Automated Exploration of Pareto-optimal Configurations in Parameterized Dynamic Memory Allocation for Embedded Systems

Stylianos Mamagkakis [1], David Atienza [2,3], Christophe Poucet [4], Francky Catthoor [4], Dimitrios Soudris [1] and Jose M. Mendias [2]

[1] VLSI Center-Democritus Univ., 67100 Xanthi, Greece. {smamagka, dsoudris}@ee.duth.gr
[2] DACYA/UCM, Juan del Rosal 8, 28040 Madrid, Spain. mendias@dacya.ucm.es
[3] LSI/EPFL 1015-Lausanne, Switzerland. david.atienza@epfl.ch
[4] IMEC vzw, Kapeldreef 75, 3001 Heverlee, Belgium. {poucet, catthoor}@imec.be
F. Catthoor also professor at ESAT/K.U.Leuven-Belgium.

Abstract

New applications in embedded systems are becoming increasingly dynamic. In addition to increased dynamism, they have massive data storage needs. Therefore, they rely heavily on dynamic, run-time memory allocation. The design and configuration of a dynamic memory allocation subsystem requires a big design effort, without always achieving the desired results. In this paper, we propose a fully automated exploration of dynamic memory allocation configurations. These configurations are fine tuned to the specific needs of applications with the use of a number of parameters. We assess the effectiveness of the proposed approach in two representative real-life case studies of the multimedia and wireless network domains and show up to 76% decrease in memory accesses and 66% decrease in memory footprint within the Pareto-optimal trade-off space.

1. Introduction

Many Dynamic Memory (DM) allocation solutions are available today for general purpose systems. These are activated with the standardized malloc/free functions in C and the new/delete operators in C++. Support for them is already available at the Operating System (OS) level [2]. Each one of those DM allocators provides a general solution that ignores the special de/allocation behavior and fragmentation outlook of the application that needs them or the underlying memory hierarchy. The same approach is followed in embedded system designs, which rely solely on their embedded O.S. for DM allocation support. The use of OS-based, general-purpose DM allocation usually has an unacceptable overhead in embedded designs, considering the limited resources and hard real-time constraints of embedded systems. Therefore, to achieve better results, application-customized DM allocators are needed [3, 1]. Note that they are best realized in the middleware and not in the platform hardware which would require undesired platform changes for each application (domain) target. In this paper, we introduce a novel tool support to automatically create and explore the trade-offs in the DM allocation parameters. With our new fully automated technique we generate Pareto-optimal DM allocator configurations for the embedded system designer to use according to the application's specific needs. For the first time, our automation support gives embedded system designers a real choice between tens of thousands of highly customized DM allocators instead of the very restricted group of a few OS-based DM allocators.

2. Automated Exploration Tool Overview

The most significant contribution of this paper is the development of a framework to automatically create, map in the memory hierarchy and test any number of DM allocation configurations (see Figure 1). The only input that our tool requires is the list of arrays with the parameter values to be explored for the different configurations. Additionally, our tool can map the DM allocator pools in any memory hierarchy. For example, we can declare that a dedicated pool for 74-byte blocks must be placed onto the L1 64 KB scratchpad memory, while a general pool and a dedicated pool for 1500-byte blocks must use the 4 MB main memory. Then, our tool takes care of the DM allocator implementation to support the mapping of these pools in the corresponding memory hierarchy layers. To this end, we have developed a C++ library that includes more than 50 modules, which can be linked in any way with the use of templates and MIXINS inheritance to create custom DM alloca-

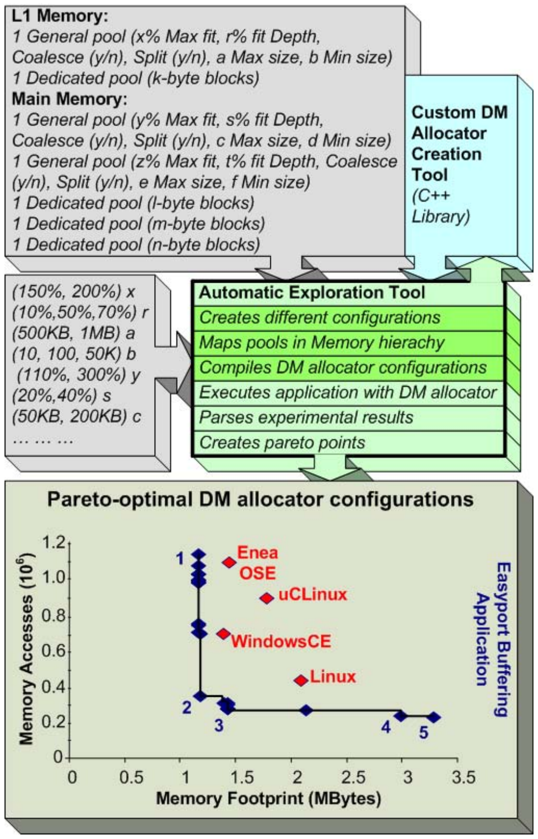
L1 Memory:
1 General pool (x% Max fit, r% fit Depth, Coalesce (y/n), Split (y/n), a Max size, b Min size)
1 Dedicated pool (k-byte blocks)
Main Memory:
1 General pool (y% Max fit, s% fit Depth, Coalesce (y/n), Split (y/n), c Max size, d Min size)
1 General pool (z% Max fit, t% fit Depth, Coalesce (y/n), Split (y/n), e Max size, f Min size)
1 Dedicated pool (l-byte blocks)
1 Dedicated pool (m-byte blocks)
1 Dedicated pool (n-byte blocks)
Custom DM Allocator Creation Tool (C++ Library)
(150%, 200%) x
(10%,50%,70%) r
(500KB, 1MB) a
(10, 100, 50K) b
(110%, 300%) y
(20%,40%) s
(50KB, 200KB) c
… … …
Automatic Exploration Tool
Creates different configurations
Maps pools in Memory hierachy
Compiles DM allocator configurations
Executes application with DM allocator
Parses experimental results
Creates pareto points
Pareto-optimal DM allocator configurations
Memory Accesses (10^6)
Memory Footprint (MBytes)
Easyport Buffering Application
Enea OSE
uCLinux
WindowsCE
Linux
1
2
3
4
5

Figure 1. Automated Exploration Tool Flow

tors. The tool works in a plug-and-play manner and the dynamic application's source code is not altered to call the appropriate DM allocator from the library.

The next step of our tool is the automated selection of Pareto-optimal configurations and involves the simulation (i.e. execution) of our dynamic application for each one of the different DM allocator configurations. These configurations were already defined, constructed and implemented automatically in the previous step. We have implemented profiling tools to test and profile all the different DM allocator configurations for the defined memory hierarchy, and get results for mem. accesses, mem. footprint and energy consumption for each level of the memory hierarchy. The results are provided either on a GUI or in a format easy to import to Excel or Gnuplot. Then, the Pareto-optimal curves to evaluate the tradeoffs of the configurations can be provided automatically with the use of our tool (as shown in the upper part of Figure 1). The tool (written in Perl and O'Caml) parses all the experimental results data and provides Pareto-optimal curves for the chosen metrics (as shown in the lower part of Figure 1). Note the importance of our fast parsing of the profiling data (less than 20 seconds), which can reach Gigabytes for one single configuration.

3. Case Studies and Experimental Results

Our first case study is the Easyport wireless network application produced by Infineon [4]. We have obtained a range in the total memory footprint of a factor 11 and for the memory accesses of a factor 54 within all the available DM allocator configurations. Then, we have used our tool to parse all the configurations to produce the Pareto-optimal configurations. We conclude that we have 15 Pareto-optimal configurations. We can decrease the total amount of memory footprint up to a factor of 2.9 and the memory accesses up to a factor of 4.1 within all the Pareto-optimal DM allocator configurations (as shown in Figure 1). Also, we can decrease the total memory energy consumption up to 71.74% and the execution time up to 27.92% within all the Pareto-optimal DM allocator configurations. Our second case study is the MPEG4 Visual Texture deCoder (VTC) [5], which deals with still texture decoding. We have managed a reduction of up to 82.4% for energy consumption and up to 5.4% for execution time within the available Pareto-optimal configurations.

4. Conclusions

New design tools must be available to the designer in order to explore the tradeoffs between various DM allocation configurations and to suitably use the resources in final embedded devices. In this paper we have presented a full automation support (including GUI) to explore the parameters of DM allocation subsystems and evaluate their possible tradeoffs in a Pareto-based fashion.

5. Acknowledgements

This work is partially supported by the European program AMDREL IST-2001-34379 and the Spanish Government Research Grant TIN2005-5619. We want to thank Matthias Wohrle (Advanced Systems and Circuits group, Infineon Tech) and Arnout Vandecappelle (IMEC) for their help with the Easyport application.

References

[1] D. Atienza et al. Dynamic Memory Management Design Methodology for Reduced Memory Footprint in Multimedia and Wireless Network Applications, DATE '04, France 2004, IEEE Press, ISSN: 1530-1591/04, pp. 532 - 537.

[2] P. R. Wilson, et al. Dynamic storage allocation, a survey and critical review. In *Int. Workshop on Mem. Manag.*, UK, 1995.

[3] E. D. Berger, et al. Composing high-performance memory allocators. In *Proc. of ACM SIGPLAN PLDI*, USA, 2001.

[4] Infineon Easyport. http://www.itc-electronics.com/CD/infineon

[5] MPEG-4. http://www.chiariglione.org/mpeg/standards/mpeg-4/mpeg-4.htm