Some Experiences on Dynamic Memory Management Refinement at System-Level for Multimedia Applications

David Atienza*, Marc Leeman†, Jose M. Mendias*, Francky Catthoor‡, Chantal Ykman‡
Vincenzo De Florio†, Geert Deconinck†, Rudy Lauwereins‡
*DACYA/UCM, Avda. Complutense s/n, 28040 Madrid, Spain. Email: {datienza, mendias}@dacya.ucm.es
†ESAT/K.U. Leuven, Kasteelpark Arenberg 10, B3001 Heverlee, Belgium. Email: {name.surname}@esat.kuleuven.ac.be
‡IMEC vzw, Kapeldreef 75, 3001 Heverlee, Belgium. Email: {name.surname}@imec.be

Abstract— **Nowadays, 3D multimedia applications have grown rapidly in number and consist of complex systems (e.g. 3D graphical processing or games) that process extensive amounts of data to create 3D images and results. This produces high-cost and high-power consumption systems whereas a superior portability demands cheap and low-power consumption ones. In these multimedia applications, the dynamic memory subsystem is currently one of the main sources of power consumption and its inattentive management can affect severely the performance and power consumption of the whole system. In this paper, we illustrate a new system-level method to explore and refine the dynamic memory management of multimedia systems on current typical case studies, i.e. a relatively new 3D image reconstruction system and a 3D simulation game. This method is based on an analysis of the access pattern, amount of memory used and power consumption estimations. With this information, a phase-wise exploration and refinement flow is used to optimize the system at the different phases of its hardware-oriented design process. As the results in the case studies show, our system-level method achieves great improvements in memory footprint, power consumption and performance for multimedia applications.**

I. Introduction

The complexity of 3D multimedia applications has increased enormously in the last years. Presently, they process large amounts of data and require a good level of performance. Representative examples of their use can be archaeological site reconstruction, film industry and 3D games [6], [13], [1]. These examples need handheld devices as final platforms to achieve quick on-site visualization, processing and real-time interaction with the user. However, the complexity of multimedia applications results in high cost and power consumption systems while current markets demand the opposite. New multimedia algorithms greatly rely on dynamic memory (DM) due to the unpredictability of the input data at compile-time. Furthermore, they demand large amounts of memory that make the DM subsystem one of the main sources of power consumption, memory usage and (due to its latency and delay) heavily affect the global performance of the system.

With the previous characteristics, classical hardware design refinements can only partially compensate for the growing hardware/software gap in power and speed [11]. Recently, the critical design improvementss for very large scale integration systems have been shifting toward the software part.

In order to optimize embedded multimedia systems, detailed power consumption, memory use and performance profilings must be available at an early stage of the design flow. For statically allocated data and operations, this information is available from modern analysis and profiling tools. However, for dynamically (de)allocated data types (further called DDTs) implementations, the situation is much worse. At the most, summarized information can be available for the pools of data, but the details about individual DDTs are lost then. Also, simulation data can be generated at a much closer level to the final implementation on a certain platform, e.g. at instruction or cycle accurate hardware level. This is however very CPU-time consuming and requires the complete mapping trajectory, thus we do not include it currently since it is not acceptable for system-level exploration purposes. In dynamic multimedia applications, DDTs are a very relevant power consumption and speed factor (in our case studies, they can consume up to 60% of the total power used in the memory subsystem).

In this paper, we illustrate a new phase-wise system-level method to explore and refine DM management on embedded multimedia systems in an early stage of the system design and integration flow. As the case studies show, it uses a high level (i.e. from `C++` code) profiling method [10] to extract the necessary run-time information for a power-aware, performance and memory usage refinement of the DDT implementations and DM managers involved. To this end, the three important factors that influence power consumption and global performance of the memory subsystem are analyzed per DDT (and not for the entire pool of data only), i.e. memory usage pattern over time, memory accesses and data access mechanisms. This analysis allows the optimizations in the latter stages of the method, where it is used to optimize the whole DM management subsystem early on during system integration, enabling large savings in the design-time of the system.

The remainder of this paper is organized as follows. In Section II we motivate the optimization method and summarize its phases. This is however not the focus of this paper. In Section III we describe the related work. In Section IV we illustrate the large effects of our approach on the cost and performance of modern dynamic multimedia applications using our case studies. This is the core of the paper. Finally, in Section V we draw our conclusions.

DCIS 2003 ISBN: 84-87087-40-X

II. Motivation and Method

The design flow of new multimedia applications (e.g. 3D graphical processing or games) involves a very time-consuming effort in the correct design of the software and dynamic part of the system. It includes the definition of the interdependent and variable at run-time DDTs of the complex algorithms used. These systems depend on an efficient DM management, which is one of the most difficult design challenges for an efficient mapping on low-power and high-speed platforms without extensive system support for DM.

As mentioned in Section I, the use of DM can have important influence on the overall algorithm performance. Our system design method (see [9], [10] for a more detailed description) analyzes this influence using a phase-wise method, which starts from a high level specification and more information concerning the final implementation is added on each phase. As a result, estimations from previous phases are refined with new and more precise information. Then, an early idea of implementation costs and requirements is achieved and the developers can make changes in the system without expensive re-iterations through the entire design flow.

The illustrated phase-wise method is divided in three main phases according to a virtual structure of typical embedded multimedia systems where data is transferred into memory via DMA, and then the execution is triggered and finished by software interrupts. This structure is depicted in Figure 1. First, the method analyzes the access pattern of the DDTs involved and optimizes their implementations preserving the system constraints, this phase is called Dynamic Data Type Transformation and Refinement (DDTTR), marked as number 1 in Figure 1. Secondly, global DM managers are studied to efficiently tackle the (de)allocation requests of the DDTs. This phase is named Dynamic Memory Management Refinement (DMMR), number 2 in Figure 1. Finally, an additional phase can be used to optimize further considering the physical characteristics of the final platform and memory subsystem, e.g. bank conflicts or cache policies. This final phase is Physical Memory Management (PMM), number 3 in Figure 1.

In the following paragraphs, the first two phases of the method are only briefly summarized because they are not the focus. They will also be clarified in Section IV for the presented case studies, in so far as needed. The PMM optimizations are not applied because our aim is not a specific platform yet and lie out of the scope of this paper.

The first phase of the method is DDTTR, which refines the DDTs of the application under study. Overall, they define the way in which memory is allocated, accessed and freed.

The basis of all DDTs is the combination of a number of basic data types, e.g. trees, arrays or lists, in a single or multi-layered organization [9]. Then, they implement a certain set of basic operations as `add`, `get` or `set` to provide the interface to the user. Our method uses an object oriented programming approach that offers a common interface of operations for the useful multi-layered DDTs implementations in embedded multimedia systems [9]. This allows to design the system in a completely platform and implementation-independent way. Different combined DDTs offer possible trade-offs between performance, memory use and power consumption. For example, an array is rigid in size and memory usage but has fast access, while a simple linked list is the other way around. Finally, these trade-offs for the final solution of the DDT implementations are exploited according to the required restrictions of the system.

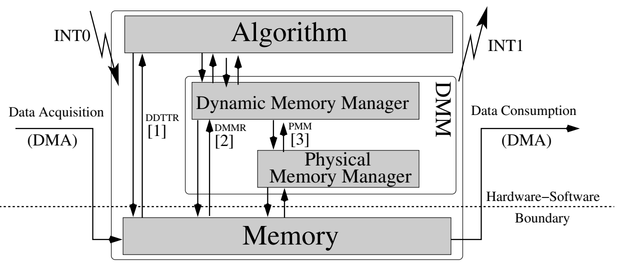

Fig. 1. Summary of the different optimization phases of the method

After the DDTs have been optimized, a suitable use of the global DM available must be provided according to the design constraints. In fact, an inappropriate management of the DM used by the DDTs reduces enormously the performance of the application and increases its power consumption [9]. Therefore, a well-adjusted DM manager must be selected. This is done in the second phase of the method, i.e. DMMR.

The definition of an appropriate DM manager implies a full analysis of the dynamic behaviour of the application under study. After this specific behaviour is known and we can characterize the actions to be performed by the adjusted DM manager (e.g. (de)allocation, block splitting or coalescing), our method applies high level strategies [17] and a set of possible decision choices that has been specifically defined for embedded multimedia systems [9]. As a result, a set of possible DM managers are available for the developer and he can decide according to the needed constraints, e.g. low-power consumption, performance, etc.

III. Related Work

Currently the foundations of a good DM management in a general-context are already well established and research has been done at its different levels.

Although the number of building blocks in multi-layered data types is limited [9], each developer tends to write his own custom DDTs for each application. Therefore, the number of implementation alternatives is defined by the experience and inspiration of the developers, and the available time to implement them. Presently, suitable access methods and DDT implementations have started to be proposed for multimedia applications considering their features [7], [9]. Also, in the software community much literature is available about DM management implementations and policies [17]. They use the locality of references and other techniques to analyze and minimize fragmentation or enable a fast time-response for systems with speed constraints [17]. However, this earlier work does not provide a complete and useful search space for a systematic exploration in multimedia applications.

Usually in memory management for embedded systems [12], the DM is partitioned into fixed blocks to store the DDTs. Then, the free blocks are placed in a single linked list [12] due to performance constraints. Another recent method is the simulation of the system with different general-purpose DM managers. In this case, [4] proposes an infrastructure to build DM managers using C++ templates. However, these methods focus on performance and do not consider power consumption issues, as embedded systems require.

Work can be found to estimate power consumption based on software instead of at a circuit-level. Most of it use an instruction level analysis [14], but more recently research has been developed for assembly code and a higher abstraction level [15], [16]. Nevertheless, they employ analyses without run-time profiling that are not enough for DM applications.

Due to large scale integration systems, several analytical and abstract power estimation models at the architecture-level have received more attention lately [5]. However, they do not focus on the DM hierarchy of the system and the power consumption from the DDTs at the software level.

In a complete different field, telecom network applications, [18] proposes a power exploration method driven by memory accesses. This work handles systems with independent DDTs and the main focuses are the dynamically created tasks due to network connections. It does not use run-time behaviour analysis because, after the system initialization, a snapshot of the memory at any moment during execution gives a good run-time memory-behaviour image, which can be used to determine the memory contribution of each DDT to the system. The DMMR optimizations we propose are not studied.

New system-level power consumption optimizations for embedded systems are explained in [3]. However, system-level refinements of memory management in applications with complex dynamic behaviour have not received much attention.

IV. Demonstration of the Method

The applied method is illustrated using two case studies that represent different modern multimedia application domains: the first case study is part of a new 3D image reconstruction system and the second one is an interactive 3D game.

The first case study forms one of the sub-algorithms of a 3D reconstruction algorithm [13] that works like 3D perception in living beings, where the relative displacement between several 2D projections is used to reconstruct the 3^{rd} dimension.

The second case study presented is the game controller of a 3D simulation game that interacts with the reality thanks to a frame grabbing device. This application belongs to the new category of interactive 3D games that integrate virtual-reality objects in scenarios offered by the real world.

In the following subsections the internal DM structures of these case studies are explained and the first two optimization phases (i.e. DDTTR and DMMR) are applied to them.

A. Method Applied to a New 3D Image Reconstruction System

The software module used as our driver application heavily uses DM and is one of the basic building blocks in many current 3D vision algorithms: *feature selection and matching*.

The sub-algorithm studied has been extracted from the original code of the 3D image reconstruction system (see [2]

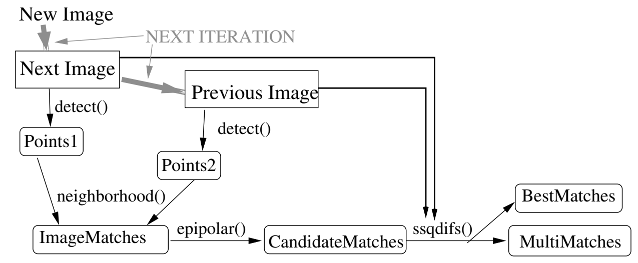

Fig. 2. Execution flow of the DDTs in the 3D image reconstruction system

for the full code of the algorithm with 1.75 million lines of high level C++), and creates the mathematical abstraction from the related frames that is used in the global algorithm. This implementation matches corners [13] detected in 2 subsequent frames. The operations done on the images are particularly memory intensive, e.g. each image with a resolution of 640×480 uses over 1Mb, and the accesses of the algorithm to the images are randomized. Thus, classic image access optimizations as row-dominated accesses versus column-wise accesses are not relevant.

The algorithm uses internally several DDTs whose sizes cannot be determined until the system is running because they depend on factors (e.g. textures in the images) determined outside the algorithm (and uncertain at compile-time). Furthermore, due to the image-dependency related data, the initial DDT implementations do not fit in the internal memory of current embedded processors. These DDTs are the following:

- ImageMatches (IMatches) is the list of pairs where one point in the first image, matches another one on the second image based on a neighborhood test [13].
- CandidateMatches (CMatches) is the list of candidates that must go through a normalized cross-correlation evaluation of a window around the points [2].
- MultiMatches (MMatches) is the list of pairs that pass the aforementioned evaluation criterion. Still one point from the first image can be listed in multiple candidate pairs for a later best match selection.
- BestMatches (BMatches) is the subset of pairs where only the best match for a point (according to the test already mentioned) is retained.

These DDTs were originally implemented using double linked lists and exhibit an unpredictable memory behaviour, typical in many state-of-the-art 3D vision systems [2] since they use some sort of *dynamic candidate selection* followed by *a criterion evaluation*. Figure 2 shows this behaviour in the generation order of the DDTs. First, IMatches is generated after a neighborhood test is applied to pairs of corners detected in two images. Then, CMatches is created using the information acquired from previous images. Finally, MMatches and BMatches are generated with the pairs that pass a normalized cross correlation evaluation [13].

Although the global interaction between these DDTs is defined by the algorithm (see Figure 2), it is not clear how each DDT affects the final system figures (e.g. memory footprint). Therefore, our method is used to explore and refine it.

First of all, DDTTR is applied to optimize the DDTs. After

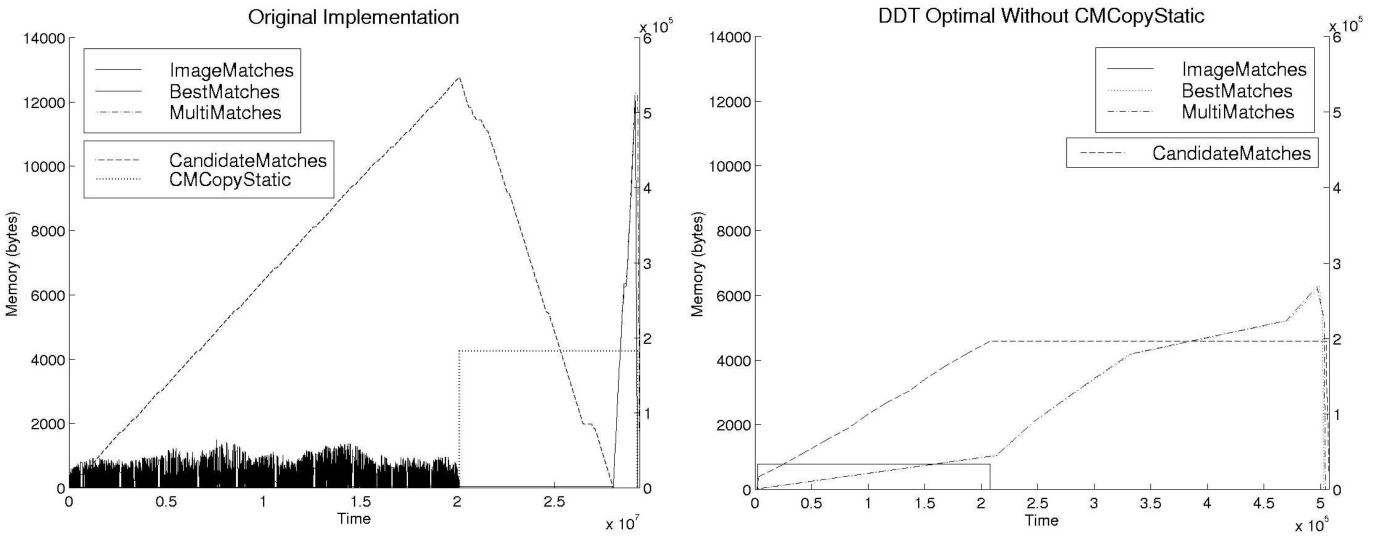

Fig. 3. Memory footprint over time. All plots mapped on the left axis, except `CandidateMatches` and `CMCopyStatic` (right axis). Left graph, original implementation. Right one, optimal implementation suggested by DDTTR with `CMCopyStatic` removed

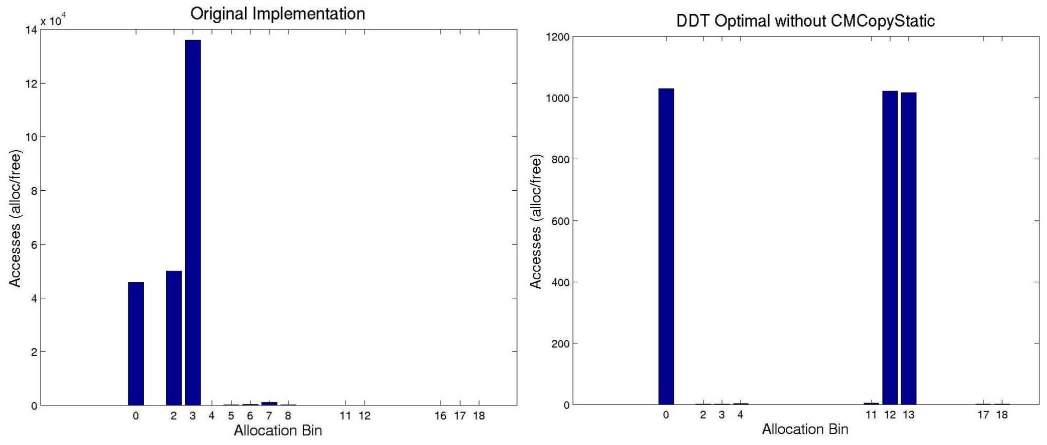

Fig. 4. Block-size allocation bins used in the Kingsley DM manager [17] by the 3D reconstruction system before (left graph) and after DDTTR (right graph)

automatically inserting the profiling code and running the tools presented in [10], profiling information is obtained. Then, a memory use graph is generated (left chart in Figure 3) and memory accesses, memory footprint and energy dissipation figures are calculated (see Table I). For the energy estimations, the memory model described in [8] is used. It is a complete energy/delay/area model for embedded SRAMs that is able to scale to different technology nodes (we use the .13 μ technology node for the results). This model depends on memory footprint factors (e.g. size, internal structure or leaks) and factors originated by memory accesses (e.g. number of accesses or technology node used). Note that any other model can be used just by replacing that module in the tools.

The analysis of the profiling information (Table I and Figure 3) shows how the DDTs affect the system. First, `CMatches` is the largest DDT. Secondly, `IMatches` has frequent accesses. Finally, `CMCStatic` (an "optimized dynamic array" that reduces the accesses to `CMatches`) consumes an important part of the energy used by the system.

After the subsequent exploration step of DDTTR (see [10] for more details), our tools suggest an ideal solution for the DDTs trying to minimize power consumption. The final DDT implementations consist of 2-layered dynamic array structures (pointer-arrays to arrays). First, an external dynamic array of 10 positions; then, each position is another array of 756, 1024 or 16384 Bytes (B) depending on which DDT. With these optimized DDTs, `CMatches` is now fast enough to interact directly with `BMatches` and `Mmatches`, thus `CMCStatic` is removed. This removal improves even more memory footprint, but has no effect on the performance (speed) of the global application, already improved by DDTTR.

TABLE I

ORIGINAL DDT IN THE 3D IMAGE RECONSTRUCTION SYSTEM

Variable	memory accesses	memory footprint (B)	energy .13μm tech.(μJ)
`IMatches`	1.20×10^6	5.14×10^2	0.18×10^3
`CMatches`	8.44×10^5	2.75×10^5	3.03×10^3
`CMCStatic`	6.24×10^4	1.08×10^5	4.48×10^4
`MMatches`	1.84×10^4	3.62×10^2	0.02×10^1
`BMatches`	1.66×10^4	3.07×10^2	0.02×10^1
`Total:`	2.14×10^6	3.86×10^5	4.80×10^4

TABLE II

DDTs after DDTTR in the 3D Image Reconstruction System

Variable	memory accesses	memory footprint (B)	energy .13μm tech.(μJ)
IMatches	4.02×10^5	6.84×10^2	2.91×10^2
CMatches	3.89×10^5	1.27×10^5	7.02×10^2
MMatches	7.68×10^3	3.81×10^3	0.03×10^1
BMatches	7.16×10^3	3.78×10^3	0.02×10^1
Total	8.06×10^5	1.28×10^5	9.98×10^2

Then, DMMR is applied. In this phase, an optimized DM manager is selected (see [9] for more details) to minimize global power-consumption and memory footprint, but preserving the performance required. The final trade-off between the constraints is decided by the designer among the solutions found by DMMR. After the exploration is performed using the profiling information obtained from DDTTR, the DM manager selected discerns the different behaviours of the DDTs, which include only few sizes after DDTTR, as Figure 4 shows. Consequently, the DM of the system is partitioned in three regions or pools with different block sizes according to the different DDTs in the application. Every block allocated stores the object size and each region uses a double linked list to optimize the time required to traverse and access the data. Each double linked list uses a LIFO order. When an allocated block is freed, the DM manager returns that memory block to the region it came from and becomes available for later block requests inside the range of sizes allowed in that pool.

The previously explained DM manager provides an excellent performance and the fragmentation is minimized. It only adds (due to internal management and fragmentation) approximately a 10% overhead in normalized memory use and 18% in energy dissipation, compared to the results of the bare DDTs shown in Table II. These results are by far better than the results obtained with standard implementations of very-well known allocators [17]. In fact, simple DM managers suffer from a high time-response delay or high degree of fragmentation (internal and external), while more sophisticated ones from standard desktop systems (e.g. the Lea Allocator [17]) that can provide a certain trade-off of performance and memory usage involve a high penalty in power consumption, due to their general-purpose design.

As a result, after DDTTR and DMMR, the final DDTs in the 3D reconstruction application require less DM and memory accesses, and consume less energy than the original ones (see Table II and Figure 3). Finally, a custom DM manager is designed for them to provide a compromise between performance, power consumption and memory footprint and do not add any unnecessary overhead in the DM management of the system. Thus, globally the method improves memory footprint up to 66.84%, estimated power consumption up to 97.93% and performance up to 95.08% compared to the original version.

B. Method Applied to a Real-World Interactive 3D Game

The second application to illustrate our method is a 3D simulation game that interacts with the real world thanks to a frame grabbing device. At real-time, images are taken from our everyday world and obstacles (i.e. walls) are detected on them. Then, in the scenario of detected walls, additional virtual objects (i.e. balls) are generated. These balls can move with 3 degrees of freedom (up/down, left/right, front/back) and interact with the detected walls. When a ball bumps into a wall, it can rebound or get blocked into it. As in the previous case study, this application contains specialized sub-algorithms for its different parts (e.g. 3D rendering or walls detection).

TABLE III

Original DDTs in the 3D simulation game

Variable	memory accesses	memory footprint (B)	energy .13μm tech.(μJ)
VWalls	4.59×10^6	1.72×10^3	2.94×10^3
HWalls	4.01×10^6	1.64×10^3	0.28×10^3
Balls	2.16×10^7	8.42×10^3	1.91×10^3
Total:	3.02×10^7	1.17×10^4	5.13×10^3

The software module studied is the global game controller, which handles the actions of the balls (e.g. movements or crashes) and the interaction between the scenario and the user. This module includes several DDTs and DM is used due to the sources of indeterminism in the system, i.e. position of obstacles in input frames, user movements and number of balls. The relevant DDTs from this module are the following:

- `VWalls` is the list of vertical walls detected in the input images. When the user moves, new walls are detected and the relative positions of the remaining ones are updated.
- `HWalls` is the list of horizontal walls detected in the images. It is also updated after a movement of the user.
- `Balls` is the list of balls that are currently in the system. It is frequently accessed to update the position of the balls in the scene due to movements of the user and their own actions (e.g. destroyed, created, etc.).

These DDTs were originally implemented using single linked lists and due to their unpredictable DM behaviour, they cannot be placed in current embedded handheld devices.

As in the first case study, the DM management of the system is refined with our method. DDTTR is applied and a detailed timing graph is produced (see Figure 5). Also, detailed profiling information of the DDTs is obtained (see Table III). Then, DDTTR explores possible implementations for each DDT considering trade-offs of power consumption, memory footprint and performance, as Figure 6 shows. Minimizing global power consumption, in the final global solution selected (among the circled points in Figure 6), the DDTs `VWalls` and `HWalls` are implemented as 2-layered structures. The first level is a dynamic array of 10 positions. Then, another one of 56 basic elements for `VWalls` and 26 for `HWalls`. In both cases, each basic element is 6 floats. Also, the DDT `Balls` is implemented as a 2-layered structure. A dynamic array of 10 positions stores dynamic arrays of 179 basic elements of 1 float each. As Table IV shows, these optimized DDT implementations reduce significantly energy dissipation and memory footprint compared to the original DDTs.

After the DDTs are optimized, DMMR is used to design a custom DM manager minimizing global power consumption for the game controller. This manager discerns two different access patterns and behaviours of the DDTs: (1) `VWalls` and

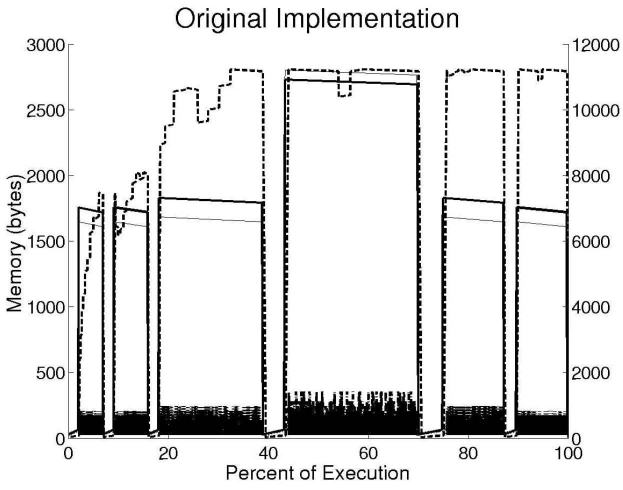

Fig. 5. DM behaviour of the game controller module for 6 frames

TABLE IV

DDTs after DDTTR in the 3D simulation game

Variable	memory accesses	memory footprint (B)	energy .13μm tech.(μJ)
`VWalls`	4.16×10^6	8.30×10^2	1.52×10^2
`HWalls`	3.62×10^6	6.99×10^2	1.28×10^2
`Balls`	2.20×10^6	7.20×10^3	7.54×10^2
`Total:`	9.98×10^6	8.72×10^3	1.03×10^3

`HWalls`. (2) `Balls`. Consequently, the DM of the system is partitioned in two regions or pools. In the pool of the walls, the block size is 24 bytes and the elements are stored in an unordered simple link list that keeps a pointer to the last accessed element for performance purposes. In the pool of the DDT `Balls`, the block size is 26 bytes and the blocks are stored in a double link list with LIFO order. Finally, no coalescing or splitting services are needed. This custom DM manager only adds a 7% overhead in memory use and 21% in energy dissipation to the final DDT figures of Table IV.

In the end, normalized memory footprint is improved up to 22.48% and power consumption up to 75.3% compared to the original implementation. Furthermore, there is a global speedup factor of one order of magnitude for the 3D game.

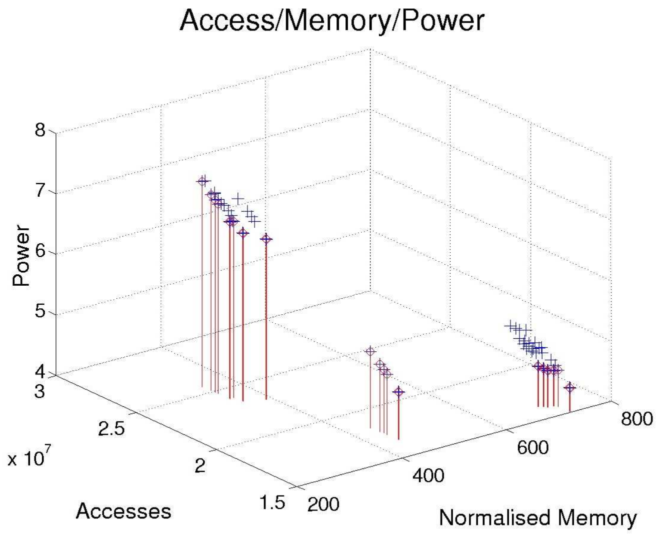

Fig. 6. Optimal DDT implementations for the 3D game. Each point is a combination of DDTs, the circled ones are global solutions (for all DDTs)

V. Conclusions

As multimedia applications have grown in complexity, the design flow of current multimedia systems require extensive time-consuming efforts to reach a correct design of their software and dynamic part for an efficient system integration. In this refinement process, DM management is one of the crucial and most difficult parts in multimedia applications due to the unpredictability of their input data and events (e.g. images or movements of the user). Furthermore, for portable devices, power consumption is a primary constraint. In this paper, we have illustrated a new system-level exploration and refinement method for DM management on two current multimedia applications, namely a new 3D image reconstruction system and a real world interactive 3D game. Our results in these case studies show that multimedia applications greatly benefit from optimal DM management systems. Places where previous implementations formed clear bottlenecks in the code were completely removed by better DDT implementations and an optimal DM manager. As a result, the optimized applications can be ported efficiently to the final embedded multimedia platforms.

Acknowledgements

This work is partially supported by the Spanish Government Research Grant TIC2002/0750, the Fund for Scientific Research - Flanders (Belgium, F.W.O.) through project G.0036.99 and a Postdoctoral Fellowship for Geert Deconinck.

References

[1] id software inc., 2002. `http://www.idsoftware.com`.
[2] Target jr, 2002. `http://www.targetjr.org`.
[3] L. Benini et al. System level power optimization techniques and tools. In *ACM TODAES*, April 2000.
[4] E. D. Berger et al. Composing high-performance mem. allocators. In *Proc. of ACM PLDI*, USA, 2001.
[5] R. Y. Chen et al. Speed and Power Scaling of SRAM's. *ACM Trans. Design Autom. of Elect. Syst.*, January 2001.
[6] J. Cosmas et al. 3D murale, 2002. `http://www.brunel.ac.uk/project/murale/home.html`.
[7] E. G. Daylight et al. Analyzing energy friendly states of dyn. app. exec. in terms of sparse data struc. In *Proc. of ISLPED*, USA, 2002.
[8] N. Jouppi. Western research lab., cacti, 2002. `http://research.compaq.com/wrl/people/jouppi/CACTI.html`.
[9] M. Leeman et al. Methodology for refinement and optimisation of dmm for embedded syst. in multimedia apps. In *Proc. of SiPS*, Korea, 2003.
[10] M. Leeman et al. Power estimation approach of dyn. data storage on a hw sw boundary level. In *Proc. of PATMOS*, Italy, 2003.
[11] T. Mudge. Power: A first class architectural design constraint. *IEEE Computer*, April 2001.
[12] N. Murphy. Safe mem. usage with DM alloc. *Embed. Syst.*, May 2000.
[13] M. Pollefeys et al. Metric 3D surface reconst. from uncalibrated image sequs. In *Lect. Notes in Comp. Science*, Springer-Verlag, 1998.
[14] D. Sarta et al. A data dependent approach to inst. level power estimation. In *Proc. of A. Volta Workshop on Low-Power Design*, Italy, 1999.
[15] T. Tan et al. High-level sw energy macro-modeling. In *Proc. of DAC*, USA, 2001.
[16] N. Vijaykrishnan et al. Evaluating integrated hw-sw optim. using a unified energy estim. framework. *IEEE Trans. on Computers*, 2003.
[17] P. R. Wilson et al. Dynamic storage allocation, a survey and critical review. In *Int. Workshop on Mem. Man.*, UK, 1995.
[18] C. Ykman et al. DMM methodology applied to emb. telecom network systems. *IEEE Trans. on VLSI Systems*, 2002.