Multicore Early Design Stage Guaranteed Performance Estimates for the Space Domain

Mikel Fernandez[†], Gabriel Fernandez[§,†], Jaume Abella[†], Francisco J. Cazorla[†]
[†]Barcelona Supercomputing Center (BSC) [§] Universitat Politecnica de Catalunya

***Abstract*—The ability to produce early guaranteed performance (worst-case execution time) estimates for multicores, i.e. before software from different providers gets integrated onto the same critical system, is pivotal. This helps reducing lately-detected costly-to-handle timing violations. An existing methodology creates 'copy' (surrogate) applications from the execution in isolation of each target application. Surrogate applications can be used to upperbound multicore contention delay, and hence WCET estimates in multicores. However, this methodology has only been shown to work on a simulation environment. In this paper we show the work we have carried out to adapt this technology to a real multicore processor for the space domain.**

I. Introduction and Background

The size of the software component in every new critical system is on the rise. To handle software complexity, *Integrated Architectures*, such as Integrated Modular Avionics (IMA) in avionics and IMA for space (IMA-SP) for the space domain, are increasingly used in critical domains. They allow OEMs (original equipment manufacturers) integrating critical and non-critical software developed by multiple software providers (SWPs) into the same system. With Integrated Architectures, the OEM provides a specification on the functional requirements of the software and its worst-case execution time, which must be fulfilled by SWPs. Time budgets are defined according to a global schedule designed by the OEM.

Applications are developed incrementally, assessing that the application fulfills its requirements in every release. Multicore processors, which are becoming the de facto computing solution in all critical domains, challenge the assessment of whether applications meet their deadlines, i.e. they fit their assigned timing budget. This occurs because the timing behavior of an application depends on how other contender applications – likely developed by other SWPs – use multicore shared resources. To make things worse, contender applications may not be shared among SWPs or with the OEM for intellectual-property restrictions. This relegates assessing timing constraints to late-design phases when potential violations of applications' deadlines are costly to handle, potentially jeopardizing the whole design and product's time to market.

Surrogate Applications (SurApp) [3] offer a path to attack this challenge. A SurApp mimics the activities generated by the real (target) application on hardware shared resources (e.g. accesses to buses, memory, and caches), effectively copying the impact the application can generate on the timing behavior of others. Let A and B be two applications: with SurApps, the slowdown that A suffers when it runs against the SurApp of B ($SurApp_B$) matches the slowdown A suffers when it runs against B. SurApps, unlike real applications, can be shared without revealing any relevant IP and enabling time estimates in early design phases, before target applications are integrated, when timing violations would be costly to handle.

While [3] provides an automatic framework to generate SurApps, it has not been validated in any real board yet, but just on a simulation environment. We cover this gap by making the first implementation of the SurApp approach in a real setup. In particular on the NGMP, a processor considered for several European Space Agency's missions.

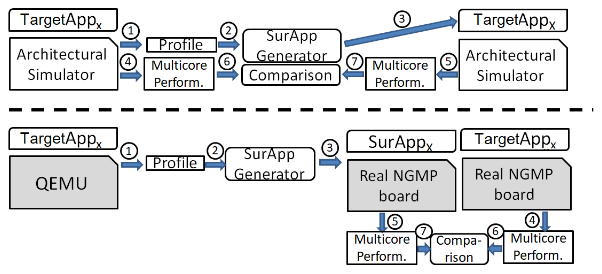


Fig. 1. Surrogate Application and experimental workload generation pipeline

Our results show that i) SurApps cause similar slowdowns on other contender applications as real applications on the NGMP processor. And ii) estimates tend to over-approximate the impact of contention, which is desirable w.r.t. under-approximating applications performance in critical systems.

II. Existing and Proposed Approach

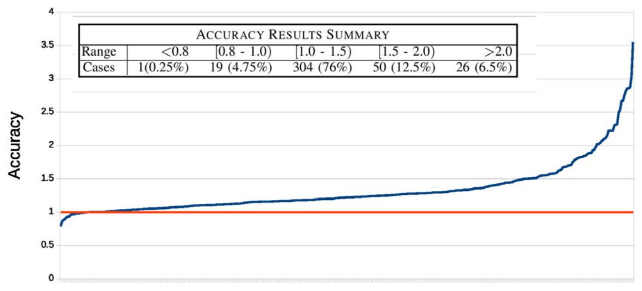
The top strand of Figure 1 shows the existing methodology proposed for hardware simulators. First, the Target Application *TargetApp*$_x$ is run in isolation on an architectural simulator to extract information about cache reuse distance. From this information, the SurApp generator creates a *SurApp*$_x$ that copies *TargetApp*$_x$. In order to make a performance comparison, in the target N-core multicore, in a first run on the simulator we run N-1 copies *TargetApp*$_x$ against the application under analysis (*AUA*). In a second experiment, we run N-1 *SurApp*$_x$ against the same *AUA* and compare the slowdown the *AUA* suffers in both cases. If they are close enough, *SurApp*$_x$ can be used instead of *TargetApp*$_x$ during timing validation and verification to capture the impact of multicore contention.

In the bottom strand we see the methodology we use for real boards. In this case, we first run *TargetApp*$_x$ in a *qemu* (a multi-architecture emulator) for SPARC, on a x86 laptop, and collect address traces that we use to extract the information that the SurApp Generator uses. Then, in two different experiments, three copies of the *SurApp*$_x$ and three copies of the *TargetApp*$_x$, respectively, are run against the *AUA* in the real 4-core SPARC board. In both cases we measure the slowdown the *AUA* suffers. Note that, in these experiments – unlike those in the upper strand – the execution time impact comparison for the *AUA* is performed on the real multicore processor rather than in a simulated environment.

III. Experimental Setup

We target a 4-core SparcV8 NGMP multicore developed by Cobham Gaisler, implemented in an Altera Terasic DE4 board. It comprises per-core instruction and data caches and a shared AMBA AHB to the shared L2 cache. We use benchmarks from the Mediabench suite [1]. It comprises multimedia applications that are increasingly used in autonomous driving systems for functions like image processing.

We create 4-task workloads with three contenders and one *AUA*. As contenders we use three copies of an arbitrary mediabench benchmark and as *AUA* we also use an arbitrary benchmark. Using this methodology we generate 400 workloads (20



Accuracy Results Summary					
Range	<0.8	[0.8 - 1.0)	[1.0 - 1.5)	[1.5 - 2.0)	>2.0
Cases	1(0.25%)	19 (4.75%)	304 (76%)	50 (12.5%)	26 (6.5%)

Fig. 2. SurApp approach accuracy

different benchmarks as the *AUA*, times 20 different triplets of benchmarks as the contenders), which we execute in the target NGMP platform. We collect the execution times of the *AUA* when it runs with the real Mediabench and when it runs against the SurApp we generate for each Mediabench. The more similar the *AUA* slowdowns are, the better the SurApp copies of the TargetApp (i.e. the Mediabench). While no particular figure has been reported for the required accuracy in timing predictions during early design stages, in the context of multicores, it has been reported that the impact of contention in execution time can be as high as 20x for some kernels and as high as 5.5x for some benchmarks [3]. These are the reference values in this work to assess accuracy.

We map each task to a different core using the `taskset` Linux command, and measure the execution time of the *AUA* with the `time` utility. We use the sparc-linux-gcc-4.4.2 cross-compiler to generate the Mediabench binaries. To ensure concurrent execution, contenders always start executing before the *AUA* and they are restarted if they finish execution before the *AUA* so that the *AUA* always shares hardware resources within the multicore executions.

TABLE I
MEDIABENCH NGMP CHARACTERIZATION

Bench.	Inst. millio	APKI	Stack Distance 0	1	2	3	≥4
mesa.o	15.7	151	0.963	0.018	0.007	0.002	0.010
adpcm.e	19.1	144	0.999	0.000	0.000	0.000	0.000
pgp.e	220.0	128	0.997	0.002	0.001	0.000	0.000
adpcm.d	15.3	126	0.999	0.000	0.000	0.000	0.000
mesa.m	81.9	118	0.990	0.006	0.001	0.000	0.003
mpeg2.e	3315.6	115	0.999	0.000	0.000	0.000	0.001
jpeg.e	17.3	101	0.972	0.018	0.005	0.003	0.002
mesa.t	167.4	99	0.967	0.023	0.007	0.001	0.002
mpeg2.d	357.9	96	0.998	0.001	0.000	0.000	0.001
rasta	52.4	87	0.907	0.035	0.028	0.019	0.011
g721.e	807.7	85	0.999	0.000	0.000	0.000	0.000
g721.d	761.8	83	0.999	0.000	0.000	0.000	0.000
pegwit.e	81.0	73	0.866	0.055	0.044	0.032	0.003
pegwit.d	44.2	71	0.867	0.056	0.043	0.030	0.003
jpeg.d	6.7	64	0.994	0.005	0.001	0.000	0.000
epic.d	15.5	61	0.927	0.014	0.006	0.005	0.049
pgp.d	0.5	58	0.978	0.016	0.004	0.002	0.000
epic.e	87.9	50	0.972	0.004	0.002	0.001	0.022
gsm.e	343.1	29	0.970	0.026	0.004	0.000	0.000
gsm.d	146.3	26	0.988	0.011	0.001	0.000	0.000

IV. Surrogate Application Generation

We start by characterizing Mediabench benchmarks including instructions count, number of bus accesses per instruction, and the stack distances of instruction and data addresses. We extract the instructions and data address trace using *qemu*. We feed the traces to a simulator based on SoCLib [4] validated for this hardware architecture [2]. Table I shows the characterization of the benchmarks in the suite, ordered by APKI (bus Accesses per thousand (Kilo) Instructions). We observe a wide range of APKI, with accesses tending to hit in the most-recently used data (stack distance 0). From this information, we generate the SurApp of each benchmark.

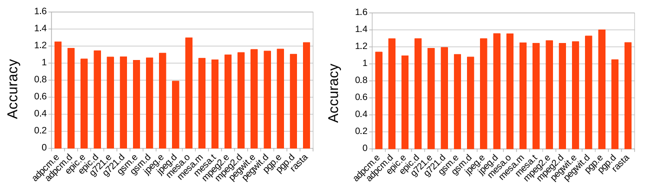


Fig. 3. Accuracy results for `mesa.o` and `adpcm.e`

V. Accuracy Results

For each of the 400 workloads we generated, Figure 2 shows the execution times of the *AUA* when executed in the real NGMP board concurrently with the SurApps (x axis) normalized to its execution time when executed against the TargetApps, i.e. the real Mediabench benchmarks. Accuracy values above 1 show that the *AUA* takes longer to execute against the SurApps than against the real applications copied by the SurApps. As it can be seen, the deviation w.r.t. the execution time against Mediabench is usually small (normalized results are usually between 1 and 1.3), with few outliers of up to 3.5x. More in detail, in the Accuracy Result Table we see that in 76% of the cases the accuracy is between 1 and 1.5x, and in 12.5% of the cases between 1.5 and 2. In 6.5% of the cases the deviation is larger than 2x. Interestingly, only 1 case (0.25% of the total) is below 0.8 – i.e. the *AUA* suffers quite lower slowdown with SurApps than with real contenders. This results show that the approach is solid against timing violations.

In terms of individual benchmarks, we focus on those with more bus accesses since they are most likely to cause interference to the *AUA*: *mesa.o* and *adpcm.e* as shown in Table I. Figure 3 (left) shows the results for *mesa.o*. We observe that the execution time of the AUA when run against the *mesa.o* SurApp is higher (more conservative) than when run against the actual *mesa.o* benchmark, except for the case of *jpeg.d* as the AUA. Likewise, Figure 3 (right) shows the results for *adpcm.e*. We observe the same trend: the *adpcm.e* SurApp provides conservative over-estimation of the contention. In both charts we observe that over-estimation comfortably fits the range $(1 - 1.4)$.

VI. Conclusions

The proposed technique helps deriving timing estimates on real multicore boards for early design software stages. Accuracy results obtained when executing the *AUA* co-running with surrogate applications show that SurApps moderately upper-bound the execution time of the *AUA* co-running with the actual applications. This allows deriving timing estimates for multicores without the need to share real applications developed by different software providers. This is pivotal in multi-provider software projects for integrated architectures to deal with IP constraints.

Acknowledgments

This work has been partially supported by the Spanish Ministry of Economy and Competitiveness (MINECO) under grant TIN2015-65316-P, and the HiPEAC Network of Excellence. Jaume Abella has been partially supported by the MINECO under Ramon y Cajal postdoc fellowship RYC-2013-14717.

References

[1] Chunho Lee et al. Mediabench: A tool for evaluating and synthesizing multimedia and communicatons systems. MICRO, 1997.

[2] Javier Jalle et al. Validating a timing simulator for the NGMP multicore processor. In *21st DASiA*, 2016.

[3] G. Fernandez et al. Surrogate applications for early design stage multicore contention modeling. In *TETC*, 2018.

[4] SoCLib. -, 2003-2012. http://www.soclib.fr/trac/dev.