GMCB: An Industrial Benchmark for use in Real-Time Mixed-Criticality Networks-on-Chip

James Harbin, Tom Fleming, Leandro Soares Indrusiak, Alan Burns

Real-Time Systems Group, Department of Computer Science, University of York, UK
{james.harbin,tdf506,leandro.indrusiak,alan.burns}@york.ac.uk

***Abstract*—This paper specifies and describes GMCB (Generic Mixed-Criticality Benchmark), a benchmark industrial application model for testing real-time mixed-criticality multicore systems incorporating a network-on-chip (NoC). Task execution information such as periods, latencies and criticality levels is specified. The communication patterns between the tasks are defined, incorporating message sizes transmitted between the source and destination nodes. Cycle-accurate NoC simulation is used to evaluate the performance of the GMCB model, evaluating the communication latencies encountered under various task mappings. The behaviour of the application under a recent mixed-criticality NoC protocol is considered.**

I. Introduction

The trend towards multi-core and many-core systems is encouraging further integration of embedded and real-time systems, and in particular the installation of multiple applications or system functions onto a single hardware platform. Complex single applications may have component tasks of distinct criticality levels sharing resources, such as processing cores or links or buffers of a communication interconnect. It is important to ensure that the goals of efficient resource utilisation do not conflict with the requirement to deliver performance guarantees for each level of criticality in the system. Furthermore, multi-criticality systems can be implemented upon a platform featuring sophisticated interconnects such as a network-on-chip (NoC) [1], which requires consideration of the communication and interference patterns that are in progress.

In order to evaluate the performance of mixed-criticality systems and protocols, it is important to have a range of consistent and standardised benchmarks which present a predefined loading for the mixed criticality system. This paper specifies GMCB (Generic Mixed Criticality Benchmark), a mixed-criticality benchmark developed from an industrial case study, providing task execution times, task criticality levels, communication patterns, and message sizes. Example results are provided for the cycle-accurate simulation of this application upon a multi-core system incorporating a NoC, demonstrating the communication latency performance of the application. These results are further extended by considering the behaviour of a mixed-criticality protocol when executing the application with two different task mappings.

II. Related Work

The benchmarks that are available for embedded and real-time systems typically provide a large number of tasks and detailed structure of the internal code. However, they do not include any criticality information for the various tasks, or a designation of the tasks into distinct criticality levels. To the best of our knowledge, this specfication of GMCB provides the first mixed-criticality benchmark derived from a real industrial case study suitable for schedulability analysis and NoC communication latency simulation.

Benchmarking in general has been an important part of performance analysis for parallel and distributed computing, since it provides consistent test loadings that can be used to evaluate a novel architecture, protocol or configuration. The SPLASH2 benchmark suite [2] consists of a number of parallel programs from various domains such as scientific and graphics computing. The benchmark characteristics are explored together with their computation to communication ratios, and their applicability to various case studies. The Mälardalen benchmarks [3] provide a set of example applications with call graphs for assessing worst-case execution time of network-on-chip applications. TACLEbench [4] provides a constantly updating and developing set of real-time benchmarks suitable for timing analysis. These include parallel and sequential benchmarks with complete C source code, and the code sizes involved range from 21 to 3985 lines. TACLEbench includes a number of parallel benchmarks. The Papabench benchmark [5] from the Paparazzi project is one benchmark included in the set of parallel benchmarks within TACLEbench, and models a fly-by-wire UAV and associated autopilot. This benchmark allows the WCET of the individual tasks to be computed from application code. However, the two distinct functions (fly-by-wire and autopilot) included are assumed to be running on different CPUs, not permitting relative performance testing on single-core vs. multi-core configurations.

In the context of NoC research, the MCSL benchmarks [6] provide a number of benchmark NoC applications including a H264 decoder, Fourier transforms, and Reed-Solomon encoders/decoders. These applications define execution times as well as communicating flow message sizes, sources and destinations, and provide statistical traffic as well as recorded application traffic patterns. However, these benchmarks do not contain any criticality information or priority levels, which

would be useful for priority-preemptive NoCs and for evaluating mixed-criticality systems in the context of task priorities.

III. Model Definition

This section defines the criticality model, giving an overview of mixed criticality fundamentals affecting the benchmark. Then the structure of the benchmark is specified by defining, firstly, its tasks and their computation intervals in the two distinct criticality modes, and secondly, the communicating flows, giving the source and destination tasks, and the data size of each communication.

A. Criticality Model

Two distinct criticality levels, high (HI) and low (LO) criticality are assumed for the benchmark. The model of criticality used assumes that in any case in which a LO-crit transmission may cause interference upon a HI-crit flow that could cause the HI-crit flow to miss a deadline, the interfering LO-crit transmissions are temporarily or permanently restricted from transmission across the NoC. This situation is accomplished by means of a criticality mode change. A criticality change is triggered by a HI-crit task transmitting data more frequently than permitted under its LO-crit parameters, or the transmission of data packets by a HI-crit task beyond the size specified in the LO-crit parameters. This model and its implementation upon the NoC is specified fully in two of our earlier protocols, defined in [7] and [8]. These models assume that the NoC architecture is priority preemptive, so as to provide reduced latency to the highest priority in the case of contention.

Depending on the NoC protocols, criticality changes triggered on one arbiter may be propagated to others. Following a criticality change, LO-crit packets may be prohibited from transmission from the affected arbiters (the model as used in WPMC [7]), or priorities may be changed so the LO-crit flows can only be serviced with a priority below any HI-crit packets (the model used in WPMC-FLOOD [8]). Although these protocols are used in evaluation with a mixed-criticality application, the specific structure of the benchmark is independent of a particular criticality change protocol.

B. Task Structure

The application consists of 20 communicating tasks, including a special system management task. Given that the application is a mixed criticality application, it is assumed that tasks may have an increased worst-case execution time in HI-crit mode. Therefore, task execution latency is specified in both two alternative modes; $C(LO)$ for LO-crit mode and $C(HI)$ for HI-crit mode. Task execution parameters as supplied by the industrial user are as defined in Table I, with $C(LO)$ assumed at 80% of the $C(HI)$ value. The majority of the tasks are HI-crit, and only four tasks are LO-crit. This is itself interesting in that the ad-hoc application case study assumed in our earlier work [7] had only a small number of HI-crit transmissions relative to total data communication. A further interesting aspect is that the LO-crit flows transmit and receive the largest amounts of data across the NoC. The system management task is assumed to communicate with every other task in the system, once per period. This communication consists of the system management task transmitting a single data packet to all other tasks defined in the system. The system management task therefore represents a highly connected component, which is capable of transmitting criticality changes widely, as explored in Section IV-A.

NAME	PERIOD (ms)	C(LO) (ms)	C(HI) (ms)	CRIT
I/O_1	20	3.6	4.5	HI
I/O_2	20	0.8	1	HI
I/O_3	20	0.8	1	HI
I/O_4	40	4.8	6	HI
I/O_5	40	4.8	6	HI
I/O_6	40	4.8	6	HI
I/O_7	20	1.6	2	HI
I/O_8	40	0.4	0.5	HI
I/O_9	100	0.2	0.25	HI
P_1	20	1.2	1.5	HI
P_2	20	0.4	0.5	HI
P_3	20	0.8	1	HI
P_4	20	3.2	4	HI
P_5	20	1.6	2	HI
P_6	20	2.4	3	HI
SYS	40	0.2	0.25	HI
P_LO_1	20	6	-	LO
P_L0_2	20	3	-	LO
P_L0_3	80	20	-	LO
IO_LO_1	40	17	-	LO

TABLE I: The definition of the tasks comprising the GMCB benchmark

C. Communication Flows Structure

The model as presented assumes that all tasks are released to begin their computation upon the start of each period. Obviously, if multiple tasks are mapped onto a single processor, then the lowest priority tasks must wait for access to the resource. Upon completion of their computation, packets are sent to the network interface for transmission to their peers in the task graph. It would be possible to apply release jitters to the tasks, however, the assumption of simultaneous release upon the start of the period is in order to produce the maximum contention on the NoC, and therefore illustrate the worst-case latencies. The task graph and its flow relationships are shown in Figure 1. Note that transmissions which are marked as HI-crit do not necessarily cause a criticality mode change. Criticality mode changes only occur if these transmissions are larger than the indicated size, or if they occur more frequently than once per task period.

D. Task Memory Requirements

In addition to the communicating flows contained in Figure 1, application tasks also communicate with memory in order to load and save relevant data. A table showing the memory communication sizes of each flow is presented in Table II. This table specifies the size of the relevant code and total data storage sizes for each task. It is evident that the memory requirements for each individual task are highly variable, and that there exists one particular task IO_LO_1 which requires significantly higher amounts of memory than the other tasks. However, although total memory requirements are presented

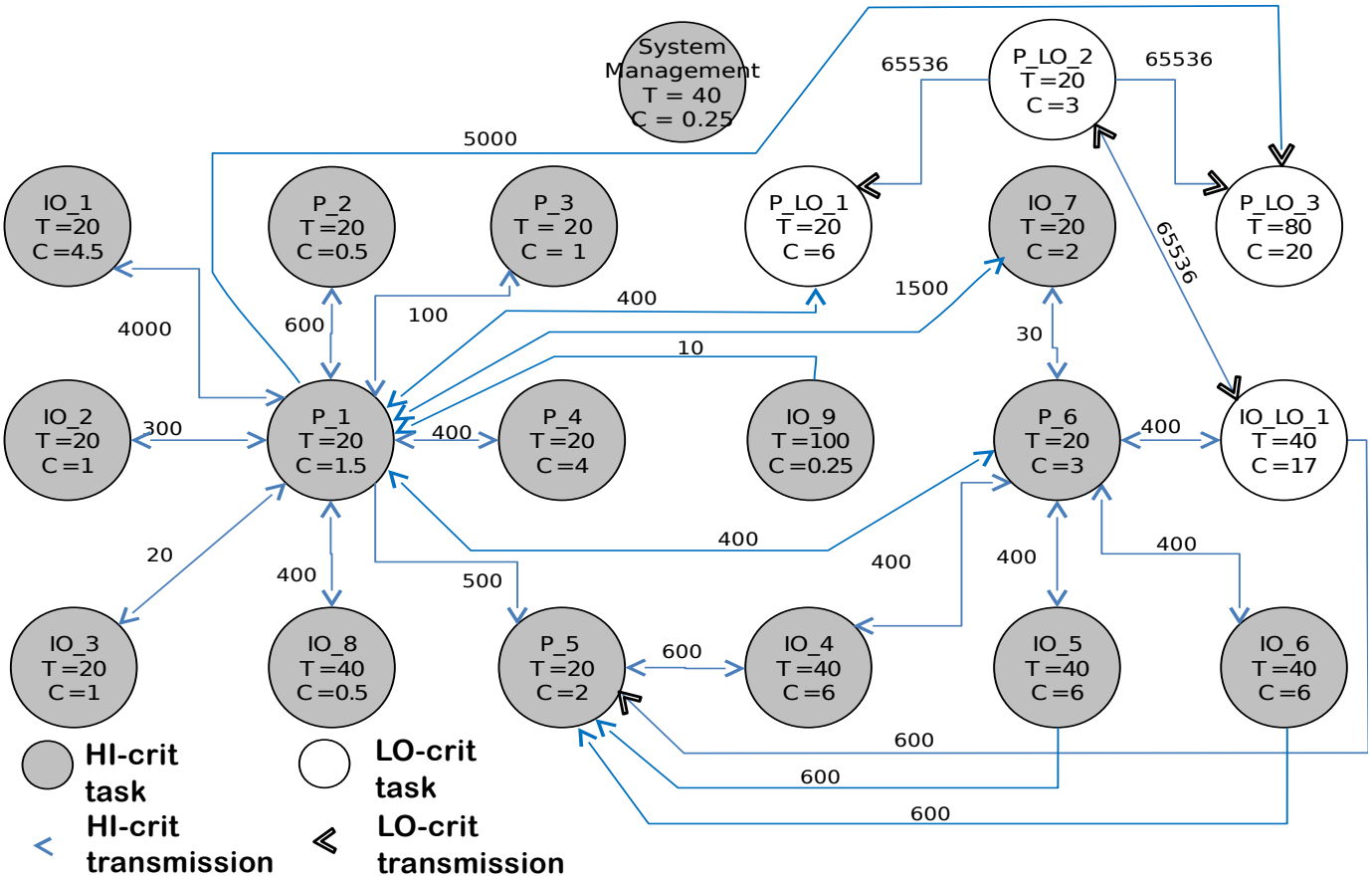


Fig. 1: The structure of communicating flows within the GMCB application. Annotations to the arcs indicate data transmission sizes in bytes in LO-crit mode. Computation latencies C are in the LO-crit mode, $C(LO)$

Task Name	CodeSize	DataSize	CodeSize(MB)	DataSize(MB)
IO_1	327208	196830	0.31	0.19
IO_2	95572	8466	0.09	0.01
IO_3	85092	17922	0.08	0.02
IO_4	976880	3118806	0.93	2.97
IO_5	976880	3118806	0.93	2.97
IO_6	976880	3118806	0.93	2.97
IO_7	512192	382078	0.49	0.36
IO_8	88992	11202	0.08	0.01
IO_9	81096	9170	0.08	0.01
P_1	1436608	925310	1.37	0.88
P_2	102340	65420	0.10	0.06
P_3	67320	104338	0.06	0.10
P_4	231056	85968	0.22	0.08
P_5	66504	4226	0.06	0.00
P_6	177224	101786	0.17	0.10
P_LO_1	495180	8394340	0.47	8.01
P_LO_2	140224	467082	0.13	0.45
P_LO_3	6369620	1565474	6.07	1.49
IO_LO_1	949800	112916262	0.91	107.69
Sys. Man.	104296	1339930	0.10	1.28

TABLE II: The code and data memory requirements for the benchmark tasks

for each task, the actual low-level memory access patterns (recorded traces giving access patterns and timings) are not currently available from the industrial user. Therefore, memory requirements and the impact of memory read/write requests upon NoC latency are not considered further in this paper.

IV. Example Results

The section presents preliminary tests of communications latency for the GMCB benchmark. **Detailed results for reference in independent implementations will be made available at [9].**

A. Criticality Change Scenarios

In order to evaluate the behaviour of the benchmark following criticality change events, three different criticality change scenarios, C1, C2 and C3, are specified. In each scenario, the size of data transmitted from the relevant flow is increased to just beyond the sizes indicated in Figure 1, forcing a criticality change. The interesting aspect of modelling these different scenarios is that criticality changes may be propagated to different regions of the NoC. Some regions of the NoC may remain in the LO-crit mode, and therefore may respond differently during arbitration and permit LO-crit flows to pass.

- Scenario C1 - Transmission by P_1 $\rightarrow$ IO_1 becomes HI-crit
- Scenario C2 - Transmission by P_6 $\rightarrow$ IO_4 becomes HI-crit
- Scenario C3 - Transmission by System Management task to IO_1 becomes HI-crit

Task src.	Task dst.	Priority	Task src.	Task dst.	Priority
IO_1	P_1	1	P_6	IO_7	32
IO_2	P_1	2	P_6	IO_5	33
IO_3	P_1	3	P_6	IO_6	34
IO_4	P_5	4	P_6	IO_4	35
IO_4	P_6	5	P_6	IO_LO_1	36
IO_5	P_5	6	SYS	IO_1	37
IO_5	P_6	7	SYS	IO_2	38
IO_6	P_5	8	SYS	IO_3	39
IO_6	P_6	9	SYS	IO_4	40
IO_7	P_1	10	SYS	IO_5	41
IO_7	P_6	11	SYS	IO_6	42
IO_8	P_1	12	SYS	IO_7	43
IO_9	P_1	13	SYS	IO_8	44
P_1	IO_1	14	SYS	IO_9	45
P_1	IO_2	15	SYS	P_1	46
P_1	IO_3	16	SYS	P_2	47
P_1	IO_7	17	SYS	P_3	48
P_1	IO_8	18	SYS	P_4	49
P_1	P_2	19	SYS	P_5	50
P_1	P_3	20	SYS	P_6	51
P_1	P_4	21	SYS	P_LO_1	52
P_1	P_5	22	SYS	P_LO_2	53
P_1	P_6	23	SYS	P_LO_3	54
P_1	P_LO_1	24	SYS	IO_LO_1	55
P_1	P_LO_3	25	P_LO_1	P_1	56
P_2	P_1	26	P_LO_2	P_LO_1	57
P_3	P_1	27	P_LO_2	P_LO_3	58
P_4	P_1	28	P_LO_2	IO_LO_1	59
P_5	IO_4	29	P_LO_3	P_LO_2	60
P_5	P_1	30	P_LO_3	P_1	61
P_6	P_1	31	IO_LO_1	P_5	62

TABLE III: Priorities assigned for data flows between sources and destinations

B. NoC Evaluation Scenario

In our evaluation of the network-on-chip scenario, we utilise a cycle-accurate priority-preemptive NoC architecture [10] with virtual channels, extended to include multi-criticality arbitration as specified in [7]. The tasks are mapped onto the NoC platforms of various 2D mesh sizes using the task mappings defined in Table IV. As an example the frequency of the NoC is taken as 100 MHz, and 32 bit flits are used. In this model, it is assumed that the tasks compute for their specified time and then make a simultaneous transmission to all their associated peers as defined within the task graph. Multiple tasks may be mapped onto a single CPU. Priority preemption is used to interrupt a task if a higher priority task executes, with the task priorities defined in descending order of Table I. Within this section, the communications latency across the NoC is recorded.

In order to implement the scenario given upon a priority-preemptive NoC, flow priorities and task mappings have to be defined. When contention occurs between two flows, priority preemption permits the highest priority flow to receive arbitration and therefore access the output port. (This may be modified by the criticality protocol as described in Section III-A). The priorities are as defined in Table III.

C. NoC Communication Latency in LO-crit mode

In this model, all tasks are using their LO-crit computation and transmission parameters. Figure 2 shows the communication latency across a 3x3 NoC, illustrating the latencies of flows of individual priority levels. The results illustrate that

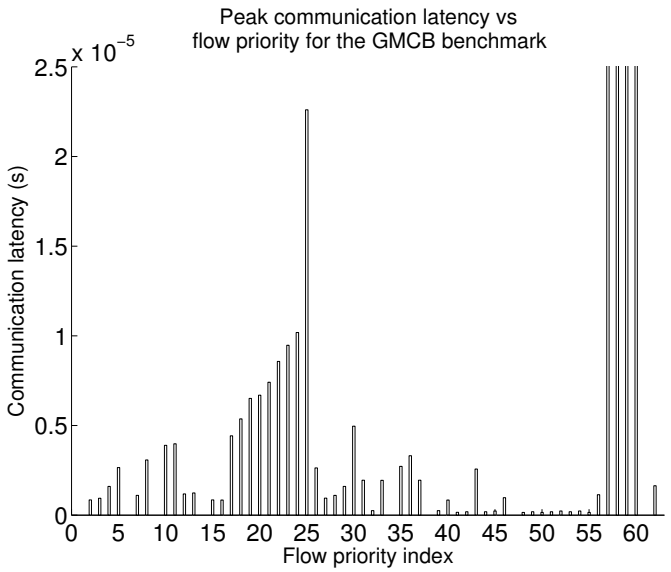


Fig. 2: Communication latencies for flows in GMCB with an example task mapping (mapped onto 3x3 NoC)

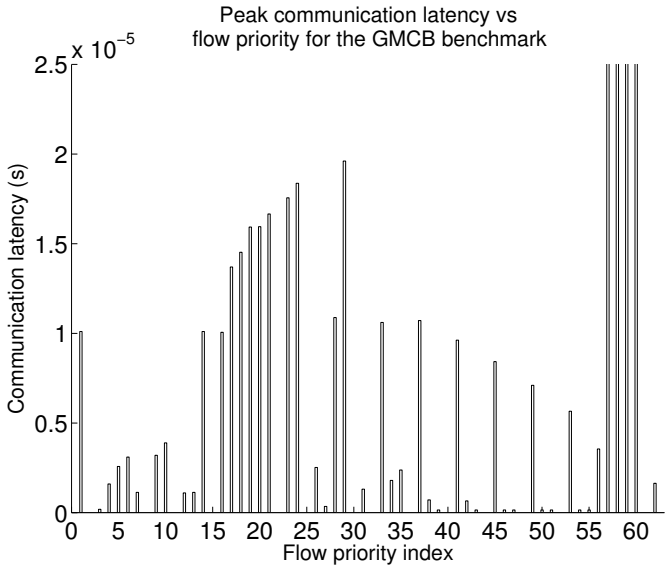


Fig. 3: Communication latencies for flows in GMCB with an example task mapping (mapped onto 2x2 Noc)

the communication latencies of the majority of flows are fairly close to their basic latency, however, the range of flows with priorities from 16 to 24 exhibit increasing latencies, since due to interference the lower priority flows must wait for the higher priorities before they can receive arbitration. In Figure 2 and throughout the remaining result graphs, the latencies of the flows from 57 to 61 are not shown directly, since they are significantly above the range of the rest of the latency values. Flow 59 has a latency of $3.2 \times 10^{-4}s$, while other LO-crit flows transmitting 65536 bytes have a latency around $1.6 \times 10^{-4}s$. The results in Figure 3 show the equivalent LO-crit case with the benchmark application mapped onto a 2x2 NoC. The results in Figure 3 show an increased latency for many flows due to the increased contention and blocking

during transmissions upon the NoC, with a larger number of flows receiving an increased contention, particularly for flows with priorities above 30. It is interesting that the flow with priority 25 has a higher peak latency in the 3x3 case than 2x2, since the round-robin 2x2 mapping places both endpoints on the same core resulting in a zero latency since there is no requirement for communication across the NoC.

D. NoC Communication Latency with Criticality Changes

This section considers how criticality changes affect the behaviour of the NoC under different candidate task mappings, presenting results on the criticality changes and communication latencies of the communicating tasks. The criticality changes are frequently propagated across the NoC, according to the protocol requirements. For the WPMC protocol [7], criticality changes are propagated along the routes traversed by a packet. Any HI-crit transmissions that cross an arbiter inherit that HI-crit mode change, and carry it with them throughout the network. The scenario used to enforce criticality changes consists of sending larger packets for a particular flow than permitted under its LO-crit parameters. In this case, scenario C1 is used, although the behaviour of the system is the same in any case due to the inherited propagation of HI-crit changes.

Figure 5a shows the effects of criticality changes during mapping onto a 2x2 NoC. The criticality status following execution is shown via screenshot from a simulation GUI. Arbiters which have changed to HI-crit mode are indicated in red. Following the first transmission of excess data from P_1 to IO_1, a criticality change is required. According to the criticality change rules in WPMC [7], all arbiters soon change to the HI-crit mode, since transmissions to all other tasks from the system management task SYS carry an inherited criticality change throughout the NoC. Given the arbitration rules in WPMC, this prevents the LO-crit transmissions from reaching their destinations. The dense packing of the HI-crit and LO-crit tasks together upon the 2x2 NoC ensures that all LO-crit tasks cannot make a transmission following the criticality change. This is illustrated in Figure 4 in which an absence of the latency bars for priority levels above 56 indicates that these LO-crit flows did not complete transmission to the destination successfully, and therefore received an undefined latency.

Figure 5b shows the effects of triggering criticality changes while mapping onto a 4x4 NoC. In this case, the mapping provided partitions the tasks so as to map the LO-crit tasks to the cores along the top row of the NoC. Since these tasks have the majority of their communication with each other in the LO-crit mode, it is possible for some of the arbiters in the top row of the NoC to remain in the LO-crit mode, since another HI-crit transmission never carries an inherited criticality change to these arbiters. Arbiters which have changed to HI-crit mode are indicated in red. There are two phases of criticality change. In the first phase, the inherited criticality change from P_1 is propagated through the cores which carry HI-crit flows, and to the top-left arbiter. In the second phase, the criticality change reaches the top-right arbiter. However, two arbiters remain in the LO-crit mode, and the latency results in Figure 6 show that

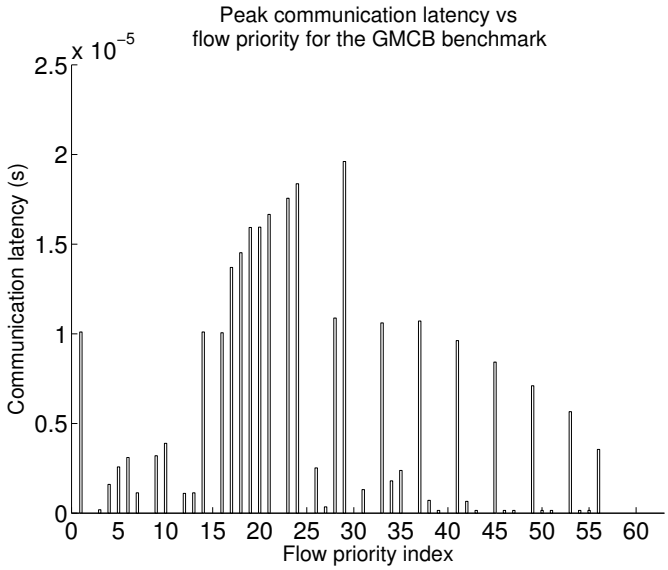
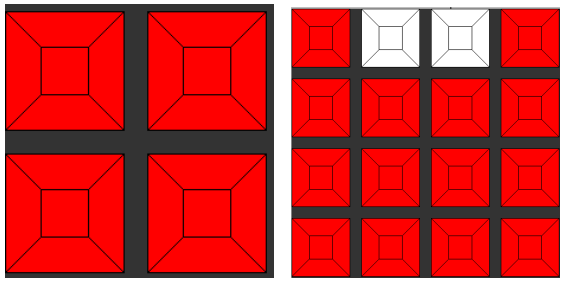


Fig. 4: Communication latencies for flows in GMCB with an example task mapping in 2x2 NoC with C1 criticality change



(a) 2x2 NoC (b) 4x4 NoC

Fig. 5: GUI screenshots for 2x2 and 4x4 NoCs showing criticality changes

some flows with priorities 57-60 still complete transmission and receive a defined latency. This indicates that the 4x4 NoC with this separated mapping can still provide some service to these flows, isolating the impact of criticality change under the WPMC protocol.

V. Further Work

This benchmark presented has defined task computation and computation parameters and the total memory consumption of each task. However, it does not assess the impact of the memory hierarchy, caching and the exact patterns of memory access times. A naive assumption, for example assuming that tasks transfer a fixed portion of their memory footprint across the NoC every period, would likely produce unrepresentative results, given the wide variation in memory access patterns exhibited in Table II, Section III-D. Specific trace timings for this model remain to be obtained from actual industrial use, and would permit accurate evaluation incorporating the dynamic behaviour of memory access requests. The benchmark

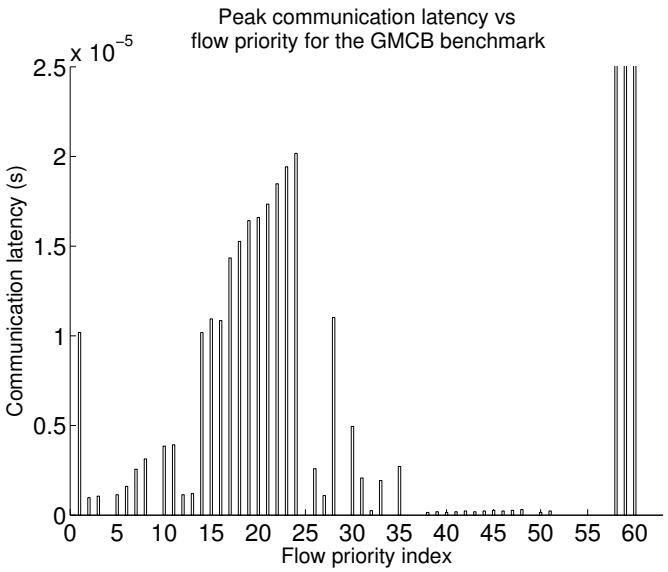


Fig. 6: Communication latencies for flows in GMCB with an example task mapping in 4x4 NoC with C1 criticality change

Task	M2x2	M3x3	4x4 (C1 crit. change)
IO_1	0,0	0,0	0,0
IO_2	0,1	0,1	0,1
IO_3	1,0	0,2	0,2
IO_4	1,1	1,0	1,0
IO_5	0,0	1,1	1,1
IO_6	0,1	1,2	1,2
IO_7	1,0	2,0	2,0
IO_8	1,1	2,1	2,1
IO_9	0,0	2,2	2,2
P_1	0,1	0,0	3,0
P_2	1,0	0,1	3,1
P_3	1,1	0,2	3,2
P_4	0,0	1,0	0,0
P_5	0,1	1,1	1,0
P_6	1,0	1,2	1,2
P_LO_1	1,1	2,0	0,3
P_LO_2	0,0	2,1	1,3
P_LO_3	0,1	2,2	2,3
IO_LO_1	1,0	0,0	3,3
SYS	1,1	0,1	0,0

TABLE IV: Task mappings used in the various NoC sizes for the evaluation, showing the assignment of the different tasks to processing cores

will be updated online at [9] as further information and results become available.

It is easily possible to tune the benchmark load provided by adjusting the frequency of packet release for each task. At the moment the benchmark assumes that each application node performs its computation and then makes a transmission simultaneously to its task peers. In order to provide additional load, it is possible to increase the data sizes of any flow, either by injecting additional packets or increasing packet size to increasing the number of flits that constitute a single packet.

Although this benchmark scenario model accomodates criticality changes resulting from application transmissions (specifically the need to transmit more data than possible under LO-crit parameters) it is possible that in real situations criticality changes could be application-determined. For example, a criticality change could be triggered from an application-specific fault model or other event model external to data transmission across the NoC platform. The impact of this, and the integration of this with NoC data processing, will need to be considered. It is also important to consider how the application-specific issues of how a mixed-criticality system can recover and return to LO-crit mode following a criticality change, and whether criticality changes are frequent and handled automatically by the platform or require application support in order to recover. An architecture or protocol allowing criticality mode changes to be reversed is currently under development.

VI. Conclusion

This paper has described and specified a test mixed-criticality application suitable for use as a benchmark for mixed-criticality systems implemented upon a network-on-chip (NoC). The benchmark has been defined including task periods, execution times in different criticality modes, and communication data sizes. Example results have been presented, illustrating the communication latencies generated for task mappings onto NoCs of different sizes. The issues involved in the production of realistic benchmarks, and the extension of the benchmark to enhance its realism for mixed-criticality scenario evaluation have been considered.

Acknowledgements

Financial support for this work was provided by the EPSRC, under project 'MCC' (EP/K011626/1).

References

[1] R. Marculescu, U. Y. Ogras, L.-S. Peh, N. E. Jerger, and Y. Hoskote, "Outstanding research problems in NoC design: System, microarchitecture, and circuit perspectives," *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, vol. 28, no. 1, pp. 3 –21, Jan. 2009.

[2] S. Woo, M. Ohara, E. Torrie, J. Singh, and A. Gupta, "The splash-2 programs: characterization and methodological considerations," in *Computer Architecture, 1995. Proceedings., 22nd Annual International Symposium on*, June 1995, pp. 24–36.

[3] J. Gustafsson, A. Betts, A. Ermedahl, and B. Lisper, "The Mälardalen WCET benchmarks – past, present and future," B. Lisper, Ed. Brussels, Belgium: OCG, Jul. 2010, pp. 137–147.

[4] "Taclebench benchmark suite," http://www.tacle.knossosnet.gr/activities/taclebench, accessed: 2015-03-29.

[5] F. Nemer, H. Cassé, P. Sainrat, J. P. Bahsoun, and M. De Michiel, "Papabench: a free real-time benchmark," *WCET*, vol. 4, 2006.

[6] W. Liu, J. Xu, X. Wu, Y. Ye, X. Wang, W. Zhang, M. Nikdast, and Z. Wang, "A NoC traffic suite based on real applications," in *ISVLSI 2011: IEEE Com. Soc. Annual Symp.*, 2011, pp. 66–71.

[7] A. Burns, J. Harbin, and L. Indrusiak, "A wormhole noc protocol for mixed criticality systems," in *Real-Time Systems Symposium (RTSS), 2014 IEEE*, Dec 2014, pp. 184–195.

[8] L. Indrusiak, A. Burns, and J. Harbin, "Average and worst-case latency improvements in mixed-criticality wormhole networks-on-chip," in *Euromicro Technical Committee on Real-Time Systems (ECRTS) (accepted)*, Jul 2015.

[9] "Gmcb benchmark definition," https://sites.google.com/a/york.ac.uk/gmcb/, accessed: 2015-06-11.

[10] L. S. Indrusiak and O. M. dos Santos, "Fast and accurate transaction-level model of a wormhole network-on-chip with priority preemptive virtual channel arbitration," in *DATE 2011: Design, Automation Test in Europe Conf.*, Mar. 2011, pp. 1–6.