Extended Analysis With Reduced Pessimism For Systems With Limited Paralellism

K. Bletsas and N. C. Audsley
Dept. of Computer Science, University of York, UK.
[bletsas, neil]@cs.york.ac.uk

Abstract

Under limited parallelism, processes competing for a single processor may issue at any time operations on remote co-processors, during which the processor is not idled but granted to other ready processes instead. We reduce the pessimism in existing Worst-case Response Time (WCRT) analysis for such systems by examining temporal patterns of local/remote execution. We extend to multi-CPU variants of the model and offer a WCRT-based feasibility test for Symmetric Multiprocessor (SMP) systems.

1. Introduction

In *limited parallel* systems [2, 3], processes competing for a single processor may at any time issue operations on remote co-processors. While a process awaits completion of a remote operation, another ready process may execute on the processor. Such parallelism is commonplace but often overlooked. The established scheduling literature (summarised in [10]), by not considering co-processors, is pessimistic if applied. Analysis for the limited parallel model [2, 3] removes some pessimism. That "macroscopic" analysis considered the execution requirements of a process locally (on the processor) and remotely (on co-processors) as worst-case scalars, ignoring when local or remote execution actually occured inside the process activation. This more exact analysis considers "microscopic" temporal patterns of local/remote execution.

Application-specific co-processors may appear as distinct devices on the system bus (Figure 1(a)) or as partitions of a single device operating in parallel (Figure 1(b)). They are often implemented in reconfigurable logic using Field Programamble

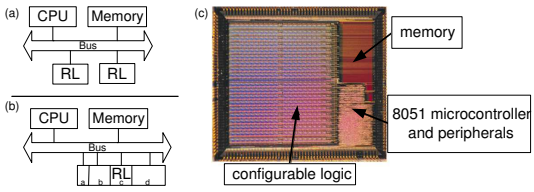

Figure 1. Embedded Architectures

Gate Arrays (FPGAs) [1, 13]. Increasingly often, reconfigurable co-processors share a package with the processor core (as in the Triscend E5 [12] (Figure 1(c))) or even the core is implemented in reconfigurable logic.

We term a *block* what others call a *task* [11, 6] or *process* and call a *process* what is often termed a *linear transaction* [8] of processes/tasks. This is consistent with codesign practice where (local/remote) code blocks forming a transaction are extracted from a single process. Local/remote blocks are also termed *LBs*/*gaps* respectively.

2. Background

Redefining the WCET of a process C as the sum of WCETs for local and remote execution (X, G resp.), the WCRT equations familiar from uniprocessor analysis [7] become:

$$R_i = (X_i + G_i) + \sum_{j \in hp(i)} \lceil R_i/T_j \rceil (X_j + G_j)$$

As seen, remote execution (G) is treated as exerting interference, while only local execution (X) does. The analysis in [2] rectifies this, the respective equation being:

$$R_i = X_i + G_i + \sum_{j \in hp(i)} \lceil (R_i + G_j)/T_j \rceil X_j \quad (1)$$

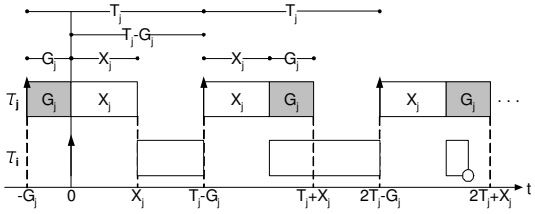

Figure 2. Worst-case Interference For The Original Limited Parallel Model

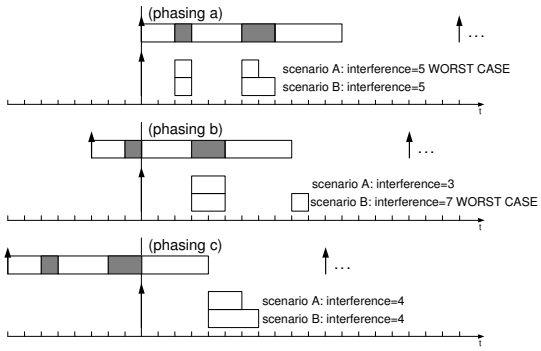

Figure 3. Candidate Phasings Tested

The term G_j expresses the worst-case scenario under that analysis, which assumes block positions may vary in subsequent activations of the same process (perhaps due to control flow). This "intra-activation jitter" is at most G_j. This worst case is depicted in Figure 2: τ_i is released at $t = 0$; this scenario maximizes interference exerted by any higher-priority process τ_j. Note that the worst-case scenario is not that of synchronous releases: the release of each higher-priority process τ_j precedes by G_j the release of the process considered.

We explain why using a uniprocessor model instead and treating remote execution as blocking on an external event would be problematic: If the respective blocking term is treated as exerting interference, this is pessimistic. If not, the variability in the periodicity of local execution is not accounted for, thus the analysis may be optimistic [5].

Some pessimism persists in the original limited parallel analysis [2, 3] because the worst-case scenario for that model treats any process as suffering an initial preemption of at least the lengths of all higher-priority LBs summed. However there are cases where at most a fraction of the LBs of an instance of a higher-priority process may be interfering. This observation motivates our approach. The intuition behind it is described by this example:

In a two-process set $\{\tau_i,\tau_j\}$, the high-priority process τ_j consists of interleaved local/remote blocks; τ_i only executes locally. The relative release offsets of τ_j, τ_i are unknown. The worst case according to [2, 3] would have τ_i be preempted for the sum of length of the LBs of τ_j before ever getting to execute. This is pessimistic because, whatever its release offset, τ_i cannot suffer **contiguous** preemption longer than the longest LB of τ_j.

3. Block Sequence Examined

For the ensuing analysis we require invocations of the same process exhibit the same sequence of (local or remote) execution blocks. Block execution times have upper and lower bounds (derived by WCET/BCET analysis). The *execution pattern* of a process is an interleaved sequence of LBs and gaps e.g. $xgxgx$ or $gxgx$ (x for LBs, g for gaps).

Consider a process with local-only execution. It is trivial to show that in the worst-case scenario for interference suffered, the release of the process in consideration would have to be coincident with the start of LBs from all higher-priority processes. If the number of LBs in a process τ is given by $n(\tau)$ the question then is which one of the $n(\tau_j)$ blocks (or equivalently phasings) this is, for each interfering process τ_j.

Consider an example: In a two-process system, the high-priority process has an execution distribution pattern of $xgxgx$ with (fixed) respective block lengths of 2, 1, 3, 2, 4 and a period of 19 (yielding 7 time units between the termination and the next release of the process). To identify the worst-case phasing in terms of interference suffered by the low-priority process we examine all 3 candidate phasings for each of two scenarios: A) the low-priority process having an execution requirement of 2 or B) 3 respectively (see Figure 3). The example shows that, in the general case, which phasing yields the worst case depends on the execution requirement of the process suffering the interference. Moreover, for multiple higher-priority processes $n(\tau_j)$ phasings from each higher-priority process τ_j need to be considered in combination, which might not be tractable.

3.1. Notation and Transformations

The execution pattern of a process τ_j is represented as: $x_{j_1}g_{j_1}x_{j_2}g_{j_2}...x_{j_{n(\tau_j)}}g_{j_{n(\tau_j)}}$. X_{j_k}, $\hat{X}_{j_k}$ denote the maximum/minimum length of block

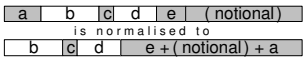

Figure 4. Distribution Normalisation

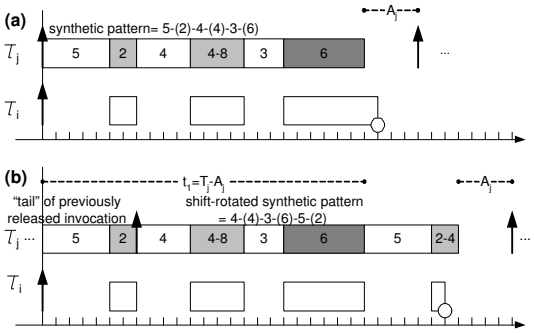

Figure 5. Jitter-like Effects

x_{j_k}. G_{j_k}, $\hat{G}_{j_k}$ denote the maximum/minimum length of gap g_{j_k}. With specific values assigned to an execution pattern we obtain the exact distribution for the respective process activation.

The above pattern starts with a LB and ends with a gap, thus remote and local blocks are equal in number. This is not always the case. For convenience, we append a notional gap to the end of the distribution, of length $N_j = T_j - C_j$. This corresponds to the minimum idle interval between successive invocations of the process if the latter were executing without competition from other processes for the processor. This transformation does not affect the schedulability of lower-priority processes as gaps exert no interference. As processes are periodic, we also "normalise" the distribution (by shift-rotating left if required) to start with a LB. Our approach covers the worst-case under all possible release phasings for the process set so this transformation is also valid. Adjacent blocks mapped both local or remote then merge. Then the number of LBs and gaps is the same, denoted $n(\tau_j)$ - this facilitates the analysis. An example of normalisation transformation is shown in Figure 4: letters denote block lengths; gaps appear shaded. The concept of a notional gap allows us to transform a given distribution by "shift-rotation" of its constituent blocks, an operation our approach depends on so as to find a bound on the worst case that covers all possible relative release phasings.

We use the following notation: the function $big\_x(\tau_j, m)$ gives the m-th of the LBs of the normalised distribution of τ_j, if these are ordered by decreasing maximum length. The function $small\_x(\tau_j, m)$ gives the length of m-th of the LBs of the normalised distribution of τ_j sorted by increasing minimum length. Similarly with functions $big\_g(\tau_j, m)$, $small\_g(\tau_j, m)$ for gaps.

3.2. Bounding The Worst Case

We introduce a tractable function that provides an upper bound for interference suffered from higher priority processes for all possible offsets and phasings. It provides tighter bounds on the exact worst case than the existing analysis [2, 3]. The main idea is to transform process distributions by reordering the constituent blocks so that there exists one phasing always providing worst-case interference on lower-priority processes regardless of their execution requirement. Worst-case interference under this *synthetic* distribution is shown to be an upper bound for the worst-case under the actual distribution.

Since processes are periodic, the sequence of activation of their constituent blocks repeats itself indefinitely. Thus LBs in a process distribution can be treated as standalone periodic processes with offset releases. In these terms, we can generalise the notion of a *release* to refer to blocks also (local or remote), rather than just processes: a block is said to be released when it is ready to execute. This depends on the completion time of the block preceding it, which may vary for subsequent invocations of the process. Thus block releases are not strictly periodic. Only the starting block of each process would be strictly periodic, like the process itself while the k-th block of some process τ_j is characterised by some jitter J_{j_k}.

In this context, for synchronous releases of τ_i, τ_j, bounds on interference can be derived:

$$I_i = \sum_{j \in hp(i)} \sum_{k=1}^{n(\tau_j)} \left\lceil \frac{R_i - O_{j_k} + J_{j_k}}{T_j} \right\rceil u(R_i - O_{j_k}) X_{j_k} \tag{2}$$

where $R_i = C_i + I_i$ and $u(t)=1 \forall t \geq 0$; else $u(t)=0$. The term $u(R_i - O_{j_k})$ describes offset releases that exhibit worst-case jitter. (Releases of the k-th block of τ_j at $t = O_{j_k}$, $T_j + O_{j_k} - J_{j_k}$, $2T_j + O_{j_k} - J_{j_k}$, ...) The inner "$\sum$" sums up the contributions from LBs of a given higher-priority process; the outer one sums up the contributions from all such processes. The equation is solved by forming a recurrence relationship, as in uniprocessor analysis [4]. Variables O_{j_k} reflect the earliest

time that the release of the k-th LB of process τ_j can be offset from the release of the process itself.

With this transformation the interference can be computed from given execution distributions and phasings. Yet the exact worst case is not identified. We derive that using this theorem:

Theorem 1: For a process τ_i suffering interference from a higher-priority process τ_j, an upper bound for the interference suffered by τ_i due to instances of τ_j released after or at the same time as τ_i is obtained if instead of the actual execution distribution of τ_j the following execution distribution is considered: $big\_x(\tau_j,1)small\_g(\tau_j,1)...big\_x(\tau_j,n(\tau_j))small\_g(\tau_j,n(\tau_j))$

Proof: Consider the actual execution pattern of τ_j, normalised (as described above) to start with local execution (without loss of generality) and augmented by the notional gap.

• Shifting right along the time axis (if necessary) the release of τ_i to coincide with a release of a LB of τ_j cannot decrease interference on τ_i.

• Then if all LBs acquire the respective maximum length, interference cannot decrease.

• If all gaps acquire their respective minimum lengths interference cannot decrease.

• Then if the first and the longest LB in each invocation of τ_j swap lengths, interference cannot decrease. Similarly with the second and the longest remaining LBs and so on, until we run out of LBs.

• Then if the first and the shortest gap exchange lengths in each invocation of τ_j, interference cannot decrease. Similarly then for the remaining gaps, as with LBs.

We obtain the execution distribution $big\_x(\tau_j,1)small\_g(\tau_j,1)...big\_x(\tau_j,n(\tau_j))small\_g(\tau_j,n(\tau_j))$ released at the same instant as τ_i. We term this the *synthetic worst-case execution distribution*.

Assume that another distribution yields greater interference. Transforming it as described cannot decrease the interference. But the product of the transformation would again be the synthetic worst-case distribution (which is a contradiction).

□

If a process suffers interference only from instances of higher priority processes released not earlier than its own release, it is trivial to show that interference is maximized under coincident releases and higher-priority process invocations characterised by the synthetic distribution. However, in the general case a process may also suffer interference from instances of higher-priority processes released earlier than it. Even if all invocations of a process maintain the same sequence of blocks, the variability in the release times of LBs (only the one coincident with the process release is strictly periodic) allows for scenarios that yield greater interference than coincident synthetic distributions. Observe this example (Figure 5):

A process τ_j is converted to its synthetic distribution, which is $5{-}(2){-}4{-}(4){-}3{-}(6)$. (In this notation the parentheses denote gaps.) Note that even though this synthetic distribution bears the notional gap at the end (of length $T_j{-}C_j$ which represents the minimum possible time interval between two activations of τ_j) there is a time interval where τ_j is idle, between two invocations (exclusive of that notional gap). In our example, this is because the gap lengths prescribed by the synthetic distribution are the respective minimum values, while for process execution time to reach a maximum (the worst-case) the respective maximum values are required. The upper bound for this idle interval is A_j. (We stress that A_j does not include the notional gap; the latter is treated, for the purposes of considering the interference exerted by τ_j as remote execution belonging to the synthetic distribution, even though in fact it is idle time). We can see that for our simple example $A_j = T_j - X_j - \hat{G}_j - N_j = T_j - X_j - \hat{G}_j - (T_j - X_j - G_j) = G_j - \hat{G}_j$.

Figure 5(a) shows the synchronous release scenario and the interference suffered by lower-priority process τ_i. In Figure 5(b) the synthetic distribution for τ_j is modified (by shift-rotating to the left) to $4{-}(4){-}3{-}(6){-}5{-}(2)$. Also the release of τ_j is shifted to the right so that the "tail" of a previous invocation of the same process gets to interfere with τ_i. Allowing the two invocations to run back-to-back (ie. with the minimum "spacing" that the final gap of length 2 allows between the last LB from the earlier invocation and the first LB of the next one) has the effect of reducing the interarrival time for the longest LB (that of length 5) to $T_j - A_j$ from T_j that it was in the previous scenario. Choosing a suitable relative offset for the release of τ_i (like the one shown in Figure 5(b)) observed interference increases (as displayed). The actual effect is equivalent to having synchronous releases of τ_j (following the synthetic distribution) but with a release jitter of A_j (which equals the variability in the response time of τ_j). Indeed $T_j - A_j$ is the minimum interarrival time that can be observed for releases of the same LB in successive invocations of the process. Revisiting Equation 2: it becomes evident that an upper bound on the individual "release jitters" J_{j_k}

of all LBs when the synthetic distribution holds (being the worst-case for interference on lower-priority processes exerted by a single instance of τ_j) is A_j. Otherwise, consecutive invocations of the same process τ_j would overlap in time, thus the system would not be schedulable. Thus we have identified an upper bound for interference suffered for all possible offsets and phasings. Extrapolating from Equation 2 we then obtain for the interference $I_{j\to i}$ exerted on τ_i by τ_j:

$$I_{j\to i} = \sum_{k=1}^{n(\tau_j)} \left\lceil \frac{R_i - O_{j_k} + A_j}{T_j} \right\rceil u(R_i - O_{j_k}) X_{j_k} \quad (3)$$

The indices correspond to the block order in the synthetic distribution, without loss of generality. The cumulative interference on τ_i is again $I_i = \sum_{j\in hp(i)} I_{j\to i}$. For the equation to be usable though, valid values for O_{j_k}, A_j are needed.

In our simple two-process example the offsets O_{j_k} for the higher priority process are derived simply by adding up the lengths of the preceding blocks in the synthetic distribution (maximal local blocks, minimal gaps). Also $A_j = G_j - \hat{G}_j$. We will show that even in the case of multiple higher-priority processes we can use

$$O_{j_k} = \sum_{m=1}^{k-1} (X_{j_k} + \hat{G}_{j_k}) \text{ and } A_j = G_j - \hat{G}_j$$

(k-indexes referring to block order as per the synthetic distribution) and the worst case will still be covered. We elaborate:

With multiple higher-priority processes, the variability in release time of a block may be due not only to variability in execution time of the preceding blocks but also due to preemption of preceding blocks by even-higher-priority processes. Thus the release jitter for a block of τ_j may appear to exceed $G_j - \hat{G}_j$ (say by ΔJ_j). However, without loss of generality, we can take the release in consideration to have occured earlier by ΔJ_j. This does not decrease (and in fact may increase) interference, so the worst-case analysis is not compromised. At the same time we are able to bound the variability in the release time of the block to $G_j - \hat{G}_j$. Meanwhile, acceptable values for O_{j_k} should be lower bounds on the release offset of the k-th block of τ_j relative to the process itself (given the synthetic distribution). Thus, with or without interference on τ_j we can use

$$O_{j_k} = \sum_{m=1}^{k-1} (X_{j_m} + \hat{G}_{j_m})$$

3.3. Response Time Equations

The WCRT equations now take the form

$$R_i = C_i + \sum_{j\in hp(i)} \sum_{k=1}^{n(\tau_j)} \left\lceil \frac{R_i - O_{j_k} + A_j}{T_j} \right\rceil u(R_i - O_{j_k}) X_{j_k} \quad (4)$$

where $O_{j_k} = \sum_{m=1}^{k-1} (X_{j_m} + \hat{G}_{j_m})$, $A_j = G_j - \hat{G}_j$. We see reduced pessimism over the original limited parallel model (Equation 1) due to i) the "fragmentation" of local execution in many distinct blocks (with gaps in between, during which lower-priority processes may execute) as opposed to one big block and ii) the reduction of the term acting as jitter: $A_j = G_j - \hat{G}_j \leq G_j$.

We note that the term acting as jitter is the same as in the Original Limited Parallel Model for the specific case that gaps always precede any local execution in the process activation [3]. Conversely we observe that for the inverse pattern (local execution preceding all remote execution) the term A_j may be omitted altogether, as the single contiguous LB is strictly periodic.

If one process is not amenable to the synthetic analysis (eg. its activations are not consistently characterised by the same sequence of blocks), the interference it exerts on other proceses can still be derived according to the worst case for the original analysis for the limited parallel model [2]. The complexity of the algorithm is discussed in [5].

3.4. Evaluation Of The Analysis

We evaluate the synthetic analysis against the existing one for an arbitrary process set. Process set attributes are presented in Table 1 whereas the actual and derived synthetic distributions are shown in Figure 6. WCRTs are compared in Table 1 with those derived under the original analysis [2, 3]. Improvements are input-dependent but may be noticeable, as shown.

4. Multiprocessor Extension

We introduce a multiprocessor extension to the limited parallel model. The general architecture depicted in Figure 1(b) is augmented by additional instruction-set processors (where there was

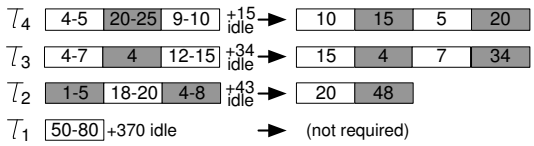

Figure 6. Actual → Synthetic

τ	T	X	G	C	A	R^{synth}	R^{orig}
τ_4	55	15	25	40	5	40	40
τ_3	60	22	4	26	0	41	56
τ_2	160	20	13	33	8	117	159
τ_1	450	80	0	80	0	402	414

Table 1. Process set and results

previously only one). All processors are identical, sharing the same address space. All system resources are global (including the memory and the co-processors) and symmetrically accessed. Processes are not attached to specific processors and can even migrate during execution. By having process state be global and symmetrically accessed we can assume negligible migration costs. Thus the N highest-priority processes among those competing for the processor are executing on any given instant. Our extension could be termed "Symmetric Multiprocessor (SMP) with co-processors".

Regarding the control of access to the shared system bus, we assume that is possible for memory accesses to be blocked on accessing the bus for very short intervals. However we assume that bus access arbitration is handled in a fair manner by the communication controllers of the processors. Its impact on latencies is an aspect of the system implementation, accounted for in WCET analysis (outside the scope of this paper).

4.1. Multiprocessor Analysis

Revisiting the previous analysis, the term "local block" would, in the new context, mean "block local to the processor array" or "software block". Intuitively the existence of $N > 1$ processors would increase availability and reduce interference suffered by processes. We proceed to quantify this effect and calculate bounds on WCRTs for such systems. For the ensuing analysis we again require that no resources be shared; this is waived later.

From release to termination of a process τ_i, an upper bound on the time (in clock ticks) spent executing locally to the processor array (ie. in software) by all higher-priority processes is:

$$W_i = \sum_{j \in hp(i)} \sum_{k=1}^{n(\tau_j)} \left\lceil \frac{R_i - O_{j_k} + J_{j_k}}{T_j} \right\rceil u(R_i - O_{j_k}) X_{j_k}$$

For $N=1$ processors, $W_i = I_i$. But with $N>1$ processors, a process competing for a processor to execute on can only be denied one (thus suffering interference) on the following condition:

There are N or more higher-priority processes also competing for a processor at the given instant (thus the N processors are granted to those with the N highest priorities among them).

An upper bound for the cumulative time that this condition may hold true for a process τ_i over a time window of length equal to its WCRT thus is an upper bound for the worst-case interference suffered by it. We show that such a bound is:

$$I_i = \begin{cases} 0 \text{ if } \tau_i \text{ among the } N \text{ highest-priority processes} \\ \lfloor \frac{W_i}{N} \rfloor \text{ otherwise} \end{cases}$$

Proof: If τ_i is among the N highest-priority processes then it may never be preempted by another process. If not: Assume that the worst-case interference for τ_i is $\breve{I}_i = I_i + a$, a being a positive integer. Then there could be at least $\breve{I}_i$ instants from release to termination of τ_i where all processors would be busy executing higher-priority processes. Thus the cumulative time spent executing higher-priority processes (in ticks) for the whole array would have been at least

$$N\breve{I}_i = N(I_i + a) = Na + N\lfloor \tfrac{W_i}{N} \rfloor > Na + N\tfrac{W_i}{N} - N$$
$$= N(a-1) + W_i \geq W_i \Rightarrow N\breve{I}_i > W_i$$

which is impossible (see the definition of W). □

Thus, bounds on interferences derived by the analysis are inversely proportional to the number of processors in the array for a given process set. A refined model is obtained by noting that the N highest-priority processes may not suffer any interference at all. Thus for a task τ_i belongs to that group of processes: $I_i = 0$.

Substituting the above expression for I_i into the equation $R_i = C_i + I_i$ and solving the recurence relation computes the WCRT for τ_i. Note that the analysis also applies for pure symmetric multiprocessor (SMP) systems, without any hardware co-processors, as a subcase. To the best of our knowledge this is the first WCRT-based feasibility test

for FP-scheduled SMP systems. Existing analyses (summarised in [10]) offer utilisation-based feasibility tests only, which do not cover systems where deadlines are less than the process periods.

WCRT is a decreasing function of process priority for a given relative priority ordering of remaining processes, ie. no anomalies exist.

4.2. Resource Management

The (uniprocessor) Priority Ceiling Protocol (PCP) [9] has been shown to ensure liveness, bound blocking times and prevent chained blocking for limited parallel systems if the coprocessors themselves are treated as shared resources [3]. Blocking terms are given by the same equations. We show that the same protocol covers the multiprocessor limited parallel model if lock requests originating from processes on different processors are serialised.

Rajkumar [9] modifies the PCP for multiple processor systems with/without shared memory. Local resources are accessed under uniprocessor PCP on each processor and global resources under the Distributed/Multiprocessor PCP respectively. In either case, global critical sections reside on dedicated *synchronization processors*, arbitrating access requests by processes on other (application) processors. With all global critical sections on a single synchronisation processor (not allowing parallel execution), their execution is serialised.

However, our model does not require the overhead of a separate synchronisation processor. All shared resources are global, accessible by any process on any processor, allowing arbitrary process migration (cf. [9] assumes processes statically assigned to processors). Uniprocessor PCP thus suffices, an arbiter is only needed for race condition avoidance with the execution of the global critical section occuring on the processor of the calling process (not the synchronisation processor). This greatly reduces blocking overheads: Many global critical sections can be executed in parallel, by processes on different processors, with only the access requests themselves serialised by an arbiter. The latter may be implemented in hardware [5], as a specialised co-processor, accessed as a shared location in the common address space. By implementing lock/unlock requests as atomic single instruction read/write operations, accesses to the arbiter are in turn serialised by the bus itself [5].

For priority assignment in the presence of blocking, an optimal algorithm is detailed in [5].

5. Conclusion

The analysis formulated provides tighter bounds on process WCRTs than the existing analysis for the the limited parallel model [2, 3]. Processes however must always be characterised by the same sequence of code blocks in their invocations so as to be able to be modelled in this way. Derived WCRTs are upper bounds for all possible process release offset combinations. We then extend to a multiprocessor model and its WCRT analysis. Our analysis is intended for use in a hardware/software codesign environment.

References

[1] Altera Corporation. *Altera Product Information : http://www.altera.com/products*, 2005.

[2] N. C. Audsley and K. Bletsas. Fixed Priority Timing Analysis of Real-Time Systems with Limited Parallelism. In *Proc. Euromicro Conference on Real-Time Systems*, 2004.

[3] N. C. Audsley and K. Bletsas. Realistic Analysis of Limited Parallel Software / Hardware Implementations. In *Proc. 10th Real Time Applications Symposium*, 2004.

[4] N. C. Audsley, A. Burns, M. F. Richardson, K. Tindell, and A. J. Wellings. Applying New Scheduling Theory to Static Priority Pre-emptive Scheduling. *Software Eng. J.*, 8(5):284–292, 1993.

[5] K. Bletsas. Extending the limited parallel model. Technical report, Department of Computer Science, University of York, 2005.

[6] J. C. P. Gutierrez, J. J. G. Garcia, and M. G. Harbour. On The Schedulability Analysis For Distributed Hard Real-Time Systems. In *Proc. Euromicro Conf. on Real-Time Systems*, 1997.

[7] C. L. Liu and J. W. Layland. Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment. *J. of the ACM*, 20(1):40–61, 1973.

[8] R. Pelizzoni and G. Lipari. Improved schedulability analysis of real-time transactions with earliest deadline scheduling. In *Proc. 11th Real Time Applications Symposium*, 2005.

[9] R. Rajkumar. *Synchronization In Real-Time Systems - A Priority Inheritance Approach*. Kluwer, 1991.

[10] L. Sha, T. Abdelzaher, K. E. Arzen, A. Cervin, T. Baker, A. Burns, G. Buttazzo, M. Caccamo, J. Lehoczky, and A. K. Mok. Real time scheduling theory: A historical perspective. *Journal of Real Time Systems*, 28(2/3):101–155, 2004.

[11] K. Tindell and J. Clark. Holistic Schedulability Analysis for Distributed Hard Real-Time Systems. *Euromicro J.*, Nov.-Dec. 1993.

[12] Triscend Corporation. *Triscend Products : http://www.triscend.com/products*, 2005.

[13] Xilinx Corporation. *Xilinx Product Information : http://www.xilinx.com/products*, 2005.