High Performance Computing for Embedded System Design: A Case Study

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

In this paper we assess the use of High Performance Computing in Design Space Exploration of a complex highly parameterized Very Long Instruction Word based System-on-a-Chip platform. Experiments show that the conventional belief of linear decrease in exploration time as the number of available processors increases is discredited starting from a relatively low number of processors mainly due to communication overhead and I/O bottleneck.

1. Introduction

The increasing complexity of modern electronic systems, together with increasing time-to-market pressure, has led the embedded system industry to adopt pre-designed System-on-a-Chip (SoC) architectures known as *SoC platforms* [3, 6, 4].

A SoC platform is typically made up of a number of parameterized intellectual property (IP) cores communicating via an interconnection system which is also parametric. Each parameter has an associated range of discrete values within which it can vary. As the number of possible configurations that can be mapped on a parametric system is equal to the product of the cardinalities of the ranges of variation for each parameter, the size of the space of configurations grows exponentially as the number of parameters increases. Exhaustively exploring the design space looking for the best system configuration for a given application (or, more in general, for a set of applications) is computationally infeasible in practical cases. In fact, each visited system configuration must be evaluated by means of a system-level simulation which typically is a computationally intensive task. It is therefore of fundamental importance to define exploration methods that can combine two basic but unfortunately conflicting requirements: efficiency and accuracy. Efficiency is a measure of the effort required to complete an exploration. Accuracy is a quality index for the obtained solutions. The ability to find a good trade-off between these requirements is the challenge now facing Design Space Exploration (DSE) methods. Another important issue in this context is that the search for optimal configurations is a multi-objective optimization problem, where some of the objectives conflict with others, for example performance and power consumption. Although this causes a considerable increase in the complexity of DSE strategies, it has the advantage of offering the SoC designer not one but a set of optimal configurations (Pareto optimal set) from which he/she can choose the one that best represents the desired trade-off level allowing for the constraints which have to be met.

In this paper we present 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 DSE related problems. To the best of our knowledge, all the DSE techniques proposed in literature rely on the use of a computation infrastructure which is mono-processor in nature. We believe that the availability of distributed or grid computational infrastructures should be exploited and analyzed in the context of DSE to cope with the ever growing increase in complexity of current and next generation SoC platforms. It should be pointed out that HPC is not as cheap as statistical or machine learning approaches (e.g. an HPC machine, like the one we used for this work, costs about 50 thousands euros).

2. Simulation environment and the HPC infrastructure

To evaluate and compare the performance indexes of different architectures for a specific application, one needs to simulate the architecture running the code of the application. In this work we use the Epic-Explorer [1] simulation environment. To make architectural exploration possible both the compiler and the simulator have to be retargetable. Trimaran [5] provides these tools and thus represents the pillar around which Epic-Explorer is built. Epic-Explorer is an environment that not only allows to evaluate any instance of a platform in terms of area, performance

978-0-7695-3277-6/08 $25.00 © 2008 IEEE
DOI 10.1109/DSD.2008.110

Table 1: Evaluation time for a simulation (compilation + execution) for several multimedia benchmarks on a Opteron 2.6 GHz Linux Workstation.

Benchmark	Description	Input size (KB)	Evaluation time (sec)
`wave`	Audio Wavefront computation	625	7.7
`g721-enc`	CCITT G.711, G.721 and G.723 voice compressions	8	25.9
`jpeg-codec`	jpeg compression and decompression	128	33.2
`mpeg2-dec`	MPEG-2 video decoding	400	143.7
`adpcm-enc`	Adaptive Differential Pulse Code Modulation speech encoding	295	22.6
`adpcm-dec`	Adaptive Differential Pulse Code Modulation speech decoding	16	20.2
`fir`	FIR filter	64	9.1

and power, but also implements various design space exploration techniques.

The parameterized system architecture used in this work is based on HPL-PD [2] which is a parametric processor meta-architecture designed for research in instruction-level parallelism of EPIC/VLIW architectures[1].

Table 1 reports the computation effort needed for one evaluation (i.e. simulation) of just a single system configuration for several media and digital signal processing application benchmarks.

The first step to integrate the Epic-Explorer suite into the HPC environment was using MPI (Message Passing Interface) to create a parallel program. Figure 1 shows the exploration flow in the 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 configuration identifiers as intermediate nodes and the cache configuration identifiers as leaves. This way multiple instances of Trimaran can compile codes for different architectures under different directories without interfering with each other. 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.

We wrote the JDL (job description language) for the MPI job and started launching Epic-Explorer on the cluster, but

[1]EPIC is an extension of the VLIW approach; where it is not necessary to make a distinction we will only use the term VLIW for the sake of simplicity.

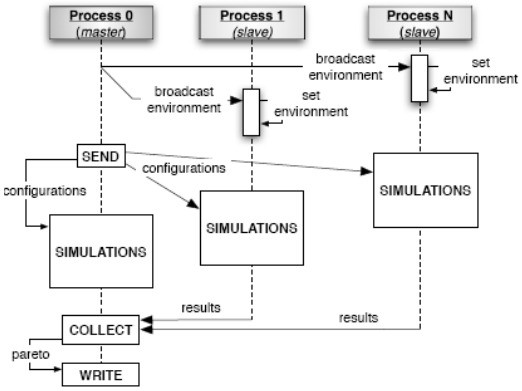

Figure 1: Exploration flow on HPC environment.

encountered some problems during the first phases of testing. The program executed correctly with 1, 2 and 4 processors, but as soon as we required 8 processors the program would crash reporting communication problems. Analyzing the problem we found that it was caused by an incongruence in the environment the job found on different hosts. In fact in our setup up to 4 processes can be executed on one host, and only using 5 or more we need to employ a second host. More specifically in the second host the environment variable that defines the workspace directory was incorrectly set by the middleware/queue manager as result of a MPI integration bug. We know the environment is correct on the host that starts the computation (master host), and the home directory tree gets mirrored via SCP to all the hosts. So to workaround the problem we have only to broadcast the correct environment variable from the master host to all the slave hosts, and then correct the environment before starting the actual computation.

3. Experiments and Results

In this section we perform an extensive evaluation of the proposed framework with an exploration of 1000 configurations of the parameterized VLIW based system architecture being studied. The configurations to be simulated were randomly chosen in the design space in Table 2 and executed on the cluster infrastructure described in subsection 3.1. The numerical results are presented in subsection 3.2.

3.1. The HPC Infrastructure

The testing was done using the following configuration: $16 \times$ *IBMLS*21 Blades with 2 Opteron 2.6 GHz dual core processors (for a total of 4 cores per blade) equipped with 8GB of DDR2 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 $4 \times 15 = 60$.

Table 2: 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}

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.

Another configuration was also tested in which all the hosts share a single distributed file system (GPFS). This eliminates the need to copy the package on all the hosts, but poses a new problem because of the I/O bottleneck in writing concurrently on a single file system.

The former configuration shows a performance advantage over the latter one. The disadvantage is that big jobs that produce a lot of output cannot be run on a single host, for example the 'mpeg2dec' benchmark produces about 130GB of output with a random exploration of 1000 configurations, this does not fit in a single HD so we had to scale down to 100 configurations to evaluate the performances for 1, 2 and 4 processes (that run on a single host). The values obtained were therefore multiplied by 10 as an approximation with linear interpolation.

Better performances could possibly be achieved if the software package were exported via NFS to all the blades for reading, while the output would still be written on the local disk. This way the performance advantages of both the previous solutions can be obtained, namely eliminating the need for copies and avoiding the I/O bottleneck of distributed concurrent writes on a single shared file system. This solution has yet to be explored because of a bug in the HPC infrastructure.

3.2. Experiments

Figure 2 reports the total time for an exploration that requires the simulation of 1000 configurations using a growing number of processors. The total time is normalized by the time needed on a single processor. As we can see in Figure 2 even using a growing number of processors the wall clock time was reduced only by an order of magnitude.

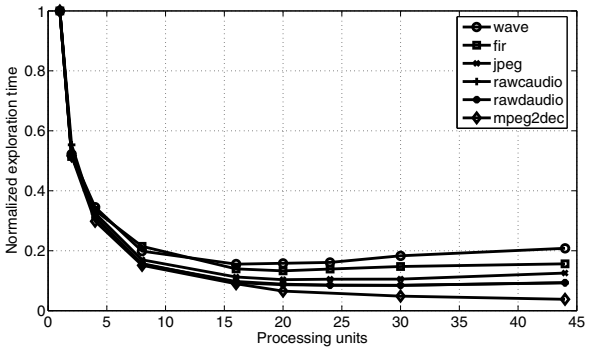

Figure 2: Normalized time required for an exploration of 1000 configurations with an exponentially growing number of processors.

Increasing the number of processes quickly leads to the saturation of computation time. We have a very steep performance increment passing from 1 to 2 processes, almost cutting in half the exploration time. We still get good benefits from the parallelization employing 4 or 8 processes, but soon after that we reach the saturation point. The simpler benchmarks as `wave` even start to get worse performances while the more complex ones like `mpeg2dec` still have some little benefits using a number of processes as high as 60. The reasons behind this behavior should be found in the increased overhead of managing a large number of processes on an equally large number of hosts, like copying the files needed across them and gathering the final results. As an example Table 3 reports the time needed to run just one simulation of the JPEG-codec with a varying number of hosts. In practice just one processing unit actually executes the simulation, all the others just setup the environment needed for the simulation. We can see in Figure 3 that time needed to setup the environment increases with the number of hosts. In fact, as said before, in our HPC environment each host has its own storage unit, where the simulation environment has to be copied.

In Table 3 we can also observe that the HPC enviroment wastes 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.

A different but equally interesting point can be made looking at Figure 4, in which the efficiency of the explorations with an exponentially growing number of processors is shown.

The efficiency is reported on the Y axis, expressed as the

Table 3: Time needed to run just one simulation of jpeg-codec with a growing number of hosts.

Processing Units	Hosts	Wall time	CPU time
Single Machine	1	34.5	33.2
1	1	90.5	33.2
4	1	99.4	33.2
8	2	262.9	62.0
16	4	426.7	123.2
32	8	982.7	294.8
44	11	1469.7	457.7
60	15	1965.4	602.5

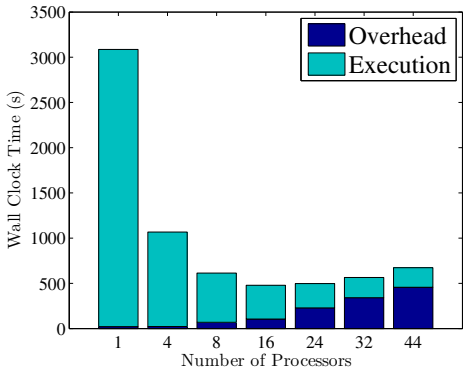

Figure 3: Time needed to run just one simulation of JPEG-codec with a growing number of hosts.

ratio of cpu time to the whole exploration time used. 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.

Strategies could be taken to alleviate this problem, such as exporting via NFS the read-only files of the software package, or implementing a tree-like copy algorithm to speed up the process, but we still would incur in the bandwidth cap of the underlying network.

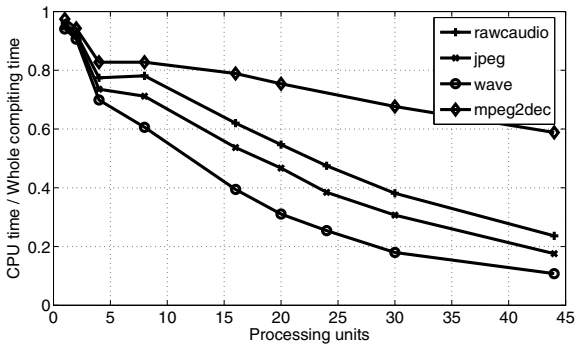

Figure 4: Efficiency ratio for an exploration of 1000 configurations with an exponentially growing number of processors.

4. 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 over classical approaches. This result remarks that the use of a combination of statistical and/or machine learning approaches in HPC environments is likely to be the most promising way to significantly reduce the exploration time.

References

[1] G. Ascia, V. Catania, M. Palesi, and D. Patti. EPIC-Explorer: A parameterized VLIW-based platform framework for design space exploration. In *First Workshop on Embedded Systems for Real-Time Multimedia (ESTIMedia)*, pages 65–72, Newport Beach, California, USA, Oct. 3–4 2003.

[2] V. Kathail, M. S. Schlansker, and B. R. Rau. HPL-PD architecture specification: Version 1.0. Technical report, Compiler and Architecture Research HP Laboratories Palo Alto HPL-93-80, 2000.

[3] K. Keutzer, S. Malik, R. Newton, J. M. Rabaey, and A. Sangiovanni-Vincentelli. System-level design: Orthogonalization of concerns and platform-based design. *IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems*, 19(12):1523–1543, Dec. 2000.

[4] P. Nsame and Y. Savaria. A customizable embedded soc platform architecture. In *4th IEEE International Workshop on System-on-Chip for Real-Time Applications*, pages 299–304.

[5] An infrastructure for research in instruction-level parallelism. `http://www.trimaran.org/`.

[6] F. Vahid and T. Givargis. Platform tuning for embedded systems design. *IEEE Computer*, 34(3):112–114, Mar. 2001.