOS, which features a high-performance kernel design with small memory footprint, and an advanced virtual single-processor (VSP) architecture for the development of embedded multiprocessor and distributed applications.

In total, 40 configurations of different DM managers needed to be evaluated to achieve the Pareto curve of Figure 3. The creation, implementation and evaluation of all the DM allocators took 13.3 hours in total. This is a memory footprint – memory accesses Pareto-optimal curve, which shows an available reduction up to 4.88% for memory footprint and up to 4% for memory accesses, within the available Pareto configurations. Furthermore, the general purpose DM allocator for Windows-XP based systems [18] was tested to compare it with our custom DM allocators, showing that our Pareto solutions reduce the energy consumption up to 82.9% and the execution time up to 3.8%.

Initially a model of the application is built featuring the different tasks and subtasks. From the model and application profiling we observe that the decoding of the image blocks takes up to 65% of the execution time. So, this is the function we focused on. We considered as a task the decoding of a block and as subtasks the decoding of each color coefficient (Y, U and V).

The next step was to convert manually from C++ to C (due to compiler limitations) the DM manager that derived from the DMMR stage. Furthermore, the functionality of the manager was enhanced, making it capable of allocating dynamic data in different banks of the SDRAM memory (bank-aware). In each function of the allocator (e.g. `malloc/free/calloc`), an ID denotes in which bank of the memory the function should act upon. With the use of the bank-aware memory allocator we manage to achieve additional gains of 6.3% in execution

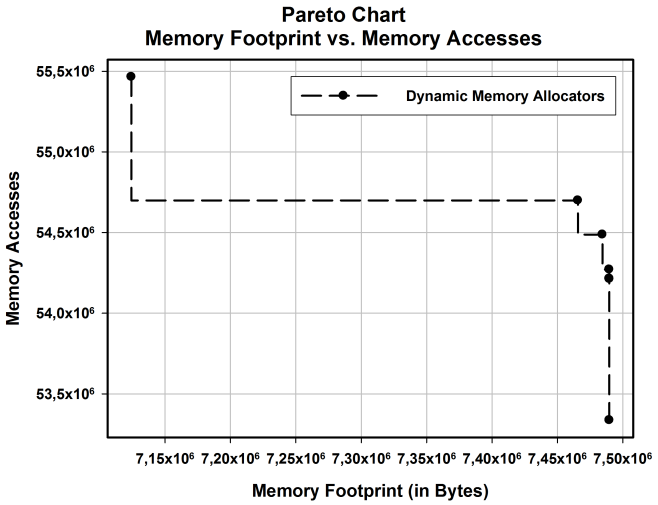

Fig. 3. Memory Footprint and Memory Accesses Pareto-optimal curve of DM allocators for VTC application. Each point represents a DM allocator.

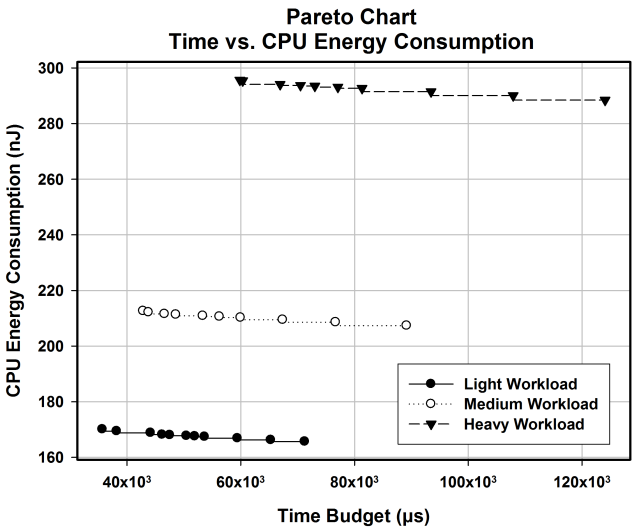

Fig. 4. Pareto curve of schedules for three different workloads.

time. Also, the use of the bank-aware allocator enable us to achieve additional 6% reduction in CPU energy consumption.

Next, in TCM stage a design-time schedule was built exploring different platform configurations (different cpu speed) and different application workloads (different size of images ranging from 352×288 to 1024×1024 pixels) producing Pareto curves. In Figure 4, such a Pareto curve is depicted, offering different schedules for three different workloads and for two TI C6202 processors (one running at 250 and the other at 300 MHz). Each Pareto-point is a schedule offering different solution according to the requirements of the application (e.g. if the application is time critical, then we have to select a schedule at the left region of the curve paying a penalty in energy and vice-versa).

V. Conclusions

The trend of future embedded systems is now moving clearly toward wireless, multimedia, multi-functional and ubiquitous applications. These features challenge the existing solutions on performance, power, flexibility and costs, calling for innovations in both architecture and design methodology. To cope with these complex tasks embedded systems employ more than one processor. The parallelism offered by the underlying hardware must be efficiently exploited by the applications. The proposed design flow, consisted by three methodologies, is helping the designer to efficiently handle the dynamic data of modern embedded applications. The first action is to optimize the dynamic memory managers by creating highly customized optimal ones. The second step is to optimize the dynamic data assignment to memories and eliminate the conflicts. The third step is to define an optimal schedule of execution of the various tasks of the application. The proposed design flow achieves 10% execution speed-up and 5.5% memory footprint reduction. In order the design flow to be more helpful to the designer, what is need to be added, is an additional tuning of T-DTSE and TCM methodologies towards dynamic data.

VI. Acknowledgements

This work is partially supported by the E.C. funded program AMDREL IST-2001-34379, `http://vlsi.ee.duth.gr/amdrel`, the Spanish Government Research Grant TIN2005-05619 and E.C. Marie Curie Fellowship contract HPMT-CT-2000-00031.

References

[1] H. de Man, "Ambient intelligence: Gigascale dreams and nanoscale realities," in *Proc. of ISSCC, Keynote speech*, 2005.

[2] P. Panda, N. Dutt, and A. Nicolau, *Memory issues in embedded in systems-on-chip: optimization and exploration*. Kluwer Acadademic Publishers, 1999.

[3] D. Atienza and et al., "Dynamic memory management design methodology for reduced memory footprint in multimedia and wireless network applications," in *Proc. of DATE*. IEEE Computer Society, 2004.

[4] P. Marchal and et al., "Integrated task scheduling and data assignment for sdrams in dynamic applications," *IEEE Des. Test*, vol. 21, no. 5, pp. 378–387, 2004.

[5] P. Yang and et al., "Energy-aware runtime scheduling for embedded-multiprocessor socs," *IEEE Des. Test*, vol. 18, no. 5, pp. 46–58, 2001.

[6] E. D. Berger, B. G. Zorn, and K. S. McKinley, "Composing high-performance memory allocators." in *Proc. of PLDI*. ACM Press, 2001.

[7] P. R. Wilson, M. S. Johnstone, M. Neely, and D. Boles, "Dynamic storage allocation, a survey and critical review." in *Proc. of International Workshop on Memory Management*, 1995.

[8] Microsoft Windows CE, 2005. [Online]. Available: http://msdn.microsoft.com/library/default.asp?url=/library/en-us/wcecoreos5/html/wce50conheaps.asp

[9] D. Atienza and et al., "Efficient system-level prototyping of power-aware dynamic memory managers for embedded systems," *Integration - The VLSI Journal*, vol. 39, no. 2, pp. 113–130, December 2005.

[10] M. Kandemir, and et al., "Dynamic management of scratch-pad memory space," in *Proc. of DAC*. ACM Press, 2001, pp. 690–695.

[11] M. Verma, L. Wehmeyer, and P. Marwedel, "Cache-aware scratchpad allocation algorithm," in *Proc. of DATE*. IEEE Computer Society, 2004.

[12] Verdoolaege and et al., "Multi-dimensional incremental loop fusion for data locality," in *Proc. of ASAP*. IEEE, 2003, pp. 17–27.

[13] F. Catthoor and et al., *Data access and storage management for embedded programmable processors*. Kluwer Acadademic Publishers, 2002.

[14] A. Macii, L. Benini, and M. Poncino, *Memory Design Techniques for Low Energy Embedded Systems*. Kluwer Acadademic Publishers, 2002.

[15] ISO, *ISO/IEC 14496-2:1999: Information Technology – Coding of audio-visual objects – Part 2: Visual*.

[16] F. Thoen and F. Catthoor, *Modeling, Verification and Exploration of Task-level Concurrency in Real-Time Embedded Systems*. Kluwer Academic Publishers, 1999.

[17] Texas Instruments, 2005. [Online]. Available: http://www.ti.com

[18] Microsoft Windows XP, 2004. [Online]. Available: http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dngenlib/html/heap3.asp