Multi-Objective Design Space Exploration of Embedded Systems in a Grid Environment

Vincenzo Catania, Gianmarco De Francisci Morales, Alessandro G. Di Nuovo, Maurizio Palesi, Davide Patti

Dipartimento di Ingegneria Informatica e delle Telecomunicazioni, Università degli Studi di Catania, Italy

Abstract

Design Space Exploration (DSE) is a hot issue in embedded system design. One of the main difficulties which characterizes DSE problems is related to the computational power requirements for implementing any reliable DSE strategy. To be reliable, a DSE strategy has to operate at the appropriate abstraction level. It cannot be too high as important details which affect the objectives to be optimized might be not modeled appropriately. On the other hand, operating at a low abstraction level, could make it unfeasible to evaluate the objectives for large real life data sets. The use of system-level simulators built around an instruction-set simulator is considered as the most viable solution in trading-off both the aspects of accuracy and efficiency. Although they allow to evaluate any system configuration running a given application in a reasonable amount of time, they are still too expensive to be used as a core of a DSE strategy which requires the simulation of thousands of system configurations. In this paper we assess the use of High Performance Computing (HPC) in DSE of a complex highly parameterized VLIW based System-on-a-Chip (SoC) platform. Experiments show that the conventional wisdom of linear decrease in exploration time as the number of available processors increases is violated starting from a relatively low number of processors mainly due to communication overhead and I/O bottleneck.

Simulation Environment in HPC infrastructure

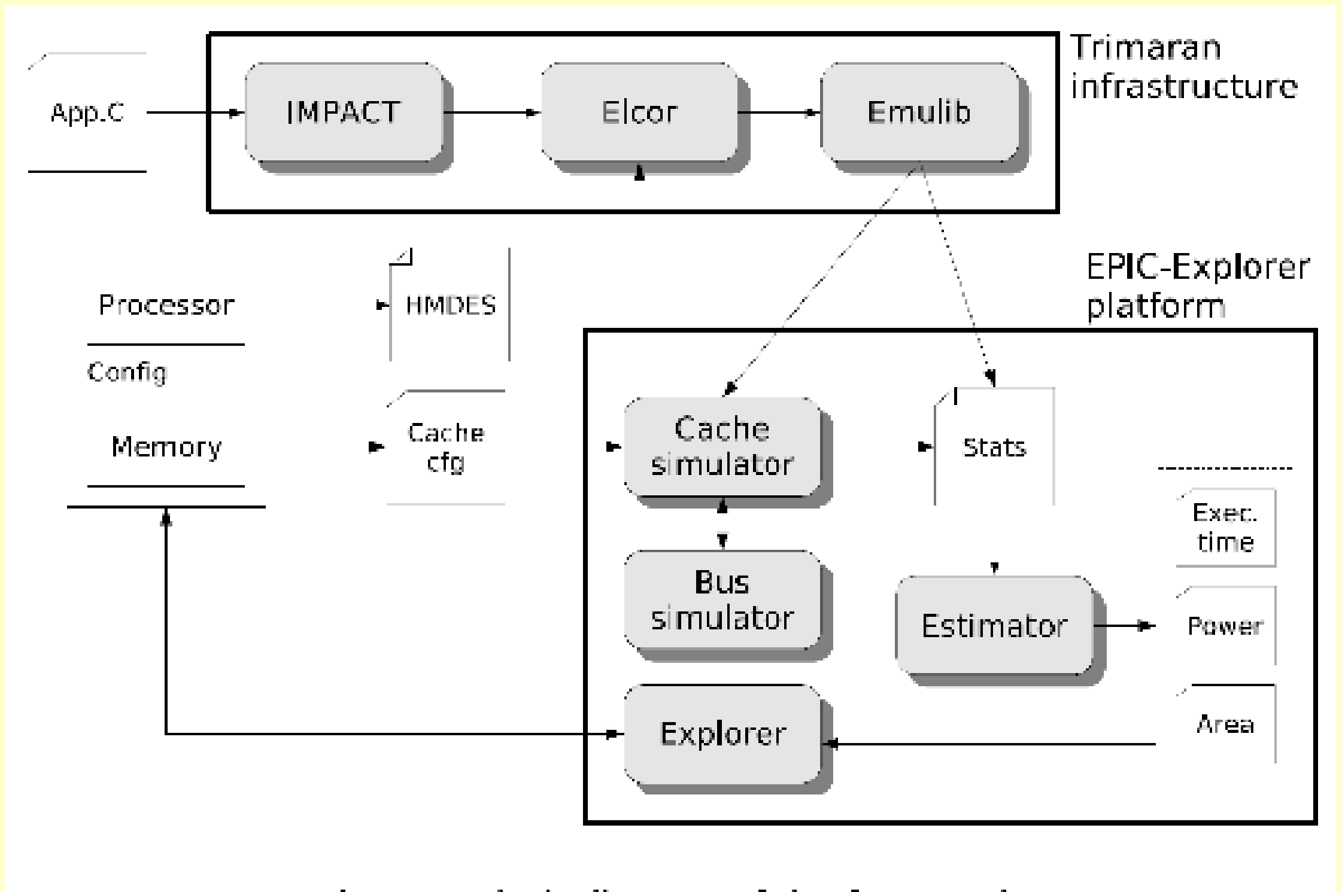


Fig. 1. Block diagram of the framework.

The main step was to integrate the Epic-Explorer suite into the HPC environment. Figure 2 shows the exploration flow on HPC environment. Trimaran required some modifications to allow for multiple processes to operate on the same disk without conflicts. A tree directory structure was employed under the workspace root with the processor configurations' identifiers as intermediate nodes and the cache configurations' as leaves. This way multiple instances of Trimaran can compile codes for different architectures under different directories without interfering with each other.

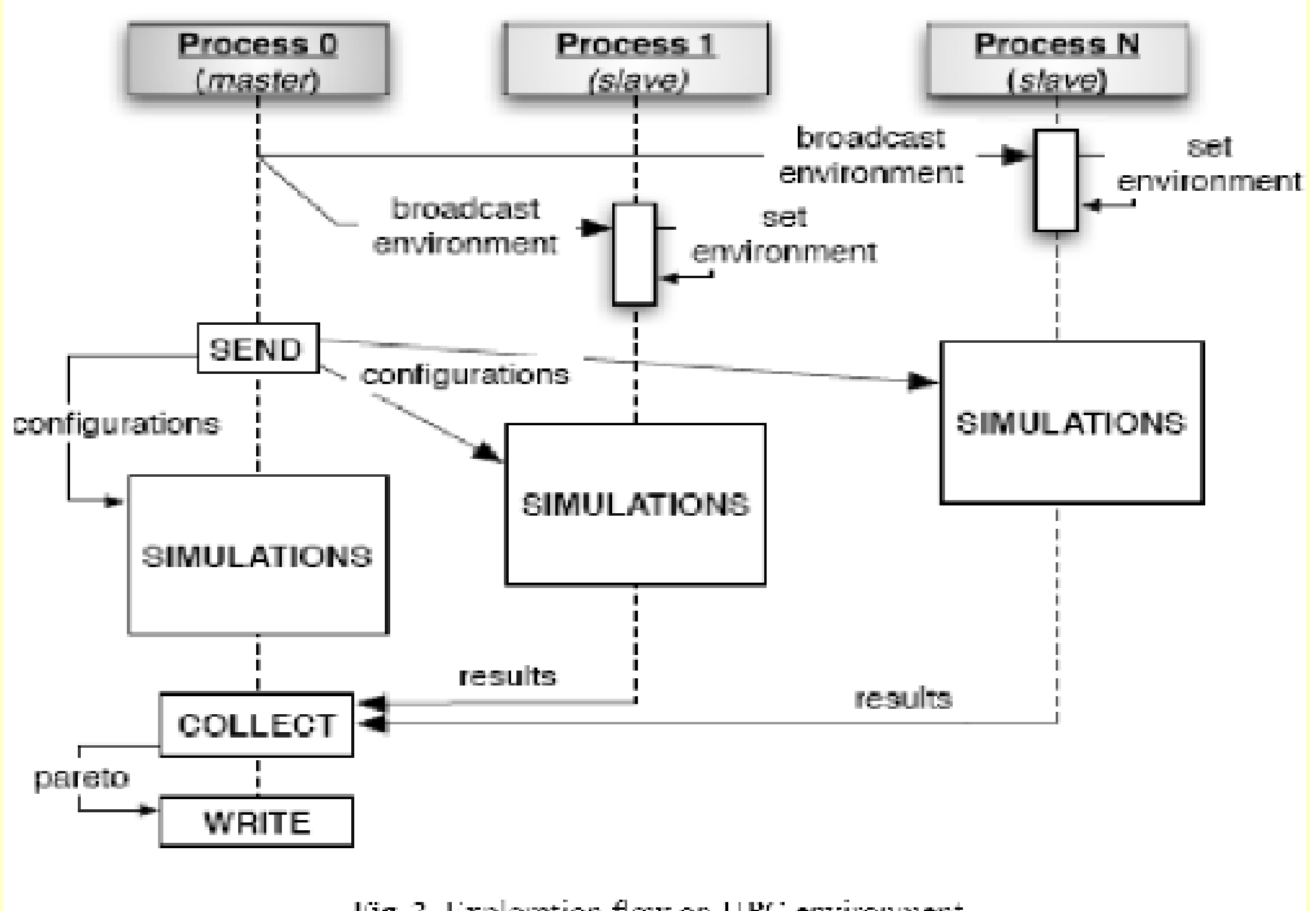
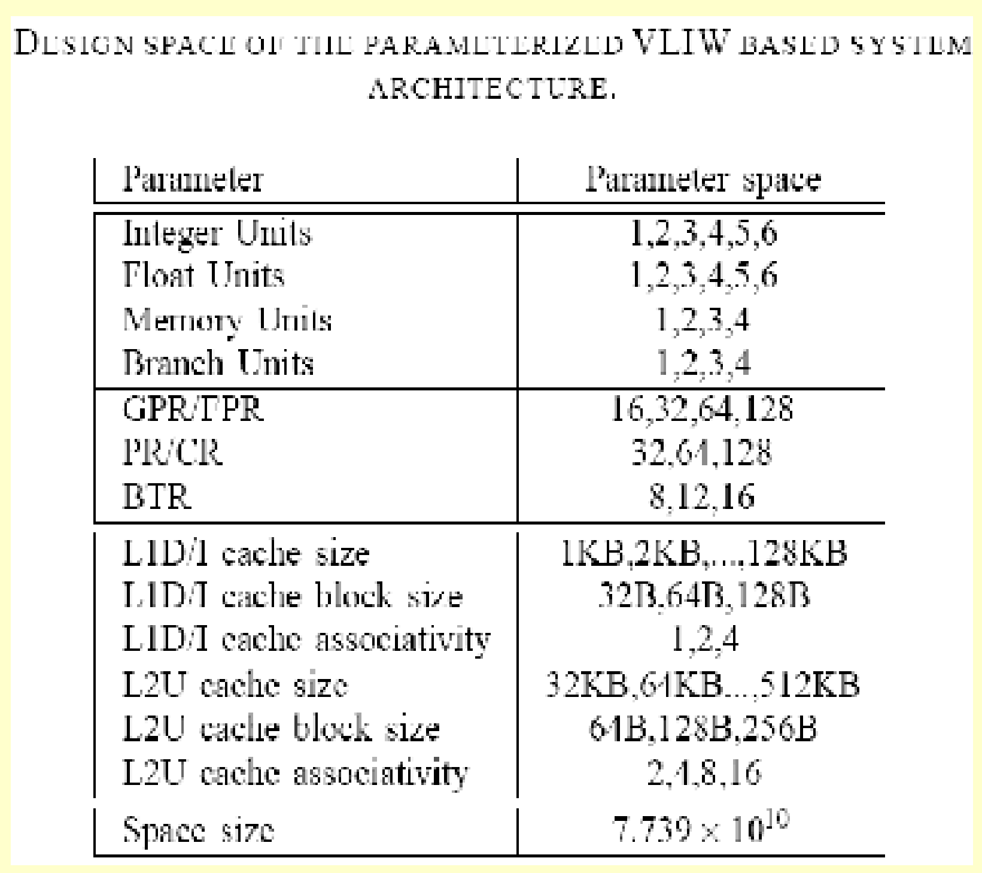


Fig. 2. Exploration flow on HPC environment.

A bit of care had to be taken with benchmarks needing external input, because of the multi-directory structure created the initialization sequence for the benchmarks had to be modified to integrate an in-place setup of the executable. This way the simulation library of Trimaran (emulib) runs in an environment equivalent to the original one.

The parameterized system architecture used in this work is based on HPL-PD [17] which is a parametric processor Fig. 1. Block diagram of the framework meta-architecture designed for research in instruction-level parallelism of EPIC/VLIW architectures1. The HPL-PD opcode repertoire, at its core, is similar to that of a RISC-like load/store architecture, with standard integer, floating point (including fused multiply-add type operations) and memory operations. The configurations to be simulated were randomly chosen in the design space in the following Table:



DESIGN SPACE OF THE PARAMETERIZED VLIW BASED SYSTEM ARCHITECTURE.

Parameter	Parameter space
Integer Units	1,2,3,4,5,6
Float Units	1,2,3,4,5,6
Memory Units	1,2,3,4
Branch Units	1,2,3,4
GPR/FPR	16,32,64,128
PR/CR	32,64,128
BTR	8,12,16
L1D/I cache size	1KB,2KB,...,128KB
L1D/I cache block size	32B,64B,128B
L1D/I cache associativity	1,2,4
L2U cache size	32KB,64KB...,512KB
L2U cache block size	64B,128B,256B
L2U cache associativity	2,4,8,16
Space size	7.739×10^{10}

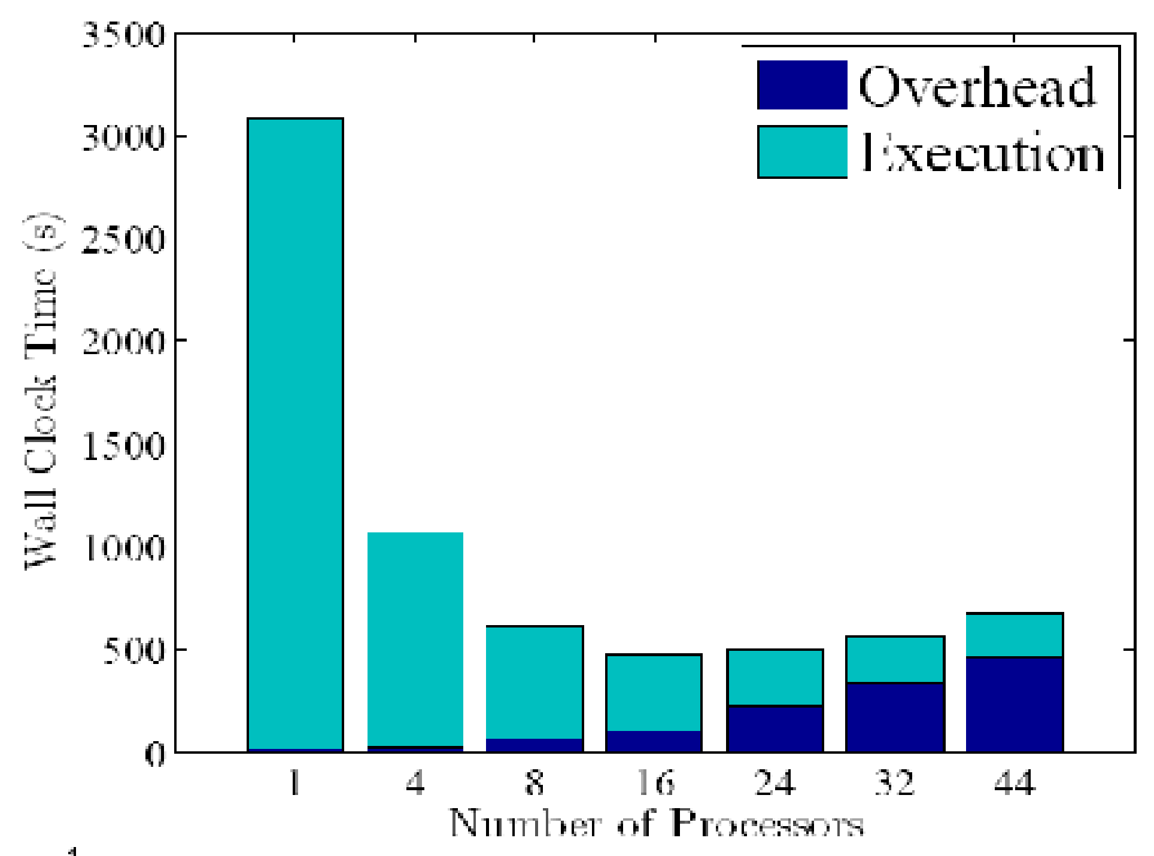
Architectural parameters can be classified in three main categories: *register files*, *functional units* and *memory subsystem*. The first two depend on the implementation of the VLIW core and regard the size of the register files, in terms of the number of registers contained in each of them, and the number of functional units for each type of unit supported. As far as the former are concerned, five different types of register files can be identified: GPR (32-bit registers for integers), FPR (64-bit registers for floating point values) PR (1-bit registers used to store the Boolean values of predicated instructions), BTR (64-bit registers containing information about possible future branches) and CR (32-bit control registers containing information about the internal state of the processor). The functional units involved are: *Integer units*, *floating point units*, *memory units* (associated with load/store operations) and *branch units* (associated with branch operations). With respect to the memory sub-system, the parameters that can be modified are the *size*, *associativity* and *block size* for each of the three caches: First-level data cache (L1D), first-level instruction cache (L1I) and second-level unified cache (L2U).

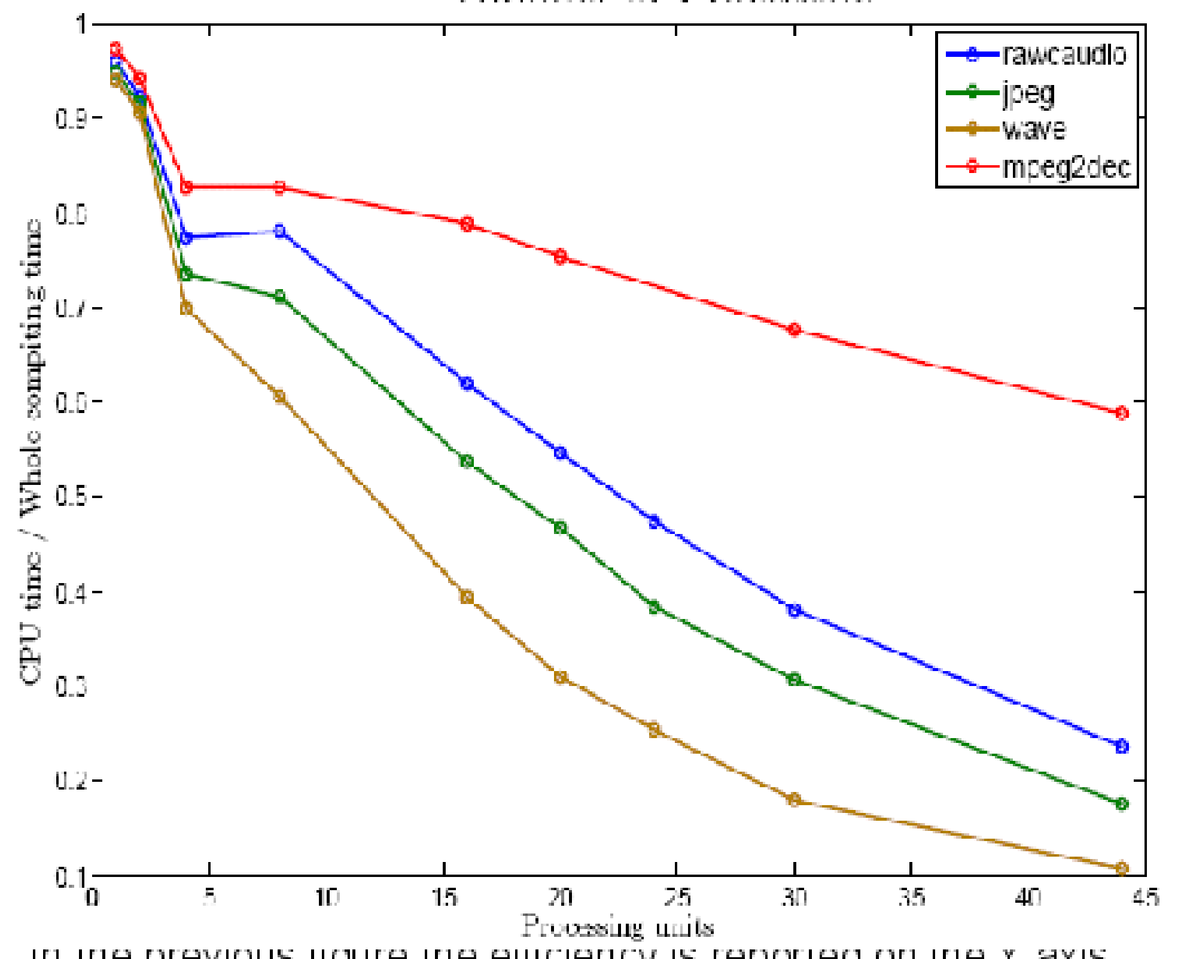
Conclusions

In this paper we presented a case study of design space exploration of a complex highly parameterized VLIW based SoC platform. The 18 free parameters which characterize the platform span a design space of over 10^9 system configurations. Even considering an evaluation time of a few seconds for each configuration, exhaustive exploration would take hundreds of years on single machine. We test the use of High Performance Computing (HPC) as a viable solution to tackle with DSE related problems. However, a maximum reduction of one order of magnitude in exploration time has been observed in our experiments. Meanwhile statistical or machine learning approaches to the DSE problem presented in the literature, achieved savings of two order of magnitude than classical approaches. This result remarks that the use of a combination of statistical and/or machine learning approaches in HPC environments likely to be the most promising way to significantly reduce the exploration time.

Tests and Results

The testing was done using the following configuration: 16xIBM LS21 Blades with 2 Opteron 2.6 GHz dual core processors (for a total of 4 cores per blade) equipped with 8GB of DDR and a 73GB SATA HD, linked by Gigabit Ethernet.
The cluster used for the tests was configured to allow 1 process per core (4 processes per host) and one of the hosts is reserved to run cluster services and to manage the jobs, so the maximum reachable number of MPI processes was 60.
The software package was installed on the dedicated host that was also the cluster coordinator. The package was mirrored via scp by the batch queue manager on a per-job basis on all the hosts involved in the parallel computation.
The tests were done with no interference from other jobs in the cluster, so the scheduler tried to allocate the processes as close as possible in order to fill one host before employing the next one. This is an advantage because it reduces the number of copies of the software to be made.
We run the simulation of 1000 configurations using a growing number of processors, and even using an exponentially growing number of processors the wall clock time was reduced only by an order of magnitude that time needed to setup the environment increases with the number of hosts. In our HPC environment each host has its own storage unit, where the simulation environment has to be copied. For this reason it waste time even within a single host due to remote dispatching to the queue manager, and, in addition, due to the use of computational resources for the process creation and management, and for the allocation and deallocation of environment data.





In the previous figure the efficiency is reported on the Y axis, expressed as the ratio of cpu time to the whole exploration time used, obtained with the formula
CPU TIME/(WALL CLOCK TIME×NUM PROCESSES)
This gives us an idea of how much of the whole computing time was spent doing something useful, and indirectly an index of the efficiency of the calculation. This is also an indirect estimate of the I/O boundness of the job configuration. As we can notice we get high efficiency for 1 or 2 processes, but start to get worse results with 4 processes. After that we get again good results with 8 processors and then start to loose efficiency almost linearly. This can be interpreted as an I/O bottleneck effect, in fact up to 4 processes we are adding more CPU power but we are using the same disk, so we quickly fill the available disk bandwidth resulting in more idle wait. As soon as we get to 8 processes the benchmarks' efficiency grows again because of the added disk. Simpler benchmarks as wave that do not produce a lot of output are less affected by this and get worse efficiency because of the copying overhead. Furthermore the benefits of more disks rapidly get overtaken by the increased overhead.

UNIONE EUROPEA

Ministero dell' Università e della Ricerca