Proceedings of the Work-in-Progress Session

of the 20th IEEE Real-Time and Embedded Technology and Applications Symposium

Berlin, Germany
April 15, 2014

Edited by Björn B. Brandenburg
Max Planck Institute for Software Systems

Message from the Work-in-Progress Chair

The Work-in-Progress (WiP) session at RTAS 2014 is dedicated to new and on-going research in the field of real-time and embedded systems. The WiP session is an integral and important part of the RTAS program, as it exposes promising research directions to a wider audience and provides researchers with an opportunity to discuss evolving and early-stage ideas, and to solicit feedback from the real-time systems community at large.

First of all, I would like to thank the authors for submitting their work, and the members of the program committee for their quick and helpful reviews. This year, the WiP track received a total of 17 high-quality submissions, each of which was reviewed by three members of the program committee. After a final online discussion round, twelve excellent papers were selected for presentation at the WiP session.

The resulting technical program covers a broad range of topics—spanning from, on the one hand, timing analysis, schedulability analysis, and mapping heuristics for many-core platforms to, on the other hand, air data estimation in UAVs, a novel kernel design for mixed-criticality systems, and support for parallel execution in a time-triggered operating system—and thus reflects the breadth of the real-time systems community. I'm convinced that the diverse mix of foundational as well as applied topics will provide for stimulating discussions and encourage lively interactions at the poster session.

It is my hope that the WiP track and the poster session will be interesting to and enjoyable for presenters and the audience alike, and I invite you to join me in taking advantage of this excellent opportunity to learn, to discuss, and to network.

Björn B. Brandenburg
Max Planck Institute for Software Systems
Kaiserslautern, Germany
RTAS 2014 WiP Chair

RTAS 2014 Work-in-Progress Technical Program Committee

Andrea Bastoni, SYSGO AG, Germany

Marko Bertogna, University of Modena, Italy

Bernard Blackham, NVIDIA Corp., UK

Aaron Block, Austin College, USA

Tommaso Cucinotta, Bell Laboratories Alcatel-Lucent, Ireland

Wanja Hofer, Brose Fahrzeugteile GmbH & Co. KG, Germany

Cong Liu, The University of Texas at Dallas, USA

Alex Mills, Indiana University at Bloomington, USA

Harini Ramaprasad, Southern Illinois University Carbondale, USA

Michael Roitzsch, Technische Universität Dresden, Germany

Technical Program

A hybrid system approach to air data estimation in unmanned aerial vehicles

Mohammad Shaqura, Mechanical Engineering and Christian Claudel, Electrical Engineering,
King Abdullah University of Science and Technology

***Abstract*—Fixed wing Unmanned Aerial Vehicles (UAVs) are an increasingly common sensing platform, owing to their key advantages: speed, endurance and ability to explore remote areas. While these platforms are highly efficient, they cannot easily be equipped with air data sensors commonly found on their manned counterparts since these sensors are bulky, expensive and reduce the payload capability of the UAV. In consequence, UAV controllers have little information on the actual mode of operation of the wing (normal, stalled, spin) which can cause catastrophic failures when flying in turbulent weather conditions. In this article, we propose a real-time air parameter estimation scheme that can run on commercial, low power autopilots in real-time. The computational method is based on an hybrid decomposition of the modes of operation of the UAV. An implementation on a real UAV is presented, and the efficiency of this method is validated using a hardware in the loop (HIL) simulation.**

I. Introduction

Unlike their ground or water-based counterparts, Unmanned Aerial Vehicles (UAVs) have the potential to be deployed extremely rapidly for surveillance and monitoring applications. Among all types of UAVs, fixed-wing UAVs are the most fuel efficient and the fastest for a given weight and propulsive power. However, manual or automatic flight of UAVs can be complex, as existing air data probes are too bulky, too expensive and too heavy [2] and are thus restricted on heavier, larger and more expensive UAVs. The lack of air data severely restricts pilot or autopilot actuation, and does not allow the operators of the UAV to explore its full flight domain due to the risk of stalls. Numerous articles have been written on the issue of air data estimation, in particular the work of [4], [3] which is based on classical Extended Kalman Filters and LQR methods. While these methods are very accurate, they typically draw significant computational resources to compute the required matrices. In this article, we propose an hybrid system formulation of the UAV dynamics for this purpose. For each mode, we show that the airspeed, angle of attack and angle of sideslip are given by analytical formulas.

II. Hybrid system modeling

We use the 6-degrees of freedom (6-DOF) equations of motion to model the dynamics of the UAV as a rigid body. For compactness, we choose not to write these equations in the present article, though these equations relate inertial measurements (acceleration and angular acceleration) to control inputs and the airflow parameters to be estimated. These relations are non linear in terms of the angle of attack α, angle of sideslip β and airspeed V_a.

M. Shaqura is a PhD student, Department of Mechanical Engineering, KAUST, 23955-6900 Saudi Arabia. Email: mohammad.shaqura@kaust.edu.sa. Corresponding author

C. Claudel is an Assistant Professor, Department of Electrical Engineering, KAUST, 23955-6900, Saudi Arabia

III. Estimation method

To estimate α, β and V_a in real time, we express the nonlinear model as an hybrid model where the dynamics are linear in terms of angle of attack and angle of sideslip in each mode. An analytical expression of the airspeed is computed offline in terms of measurements and inputs. For mode i, the linear 6-DOF system of equations can be written as $A_i x_i = b$ where A is a 6×2 matrix that contains the coefficients of α and β for mode i and b is a 6×1 vector contains the measurements, control inputs and linear coefficients of α and β, which are mode dependents.

$$b = [b1 \;\; b2 \;\; b3 \;\; b4 \;\; b5 \;\; b6]^T \tag{1}$$

$$A_i = \begin{bmatrix} C_{D_{\alpha i}} & 0 & C_{L_{\alpha i}} & 0 & C_{m\alpha} & 0 \\ C_{D_{\beta i}} & C_{Y_\beta} & C_{L_{\beta i}} & C_{l_\beta} & 0 & C_{n_\beta} \end{bmatrix}^T \tag{2}$$

The least square solution is computed analytically:

$$\begin{bmatrix} \alpha(V_a) \\ \beta(V_a) \end{bmatrix} = (A^T A)^{-1} A^T b \tag{3}$$

We then substitute back α and β into the nonlinear dynamical model. The explicit formula of V_a is computed by minimizing the difference between the estimated aerodynamical forces and torques and the actual ones in the norm 2 sense:

$$N1 = F_{AX_{est}} - F_X(V_a) - F_{GX} - F_{throttle} \tag{4}$$
$$N2 = F_{AY_{est}} - F_Y(V_a) - F_{GY} \tag{5}$$
$$N3 = F_{AZ_{est}} - F_{GZ} \tag{6}$$
$$N4 = T_{AX_{est}} - T_X(V_a) \tag{7}$$
$$N5 = T_{AY_{est}} - T_Y(V_a) - T_{Y_{throttle}} \tag{8}$$
$$N6 = T_{AZ_{est}} - T_Z(V_a) \tag{9}$$
$$N(V_a) = (N1^2 + N2^2 + N3^2 + N4^2 + N5^2 + N6^2)(V_a) \tag{10}$$
$$V_{analytic} = \arg\min_{V_a} N(V_a) \tag{11}$$

In the present case, the minimization problem yields an analytic solution, which is a function of the inertial measurements, inputs (from the aileron, elevator and throttle) as well as aerodynamic model parameters.

$$V_{analytic} = f(ax, ay, az, p, q, r, \dot{p}, \dot{q}, \dot{r}, \Delta A, \Delta E, throttle, C_{D_{\alpha i}}, C_{D_{\beta i}}, C_{L_{\alpha i}}, C_{L_{\beta i}}) \quad (12)$$

We thus compute the estimated airspeed, angle of attack and sideslip angle for all possible modes (this boils down to applying a 2×6 matrix on a vector, and to computing a function of the parameters. Obviously, since each mode i yields a different value for the estimated airspeed, angle of attack and angle of sideslip, we identify the mode in which the UAV flies by computing the residuals in the norm 2 sense between the original nonlinear model (in which we apply the estimated values of airspeed, angle of attack and sideslip) and the measurements, then minimizing these residuals.

IV. Implementation

A. System

The system consists of an RC C-17 Globemaster aircraft [1] equipped with an `Ardupilot Mega (APM)` v2.6 microcontroller and a number of sensors including barometer, magnetometer, GPS, inertial measurement unit (IMU) and an ultrasound ground proximity sensor. The firmware used in `APM` is `Arduplane`, which is an open source code for flight management. The onboard processing is done by the `APM` processor itself, an `Atmel ATMEGA 2560` 8-bit chip with 256KB flash memory and a maximum operating frequency of 16 MHz. Given the relatively low computational power of this platform (which also handles a large number of processes related to guidance and attitude estimation), we want to validate that the estimation of V_a, α and β can be performed in real-time.

B. Simulated computational performance

To validate this approach, we use *Hardware-In-The-Loop* (HIL) to test the code execution when the `APM` microcontroller is interfaced with a commercial flight simulator. We choose the flight simulator `X-Plane`, and create a UAV matching the specifications of our current UAV, using both computer assisted design and computational fluid mechanics software. We break down the dynamics of the UAV into 15 different modes. The estimation function is implemented as part of the `Arduplane` code in the 50 Hz loop. We choose this very fast update rate (compared to the typical dynamics of the UAV) since more advanced mode selection schemes will be used in the future. The interface between the `APM` and `X-Plane` is done using a customized version of the `APM Mission Planner` (open source software). The complete setup is illustrated in Figure 1.

To validate this approach, we used manual flight mode to voluntarily place the UAV in an aerodynamic stall. Aerodynamic stalls occur whenever the angle of attack α exceeds a threshold (for our UAV: 12°) that is a function of the wing shape and size. As can be seen from figure 2, the estimation process run on the APM detects this event easily. The estimate of the angle of attach for this case was 16°, which can be confirmed independently by computing the difference between the pitch angle (about 15°) and the angle of climb (about 0°).

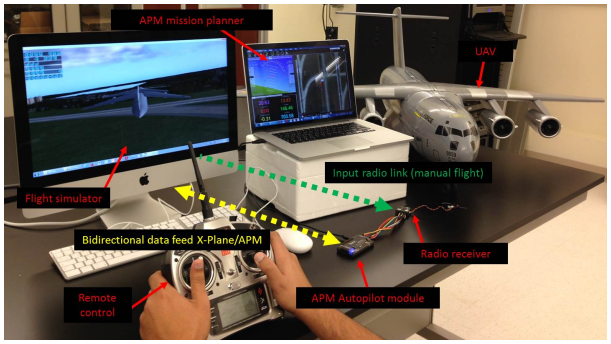


Fig. 1. Hardware in the loop simulation setup

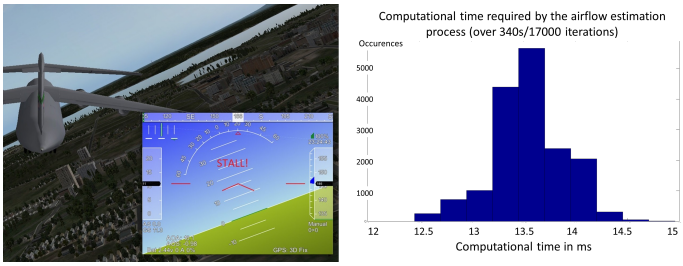


Fig. 2. Left: aerodynamic stall detection during a manual HIL flight test. Right: Distribution of the measured computational times of the air data estimation process (running at 50Hz)

V. Conclusion and Future Work

In this article, we presented a computational method for air data estimation in Unmanned Aerial Vehicles (UAVs), that can run on the low-power hardware that is typically used on low-cost Micro Air Vehicles (MAVs). This method is based on a hybrid representation of the dynamical model of the UAV, and consists in computing a set of explicit functions of the inertial measurements and input variables, which is very fast. Estimating these airflow parameters in real time is very important in practice to detect stalls (which is a leading cause of crashes, and which is not considered in most UAV autopilots (including the `APM`) because of the lack of angle of attack sensors), and to detect airspeed sensor faults. Future work will deal with enhanced mode-selection methods based for instance on compressed sensing or on machine learning, taking into account the dynamics of the evolution of the estimated values and residuals.

References

[1] M. Abdelkader, M. Shaqura, C.G. Claudel, and W. Gueaieb. A UAV based system for real time flash flood monitoring in desert environments using Lagrangian microsensors. In *ICUAS, 2013*, pages 25–34, 2013.

[2] M. Ghommem V. Calo C. Claudel. Micro cantilever flow sensor for small aircrafts. *SAGE Journal on Vibration and Control*, 2013. doi: 10.1177/1077546313505636.

[3] M.Z. Fekri and M. Mobed. Angle of attack estimation and aircraft controller design using lqr methods. In *Electrical and Computer Engineering, 2005. Canadian Conference on*, pages 2253–2256, May 2005.

[4] F Adhika Pradipta Lie and Demoz Gebre-Egziabher. Synthetic air data system. *Journal of Aircraft*, 50(4):1234–1249, 2013.

A Refined Approach for Stochastic Timing Analysis

Nathan C. Cox, Darius G. Luchian, Thomas E. Herschede, Christopher A. Healy
Department of Computer Science
Furman University
Greenville, South Carolina, USA

***Abstract*—In real-time scheduling, the worst-case execution time (WCET) of a task needs to be known in advance. However, the actual WCET of a task is unlikely to occur in practice. Therefore, scheduling for a soft real-time system can make use of execution times that have a finite probability of being exceeded. As a result, it is desirable to statically compute a program's execution time distribution, rather than just a single number representing an extreme value. In this short paper, we describe ongoing work to produce accurate execution time distributions.**

Keywords—WCET analysis, probabilistic timing analysis, scheduling, control-flow analysis

I. INTRODUCTION

In order to support real-time scheduling, it is necessary to statically predict the worst-case execution time of each scheduled task. Considerable research has been conducted in static WCET analysis [11]. The purpose of such analysis is to guarantee that all deadlines will be met. By contrast, a soft-real time system is one in which an occasional deadline miss degrades performance but is tolerated [7]. The motivation for our present work is that the predicted WCET may be unrealistically high. For soft-real time systems, we may instead want to pursue, say, the 99th percentile of the execution time probability distribution. Our aim is to generate the distribution of execution times statically.

Our approach to stochastic execution time looks at the whole distribution, rather than the execution time with a small probability, e.g. 10^{-12}, of being exceeded [5] [2]. Keim et al. introduced an algorithm to statically produce the stochastic timing analysis of a single loop [6]. This approach reduced the problem to using repeated applications of a binomial probability distribution throughout. The approach combined paths in pairs, and results for code having more than two paths were not very accurate. Our motivation was to improve upon this approach and to be able to tackle a wider class of benchmarks.

II. APPROACH

Our approach follows two directions. One is to gather a time distribution estimate from the timing analyzer, and the other is to glean a corresponding time distribution based on observed execution times. The following is a high-level outline of our approach.

```
Compute estimated time distribution:
-  A compiler produces ARM assembly code.
-  RALPHO produces control-flow information.
-  Timing analyzer generates distribution.
Compute observed time distribution:
-  A compiler produces ARM executable code.
-  Run the benchmark repeatedly to gather
     observed time distribution.
Compare the two distributions, using
  statistical and integral tests.
```

This methodology relies on three pieces of software. First is an off-the-shelf compiler such as gcc, which can produce assembly and executable code. The other two software tools are our own analysis suite. We modified two existing analysis tools. The assembly code is fed into a revised version of RALPHO [3], in order to produce a control-flow file. This control-flow information is input into a modified timing analyzer [8] to give us the execution time distributions. The modifications to RALPHO are beyond the scope of this paper.

A. Static calculation

The timing analyzer computes the probability that each path is selected during a loop iteration. This is based on the branch instructions that are encountered in each path. Next, we enumerate the cases of selecting each path for all the iterations of the loop. Note that we are only interested in the number of times that each path is taken, not the sequence in which they occur. This assumption simplifies the analysis.

The approach suggested by [6] works well for two paths, but in our experimentation we found it to be inaccurate for more than two paths. Therefore, we decided to modify the algorithm to use multinomial probabilities [9], instead of repeated applications of the binomial probability. To avoid a combinatorial explosion, we restrict our algorithm to handle up to eight paths. If a loop has more than eight paths, then only the eight most likely taken paths are considered.

Our algorithm computes an execution time distribution for every loop and function in the program. Then, it computes the distribution for the entire program, using the convolution technique of [1]. This step is taken when we need to combine two or more loops, or when a loop or function is nested inside another loop or function. Essentially, the technique is to take the Cartesian product of two distributions, and then re-sort the result so that it has N buckets instead of N^2.

This work was supported in part by grants from the Furman Advantage and Francis M. Hipp Research Fellowship and the Howard Hughes Medical Institute.

B. Comparing with Dynamic Calculation

We consider three possible metrics to evaluate the accuracy of the timing analyzer's stochastic distribution versus the observed one. First, we compare the means and standard deviations of the two distributions. Second, we compare the top quantile of each distribution. This is analogous to the usual way of comparing statically predicted versus observed WCETs. This metric also highlights how much lower a likely WCET is compared to its guaranteed WCET.

A third metric is an integral test. It seeks to capture the differences between the probability curves throughout their domain $0 \leq x \leq 1$. The estimated and observed functions (est(x) and obs(x), respectively) are normalized so that the integrals of each function are equal to 1. This is done so that we can allow for a varying number of trials performed when computing obs(x). Then the error can be defined as:

$$e = \frac{1}{2}\int_0^1 |obs(x) - est(x)|\,dx \qquad (1)$$

III. EXPERIMENTAL EVALUATION

We tested our approach on six benchmarks, each having up to five loops. The loops had between two and seven execution paths, inclusive. The code was taken from established set of benchmarks used in WCET analysis [4] [11].

TABLE I. RESULTS OF EXPERIMENTS

Test case	Observed mean	Estimated mean	Observed 99 %ile	Estimated 99 %ile
A*	0.77 W	0.74 W	0.86 W	0.97 W
A	0.77 W	0.76 W	0.86 W	0.87 W
B	0.37 W	0.36 W	0.49 W	0.49 W
C	0.71 W	0.86 W	0.81 W	0.88 W
D	0.90 W	0.88 W	0.94 W	0.92 W
E	0.90 W	0.89 W	0.94 W	0.90 W
F	0.93 W	0.90 W	0.94 W	0.92 W
Avg.	0.76 W	0.78 W	0.93 W	0.83 W

These results show that the predicted execution time distribution was often very close to the observed distribution. In each case, W represents the predicted WCET of the benchmark. To normalize the results, the second through fifth columns express the execution time as a factor of W. Benchmark A* is the same as A, and is presented here to compare previous work with ours. The A* row refers to the result reported by [6]. This benchmark features a loop with seven paths and is now much more accurately predicted, as a result of using multinomial rather than binomial probabilities.

Considering the absolute value of the error of the means, the predicted mean was within 4 percentage points of the observed mean of the execution times. The absolute value of the error in the 99th percentile statistics was less than 3 percentage points. For sake of brevity, results of the standard deviation comparison and integral test are not shown.

IV. ONGOING WORK

Our static analysis tool can generate a statistical distribution of execution times of whole benchmarks. The analysis can handle multiple paths, nested or consecutive loops, as well as the nesting of function calls and/or loops.

We are currently working on improving the accuracy of our stochastic predictions, and expanding the class of programs that we can analyze. For example, we are addressing the problem of individual branch probabilities, to determine which branches govern loop control. We are also modifying RALPHO so that it can accurately compute loop iterations for more complex loop nests.

We also plan to investigate probabilistic runs, i.e. the problem of being temporarily lucky or unlucky in the execution time [9]. This concept has recently been studied elsewhere in the context of predicting performance in the presence of random errors and bursts of random errors in a system [10].

ACKNOWLEDGMENT

The authors would like to thank Zach Hall and Joey Iannetta for their assistance with RALPHO. Kory Kraft helped to implement the core of the timing analysis algorithm.

REFERENCES

[1] G. Bernat, A. Colin, S. Petters, "WCET analysis of probabilistic hard real-time systems," IEEE Real-Time Systems Symposium, December 2002, pp. 279-288.

[2] R. Davis, L. Santinelli, S. Altmeyer, C. Maiza, L. Cucu-Grosjean, "Analysis of probabilistic cache related pre-emption delays," Euromicro Conferences on Real-Time Systems, July 2013, pp. 168-179.

[3] J. Estep, Design and Implementation of the Retargetable Assembly-Level Program Hierarchical Organizer, senior thesis, Furman University, 2003.

[4] J. Gustafsson, B. Lisper, A. Betts, A. Ermedahl, "The Malardalen WCET benchmarks: past, present and future," International workshop on worst-case execution time analysis, July 2010, pp. 137-147.

[5] J. Hansen, S. Hissam, G. Moreno, "Statistical-based WCET estimation and validation," International workshop on worst-case execution time analysis, June 2009, pp. 129-133.

[6] P. Keim, A. Noyes, A. Ferguson, J. Neal, C. Healy, "Extending the path analysis technique to obtain a soft WCET," International workshop on worst-case execution time analysis, June 2009, pp. 134-142.

[7] P. Laplante, Real-Time Systems Design and Analysis, Wiley-IEEE Press, 2004.

[8] S. Mohan, F. Mueller, W. Hawkins, M. Root, C. Healy, D. Whalley, "Parametric timing analysis and its applications to DVS," IEEE Trans. Embed. Compt. Syst., December 2010.

[9] S. Ross, A first course in probability, New York: Macmillan, 3rd edition, 1988.

[10] M. Short, J. Proenza, "Towards efficient probabilistic scheduling guarantees for real-time systems subject to random errors and random bursts of errors," Euromicro Conference on Real-Time Systems, July 2013, pp. 259-268.

[11] R. Wilhelm et al., "The worst-case execution time problem – overview of methods and survey of tools," ACM Trans. Embed. Comput. Syst. 7(3):1-53, 2008.

Thread Migration for Mixed-Criticality Systems

Alexander Zuepke
RheinMain University of Applied Sciences, Wiesbaden, Germany
Email: alexander.zuepke@hs-rm.de

***Abstract*—This work-in-progress paper presents a thread migrating operating system concept for mixed-criticality systems on multi-core platforms. Thread migration provides fast context switching between isolated software components which handle shared resources. Combined with criticality inheritance protocols and a multi-policy scheduler, the described operating system concept aims to meet the level of determinism and analysability which is required for safety-critical applications.**

I. Introduction

With *Cyber Phyiscal Systems* and the *Internet of Things*, mixed-criticality systems have become a reality in the embedded computing world. Combined with the recent availability of multi-processor systems, it imposes a new challenge on operating systems when different functional units are combined in a single computer system. Similarily, regulatory standards like ISO 26262 require *freedom of interference* between these independent functional units [1]. On the other hand, tight integration of today's hardware technology results in problematic sharing of computational resources like caches and memory bandwidth, and functional resources like I/O devices and buses. An operating system for such scenarios should therefore help to make the side effects of resource sharing *predictable* and enforce the required level of *determinism*.

From a real-time perspective, this means that applications of different *criticality levels* (in the sense of importance to a device's overall function and cost of malfunction) need to be scheduled concurrently. It also requires that access to shared resources and any resulting priority inversion problems need to be solved in a bounded worst-case execution time (WCET) to guarantee that deadlines are met. From a safety perspective, a high degree of separation between different application components and shared components is necessary to guarantee the required freedom of interference and fault isolation. State of the art techniques place applications, drivers, and services into separate address spaces, protected by means of the processor's memory management unit (MMU) [2] [3].

However, while decomposition of a system's components into multiple address spaces helps to fulfill the safety requirements, it entails overhead due to the cost of additional context switches. Therefore, operating system support for mixed criticality systems should include:

- isolation of components in separate address spaces,
- fast context switches between isolated components,
- bounded WCET of all internal operations of the kernel,
- solving of priority inversion problems on shared resources,
- concurrent scheduling of threads of different criticality.

This paper briefly presents the design principles of the WINGERT operating system, which addresses the goals discussed above. Its overall architecure is shown in section II, the benefits of thread migration is described in III, scheduling in IV, resource sharing in V, and related work in VI. We conclude and give an overview of future work in section VII.

II. System Architecture

The WINGERT OS is built upon a small kernel running in the CPU's privileged mode and a hierarchically structured set of isolated address spaces (tasks) of different criticality in user mode, which comprise applications or services like shared drivers. Following the design principle of a small trusted computing base, critical application tasks only depend on the required subset of tasks providing shared services for them. Communication between tasks is implemented by remote procedure calls (RPCs), which are described in detail in the following section. Starting from the application tasks as the leaves of the task tree, the hierarchy of depending tasks down to the kernel as the root node never decreases in the criticality level. This implies that applications can trust the tasks down in the chain.

Further, all system resources like memory, I/O and time budgets are statically assigned to the tasks at startup. This *resource partitioning* approach eliminates the later need to transfer system resources or access permissions via task communication at runtime, keeping the RPC implementation in the kernel fast and simple.

Each task manages its capabilities in its own name space. Capabilities address the user and kernel parts of threads, interrupts, child tasks, child address spaces, and communication channel endpoints. Memory is addressed differently by its implicit virtual address in the task's page tables. Additionally, tasks can freely repurpose their assigned amount of page-sized *kernel memory* to page tables for dynamically created memory mappings or in-kernel stacks for threads. This degree of freedom allows for example a para-virtualized Linux task to reconfigure itself for different use cases at runtime, without violating the static partitioning approach.

III. Thread Migration

The main difference of WINGERT compared to other micro kernels like L4 lies in its low-level abstraction model named *body and soul* instead of threads as the basic entities of execution. The *soul* is a scheduling entity with priority, deadline, and a kernel stack. It migrates synchronously between different *bodies*, which comprise of an entry point and dedicated stack in user space. The invocation of a new body resembles an RPC call to the same or a remote address space, while keeping the calling soul to have a unique entity to control the execution flow and to reduce context switching overhead in the kernel.

For asynchronous communication and decoupling of potentially blocking calls, the kernel provides *fork-join* operations: a soul forks and instructs its forked sibling to issue a synchronous RPC. With a technique named *lazy forking*, the kernel follows the forked path using the original soul first and performs the real fork operation when a blocking point is encountered. Assuming there is no blocking point, the forked one returns the result and joins gracefully without any overhead.

IV. Scheduling

The practical challenge of mixed-criticality scheduling lies in reclaiming scheduling reservations of higher critical tasks at run time caused by their overly pessimistic WCET analysis. Inspired by MC^2 [4], the kernel scheduler provides multiple scheduling policies for different levels of criticality. In descending order of criticality, these are:

1) P-FP: partitioned fixed-priority scheduling
2) P-EDF: partitioned earliest deadline first scheduling
3) G-FP: global fixed-priority scheduling
4) G-EDF: global earliest deadline first scheduling
5) BE: best effort scheduling for non-real-time applications
6) IDLE: scheduling of idle threads of the lowest level

All scheduling policies are mapped into the same priority space, but have disjoined priority ranges and different queueing policies (FIFO or deadline ordered). The highest priority level ready queues are kept exclusive per processor to ensure partitioned scheduling, the lower levels share a single set of ready queues. The dispatcher picks the highest eligible thread for scheduling on its CPU. Supporting other scheduling policies, like the ones used by Linux, is not the responsibility of the OS scheduler. On top of this system, a para-virtualized Linux implementation would use its built-in scheduler and dispatch its processes by thread migration.

V. Resource Sharing

WINGERT provides two different mechanisms for synchronization and resource sharing: thread migration across tasks; and mutexes and condition variables shared by threads in the same task. The latter use *Deterministic Futexes* described in [5] as the underlying kernel mechanism. The implementation enters the kernel only on contention and uses atomic operations on variables in user space in the fast path.

As bodies have a single user space stack only and therefore do not support multiple souls inside, migrating souls have to wait when a body is already occupied. With an extension to let souls *wait* outside the body and let them stay there until they are *signalled* again by the body, the body effectively becomes a *Monitor* [6].

On contention, both bodies and futexes need to properly solve priority inversions problems. The standard *priority inheritance protocol* (PIP) [7] solves this issue for the class of highest criticality P-FP scheduling. Additionally, the protocol covers P-EDF by prefering earlier deadlines on priority ties. Finally, with *migratory priority inheritance* [8], the scheduler migrates preempted threads across CPUs and solves priority inversions in global scheduling scenarios. With these extensions for mixed-criticality scheduling, the described protocol effectively becomes a *criticality inheritance protocol*.

VI. Related Work

WINGERT has in common with micro kernels like L4 [9] a similar overall system structure of decomposed software components in isolated address spaces and the use of a synchronous context switch mechanism as a means of communication [2]. However, our approach is more specific to the mixed-criticality use case than the policy-free approach in L4.

Thread migration was previously used in [10], [11], and [12]. Compared to COMPOSITE, which uses thread migration and solves contention on user stacks with PIP and PCP (priority ceiling protocol) [3], the presented approach scales to multi-processor platforms.

VII. Conclusion and Future Work

This paper presented the WINGERT operating system, which aims to exploit thread migration for real-time systems. Using thread migration for strictly hierarchical system designs such as mixed-criticality systems seems to be a good trade-off between software component isolation for safety reasons on the one hand and fast performance on the other hand, while at the same time reducing the number of possibly misbehaving actors and keeping the overall system complexity low.

In future work, we plan to evaluate the system performance and provide an in-depth analysis of the presented criticality inheritance protocol, with a special focus on an implementation with a bounded WCET.

References

[1] ISO 26262, "Road vehicles – Functional safety," 2011.

[2] J. Liedtke, "On μ-Kernel Construction," in *SOSP*, 1995, pp. 237–250.

[3] Q. Wang, J. Song, and G. Parmer, "Execution Stack Management for Hard Real-Time Computation in a Component-Based OS," in *RTSS*, 2011, pp. 78–89.

[4] J. L. Herman, C. J. Kenna, M. S. Mollison, J. H. Anderson, and D. M. Johnson, "RTOS Support for Multicore Mixed-Criticality Systems," in *RTAS*, 2012, pp. 197–208.

[5] A. Zuepke, "Deterministic Fast User Space Synchronisation," in *OSPERT Workshop*, 2013.

[6] C. A. R. Hoare, "Monitors: An Operating System Structuring Concept," *Commun. ACM*, vol. 17, no. 10, pp. 549–557, Oct. 1974.

[7] L. Sha, R. Rajkumar, and J. P. Lehoczky, "Priority Inheritance Protocols: An Approach to Real-Time Synchronization," *IEEE Trans. Computers*, vol. 39, no. 9, pp. 1175–1185, 1990.

[8] B. B. Brandenburg and A. Bastoni, "The case for migratory priority inheritance in linux: Bounded priority inversions on multiprocessors," in *Fourteenth Real-Time Linux Workshop*, 2012.

[9] K. Elphinstone and G. Heiser, "From L3 to seL4 What Have We Learnt in 20 Years of L4 Microkernels?" in *SOSP*, 2013, pp. 133–150.

[10] B. Ford and J. Lepreau, "Evolving Mach 3.0 to A Migrating Thread Model," in *USENIX Winter Conference*, 1994, pp. 97–114.

[11] G. A. Parmer, "Composite: A Component-based Operating System for Predictable and Dependable Computing," Ph.D. dissertation, Boston, MA, USA, 2010.

[12] E. Gabber, C. Small, J. Bruno, J. Brustoloni, and A. Silberschatz, "The Pebble Component-based Operating System," in *USENIX ATC*, 1999.

What if we would degrade LO tasks in mixed-criticality systems?

Marcus Völp
School of Computer Science, Logical Systems Lab
Carnegie Mellon University
Pittsburgh, PA, USA
mvoelp@cs.cmu.edu

I. Introduction

Mixed-criticality (MC) systems [1] allow tasks of different importance (or criticality) to be consolidated into a single system. Consolidation facilitates resource sharing (even across criticality levels) and hence bears the potential to reduce the overall amount of resources needed. However, there is a common misconception that recurs in literature about Vestal's model: *the false believe that low criticality tasks are degraded to soft real-time or even best effort tasks*. In this work, we not only wish to clarify this misconception but also ask ourselves what would happen if we degrade LO tasks. Revisiting Quality Assuring Scheduling (QAS) [2], [3], our goals are stochastic guarantees for LO completion in addition to and replacing the hard MC guarantees if LO tasks are soft real-time. In this WIP report, we focus on properties of dropped LO tasks (while keeping hard MC guarantees) such as: *"What is the likelihood of lower criticality jobs being dropped because higher criticality jobs exceed their low WCET estimates?"*, *"What is the likelihood of dropped jobs to still make their deadline?"*, and *"What is the expected time / Q-percentile for dropped jobs to catch up with their execution?"*. Part of our future work will be to extend these guarantees and to develop MC schedulers for a combination of hard and soft real-time tasks. Notice though, that the assumptions in this report still limit the applicability of our results. We indicate how we plan to relax them in the future. Most notably, we assume that jobs arrive in their synchronous arrival sequence and that execution-time distributions are known. The latter we plan to replace with confidence of WCET estimates.

II. Mixed-Criticality Scheduling

Let $\mathcal{T}$ be a set of sporadic tasks. As usual, we characterize $\tau_i \in \mathcal{T}$ by tuples (l_i, D_i, T_i, C_i) where l_i is a criticality level in the ordered set of criticality levels $\mathcal{L}$ (e.g., $\mathcal{L} = \{LO, HI\}$ with $LO < HI$), $D_i \leq T_i$ is the relative deadline and T_i the minimal interrelease time. We subject tasks to execution time analyses suitable for the individual levels. The result is a vector of increasingly more pessimistic WCET estimates $C_i(l)$. The set $\mathcal{L}$ and the requirements for considering an analysis suitable may be drawn from evaluation criteria such as DO-178C [4] but other metrics are also conceivable. We assume the system enforces budgets $C_i(l_i)$ and subject τ_i only to the WCET analyses for all $l \leq l_i$. The feasibility criteria and hence the guarantee given to all admitted tasks is:

Definition 1 (Feasibility): The set $\mathcal{T}$ is feasible if every job $J_l = \tau_{i,j}$ receives $C_i(l_i)$ time in between its release time r_l and its absolute deadline $r_l + D_i$ provided no job J_h of a higher criticality task with $l_h > l_i$ exceeds its low WCET estimate $C_h(l_i)$.

There are two important points to notice:

1) If no higher-criticality job J_h exceeds its low estimate $C_h(l_l)$, lower criticality jobs J_l receive the same hard real-time guarantees as in a classical system. That is, provided all WCET estimates are safe, they receive sufficient time to complete before their deadlines; and
2) No guarantee is given to these jobs once a higher criticality job exceeds its low WCET estimate.

Notice also that dropped jobs merely loose their real-time guarantees (and possibly their high prioritized budget). There is no necessity to terminate these jobs. The question about low-criticality guarantees now boils down to whether WCET estimates are safe or whether in some rare situations the actual execution time may exceed these estimates. Either way, MC schedulers convey no guarantee about the subset of deadlines low criticality tasks meet, which is essential for soft real-time.

III. Quality Assuring Scheduling

QAS [2] offers stochastic guarantees for imprecise computations [5] where jobs are composed of mandatory parts, which must execute to completion, and optional parts, which improve the final result. For the first, a safe WCET estimate ($\mathbf{P}[X_i \leq C_i] = 1$) is anticipated whereas for the latter QAS assigns budgets b_i^k resulting from requested Q_i-percentiles of the execution time distribution ($\mathbf{P}[X_i \leq b_i^k] = Q_i$). Here and in the following X_i and Y_i^k denote non-negative random variables capturing actual execution times and $\mathbf{P}[X_i \leq c]$ stands for the probability that $X_i \leq c$. QAS aborts parts at their assigned budgets but considers the actual execution time distribution to determine the likelihood of lower prioritized jobs meeting their deadlines. The k^{th} optional part o_i^k is executed (possibly at a different priority) only if all prior optional parts completed in time. For the special case where tasks shares a common release time r and deadline D, the job J_i completes o_i^k with probability p if

$$\mathbf{P}[Y_i^k < b_i^k \wedge (\sum_{j \in \mathbf{hp}_i} X_j + \sum_l \min(Y_j^l, b_j^l) + X_i + \sum_{m \leq k} \min(Y_i^m, b_i^m)) < D] = p$$

IV. Optional Parts meet Mixed Criticality

Our goal is to translate MC scheduling into QAS to (1) reuse the feasibility and abortion results and (2) devise new algorithms to convey stochastic guarantees for LO jobs.

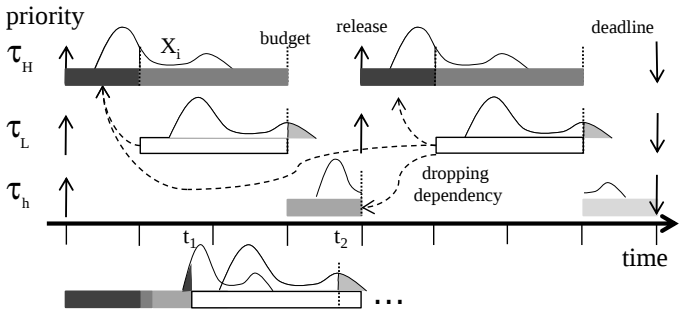


Fig. 1. Mixed-criticality schedule with execution-time distributions X_i and optional-part dependencies (dashed lines) for the tasks $\tau_H = (HI, 4, 4, (1, 3))$, $\tau_L = (LO, 4, 4, (2, -))$ and $\tau_h = (HI, 8, 8, (1, 2))$ with $J_1 = \tau_{H,1}$, $J_2 = \tau_{L,1}$, $J_3 = \tau_{h,1}$, $J_4 = \tau_{H,2}$, $J_5 = \tau_{L,2}$. There is still a chance to complete J_2 after dropping it, even if J_2 exceeds its initial budget.

a) Likelihood to drop low criticality jobs: Fig. 1 shows an example of a standard (i.e., not QAS) adaptive MC algorithm. The algorithm is adaptive in that τ_L must be dropped (i.e., the priority of τ_L must change) to guarantee the completion of the two HI tasks in case J_1 exceeds $C_H(LO)$. In other words, $X_1 < C_H(LO)$ must hold for J_2 to not be dropped and in addition $X_3 < C_h(LO)$ and $X_4 < C_H(LO)$ to start executing J_5. But these are exactly the conditions for the execution of optional parts. By regarding the LO parts of HI jobs whose criticality decision point is before the worst-case response time R_k^{LO} of a job J_k as preceding optional parts of this job, QAS gives us the likelihood of this job being dropped as:

$$1 - \prod_{\tau_i \in T | R_i^{LO} \leq R_k^{LO}} \mathbf{P}[X_i < C_i(LO)] \tag{1}$$

where $R_i^{LO} = C_i(LO) + \sum_{\tau_j \in \mathbf{hp}(i)} \left\lceil \frac{R_i}{T_j} \right\rceil C_j(LO)$ is the worst-case response time of J_i assuming all higher prioritized job J_j require no more than $C_j(LO)$. Fig. 1 indicates this *dropping dependency* as dashed lines. The criticality decision point of a job J_k is the worst-case response time of its low part. At this point, we know whether J_k causes LO tasks to be dropped [1].

b) Likelihood of dropped jobs meeting their deadlines: The bottom part of Fig. 1 shows that it is possible to complete the dropped job J_2 at a priority level where it can no longer defer the execution of J_3, which could have caused a deadline miss. J_2 completes in time if after J_1 has exceeded $C_H(LO)$ the combined execution time $X_1 + X_2 + X_3$ is less than or equal to D_L. J_2 may even complete if $C_L(LO)$ is an unsafe WCET estimate, for example if J_3 stops before time t_1 and if additional time is given to J_2 after t_2. More generally, if a HI job J_k does not complete, it receives an additional part with $Y_k = \max(X_k - C_k(LO), 0)$ and the originally scheduled parts of LO jobs J_l in dropping dependency with J_k are aborted. Instead of not executing dropped jobs J_l, we assume they receive a possibly larger budget at a priority level that is sufficiently low to not risk high completion. Priorities in a strictly lower band fulfill this condition, however, we expect to find less pessimistic setups in the future. The important constraint (in particular when considering more than two criticality levels) is to preserve the relative priority ordering of dropped jobs because then MC guarantees extend to the stochastic MC guarantees for dropped jobs. That is, dropped jobs with a higher criticality level than other dropped jobs complete more likely. As a preliminary result, the likelihood that a dropped job J_i meets its deadline under the condition that $\pi(J_1) > \ldots > \pi(J_{i-1})$ denotes the priority ordering of higher than J_i prioritized jobs after jobs have been dropped is $\mathbf{P}[\max(T_{i-1}, r_i) + X_i < r_i + D_i]$ where $T_{-1} = T_0 = 0$ and

$$T_k = \begin{cases} \max(T_{k-1}, r_k) + X_k & \textbf{if } \max(T_{k-1}, r_k) + \\ & \quad X_k < r_k + D_k \\ \max(T_{k-2}, r_k + D_k) & \textbf{otherwise} \end{cases} \tag{2}$$

c) Time to catch up: Even without the above precaution LO jobs $J_k = \tau_{l,m}$ may catch up with their execution by exploiting the budgets of the next jobs of their tasks. Notice, the result is late and its value degraded. $\tau_{l,m}$ catches up after consuming $\max(X_{l,m} - F, 0)$ of $\tau_{l,m+1}$'s budget and both $\tau_{l,m}$ and $\tau_{l,m+1}$ complete before $\tau_{l,m+1}$'s deadline if $\tau_{l,m+1}$ would complete with its execution-time distribution changed to $\max(X_{l,m} - F, 0) + X_{l,m+1}$ where F is the time that $\tau_{l,m}$ did run before consuming $\tau_{l,m+1}$'s time.

V. Related Work

To our best knowledge, this is the first attempt to cast MC scheduling into an imprecise computation context. There is of course a large body of work on probabilistic analyses and scheduling of non-MC systems. Alahmad et al. [6] investigate probabilistic execution-behavior models and identify stochastic MC scheduling as an open problem [7]. In contrast to our work, they seek to optimize the feasibility of the MC schedule itself, not only of dropped tasks. Also they do not consider the possibility of unsafe WCET estimates for LO jobs.

VI. Conclusions

We present first results connecting quality assuring and mixed-criticality scheduling to give stochastic guarantees for dropped jobs[2].

References

[1] S. Vestal, "Preemptive scheduling of multi-criticality systems with varying degrees of execution time assurance," in *Real-Time Systems Symposium*. Tucson, AZ, USA: IEEE, December 2007, pp. 239–243.

[2] C.-J. Hamann, J. Löser, L. Reuther, S. Schönberg, J. Wolter, and H. Härtig, "Quality Assuring Scheduling - Deploying Stochastic Behavior to Improve Resource Utilization," in *22nd IEEE Real-Time Systems Symposium (RTSS)*, London, UK, Dec. 2001.

[3] C.-J. Hamann, L. Reuther, J. Wolter, and H. Härtig, "Quality-assuring scheduling," TU Dresden, Dresden, Germany, Tech. Rep. TUD-FI06-09, December 2006.

[4] *DO-178C: Software Considerations in Airborne Systems and Equipment Certification*, RTCA, Dec. 2011.

[5] J.-Y. Chung, J. W. S. Liu, and K.-J. Lin, "Scheduling periodic jobs that allow imprecise results," *IEEE Transactions on Computers*, vol. 39, no. 9, pp. 1156–1173, Sep. 1990.

[6] B. Alahmad, S. Gopalakrishnan, L. Santinelli, and L. Cucu-Grosjean, "Probabilities for mixed-criticality problems: Bridging the uncertainty gap," in *Real-Time Systems Symposium - Work in Progress*, Vancouver, Canada, Nov. 2011.

[7] B. Alahmad and S. Gopalakrishnan, "Can randomness buy clairvoyance? a look into stochastic scheduling of mixed criticality real-time job systems with execution time distributions," in *3rd Int. Real-Time Scheduling Open Problems Seminar (RTSOPS)*, Pisa, Italy, July 2012.

[1] Notice, there is no need to make this decision earlier because the completion of no high task is at risk.

[2] This work is partially funded through NSF Grant CNS-0931985 and by the DFG through the cluster of excellence *Center for Advancing Electronics Dresden*.

Synchronous Execution of a Parallelised Interrupt Handler

Christian Bradatsch, Florian Kluge, Theo Ungerer
Department of Computer Science
University of Augsburg
86159 Augsburg, Germany
{bradatsch,kluge,ungerer}@informatik.uni-augsburg.de

***Abstract*—Upcoming multicore processors for embedded real-time domains allow the integration of multiple applications and new features in electronic control units. PharOS [1] enables an efficient parallelisation of tasks by executing these in a time-triggered manner on a multicore processor. Many processes especially in the automotive domain are event triggered. PharOS allows to define event triggered tasks with short deadlines and hence only a low complexity. We also see the need for more complex event-triggered tasks that need to be parallelised. Still, we want to profit from the benefits of the time-triggered execution model. We propose a method to execute a parallelised interrupt handler synchronously within the PharOS execution model on a network-on-chip based manycore processor.**

I. Introduction

Multicore processors are entering the domain of safety-critical real-time systems. This enables the integration of new features and of more complex control algorithms, e.g. in automotive electronic control units. With PharOS [1], an operating system was proposed to enable the use of multicore processors in safety-critical automotive systems. With increasing core numbers, those efforts have to go hand in hand with an efficient parallelisation of applications, thereby still keeping well-known safety properties. The time-triggered execution model used in PharOS enables such a parallelisation for time-triggered tasks. However, we also foresee the need for a parallelisation of event triggered tasks, like more complex crank angle triggered interrupt handlers in an engine management system. While the concept of event triggered tasks is part of PharOS, their parallelisation is not discussed.

Our aim is to execute a parallelised interrupt handler on a manycore processor, where the single cores are linked by an interconnect. In such an architecture, communication between cores takes place in the form of implicit or explicit messages. Depending on source and sink nodes, the messages can have varying worst-case traversal times (WCTTs). A multicast message sent by one core will arrive at different times at the receiver cores. Still, we must ensure that the actual processing of such a message starts at the same time on all cores.

In the following section, we briefly review the properties of the PharOS execution model and discuss their adequacy for the parallelisation of interrupt handlers. Our method for synchronous execution of a parallelised interrupt handler is presented in section III. A first prototype implementation is outlined in section IV. In section V, we give an outlook on future work.

II. Execution Model

PharOS is based on the time-triggered execution model of OASIS. OASIS [2] defines an approach for the design of safety-critical systems, e.g. in nuclear power plants. In OASIS, all tasks are executed in a time-triggered manner. Each task ω has an associated real-time clock H_ω. This clock is defined by the time instances at which input and output of the task can occur. Tasks can use an instruction *ADV*(n) to advance their clock by n instants. A task is then activated again n instants after its last activation. During clock advancement, execution of the calling task is blocked.

For communication between tasks, OASIS defines temporal variables and an asynchronous message passing mechanism. Both mechanisms are coupled to real-time clocks. Sent data is visible at the receiver at predefined time instants. The implementation of these mechanisms ensures that neither senders nor receivers experience blocking times. This model allows to implement concurrent tasks without explicit synchronisation, leading to a less pessimistic WCET analysis as no blocking times through interferences between tasks can occur. Indeed, the only blocking times that can occur are those requested by tasks themselves through the use of the *ADV* instruction.

PharOS [1] extends the OASIS concepts for automotive systems using multicore processors. PharOS partitions a system into time and event triggered domains. Each domain is assigned to dedicated cores. Time-triggered tasks are referred to as *agents* and executed on *computing cores*. In addition, PharOS places event-triggered tasks, called *handlers*, on *control cores*. The control core also processes I/O interrupts. This concept is demonstrated by an example where a handler task monitors a PWM signal for its duty cycle and sends measured data to an agent for further processing.

In our work, we go one step further. Our aim is to parallelise an interrupt handler and benefit from the advantages of the OASIS/PharOS execution model. The need for such an approach arises, if an interrupt handler not only has to read some input data, but also must process the data and set some output within a short deadline. If, furthermore, such an interrupt can occur with widely varying inter-arrival times, it is

hard to dislocate processing and output into the time-triggered domain, if possible at all.

III. Start Time Synchronization

The target of the proposed approach is a parallelised and synchronous processing of an IRQ handler. We assume that *agents* are executed exclusively on *computing cores* and *handlers* on *control cores* accordingly. An I/O interrupt is only triggered at one dedicated control core. On this core, an *initial handler* is executed which does not use the time-triggered execution model. The initial handler sends *messages* to activate processing of the IRQ by the *processing handlers* running on other control cores. Thereby, two requirements have to be fulfilled: (1) the latency between the occurrence of the IRQ and the start of the actual processing must be statically boundable and sufficiently low to allow schedulability, and (2) the processing handlers must start execution at the same time such that their implementation can also use the time-triggered execution model of OASIS/PharOS. From the second requirement follows, that an event triggered processing handler must also be able to use the *ADV* instruction like tasks executed in the time-triggered domain. For the *ADV* instruction a clock reference point is required. In the time-triggered domain this reference point is the start of the program.

To obtain such a reference point for processing handlers, we extend the messages sent by the initial handler by a future-timestamp. This future-timestamp represents the clock reference point. It is computed from the actual timestamp when entering the initial handler and the WCTT of the messages. Each receiving control core uses an *ADV* instruction to advance to this future-timestamp. Thereafter all cores start their processing handler synchronously at the same time.

To realise this approach, two requirement must be fulfilled by the underlying hardware: (1) A message mechanism must be available which signals an incoming message at receiver side immediately. (2) All cores need a common, fine-granular time base.

IV. Implementation

We have performed a first prototype implementation of our approach on the manycore simulator of the parMERASA project [3]. The simulator provides a common time base for all cores by hardware, a flexible interrupt system, and a global address space.

Our use case is an engine control application using, amongst other things, an interrupt triggered by certain crank angle positions. The occurrence of this crank angle interrupt varies within a certain range depending on the rotation speed of the crank shaft. The interrupt signal is processed by the initial handler on a dedicated core. The aim is to execute a parallelised version of the crank angle interrupt handler routine. Currently, only one processing handler is executed exclusively on each control core. Due to that circumstances, a control core is actively waiting for an incoming message and thus needs not to be interrupted by a message notification. This simplifies the implementation.

After an interrupt is asserted at the dedicated interrupt core, the initial IRQ handler is started. The initial handler requests the actual time base and adds a WCTT offset that was calculated offline. The result is the clock reference point t_r which is stored at a specific location m_t in shared memory. Afterwards, the initial handler returns, and the core can also execute a processing handler. Each control core that shall execute a processing handler is spinning on the memory location m_t until its value changes. It then loads the value t_r into a register and executes an $ADV_{\mathrm{abs}}(t_r)$ instruction, that advances to an absolute point in time. Thus, a synchronous start of all processing handlers is enabled.

In our case the WCTT of the message is calculated as the sum of the WCET of the store instruction executed by the initial handler and the WCET of the load instruction executed by the processing handlers. Measurements have shown that the deviation of the start time of each handler is up to 50 clock cycles. Additionally, the actual execution of the handlers starts 150-200 cycles after t_r elapsed. This circumstance is due to the fact that memory accesses to shared as well as private memory are routed through the interconnect of the simulated processor. Thereby, interferences on the interconnect occur and the latency for accesses is quite long. The same effect was also observed when an $ADV(n)$ instruction with the same clock value n was executed on all cores in the time-triggered domain. So the deviations are likely to result because of the underlying hardware.

V. Conclusions and Future Work

We have presented an extension of the *ADV* instruction that was introduced in OASIS. This extension, ADV_{abs}, allows to advance execution to an absolute point in time. Thus, we achieve to synchronously start the threads of a parallelised interrupt handler, and their implementation is able to profit from the time-triggered execution model. In the future, we plan to investigate in two directions. First, we will compare our polling mechanism to messages sent via inter-core interrupts. Second, we will integrate agent and handler tasks on single cores to improve utilization compared to exclusive control cores. The consequences for schedulability of both task types will be analysed.

Acknowledgements

Part of this research has been supported by the EC FP7 project parMERASA under Grant Agreement No. 287519.

References

[1] C. Aussaguès, D. Chabrol, V. David, *et al.*, "PharOS, a multicore OS ready for safety-related automotive systems: results and future prospects", in *Embedded Real-Time Software and Systems (ERTS^2 2010)*, Toulouse, France, May 2010.

[2] C. Aussagués and V. David, "A method and a technique to model and ensure timeliness in safety critical real-time systems", in *Fourth IEEE International Conference on Engineering of Complex Computer Systems (ICECCS '98)*, 1998, pp. 2–12.

[3] T. Ungerer, C. Bradatsch, M. Gerdes, *et al.*, "parMERASA – multi-core execution of parallelised hard real-time applications supporting analysability", in *Euromicro Conference on Digital System Design (DSD), 2013*, Los Alamitos, CA, USA: IEEE, Sep. 2013, pp. 363–370.

On the Schedulability of P-FRP Tasks

Yu Jiang+
School of Computer Science and Technology
Heilongjiang University
Harbin, Heilongjiang 150001, China
jiangyu@hlju.edu.cn

Xingliang Zou+, Albert M. K. Cheng
Department of Computer Science
University of Houston
Houston, TX 77004, USA
xzou@uh.edu, cheng@cs.uh.edu

***Abstract*—As a variant of Functional Reactive Programming (FRP), priority-based FRP (P-FRP) maintains both type-safety and state-less execution paradigm of FRP, and supports assigning different priorities to different tasks in real-time systems. Since the abort-and-restart execution semantics of P-FRP are different from that of the classical preemptive model, the schedulability analysis for P-FRP is much of difference and difficulty. In this short paper we briefly present our ongoing work about an exact schedulability condition and a response time bound we have newly discovered for scheduling P-FRP tasks.**

Keywords—real-time system; schedulability; functional reactive programming (FRP); priority-based FRP; response time analysis; feasibility interval; permissibility interval

I. INTRODUCTION

A. Motivations and contributions

Compared with the classical preemptive model [1], the schedulability analysis for *Priority-based Functional Reactive Programming* (P-FRP) [2] is of difficulty because of the abort-and-restart feature and scheduling uncertainty of lower priority tasks interfered by higher priority tasks. For a P-FRP *n*-task set, the current schedulability condition [3] is only being sufficient and is utilization-based with the bound 1/*n* under some restrictions on the task periods, making it less useful as *n* gets larger. If some conditions could be found without this kind of restrictions it would be much better. Furthermore, under the P-FRP model, for a given release pattern such as synchronous, for the computation-based actual response time analysis, a higher performance algorithm is needed, which is also beneficial to the *worst-case response time* (WCRT) study.

In this work-in-progress paper, we present an exact (necessary and sufficient) schedulability condition for P-FRP *n*-tasks with the *rate monotonic* (RM) scheduling, without using the utilization bound.

B. Backgrounds and related works

Functional Reactive Programming(FRP) [4] is a declarative programming language for the modeling and implementation of safety-critical embedded systems, and is getting widely used in preemptive reactive systems. However, it has no real-time guarantees. To address this limitation, P-FRP has been put forward, which maintains both type-safety and state-less execution paradigm of FRP, and supports assigning different priorities to different tasks in real-time systems.

Different from the classical preemptive model in which tasks can resume execution from the point they were preempted, in P-FRP lower priority tasks will resume in a transactional execution way, i.e., resuming execution from the very beginning, in order to meet the natural requirement of atomic execution of FRP. Therefore, P-FRP is characterized as the abort-and-restart execution semantics.

In their seminal paper, Liu and Layland [1] have shown that, in the classical preemptive model using the *RM* scheduling for periodic tasks with deadlines equal to periods, a *n*-task set is schedulable if the first instance of each task can meet its deadline when these tasks are released synchronously (at the same time). Based on the processor utilization bound, they also presented a necessary condition of schedulability for periodic tasks using the *RM* scheduling. For schedulability analysis under the classic preemptive model with a given priority assignment, there are also some other research work such as [5-8]. However, the schedulability tests discovered for the classical preemptive model, using either processor utilization bound or an iterative equation to compute the WCRT then for checking up the schedulability, do not apply directly to the P-FRP model because of the abort-and-restart feature and hence the uncertain response time of the P-FRP.

In [3] the authors put forward a necessary schedulability condition for 2-task sets, and some sufficient schedulability conditions for 2-task sets and *n*-task sets in P-FRP, based on processor utilization and under certain restrictions on periods and release scenarios. In [9] and [10] they further present the Gap-enumeration and idle-period game board algorithms, respectively, for computing the actual response time, and then the schedulability of the task set is determined. However, both algorithms have some slack in performance due to higher search costs or balance-tree maintaining costs [10]. The authors also present in [11] the feasibility of using time Petri nets for schedulability analysis in P-FRP for small task sets. In [12] the authors prove the feasibility interval for *n*-task sets.

II. BASIC CONCEPTS AND NOTATIONS

In this paper, we consider a P-FRP real-time uniprocessor system with a set of independent, periodic *n* tasks. A task τ_i has a ***(maximum) computation time*** $C_i > 0$, an ***arrival time period*** T_i, a ***relative deadline*** D_i equal to its period T_i, i.e., $D_i=T_i>0$, and a ***release offset*** $O_i \geq 0$ which is the release time of the first instance of the task and is relative to time 0. For later notational convenience we label the tasks so that task τ_i is assigned a fixed priority *i*, where *n is the lowest priority*. When using the *RM* scheduling, there will be $T_1 \leq T_2 \leq ... \leq T_n$.

For $1 \leq k \leq n$, let $\Gamma_k = \{\tau_1, \tau_2, ..., \tau_k\} \subseteq \Gamma_n = \{\tau_1, \tau_2, ..., \tau_n\}$, $O_{min}^{\{j\}} = min_{i=1}^{k}\{O_i\}$, and $O_{max}^{\{j\}} = max_{i=1}^{k}\{O_i\}$.

This work is sponsored in part by the State Scholarship Fund of China under award No. 201308230034, and in part by the US National Science Foundation under award Nos. 0720856 and 1219082.

+Corresponding authors.

We use LCM_k to represent the *least common multiple* (*LCM*) of the periods of the first k tasks, i.e., $LCM_k=LCM(T_1,T_2,...,T_k)$, $1 \le k \le n$, and $LCM_1=T_1$.

An ***absolute time*** t (or *time t*) is the time elapsed since the start, which is assumed to be at the absolute time 0, of the real-time system. A ***half-closed interval*** $[t_i, t_j)$ represents a time interval such that: $\forall t \in [t_i, t_j), t_i \le t < t_j \wedge t_i \neq t_j$.

A ***feasibility interval*** is the time interval $[t, t+H)$ such that if all tasks are schedulable in $[t, t+H)$ then the tasks will also be schedulable in the time interval $[0, Z)$: $Z \rightarrow \infty$ [12].

We define the ***i-permissibility interval*** for task τ_i as the time interval $[t_j, t_k)$ such that the length of the interval, i.e., t_k-t_j, is no less than C_i, and in which no higher priority task is awaiting execution and ready to execute strictly before t_j. Notice that this concept is different from the *k*-gap in [9] which has not taken the requirement of interval length into account in the definition of the *k*-gap.

Due to space limitation, the execution model is referred to [12].When considering abort and restore costs, the *computation time* C_i can be replaced by the *processing time* P_i for simplicity.

III. OUR RESULTS

In this section we briefly present our new results on the schedulability condition without using the utilization bound, and a method of computing the *WCRT* upper bound, for P-FRP *n*-tasks using the *RM* scheduling.

A. An exact schedulability condition

Lemma 1. Considering a schedulable *n*-task set $\Gamma_n = \{\tau_1, \tau_2, ..., \tau_n\}$ using fixed task priority, with a given release order such that for all $1 \le j \le n$, $0 \le O_j < T_j$, then, for each $1 \le k \le n$, (1) for any $t \ge O_{min}^{\{k\}}$, the release and executing patterns of $\Gamma_k = \{\tau_1, \tau_2, ..., \tau_k\} \subseteq \Gamma_n$ in the intervals $[t+m \cdot LCM_k, t+(m+1) \cdot LCM_k)$ and $[t+(m+1) \cdot LCM_k, t+(m+2) \cdot LCM_k)$, $m \ge 0$, will be the same; (2) the feasibility interval of Γ_k is $[t, t+LCM_k)$, where $t \ge O_{min}^{\{k\}}$.

There are two differences between Lemma 1 and those in [12]. First, with the assumption of $0 \le O_k < T_k$, $1 \le k \le n$, the starting point of the relevant intervals shifts to the left and is from $O_{min}^{\{k\}}$ instead of from $O_{max}^{\{n\}}$, making an extended coverage of the time line. Second, Lemma 1 is feasible for all subsets $\Gamma_k = \{\tau_1, \tau_2, ..., \tau_k\} \subseteq \Gamma_n$, $1 \le k \le n$, achieving more flexibility.

Theorem 2. Considering an *n*-task set $\Gamma_n = \{\tau_1, \tau_2, ..., \tau_n\}$ using the *RM* scheduling, with a given release order such that for all $1 \le j \le n$, $0 \le O_j < T_1$, then, for each $2 \le k \le n$, Γ_k is schedulable if and only if: (1) $\Gamma_{k-1} = \{\tau_1, \tau_2, ..., \tau_{k-1}\} \subseteq \Gamma_n$ is schedulable, (2) task τ_k is schedulable in the time interval $[O_k, O_k+max^k)$, where $max^k = max\{ O_{min}^{\{k-1\}} + LCM_{k-1}, T_k\}$, and (3) there are j_k *k-permissibility intervals* for τ_k in the time interval $[O_k, O_k + O_{min}^{\{k-1\}} + LCM_{k-1})$, denoted by $[t_1, t_2)$, $[t_3, t_4)$, ..., $[t_{2j_k-1}, t_{2j_k})$, and the next *k-permissibility interval* is denoted by $[t_{2j_k+1}, t_{2j_k+2})$, such that $j_k > 0$ and $T_k \ge l_{max}$ or $T_k = LCM_{k-1}$ when $t_1 - O_k + C_k \le LCM_{k-1}$ or $T_k \ge l_{max}$ when $t_1 - O_k + C_k > LCM_{k-1}$, where $l_{max} = max_{i=1}^{j_k} \{t_1 - O_k + C_k, t_{2i+1} - t_{2i} + 2C_k - 1\}$.

The release order requirement in this theorem aims at getting the largest response time. Due to space limitation, we omit the proofs of Lemma 1 and Theorem 2 here; they will be included in the complete version of this paper.

B. Computing WCRT upper bound

From Theorem 2 we may compute an *WCRT* upper bound using $max_{i=1}^{j_n} \{t_1 - O_n + C_n, t_{2i+1} - t_{2i} + 2C_n - 1\}$ for Γ_n.

This is a recursive method which can be applied step-by-step from the highest priority task to each of lower ones in the *n*-task set. Compared with the algorithms in [9]-[10], this computing-based schedulability examining method is much more efficient due to the reduction of the maximum search bound from LCM_n to LCM_{n-1}.

IV. CONCLUSIONS AND FUTURE WORK

In this paper we present an exact condition of schedulability for P-FRP *n*-tasks using *RM* scheduling, without using the utilization bound. We reduce the maximum search bound from LCM_n to LCM_{n-1}. It is also a recursive method which can be applied to each task in the *n*-task set. Our ongoing work includes developing a *WCRT*-computing algorithm based on this condition.

REFERENCES

[1] C. L. Liu, L. W. Layland, "Scheduling algorithms for multiprogramming in a hard-real-time environment," Journal of ACM, 1973, 20(1): 46–61.

[2] R. Kaiabachev, W. Taha, A. Zhu, "E-FRP with priorities," in 7th ACM & IEEE Int'l Conf. on Embedded Software(EMSOFT2007), pp.221-230.

[3] C Belwal, A. M. K. Cheng, "A Utilization based Sufficient Condition for P-FRP," in 9th IEEE/IFIP Int'l Conf. on Embedded and Ubiquitous Computing (EUC), 2011, pp.237-242.

[4] Z. Wan, P. Hudak, "Functional reactive programming from first principles," in ACM SIGPLAN PLDI 2000, pp. 242–252.

[5] J. Lehoczky, L. Sha, Y. Ding, "The rate monotonic scheduling algorithm: Exact characterization and average case behavior," IEEE RTSS 1989, pp. 166-171

[6] N. Audsley, A. Burns, M. Richardson, K. Tindell, A. Wellings. "Applying new scheduling theory to static priority preemptive scheduling," Software Engineering Journal, 1993, 8(5): 284-292

[7] G. Bernat, "Response time analysis of asynchronous real-time systems," Real-Time Systems, 2003, 25(2-3): 131–156.

[8] R. I. Davis, A. Burns. "Response Time Upper Bounds for Fixed Priority Real-Time Systems," IEEE RTSS 2008, pp.407-418.

[9] C. Belwal, A. M. K. Cheng, "Determining actual response time in P-FRP," in PADL 2011, LNCS 6539, pp. 250–264.

[10] C. Belwal, A. M. K. Cheng, "Determining Actual Response Time in P-FRP Using Idle-Period Game Board," in 14th IEEE Int'l Symp. on Object/Component/Service-Oriented Real-Time Distributed Computing, 2011, pp. 136–143.

[11] C. Belwal, A. M.K. Cheng, Y.F. Wen,"Time Petri nets for schedulability analysis of the transactional event handlers of P-FRP," in ACM Research in Applied Computation Symp. (RACS), 2012, pp. 257-262.

[12] C. Belwal, A. M. K. Cheng, B. Liu, "Feasibility interval for the transactional event handlers of P-FRP," Journal of Computer and System Sciences, 2013, 79(5): 530–541.

Concurrent soft-real-time execution on GPUs

Kiriti Nagesh Gowda, Harini Ramaprasad
kiriti@siu.edu, harinir@siu.edu
Southern Illinois University Carbondale

***Abstract*—Graphics Processing Units (GPUs) are computational powerhouses that were originally designed for accelerating graphics applications. However, in recent years, there has been a tremendous increase in support for general purpose computing on GPUs (GPGPU). GPU based architectures provide unprecedented magnitudes of computation at a fraction of the power used by traditional CPU based architectures. As real-time systems integrate more and more functionality, GPU based architectures are very attractive for their deployment. However, in a real-time system, predictability and meeting temporal requirements are much more important than raw performance. While some real-time jobs may benefit from the performance that all cores of the GPU can provide, most jobs may require only a subset of cores in order to successfully meet their temporal requirements. In this paper, we propose to study concurrent scheduling of soft-real-time jobs on a GPU based platform.**

Keywords GPU; Soft Real Time; Scheduling.

I. Introduction

Real-time systems not only have to satisfy logical correctness requirements like any other computing system, but also have to adhere to temporal correctness requirements, typically represented as deadlines for every job within the system. As computational demands of such systems continue to increase in the wake of the ubiquitous presence of real-time systems in today's world, traditional single-core architectures are no longer a viable option for their deployment. As a result, there is a significant body of research that studies the challenges involved in real-time execution on multi-core architectures. However, most of this work focuses on CPU based architectures.

Recently, researchers have started to explore the use of GPU based architectures in real-time systems. There is strong motivation for enabling real-time execution on GPUs. As noted by Elliott and Anderson [1], there are two fundamental aspects that make GPUs an attractive option for real-time systems. First, GPUs execute at higher frequencies, thereby accelerating the execution of jobs allocated to it. This could improve system responsiveness. Second, the power needed for a GPU to carry out an operation is much lesser than that needed by traditional CPUs, making it ideal for use in real-time embedded systems.

Having said that, execution on GPU architectures has fundamental differences compared to that on CPU based multi-core architectures. First, due to significant hardware and firmware challenges in enabling preemptive execution, GPU execution is assumed to be non-preemptive. Second, GPUs do not provide the degree of controllability of cores that is typically available on CPU based multi-core platforms. In early versions of GPU platforms, only one instruction stream or function represented by a *kernel* could execute on a GPU at any given time, regardless of GPU utilization. As such, most existing work on enabling real-time execution on GPUs treat the GPU as a black box and focus on developing techniques to determine the order (schedule) for dispatching kernels.

More recent GPU platforms such as the NVIDIA Fermi architecture [2] do support concurrent execution of kernels[1]. In practice, real-time functions may not benefit from the performance that using the entire GPU (i.e., all its cores) can provide. One reason may be simply because it does not require that amount of computational power. A second reason may be that the function is memory bound and cannot effectively utilize all computational elements due to memory transfer delays. On the other hand, concurrent execution of multiple functions could be tremendously useful in maintaining timeliness of applications. This motivates us to explore the development of policies for concurrent scheduling of kernels. In the current work, we target soft-real-time job execution, i.e., execution of jobs with non-strict deadlines.

II. Existing Research

Elliott and Anderson present potential real-time applications that can benefit from GPU architectures and discuss the limitations and constraints of current GPU architectures [1]. Several researchers propose policies for scheduling real-time jobs on the GPU [4], [5], [6] using a priority-based approach. In other recent work, researchers target a multi-GPU model, where jobs are dispatched on to an array of available GPUs [7], [6]. Research has been conducted in decoding the GPU driver or managing the GPU as a resource [8], [9]. However, this entire body of work assumes that only one kernel may execute on a GPU at a given time. Recent work has explored concurrent execution of multiple kernels on a single GPU, using hardware (e.g., NVIDIA Fermi architecture) that supports concurrent execution. Wang et al. discuss considerations involved in such execution and present an experimental evaluation of the performance impact of such execution [3]. This work does not specifically target real-time execution. Junsung et al. present frameworks for supporting adaptive GPU resource management by allowing tasks to use a variable number of cores on the GPU based on their needs and core availability [10]. The authors propose an explicit management mechanism that requires significant programmer

[1]Currently, the Fermi architecture only allows concurrent execution of kernels that share a common GPU context. However, the fundamental ideas of our work are not affected by or dependent on this limitation. Moreover, there exist software solutions such as context funneling [3] to allow kernels from multiple GPU contexts to execute concurrently.

involvement and an implicit management mechanism with less programmer involvement.

III. Our Ongoing Work

Our ongoing work aims to develop a dynamic schedule management framework for soft-real-time jobs[2] on GPU based architectures. Cores on a platform such as the NVIDIA Fermi architecture [2] are organized into clusters, termed streaming multiprocessors (SMs). Cores within each SM share resources (register file, control units, L1 cache, etc.) and execute a common kernel. Our goal is to divide a real-time job into kernels and schedule kernels on the GPU, treating each SM as an indivisible unit. We propose to achieve this goal with minimal programmer involvement.

In general, a kernel consists of a set of thread blocks. When a kernel is dispatched to the GPU, a work distribution engine in the GPU schedules thread blocks on available SMs[3]. The fundamental idea behind our technique is to divide a kernel into a set of controlled blocks of threads such that the number of threads per block is close to the total number of threads that a single SM can handle concurrently. For example, in the NVIDIA Fermi architecture [2], every SM is capable of supporting 1536 threads, among which 1024 threads can execute concurrently with minimal context switch costs. So, for this architecture, block sizes of 1024 threads are found to be a suitable choice. In this way, no more that one block may reside on a SM at a given time and hence, the number of blocks is a measure of the number of SMs that a kernel will occupy. We have conducted preliminary experimentation on the NVIDIA Fermi platform to verify and evaluate the feasibility of this basic approach towards SM scheduling. Thus far, we have found the results to be encouraging.

We propose to exploit this basic idea to perform coarse-grained scheduling of jobs on SMs. Our work lays emphasis on minimal programmer involvement. To this end, we are developing a dynamic schedule management framework (somewhat similar to the implicit adjustment approach proposed in [10]) that is responsible for 1) keeping track of current and expected SM availability; 2) determining which kernel(s) to dispatch to the GPU at a given time; and 3) determining how many SMs to assign for a given kernel. These decisions will be made based on observed and predicted system state and on job characteristics (expected execution times, deadlines, etc.). Figure 1 depicts our framework. As seen in the figure, our scheduling framework resides on a CPU core and dispatches kernels to the GPU.

IV. Conclusions

Graphics Processing Units (GPUs) are capable of providing tremendous computational power under reasonable power/energy budgets. Recent GPU based platforms allow concurrent execution of multiple functions on the GPU. This provides an avenue to explore real-time scheduling on such platforms, trading off raw performance of individual jobs for improved responsiveness and schedulability of job sets. In the ongoing work described in this paper, we envision the development of a dynamic schedule management framework for soft-real-time job execution on GPU based platforms. In ongoing work, we are targeting sets of independent soft-real-time jobs. As part of future work, we plan to extend our framework to support recurring (periodic) soft-real-time tasks.

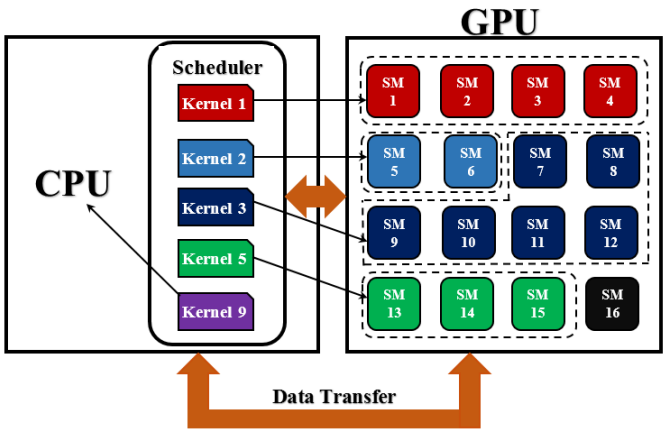


Fig. 1. Concurrent kernel execution on GPU

[2]To the best of our knowledge, there is no technique that produces safe estimates of the worst-case execution time or WCET of jobs for GPU based architectures because of their closed nature. Hence, we currently target soft-real-time jobs and propose to use measurement-based estimation of job execution times.

[3]Dependent blocks in a kernel are made part of a single stream that is guaranteed to execute in sequence.

References

[1] G. A. Elliott and J. H. Anderson, "Real-World Constraints of GPUs in Real-Time Systems," in *International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, 2011, pp. 48–54.

[2] P. N. Glaskowsky, "NVIDIA's Fermi: the first complete GPU computing architecture," *White paper*, 2009.

[3] L. Wang, M. Huang, and T. El-Ghazawi, "Exploiting concurrent kernel execution on graphic processing units," in *International Conference on High Performance Computing and Simulation (HPCS)*, 2011, pp. 24–32.

[4] G. A. Elliott and J. H. Anderson, "Globally scheduled real-time multiprocessor systems with GPUs," *Real-Time Systems*, vol. 48, no. 1, pp. 34–74, 2012.

[5] S. Kato, K. Lakshmanan, R. Rajkumar, and Y. Ishikawa, "TimeGraph: GPU scheduling for real-time multi-tasking environments," in *USENIX Annual Technical Conference (USENIX ATC)*, 2011, p. 17.

[6] G. A. Elliott, B. C. Ward, and J. H. Anderson, "GPUSync: A framework for real-time GPU management," in *Real-Time Systems Symposium (RTSS)*, 2013, pp. 33–44.

[7] G. A. Elliott and J. H. Anderson, "An optimal k-exclusion real-time locking protocol motivated by multi-GPU systems," *Real-Time Systems*, vol. 49, no. 2, pp. 140–170, 2013.

[8] G. Elliott and J. Anderson, "Robust real-time multiprocessor interrupt handling motivated by GPUs," in *Euromicro Conference on Real-Time Systems (ECRTS)*, 2012, pp. 267–276.

[9] S. Kato, K. Lakshmanan, A. Kumar, M. Kelkar, Y. Ishikawa, and R. Rajkumar, "RGEM: A responsive GPGPU execution model for runtime engines," in *Real-Time Systems Symposium (RTSS)*, 2011, pp. 57–66.

[10] J. Kim, R. R. Rajkumar, and S. Kato, "Towards adaptive GPU resource management for embedded real-time systems," *ACM SIGBED Review*, vol. 10, no. 1, pp. 14–17, 2013.

Scheduling Hard Real-Time Self-Suspending Tasks In Multiprocessor Systems

Maolin Yang[1], Hang Lei[1], Yong Liao[1], Furkan Rabee[2]
[1]School of Information and Software Engineering, [2]School of computer science and engineering,
University of Electronic Science and Technology of China (UESTC), China
maolyang@gmail.com, hlei@uestc.edu.cn, liaoyong@uestc.edu.cn, forkanr@yahoo.com

***Abstract*—The problem of self-suspensions has attracted a lot of attentions in recent years, especially when more and more special-purpose processors are adopted in systems with real-time requirements. This work presents a novel scheduling approach, based on semi-partitioning, for fixed priority hard real-time sporadic self-suspending tasks in multiprocessor systems. By splitting self-suspending tasks into non-suspending subtasks and assigning them to different processors, this scheduling approach can avoid higher priority tasks from introducing additional schedulability losses, by suspension delays, to lower priority tasks assigned on the same processor. We also conducted schedulability experiments to study the performances of different scheduling approaches.**

I. INTRODUCTION

In order to improve the performance of increasingly complex embedded applications, more and more embedded system designers rely on modern System-on-a-Chip (SoC) techniques that are able to integrate several special-purpose processors into single physical systems. Conceivably, most of these systems are heterogeneous, for example a system may contain a homogeneous multi-core processor and one or more accelerators such as Graphics Processing Units (GPUs). However, the use of such accelerators may introduce considerable suspension delays [1]. Tasks interacting with I/O devices and waiting for shared resources can also lead to such delays. A major issue in building such heterogeneous real-time systems is how to make full use of the available computing resources while ensuring real-time requirements for the self-suspending tasks in the system.

In general, tasks can be scheduled by global, partitioned, semi-partitioned, or clustered scheduling. Wherein, semi-partitioned scheduling extends partitioning by allowing part of tasks to migrate between partitions, while clustered scheduling limits global scheduling by only allowing tasks to migrate among processors within the same cluster.

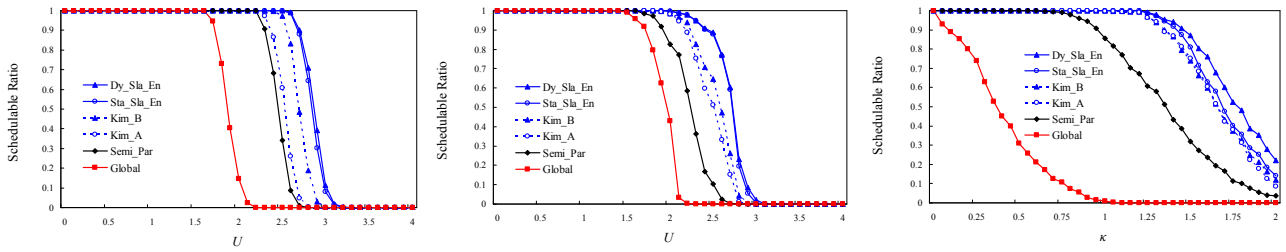
Although suspension-oblivious analysis [2][3][4] that simply incorporates suspension delays in normal execution time requirements seems to be a computationally efficient approach to deal with the schedulability problem for self-suspending tasks, it can be quite pessimistic in case suspension durations are long. On the other hand, it is an intractable problem to find an optimal solution for multiprocessor real-time scheduling under suspension-aware analysis [5]. Prior work has proved that the feasibility problem of scheduling periodic tasks with implicit deadlines and at most one suspension per task in uniprocessors is NP-hard [6]. Moreover, classical dynamic priority scheduling algorithm such as Earliest-Deadline First (EDF) can cause scheduling anomalies in self-suspending task systems.

In Fixed-Priority (FP) uniprocessor scheduling, Kim et al. [7] developed the first two Worst-Case Response Time (WCRT) analyses for tasks with only one suspension per task. Lakshmanan et al. [8] proposed two slack enforcement schemes for Sporadic Self-Suspending (SSS) tasks under RM scheduling. However, these schemes may cause a lot of run-time overheads through are considered to be efficient in terms of schedulability. Most recently, Kim et al. [9] proposed a set of heuristics for SSS tasks based on a segment-fixed priority scheduling and DM scheduling. Under multiprocessor scheduling, Liu and Anderson [10] proposed the first Global FP (GFP) schedulability analysis for Hard Real-Time (HRT) SSS tasks. So far as we know, no prior work has been reported the issue of scheduling HRT SSS tasks in semi-partitioned scheduled multiprocessors. In this work, we develop a scheduling approach based on semi-partitioning, without use any enforcement mechanism, and present its associated WCRT analysis for FP-HRT-SSS tasks, we also bring preliminary results to study the performances of different scheduling approaches.

II. SEMI-PARTITIONED SCHEDULING FOR SSS TASKS

System model. n sporadic tasks $\Gamma=\{\tau_1,\tau_2,\ldots,\tau_n\}$ (ordered by decreasing order of priority) are scheduled on m processors $p_1, p_2, \ldots, p_m$. Each τ_i contains s_i ($s_i \geq 1$) execution segments with suspension between consecutive segments. Let $\tau_{i,j}$ denote the jth segment of τ_i, and let $BC_{i,j}/WC_{i,j}$ to be the Best/Worst-Case Execution Time (B/WCET) of $\tau_{i,j}$. We use $WC_i=\sum_{\forall j}WC_{i,j}$ as the WCET of τ_i. When $\tau_{i,j}$ ($1\leq j\leq s_i-1$) finishes, it suspends for $G_{i,j}\in[G_{i,j,min}, G_{i,j,max}]$ and then $\tau_{i,j+1}$ becomes ready. The period or minimum time interval between any two consecutive jobs of τ_i is T_i, and the relative deadline is D_i. The utilization of τ_i and $\tau_{i,j}$ are $u_i=WC_i/T_i$ and $u_{i,j}=WC_{i,j}/T_i$ respectively. The system utilization is denoted by $U=\Sigma_{\forall i}u_i$. We also assume $D_i=T_i$ and $s_i\leq m$ for simplicity.

Scheduling. In the partitioning phase, tasks are split to pieces and are assigned among processors. Each execution segment of a task is considered to be a subtask, and all subtasks of the same task are assigned to different processors. Tasks are assigned one after another, and the Worst-Fit-Decreasing (WF) heuristic is used for sub-task (segment) assignment. In the scheduling phase, all subtasks assigned to the same processor are scheduled by RMS. Also,



(a) (b) (c)

Figure 1. Schedulable ratios in various scenarios where (a) $\kappa = 0.8$, $u_i \in [0.01, 0.2]$; (b) $\kappa = 0.5$, $u_i \in [0.2, 0.5]$; and (c) $U = 2$, $u_i \in [0.05, 0.3]$.

each sub-task is executed strictly conforming to its logical order and without overlapping with other sub-tasks of the same task.

According to this scheduling approach, each processor schedules a set of non-self-suspending subtasks. Once a job self-suspends, it will not resume on the same processor again in the same period, thus will not introduce additional suspension-related interference to lower priority subtasks that are assigned on the same processor.

WCRT analysis. Let the WCRT of $\tau_{i,j}$ be $r_{i,j}$, and let the release time of $\tau_{i,1}$ be φ_i, then the release time of $\tau_{i,j}$ can be upper and lower bound by Eq.(1) and Eq.(2) respectively[1]

$$UO_{i,j} = \varphi_i + \sum_{\forall k \in [0, j-1]} (r_{i,k} + G_{i,k,max}) \quad (1)$$

$$LO_{i,j} = \varphi_i + \sum_{\forall k \in [0, j-1]} (BC_{i,k} + G_{i,k,min}) \quad (2)$$

For the constraints of space, we present without giving the proof of the following theorem.

Theorem 1 The amount of computations that $\tau_{i,j}$ can execute in a time interval L ($L > WC_{i,j}$) is upper bounded by

$$I_{i,j} = (1 + \lfloor \beta / T_i \rfloor) WC_{i,j} + |\min(WC_{i,j}, \beta - \lfloor \beta / T_i \rfloor T_i - LO_{i,j} - \varphi_i)|_0 \quad (3)$$

where $\beta = \max(0, L - T_i - \varphi_i + UO_{i,j+1} - WC_{i,j} - G_{i,j,max})$ if $0 < j < s_i$, and $\beta = \max(0, L - T_i + r_i - WC_{i,j})$ if $j = s_i$.

Let $\Gamma_{i,j}$ be the set of subtasks that are assigned on the processor where $\tau_{i,j}$ is assigned. The WCRT of $\tau_{i,j}$ can be bounded by performing a fixed-point iteration of the RHS of (4) starting with $L = WC_{i,j}$ as follows.

$$r_{i,j} = WC_{i,j} + \sum_{h<i \wedge \exists y, \tau_{h,y} \in \Gamma_{i,j}} I_{h,y} \quad (4)$$

The WCRT of τ_i can be bounded by

$$r_i = \sum_{\forall j \in [1, si-1]} (r_{i,j} + G_{i,j,max}) + r_{i,si} \quad (5)$$

III. Preliminary Results

We compare the proposed method (Semi_Par) with prior work[2] through schedulability experiments. The WFD heuristic is used for task assignment under partitioned scheduling. Task period is uniformly distributed from [100, 1000], u_i is randomly chosen from [0.01, 0.5]. We assume m=4, and each task contains only two execution segments and one suspension in this preliminary experiments, i.e. s_i =2 for all i. Both $BC_{i,*}/WC_{i,*}$ and $G_{i,*,min}/G_{i,*,max}$ are set to be 0.5. Let $WC_{aver} = WC_i/(s_i - 1)$, i.e., the average WCET of each execution segment. We set $\lambda = WC_{i,j}/WC_{aver}$ lies in [0.2, 1.8], and $\kappa = G_{i,j}/WC_{i,j}$ lies in [0.05, 2]. Experimental results in Fig.1 show that, schedulability under partitioned scheduling and using dynamic slack enforcement scheme [8] performs the best, and the proposed method outperforms GFP in [10]. It can also be observed that schedulability tends to perform worse with increasing suspension durations.

IV. Conclusion

This work presents a semi-partitioned scheduling for FP-HRT-SSS tasks, which avoids higher priority tasks from introducing suspension-related schedulability losses to lower priority tasks. Preliminary experimental results show that the proposed method outperforms prior work under GFP scheduling. However, partitioned scheduling with the state-of-the-art analysis for uniprocessors seems to be more preferable at present. The ongoing work involves (1) introducing period enforcement machanisms to improve the WCRT analysis, and (2) studing about the task assigment algorithms to improve the scheduliablity.

[1] Specifically, we let $r_{i,0} = 0$, $BC_{i,0} = WC_{i,0} = 0$, and $G_{i,0,max} = G_{i,0,min} = 0$.

[2] The WCRT analyses in [7] (Kim_A and Kim_B) and [8] (Dy_Sla_En and Sta_Sla_En) are used for partitioned scheduling, and the schedulability analysis in [10] (GFP) is used for global scheduling.

References

[1] G. Elliott, J. Anderson, "Globally scheduled real-time multiprocessor systems with GPUs," *Journal of Real-Time Systems*, vol. 48, no. 1, pp. 34-74, 2012.

[2] J. Liu. Real-time systems. Prentice Hall, 2000

[3] B. Brandenburg, J. Anderson, "Optimality results for multiprocessor real-time locking," in *RTSS* pp. 49-60, 2010.

[4] B. Brandenburg, J. Anderson, "Real-time resource-sharing under clustered scheduling:mutex, realer-writer, and *k*-exclusion locks," in *EMSOFT*, pp. 69-78, 2011.

[5] K. Lakshmanan, S. Kato, R. Rajkumar, "Open problems in scheduling self-suspending tasks", in *RTSOPS*, pp 12–13, 2010.

[6] F. Ridouard et al., "Negative results for scheduling independent hard real-time tasks with self-suspensions," in *RTSS*, pp. 1-10, 2004.

[7] I. Kim, K. Choi, S. Park, K. Kim, and M. Hong, "Real-time scheduling of tasks that contain the external blocking intervals," in *RTCSA,* pp. 54-59, 1995.

[8] K. Lakshmanan and R. Rajkumar, "Scheduling self-suspending real-time tasks with rate-monotonic priorities," in *RTAS*, pp. 3-12, 2010.

[9] J. Kim, B. Andersson, D. Niz, and R. Rajkumar, "Segment-fixed priority scheduling for self-suspending real-time tasks," in *RTSS*, pp. 246-257, 2013.

[10] C. Liu, J. Anderson, "Suspension-aware analysis for hard real-time multiprocessor scheduling," in ECRTS, pp. 271-281, 2013.

Mapping Real-Time Tasks onto Many-Core Systems considering Message Flows

Matthias Becker*, Kristian Sandström†, Moris Behnam*, Thomas Nolte*†
*MRTC / Mälardalen University, Västerås, Sweden
{matthias.becker, moris.behnam, thomas.nolte}@mdh.se
†Industrial Software Systems / ABB Corporate Research, Västerås, Sweden
kristian.sandstrom@se.abb.com

***Abstract*—In this work we focus on the task mapping problem for many-core real-time systems. The growing number of cores connected by a Network-on-Chip (NoC) calls for sophisticated mapping techniques to meet the growing demands of real-time applications. Hardware should be used in an efficient way such that unnecessary resource usage is avoided. Because of the $\mathcal{NP}-$ hardness of the problem, heuristic and meta-heuristic techniques are used to find good solutions. We further consider periodic communication between tasks and we focus on a static mapping solution.**

I. Introduction

According to Moore's Law, the number of transistors on integrated circuits doubles every two years, this observation is still true. Processor frequency on the other hand has reached stagnation. Higher frequency calls for higher operating voltage which leads to an increasing power density that in the end is limited by laws of physics. Both observations lead the industry to focus on implementing multiple instances of the same element on the die to increase the computational power and meet the demand of future applications. A single processor will incorporate tenths and hundreds of cores in the upcoming years. Traditional interconnection mediums like the shared bus or the crossbar are not suitable for such a large number of interacting cores, since they all access the same medium and thus the individual cores start to block each other while trying to access the bus. The NoC is instead used as an interconnection medium between the individual cores [1]. Rather than connecting all cores to the same bus, a network is reassembled on the chip. The individual cores are located on so called tiles, together with a local memory subsystem. The tiles themselves are connected to a router, reassembling the building blocks of the NoC. Communication channels between the routers form the network, where different network topologies are possible.

The natural parallelism and the high computational power at low energy consumption make many-core processors well suited for signal processing applications in embedded systems [2]. However the increasing number of computing elements on one die leads to a number of new challenges especially for embedded real-time systems. Since energy consumption is a dominant factor for embedded systems, hardware should be used in an energy efficient way while meeting all execution deadlines of the individual tasks. In contrast to traditional applications it becomes increasingly important to consider inter task/core communication to fully exploit the available parallel resources. With this in mind task mapping onto cores is one of the main challenges. The bin packing problem can be reduced to the basic task mapping problem, without communication between tasks. Therefore even the basic task mapping problem is $\mathcal{NP}-$ hard. Several heuristics and meta-heuristics have been proposed. Two recent surveys on mapping techniques for many-core systems [3], [4] give an overview of the current state of the art. Some of the introduced algorithms target real-time systems however most of them target general computing systems. Mapping algorithms can be divided into two categories, static and dynamic. Static mapping finds a solution at design time and the tasks stay at those cores during run time, static mapping is used for most real-time applications. In contrast, dynamic mapping allows tasks to be remapped during runtime, e.g., for load balancing purposes. We consider static mapping algorithms for the purpose of this work. Ali et al. [5] have proposed a design time mapping algorithm to map real-time streaming applications. Their algorithm uses multiple heuristics to map the tasks by considering locality. Communicating tasks are mapped close together and parallel tasks are mapped on different cores to exploit parallelization. In contrast our approach applies meta-heuristic techniques to map the tasks.

II. Assumptions and Setup

A. Hardware Assumptions

Similar to most recent hardware [6]–[8] we assume a 2D-Mesh network structure connecting the n identical cores on the many-core processor. The individual tiles each contain a scratchpad memory besides the core. As most recent processors [6]–[8] we assume wormhole routing [9] on the NoC. One of the main advantages compared to transaction based routing mechanisms is the limited memory resource needed on each router. The deadlock and livelock free XY-routing is implemented to determine the path taken by the header flit. The deterministic routing strategy makes it well suited for real-time systems.

B. Task and Message Model

We consider a periodic task model with N tasks, where each task τ_i can be scheduled on each core. We further define Γ as the set of all tasks τ_i. Since we target static mapping algorithms, tasks are not allowed to migrate between cores. The tuple $\tau_i = \{C_i, T_i\}$ describes task τ_i, where C_i is the task's worst case execution time and T_i is the task period.

Tasks can communicate with each other by periodic message passing. We model those messages as traffic flow $f_{i,j}$ starting at task i and ending at task j. Each flow is represented

by the tuple $f_{i,j} = (T_{i,j}, D_{i,j}, S_{i,j}, P_{i,j})$, where $T_{i,j}$ is the flow's period and equal to the period T_i of the task initiating the flow, $D_{i,j}$ the deadline, $S_{i,j}$ the number of bytes to be transfered and $P_{i,j}$ is the priority of the flow. F represents the set of all traffic flows $f_{i,j}$. Our initial work assumes equal periods for communicating tasks, therefore the message deadlines are, as the task deadlines, equal to the period.

III. The Task Mapping Problem

We first define suitable models for both the hardware and the task set and we outline our approach followed by a discussion on open challenges inherent in the mapping problem.

A. Application and Architecture Model

We represent the application as a directed graph $G(\Gamma, F)$. Each vertex $\tau_i \in \Gamma$ represents one task. The traffic flow between two tasks is modeled as an edge $f_{i,j} \in F$, connecting task τ_i and task τ_j. As a first approach we allow only one predecessor but multiple successors.

The architecture is represented by a topology graph [10]. The topology graph is a directed graph $P(N, L)$ where each vertex $n_i \in N$ represents a node in the topology, e.g., a router, a tile or an external memory. Connections between the vertexes are represented by directed edges $l_{i,j} \in L$.

The main objective now is to find a mapping $\Omega : G \rightarrow P$ such that all traffic flows meet their deadlines.

B. Mapping and Scheduling

Mapping and scheduling are closely related. We have to find a suitable task to core mapping and further a viable scheduling policy on each core which allows all tasks and message flows to meet their deadlines. As most related work we assume preemptive fixed priority scheduling and rate monotonic priority assignment [11] on each core.

Mapping of tasks onto cores can be done by heuristics and meta-heuristics. Most related work in mapping of real-time tasks applies heuristics to find a mapping. We plan to apply meta-heuristic techniques since they are widely used to map applications in general many-core systems as well as in cloud computing. In particular we plan to apply particle swarm optimization [12]. Our approach is similar to the one proposed by Sahu et al. [13] but since meeting deadlines is the first priority for us, we need to change the fitness function used to grade the individual solutions found by the algorithm. A first approach for a fitness function is the number of tasks not meeting their deadline. Since execution times of tasks depend on their predecessors to finish and the message to arrive, we propose to use a holistic response time analysis. We use the transaction based fixed priority analysis for distributed real-time systems by Tindell and Clark [14] to compute the response time of each task. The work by Tindell and Clark assumes a shared bus, instead we apply the worst case traversal time analysis for mesh based NoC by Shi and Burns [15].

C. Open Challenges

In our current work-in-progress we have identified several open challenges that must be addressed in finding a good solution to the mapping problem.

- As the current communication model assumes each message to be sent at the beginning or end of a period, a more realistic model is required to more accurately replicate the messaging behavior of actual applications.
- Often several applications are mapped onto one many-core processor. Those applications are often developed by different teams and thus a way of guaranteeing and provisioning resources to the individual applications is useful.
- The restricted resources available in embedded systems call for energy efficient execution. Additional to guaranteeing that all deadline are met the mapping should also be optimized for a low energy consumption.

IV. Conclusion

In this paper we outline our approach to map real-time tasks onto a many-core processor. We first define the problem by supplying suitable models to represent the application and hardware architecture, and we give a short outline of the planned mapping strategy. We further identify challenges related to our approach, that we plan to address with further research.

Acknowledgment

The work presented in this paper is supported by the Knowledge Foundation via the research project PREMISE.

References

[1] L. Benini and G. De Micheli, "Networks on chips: a new soc paradigm," *Computer*, vol. 35, no. 1, pp. 70–78, 2002.

[2] J. Bengtsson, "Models and methods for development of dsp applications on manycore processors," Ph.D. thesis, Halmstad University, 2009.

[3] P. K. Sahu and S. Chattopadhyay, "A survey on application mapping strategies for network-on-chip design," *J. Syst. Archit.*, vol. 59, no. 1, pp. 60–76, 2013.

[4] A. K. Singh, M. Shafique, A. Kumar and J. Henkel, "Mapping on multi/many-core systems: survey of current and emerging trends," in *50th DAC*, pp. 1:1–1:10, 2013.

[5] H. Ali, L. M. Pinho and B. Åkesson, "Critical-path-first based allocation of real-time streaming applications on 2d mesh-type multi-cores," in *19th RTAS*, 2013.

[6] Intel. Single chip cloud computer. http://www.intel.com/content/www/us/en/research/intel-labs-single-chip-cloud-computer.html, Retrieved February 2014.

[7] Tilera. Tile64 processor. http://www.tilera.com/products/processors, Retrieved February 2014.

[8] *Epiphany Architecture Reference*, Adapteva Inc., Adapteva Inc. 1666 Massachusetts Ave, Suite 14 Lexington, MA 02420 USA, 2012.

[9] L. Ni and P. McKinley, "A survey of wormhole routing techniques in direct networks," *IEEE Comp. J.*, vol. 26, no. 2, 1993.

[10] R. Marculescu, J. Hu and U. Ogras, "Key research problems in noc design: a holistic perspective," in *3rd CODES+ISSS*, pp. 69–74, 2005.

[11] C. L. Liu and J. W. Layland, "Scheduling algorithms for multiprogramming in a hard-real-time environment," *J. ACM*, vol. 20, no. 1, 1973.

[12] J. Kennedy and R. Eberhart, "Particle swarm optimization," in *Proceedings., 4th ICNN*, pp. 1942–1948, 1995.

[13] P. Sahu, P. Venkatesh, S. Gollapalli and S. Chattopadhyay, "Application mapping onto mesh structured network-on-chip using particle swarm optimization," in *2nd ISVLSI*, pp. 335–336, 2011.

[14] K. Tindell and J. Clark, "Holistic schedulability analysis for distributed hard real-time systems," in *Microprocessing and Microprogramming*, vol. 40, pp. 117–134, 1994.

[15] Z. Shi and A. Burns, "Real-time communication analysis for on-chip networks with wormhole switching," in *2nd NOCS*, 2008.

Comparing Heuristics and Linear Programming Formulations for Scheduling of In-Tree Tasksets

Thomas Kothmayr, Jakob Hirscheider, Alfons Kemper
Chair for Database Systems,
Technische Universität München, Germany
{kothmayr, hirschei, kemper}@in.tum.de

Andreas Scholz, Jörg Heuer
Corporate Technology, Siemens AG
{andreas.as.scholz, joerg.heuer}@siemens.com

***Abstract*—Cheap but resource constrained platforms are poised to assume duties in next generation automation systems - the Internet of Things (IoT) is entering the real-time scene. For highly distributed systems like these, service oriented architectures (SOAs) are being increasingly adapted to raise the overall level of flexibility and adaptability. Our approach encompasses a SOA for hard real-time tasks in industrial automation, aimed at IoT class devices. The feasibility of task assignments to machines is verified through computation of a local schedule for the tasks assigned to each device. This reintroduces the need for efficient non-preemptive single machine scheduling. In this paper, we evaluate the efficiency of five heuristics and two linear program formulations for scheduling task sets with release times, deadlines and in-tree precedence constraints.**

I. Introduction

Our vision takes the SOA approach to automation [1] and aims to make resource constrained IoT class devices full members of a hard real-time SOA. We chose a top-down approach to this problem, focusing on the planing and data-flow aspects instead of ontologies and network protocols. In our system, automation tasks are defined as cyclic workflows of distinct subtasks with precedence constraints and end-to-end deadlines. Individual tasks are assigned to machines in a real-time network by an engineer or an automatic planning component. Local constraints (task release times and deadlines) are derived from the underlying network configuration, dependencies of the tasks and the global end-to-end deadline. The feasibility of the assignment is verified by scheduling the tasks on each machine according to their new local constraints. At this early stage, our current work focused on efficient local scheduling as a means to verify a manual task assignment.

II. Formal Problem Description

The scheduling problem analyzed in this paper is as follows: Find a feasible schedule for a set of jobs $\mathcal{J} = \{j_1, j_2, \ldots, j_n\}$ with a fixed integer processing time $wcet_i$ on a single, non-preemptive machine. Additional constraints are: Each job j_i has either exactly one successor j_k, written as $j_i \prec j_k$, or no successor. A job may be annotated with a deadline d_i and a release time time r_i. Since the generated schedule is expected to be executed cyclically, a maximum schedule length $\mathcal{D}$ is defined. It is enforced by the root element Ω with $wcet_\Omega = 0, d_\Omega = \mathcal{D}$ and $r_\Omega = \mathcal{D}$. If $\emptyset \prec j$ set $\Omega \prec j$, thus transforming the structure of $\mathcal{J} \cap \Omega$ into a single in-tree. If no values for r_i or d_i are given, 0 and $\mathcal{D}$ are assigned by default. The objective is minimizing the number of tardy jobs ΣU_j. We are only interested in finding a schedule that has no tardy jobs at all. In the traditional notation of scheduling theory we can express our problem as $1|r_j, in-tree|\Sigma U_j$.

III. Related Work

Chretienne [2] minimizes the makespan of in-tree task sets with stochastic processing times in a distributed memory environment under the assumption that the processing times are greater than or equal to the data communication times between the jobs. Liu [3] applies heuristics in a branch-and-bound approach to minimize maximum lateness under general precedence constraints and release dates. We use his BLOCK heuristic on our problem set. For minimizing the maximum tardiness, research based on Schrage's Heuristic has proven to be very efficient. Hall and Shmoys [4] improved on Pott's original work which employed Schrage's Heuristic.

IV. Linear Programming Solutions

This paper is using the mixed integer programming (MIP) formulations given by Keha et al. [5]. They compare four different approaches: Start time and completion time variables (F1), time index variables (F2), linear ordering variables (F3), and positional and assignment variables (F4). Because we are looking for feasible instead of optimal solutions we only adapted F1 and F4. These generate the highest amount of feasible solutions in a given amount of time [5]. Due to space constraints, we refer the reader to the original paper for the MIP formulations.

V. Heuristic Approaches

Heuristics pose an attractive alternative for finding feasible solutions. We compare earliest release time time first (ERF), earliest deadline first (EDF), the BLOCK heuristic [3] and Potts' algorithm [4].

Earliest release time time first: The ERF heuristic simply sorts all jobs in $\mathcal{J} \cup \Omega$ by ascending order of their release time time. If two jobs have the same release time, then their deadlines are used as a tiebreaker. Precedence constraints should be mapped to modified release times by applying: $\forall j_i, j_k \epsilon \mathcal{J} \cup \Omega : j_i \prec j_k \implies r'_k = max\{r_k, r_i + wcet_i\}$

Earliest deadline first: In contrast to the ERF heuristic we cannot simply sort jobs by their deadline because that could lead to violation of precedence constraints. EDF instead chooses the leaf of the tree with the earliest deadline j_d and the leaf with the earliest release time j_r. If j_r can be scheduled

before j_d without conflict, i.e. $r_r + wcet_r \leq r_d$, schedule j_r first, otherwise j_d. The scheduled leaf is then removed from the tree and if all predecessors of a job have been scheduled that job is then added to the set of available leaves. Effective deadlines for each job should be computed beforehand as: $\forall j_i, j_k \epsilon \mathcal{J} \cup \Omega : j_i \prec j_k \implies d'_i = min\{d_i, d_k - wcet_k\}$

BLOCK heuristic: The BLOCK heuristic [3] first sets up a schedule by ERF and divides it into blocks of jobs which are executed with no time delay between them. If the schedule is invalid, the heuristic adjusts the block by scheduling jobs with higher deadline towards the end of the block.

Potts' algorithm: Potts' algorithm [4] is setup by sorting tasks by their deadlines in topological order. If the schedule is invalid, it analyzes the critical sequences of the schedule, i.e. blocks of jobs where at least one job has an invalid deadline (= critical job j_{crit}). An interference job j_{int} is a job within a critical sequence that is scheduled before j_{crit} but has a higher deadline than d_{crit}. $r_{int} < r_{crit}$ must hold, since j_{int} would otherwise not have been scheduled this early. Interference jobs are thus scheduled after their corresponding critical jobs to reduce the amount of tardy jobs.

VI. Evaluation

The evaluation was performed on over 40 000 randomly generated scheduling problems with 16 to 128 jobs. The amount of deadline and release time constraints per workflow is uniformly distributed between one and $|\mathcal{J}|$. Similarly, the tightness factor ($\Sigma wcet_i/\mathcal{D}$) was uniformly distributed between one (maximum tightness) and zero. Since the goal of the evaluation is to evaluate the efficiency of a scheduling algorithm, i.e. for how many of the feasible workflows it can find a valid schedule within a given CPU time budget, infeasible workflows have to be discarded first. As no optimal algorithm for the non-preemptive case exists we employ the preemptive least laxity first algorithm (LLF) to filter out definitely unschedulable workflows. LLF has been shown to be optimal for the preemptive single machine case [6]. Any workflow that is not schedulable in the preemptive case will remain so in the non-preemptive case. LLF discarded about half of the generated workflows as unsolvable, the remaining 20 503 jobs where then scheduled with each of the methods described in Sections IV and V and with simulated annealing (SA). For only two of these jobs no solution could be found with any of the employed methods, meaning that we have 20 501 jobs which comprise our set of feasible test cases.

We use Gurobi[1] in version 5.5 for solving the LP formulations for feasibility, not optimality. The simulated annealing portion is based on the Opt4J[2] framework using a simple linear temperature function and ran for 250 000 iterations. The test machines are equipped with an Intel Q6700 CPU at 2.66GHz and 8 gigabytes of RAM.

The time budget for each algorithm in Figure 1 was 10 seconds. The MIP approach is outperformed by all the heuristics, ruling them out for productive use. In the case

[1]http://www.gurobi.com/

[2]http://opt4j.sourceforge.net/

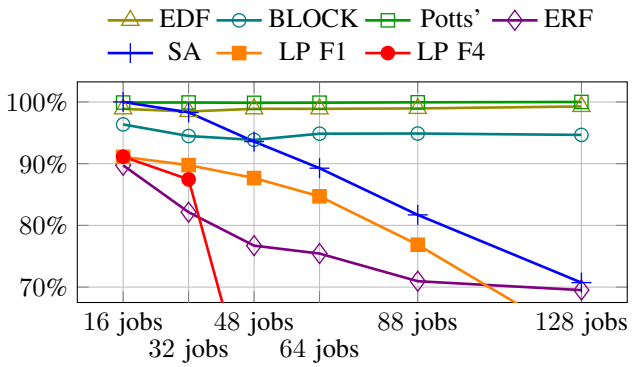


Fig. 1: Percent of test cases solvable by each method

of LP F4 the efficiency sinks to 10% for 128 jobs. ERF naturally also struggles to achieve a high efficiency because it does not take deadlines into account. However, it is more stable in the number of jobs. Simulated annealing performs well for small problem instances but looses efficiency for larger workflows because the state space it has to explore is growing exponentially. EDF, BLOCK and Potts' algorithm all perform well with over 95% efficiency on average. Potts' algorithm consistently performs at near 100% efficiency, there were only 18 out of 20 501 test cases where it did not find a solution, which equals an overall efficiency of over 99.9%. The combination of Potts' Algorithm, BLOCK and EDF finds a solution for all but 8 test cases. It is worth noting that ERF, EDF, BLOCK and Potts' algorithm all run in polynomial time and, on average, need less than one millisecond to generate a solution.

VII. Conclusion and Future Work

This paper shows that heuristics, especially Potts' algorithm, are able to solve our scheduling problem within a CPU time budget of 10s in nearly 100% of our test cases. This makes them a good fit for the scheduling component in our real-time SOA. MIP formulations are not suited to finding feasible solutions fast, but could be used to find optimal solutions in a later, separate step. Future work will implement the other building blocks, such as deriving local constraints from end-to-end deadlines and evaluate the concept in simulation as well as in real world testbeds.

References

[1] L. De Souza, P. Spiess, D. Guinard, M. Köhler, S. Karnouskos, and D. Savio, "SOCRADES: A Web Service Based Shop Floor Integration Infrastructure," in *The Internet of Things*, ser. LNCS. Springer Berlin Heidelberg, 2008, vol. 4952.

[2] P. Chretienne, "A polynomial algorithm to optimally schedule tasks on a virtual distributed system under tree-like precedence constraints," *European Journal of Operational Research*, vol. 43, 1989.

[3] Z. Liu, "Single machine scheduling to minimize maximum lateness subject to release dates and precedence constraints," *Computers & Operations Research*, vol. 37, no. 9, 2010.

[4] L. A. Hall and D. B. Shmoys, "Jackson's rule for single-machine scheduling: making a good heuristic better," *Mathematics of Operations Research*, vol. 17, no. 1, 1992.

[5] J. W. F. Ahmet B. Keha, Ketan Khowala, "Mixed integer programming formulations for single machine scheduling problems," *Computers & Industrial Engineering*, vol. 56, 2009.

[6] A. K. Mok and M. L. Dertouzos, "Multiprocessor scheduling in a hard real-time environment," in *Seventh Texas Conf. Comput. Syst.*, 1978.

Mathematical Considerations of Linear Real-Time Logic Verification

Stefan Andrei
Department of Computer Science
Lamar University
Beaumont, TX 77710, U.S.A.
stefan.andrei@lamar.edu

Albert M. K. Cheng
Department of Computer Science
University of Houston
Houston, TX 77004, U.S.A.
cheng@cs.uh.edu

Mozahid Haque
Department of Mathematics
University of Houston
Houston, TX 77004, U.S.A.
mahaque@uh.edu

***Abstract*—In [1], Cheng and Andrei introduced an extension of Real-Time Logic verification to Linear Real-Time Logic, henceforth LRTL. LRTL allows for dependencies beyond two events, and [1] introduces a reduction to a nonlinear matrix form for the specification and verification problem. In [3], we provided an alternative method using QR decomposition through Householder reflections. In this paper, we will introduce a block matrix decomposition through permutation matrices and clustering to exploit any sparseness when possible. Then, a simple mechanism of generating ratios between elements will be extended to generate a well-defined initial vector, if not the solution. This initial vector will be implemented in a select few iterative methods to resolve the null space when possible.**

Keywords—Algorithms, Linear Real-Time Logic, LRTL, Real-Time Logic, RTL, and Verification.

I. INTRODUCTION

In RTL, a set of safety specifications and safety assertions, SP and SA, respectively, dictate the constraints of a given system. Logically, we wish to show the satisfiability of SP → SA indirectly by showing the unsatisfiability of the logically equivalent form ~(SP Λ ~SA), where ~SA means the negation of SA. Path RTL is a proper subclass of RTL defined in [4] and gives us a form for timing constraints in the form of inequalities that are expressed as disjunction of inequalities. For example, the expression $\forall i\ @(e1, i) - @(e2, i) \leq k$ has the meaning that the difference between the time of the i-th occurrence of event $e1$ and the time of the i-th occurrence of event $e2$ is at most k with i and k both positive integers.

With all the constraints given in inequality form, we are then able to express this in conjunctive normal form resulting in a set of inequalities which can be further compressed to the simplified matrix form from linear algebra $\mathbf{Ax} \leq \mathbf{b}$. This gives us Linear Real-Time Logic, LRTL. For further reading, a full and better articulated derivation can be found in [1] and [2].

In [1] and [2], the solution method involved finding the null space using the well-known Gaussian elimination method with partial pivoting. In [3], the method introduced was QR decomposition with Householder matrices for increased stability at the price of an increased flop count. In this paper, we continue seeking further efficient methods with regards to stability and flop count by introducing the notion of a partial cluster and ratio of elements.

*Supported in part by the US National Science Foundation under Awards No. 0720856, No. 1219082, and No. 1154606.

II. PRELIMINARIES

A. Partial Cluster

Following the language of [5], we introduce the notion of a partial cluster in matrices. Often with sparse matrices (contains lots of zero elements in a matrix), we are able to cluster the non-zero elements to further simplify and reduce the work of any given solution method with permutation matrices. Recall, multiplication by a permutation matrix on the right of some matrix **A** will rearrange the columns of a matrix while multiplication on the left will rearrange the rows. In [5], the clustering resulted in a matrix of this form:

$$\begin{bmatrix} * & * & * & 0 & 0 & 0 \\ * & * & * & 0 & 0 & 0 \\ * & * & * & 0 & 0 & 0 \\ 0 & 0 & 0 & * & * & * \\ 0 & 0 & 0 & * & * & * \\ 0 & 0 & 0 & * & * & * \end{bmatrix}.$$

In a similar fashion, the notion of a partial cluster is of this form:

$$\begin{bmatrix} * & * & * & * & 0 & 0 & 0 \\ * & * & * & * & 0 & 0 & 0 \\ * & * & * & * & 0 & 0 & 0 \\ * & * & * & * & 0 & 0 & 0 \\ 0 & 0 & 0 & * & * & * & * \\ 0 & 0 & 0 & * & * & * & * \\ 0 & 0 & 0 & * & * & * & * \\ 0 & 0 & 0 & * & * & * & * \end{bmatrix}.$$

The difference between the two is a shared column of potentially non-zero entries - a shared row is similar. Note that the size of these matrices is not at issue; the structure of the sparseness is the purpose of the toy matrices.

B. Ratio of Elements

Consider the following:

$$\begin{bmatrix} a & b & 0 \\ 0 & c & d \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}.$$

We wish to find $\boldsymbol{x}_1$, $\boldsymbol{x}_2$, and $\boldsymbol{x}_3$ such that $\mathbf{Ax} = \mathbf{0}$. It is simple to generate the solutions simply from the given matrix elements.

First, check whether elements *c* and *d* are of the same sign or not. If they are, simply let $\boldsymbol{x}_2 = (-d/c)\boldsymbol{x}_3$, negate for opposite signs. This establishes a ratio between $\boldsymbol{x}_2$ and $\boldsymbol{x}_3$. Similarly, check sign of *a* and *b*, and if same, easily generate the other ratio $\boldsymbol{x}_1 = (-b/a)\,\boldsymbol{x}_2'$. Then, just find the least common multiple (lcm) of the coefficients of the common column (or common row if that form) to equate the common shared element.

Notice the special nature of the toy example. It is indeed in the form of a partial cluster! The extension of this ratio schema becomes apparent when applying to matrices of larger sizes in partial cluster form. Also, note that we may have several partial clustering in a matrix. It need not be two clusters. Only the sparse structure is important after using the permutation matrices. The adjustments of the **x** vector correspond to column exchanges, i.e., multiplication of matrix **A** by a permutation matrix on the right.

This has another implication related back to a block decomposition of a given matrix. We need not solve for the null space of a matrix as a whole. Instead, we can simply solve for the null space of the clusters and equate ratios of the shared column (or row) element using a lcm method. This would reduce the flop count by applying any of the matrix methods introduced in [1], [2], or [3] to the submatrices.

C. Givens Rotation QR Decomposition

Givens rotation involves rotation matrices with calculation of c and s values based on elements of the column vector we wish to zero. The advantage is its selective zeroing method.

D. Additional Principles

Some additional concepts we cannot fully explicate here will just be briefly mentioned here; details can be found in [6]. A Jordan normal form consists of a decomposition consisting of its generalized eigenvalues in Jordan block form. The Krylov subspace is the span of a subspace multiplied by powers of a matrix **A**. For matrix **A**, ind(**A**) is the smallest nonnegative integer k such that $\text{rank}(\mathbf{A}^k) = \text{rank}(\mathbf{A}^{k+1})$.

III. Solution Method Proposals

A. Iterative Procedures

We will examine a few iterative methods when the option is available. Most of the methods require ind(**A**) = 1 for the largest Jordan block and use a Krylov subspace method. However, the vast majority of iterative methods revolve around nonsingular matrices whereas our matrices are singular, i.e., non-invertible matrices. The possibility of a null basis with more than one vector also hampers the generation of good iterative procedures since we start with one initial vector when there may be many.

B. Reduction to Proposed Method

Our proposed method will implement the two notions given in *II*. For a given matrix $\mathbf{A}^{i \times j}$ and the problem $\mathbf{Ax} = \mathbf{0}$,

1. (Sparsity) When inputting elements of matrix **A**, keep a count of nonzero elements of **A** corresponding to element $\boldsymbol{x}_j$ and a count of corresponding rows in **A** with only two or three unknowns with the rest of the elements in the rows being zero.
2. (Cluster) Rearrange the matrix to resemble the partial cluster form with permutation matrices and update elements of **x** accordingly, resulting in submatrices from block decomposition. The count from step 1 is preserved.
3. (Ratios) Resolve the unknowns for any of the rows with two unknowns from step 1 in each submatrix in a fashion similar to the toy example in *II.B.* if there are enough two element rows w.r.t. unknowns and equations.
4. (Other general methods) For the unresolved elements of the submatrices, we may apply a mix of selective Gaussian elimination and a combination of 3. for particularly sparse submatrices, the QR decomposition through Householder matrices for particularly dense (less zeroes) submatrices from [3], a QR decomposition through Givens rotation for particularly sparse submatrices, or an iterative method as a last resort.
5. (LCM) Recombine the resolved $\boldsymbol{x}_i$ values corresponding to the various submatrices by finding the least common multiple of the common $\boldsymbol{x}_i$'s of the shared columns (or rows) due to clustering.

There are other stop procedures involving a positivity condition along with other notions due to various subtleties in the reduction process of [1]. Sometimes, step 3 may be unnecessary. Givens rotation method may be implemented in general with sparse matrices with advantages being less flops and parallelization as opposed to Householder method [7]. All these enhancements are being explored and will be reported together with experimental results in a future paper.

References

[1] Andrei, S. and Cheng, A. M. K. 2007. *Verifying Linear Real-Time Logic Specifications*. 28th IEEE International Real-Time Systems Symposium (RTSS), Tuscon, AZ, 2007.

[2] Andrei, S. and Cheng, A. M. K. 2009. *Efficient Verification and Optimization of Real-Time Logic Specified Systems*. IEEE Transactions on Computers, Vol. 58, No. 12, pp. 1640-1653.

[3] Andrei, S., Cheng, A. M. K., and Haque, M. 2013. *Optimizing the Linear Real-Time Logic Verifier*. 19th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS) WiP Session, Philadelphia, PA.

[4] Jahanian, F. and Mok, A.K.. *A Graph-theoretic Approach for Timing Analysis and Its Implementation*. IEEE Transactions on Computers, Vol. C-36, No. 8, pp. 961-975, August 1987.

[5] Andrei, S. and Cheng, A. M. K. *Decomposition-based Verification of Linear Real-Time Systems Specifications*, 2nd Workshop on Compositional Theory and Technology for Real-Time Embedded Systems (CRTS), Washington, D.C., USA (Co-located with IEEE RTSS 2009), December 1, 2009.

[6] Simoncini, V. 1996. *An Iterative Procedure for Computing the Null Basis*. Iterative Methods in Linear Algebra, II, IMACS Series in Computational and Applied Mathematics, Vol. 3, pp. 413-423.

[7] Golub, G. H. and Van Loan, C. F. *Matrix Computations*, 3rd ed. Johns Hopkins University Press, Baltimore, MD, 1996.

Towards a communication-aware mapping of software components in multi-core embedded real-time systems

Hamid Reza Faragardi, Kristian Sandström, Björn Lisper, Thomas Nolte
MRTC/Mälardalen University, P.O. Box 883, SE-721 23 Västerås, Sweden
{hamid.faragardi, bjorn.lisper, thomas.nolte}@mdh.se, kristian.sandstrom@se.abb.com

***Abstract*—Several studies have been done to investigate efficient mapping of software components onto a distributed embedded real-time system. However, most of these studies do not consider a distributed system where each node can be a multi-core processor. Mapping becomes even more complicated when precedence constraints between software components are taken into account. In this paper, the ongoing work on addressing the challenge of achieving a resource efficient and predictable mapping of software components onto a distributed multi-core system is discussed. Our goal is to minimize inter-components' communication cost besides meeting timing and precedence constraints between the components. The solution should solve two sub-problems; assign a set of software components onto a task set, and assign the created task set to (the cores of) the processing nodes. In this paper, we discuss some challenges and potential solutions to design a communication-efficient system.**

***Keywords*-mapping; multi-core; distributed real-time systems.**

I. Introduction

Nowadays, component based software engineering is more commonly used in design and implementation of modern embedded systems. In component based design, each Software Component (SC) is corresponding to one particular function of the system, and the application is fulfilled via composition and corporation among different SCs. This design significantly simplifies both development and maintenance of systems, and it typically also increases predictability. However, mapping and scheduling of component-based software applications is more challenging when it comes to achieving a resource efficient solution compared to the traditional resource optimization engineering of monolithic software. The reason for this is inherent in the complex nature of modern embedded hardware, which often is both heterogeneous and parallel in nature. When running a component-based application on a distributed system in which each node could be a multi-core processor, then this problem becomes even more complicated.

In this work-in-progress paper, we address the problem of achieving a resource efficient design of a component-based software application being mapped onto a heterogeneous distributed embedded hardware. We concentrate on minimizing communication cost as a prominent performance factor in such systems because, increasing the inter-components' communication cost does not only aggravate the overall performance but it can also reduce schedulability of the system. The problem can be divided into two sub-problems; mapping and scheduling. In the mapping process a set of components are assigned to a task set, and the generated task set should then be scheduled on a distributed multi-core system. The scheduling phase should deal with the assignment of tasks to the cores of the processing nodes. The scheduling of tasks is partially resolved off-line, in the sense that one task will always execute on the same core, i.e., tasks are not migrating across cores.

Although a balanced allocation of the task set onto the cores (i.e., load balancing) usually results in an acceptable performance, such a solution may increase the overall communication overhead when the components (and therefore, also tasks) are able to communicate with each other. When designing a communication-aware solution the consolidation approach, where the tasks are consolidated on a minimum number of nodes, could be more effective because the cost of transferring data over the network is dramatically higher than the communication cost on the same core. Generally, the cost of communication between a pair of SCs depends on their location that will determine resource related costs, such as communication delays on the network, memory latencies to communicate between the cores of a same node, interference patterns etc. On the other hand, the proposed solution must be able to, in addition to minimizing communication cost, satisfy all system and application constraints such as deadlines, precedence and memory limitation.

II. Related Work

Most previous works related to this paper can be categorized into two groups. In the first group, allocation of hard real-time tasks in distributed systems in the presence of precedence constraints has been investigated [1]. However, often only considering single core processors or assuming that the task set is created in advance and thus the mapping of software components to a task set is not covered. Some works also cover both the mapping and scheduling phases for single-core systems such as [2], [3]. In the second group, multi-core scheduling is taken into account however, distributed multi-core systems has not been considered [4], [5].

To the best of our knowledge, the mapping of software components onto a distributed multi-core system with the goal of achieving a resource efficient solution by minimizing the overall communication cost has not yet been carefully studied. This is the focus of this paper.

III. Problem Description

We consider a set of strictly periodic SCs which could be concurrently executed on different cores. This set of SCs should be allocated among a set of heterogeneous processing nodes of a distributed system. The nodes are connected together through a CAN bus. Because of its low cost and high reliability, the CAN bus is a quite popular solution for embedded systems [1]. Each node is assumed to be a homogenous multi-core system in which all the cores have

the same processing power. In addition, a set of independent transactions represent some end to end latency requirements between a sequence of SCs. In fact, each transaction is represented by a directed acyclic graph in which each node is a SC and the links show data dependency between the SCs. This dependency implies that to complete the successor, fresh data generated by the predecessor should have arrived before the release time of the successor. The communication between the SCs can be performed based on non-blocking read/write semantics. Nevertheless, triggering is not included in the scope of this paper, in the sense that a successor can start its execution even with obsolete data however, in this case we should wait for the next invocation of the successor in order to fulfill the transaction mission. Each transaction has a relative end to end deadline before which the execution of the transaction must finish, i.e., all SCs of the transaction must finish their execution before this deadline. The transaction deadline is corresponding to the mission of that transaction. For example, the mission could be the braking system in a car where the whole process should be done before an end to end deadline.

We assume that the transactions are sporadic in nature, and thereby in the worst-case they follow a periodic behavior with a known minimum inter-arrival time. However, the inter-arrival time between two subsequent instances of a transaction could be larger than its deadline. A conservative modeling is to assume that the period of the transaction is equal to its relative deadline. We also assume that a pair of SCs may transfer data in between each other, while there is not any precedence for their execution order. To represent this an undirected graph called Software Component Interaction Graph (SCIG) is employed. Each node of the SCIG represents a SC and the arcs between them show data communication. Furthermore, there is a label on each arc that indicates the rate of data that should be transferred between the SCs.

The communication cost to transfer data between a pair of SCs depends both on the mapping of SCs onto the cores and the final task set structure. It can be computed as follows. If the SCs are located *i)* in the same task, then we consider cost α for transferring each unit of data, or *ii)* in different tasks on the same core, then the cost of communication is β, or *iii)* in different cores of the same processor, then θ is the cost of communication, or *iv)* in different processing nodes, then γ is the cost of communication. We assume that α, β , θ, and γ are sorted in ascending order in the sense that γ is the largest value while α is the smallest.

The main goals of this paper are 1) meeting all end to end deadlines of the transactions, 2) minimizing the overall SCs' communication cost.

IV. Work-in-Progress

In our current work-in-progress we are applying different heuristic techniques to find good mapping strategies for the components of the applications. In real software applications, there are additional restrictions which impose extra complexity to the problem, and thus the problem can not be solved by the pure consolidation approach. Three of these constraints are: A) Due to heterogeneous hardware and the OS architecture of the processing nodes a SC could be restricted to run only on a subset of nodes, e.g., due to availability of specific I/O and resources. B) Some SCs should not run on the same core, e.g., because of fault tolerance considerations. C) Some SCs should run on the same node, e.g., for efficiency reasons inherent in the SCs implementation and nature. In the following, a potential candidate solution for the problem is intuitively presented following three steps:

1: In the first step, a simple task set is created where each task contains one transaction. Subsequently, an efficient heuristic is applied (for example, genetic algorithm) to find an communication-aware allocation of the created task set onto the cores.
2: In the second step we try to be more intelligent in the way that the task set generated in the first step is refined by merging some of the SCs located on the same core that communicate with each other onto the same task. To avoid extra utilization overhead only the tasks with the same period are allowed to merge into one task.
3: In the last step, the tasks with different periods communicating with each other which are allocated onto the same core are allowed to merge. Merging the tasks with different periods may increase the CPU utilization as the lower period of the two tasks has to be selected for the new task. Therefore, it forms a tradeoff between the task's utilization and the overall communication cost. To handle this case we translate the communication to CPU utilization and then the tasks will be merged if the utilization of the new task set on that core decreases.

The interesting point about the above steps is that step one independently can be considered as a solution for the problem, and step one and two can be considered as another solution and finally all three steps together can be considered as the third solution. We intend to develop these three solutions and compare them against each other. It is expected that the third solution which is also a more complete solution outperforms the other solutions. However, it may have some drawbacks, e.g., inherent in the complexity of combining the solutions. Another interesting idea is to get a feed-back from the second and third step to re-execute the first step with a better efficiency. For this purpose, the evolutionary algorithm mentioned in the first step invokes the second or third solution before evaluation of each individual, and thereby, the feedback results generated by the second or third solution are used to guide the search towards a global optimum.

References

[1] Y. Yang, "Software synthesis for distributed embedded systems," Ph.D. dissertation, Ph. D. thesis/Yang Yang, 2012.
[2] Q. Zhu, H. Zeng, W. Zheng, M. Di Natale, and A. Sangiovanni-Vincentelli, "Optimization of task allocation and priority assignment in hard real-time distributed systems," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 11, no. 4, p. 85.
[3] E. Wozniak, A. Mehiaoui, C. Mraidha, S. Tucci-Piergiovanni, and S. Gerard, "An optimization approach for the synthesis of autosar architectures," in *IEEE ETFA*, 2013, pp. 1–10.
[4] H. R. Faragardi, B. Lisper, and T. Nolte, "Towards a communication-efficient mapping of autosar runnables on multi-cores," in *IEEE ETFA*, 2013, pp. 1–5.
[5] A. Saifullah, J. Li, K. Agrawal, C. Lu, and C. Gill, "Multi-core real-time scheduling for generalized parallel task models," *Real-Time Systems*, vol. 49, no. 4, pp. 404–435, 2013.

List of Authors