Custom Dynamic Memory Allocation Tool (DMMR-MATISSE)

S. Mamagkakis, D. Atienza, C. Poucet, F. Catthoor, D. Soudris and J. Mendias

In this demonstration, we introduce a novel tool support to automatically create and explore the trade-offs in the Dynamic Memory (DM from now on) allocation parameters. With our 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. The Custom Dynamic Memory Allocation Tool was developed in collaboration of IMEC research center with DUTH and DACYA universities. The most significant contribution 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 allocators. 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.

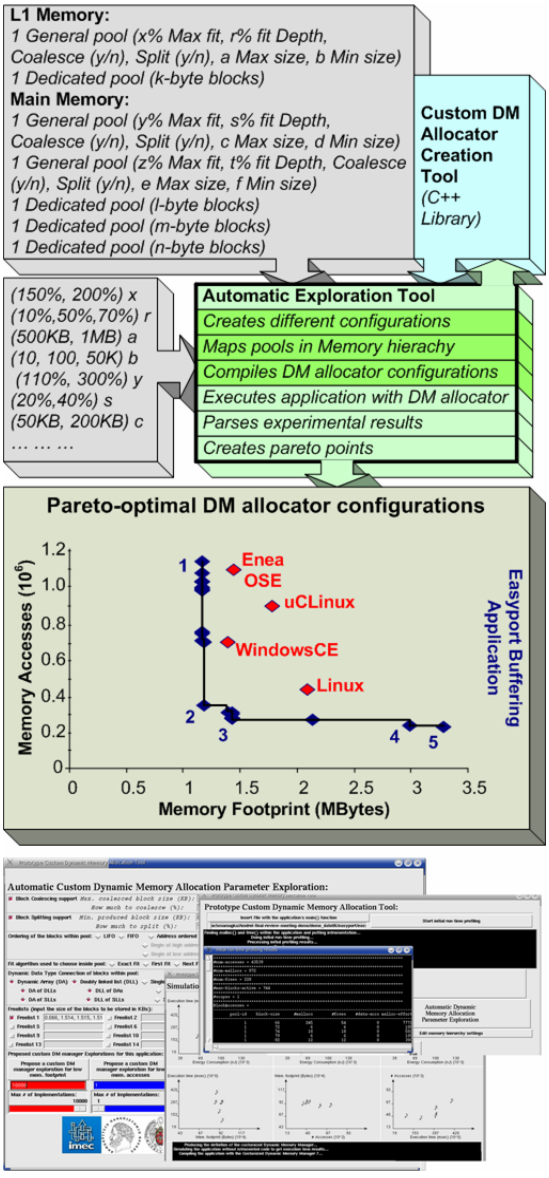

Figure 1. Tool flow and GUI