Extending The Limited Parallel Model

K. Bletsas

Dept. of Computer Science, University of York, UK.

bletsas@cs.york.ac.uk

Abstract

In the limited parallel computational model for real-time systems, processes competing for a single processor may issue at any time operations on remote co-processors. During such operations, the processor is not idled but may instead be granted to other ready processes. Such systems, though widespread, until recently had to be analysed with approaches intended for other models, at the cost of pessimism. Noting that remote operations exert no interference on locally executing processes, recent analysis for such systems provided tighter bounds on Worst-Case Response Times (WCRTs) than the application of uniprocessor analysis (but still pessimistic). Based on that work, this new approach further reduces the pessimism by examining temporal patterns of local/remote execution and extends to handle multi-CPU variants of the model under Fixed Priority Scheduling (FPS). The same approach can serve as a WCRT-based feasibility test for Symmetric Multiprocessor (SMP) systems.

1 Introduction

In *limited parallel* systems [3, 4], 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 be scheduled on the processor. By ignoring this, uniprocessor analysis [10] is too pessimistic if applied. Analysis considering the limited parallelism [3, 4], removes much of that pessimism. That analysis was "macroscopic": it only considered the execution requirements of a process locally (on the processor) and remotely (on co-processors) as worst-case scalar values, ignoring when local or remote execution actually occured inside the process activation. Instead, the analysis presented here, considers "microscopic effects" (the actual temporal pattern of local/remote execution of a process) so as to be more exact.

We focus on analysis tailored at systems with co-processors and limited parallelism because such systems, though very widespread in engineering practice, traditionally are analysed in ways that overlook their characteristics, hence the pessimism. Advances in manufacturing make *Field Programmable Gate Arrays* (FPGAs) [1, 17] ideal for the implementation of limited parallel systems.

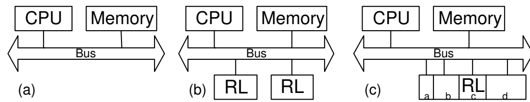

Figure 1: Architectures For Embedded Systems

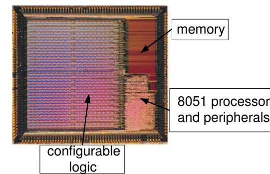

Figure 2: The Triscend E5

Limited parallel architectures are derived from conventional processor-memory architectures (Figure 1(a)) by the adddition of application-specific co-processors. The latter can be implemented as distinct devices on the system bus (Figure 1(b)) or as partitions of a single device that can operate in parallel with each other (Figure 1(c)). These co-processors are often implemented in reconfigurable logic in the form of commercially available FPGAs [1, 17].

The tendency is for greater integration and *Systems-On-a-Chip* (SOCs) so the co-processors often come in the same package as the processor core. Such an example is the Triscend E5 customizable microcontroller [16], depicted in Figure 2. Often, even the core itself is implemented in reconfigurable logic. Current transistor budgets in FPGAs enable such levels of integration. Still, as resource usage efficiency is important in embedded design, we aim to reduce the pessimism in the existing timing analysis for such systems. Then over-engineering will not be required for systems to qualify as meeting their timing constraints.

The established scheduling literature (summarised in [13]), addresses standard models of computation, where hardware co-processors are not considered. To the best of our knowledge, the analysis given in [3, 4] was the first to be directly applicable. Subsequently published analysis[11] is applicable to a similar model under Earliest Deadline Scheduling (EDF). The analysis in [3, 4] does provide tighter bounds on interference than the application of the standard uniprocessor approach [10, 9]. However, it fails to utilise information about the actual location of code blocks executing locally (termed *local blocks* or *LBs*) and remotely (*remote blocks* or *gaps*) in a process activation, hence some pessimism.

Note that what we term a (local or remote) *block* in the context of this paper corresponds to what others call a *task* [15, 8] or a *process* and what we call *process* is often refered to as a *linear transaction* [11] of processes/tasks. This is because our analysis is primarily intended for use in a codesign environment where the various code blocks (executing locally or remotely) that form a linear transaction are extracted from the code of a single process.

2 Background

The limited parallel model allows for multiple processes executing at a given instant: at most one locally, the rest remotely. A process switching to remote execution gives the next-highest-priority process among those runnable an opportunity to advance in computation when it would otherwise have remained preempted. This way, lower-priority processes suffer

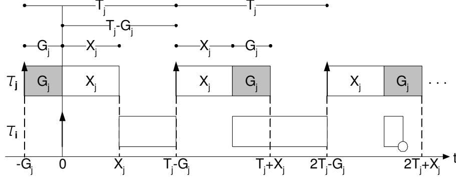

Figure 3: Worst-case Interference Under The Original Limited Parallel Analysis

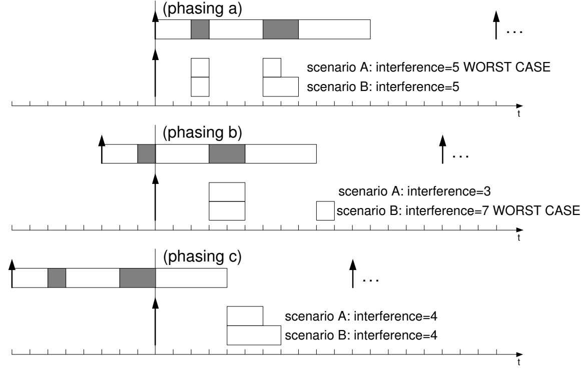

Figure 4: Examining Candidate Phasings

less interference than that suggested if uniprocessor analysis [10] were applied. To cope with these properties of the new model, the analysis introduced in [3, 4] redefines the worst-case execution time of a process C as the sum of the WCETs for local and remote execution – X, G respectively. The familar WCRT equations are then rewritten as:

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

This expression highlights the pessimism in applying that analysis, as both local (X) and remote execution (G) are treated as exerting interference, while only local execution (X) does. The existing analysis tailored at the limited parallel model [3] does however account for the fact that remote execution exerts null interference on lower-priority processes, thus achieving tighter bounds on interference suffered by them. The respective WCRT equations become:

$$R_i = X_i + G_i + \sum_{j \in hp(i)} \left\lceil \frac{R_i + G_j}{T_j} \right\rceil X_j \tag{1}$$

The term G_j inside the ceiling function expresses the worst-case scenario for interference under that analysis, which assumes that the position of code blocks of local and remote execution can vary for subsequent activations of the same process (perhaps due to different control flows). The maximum such "intra-activation jitter" is bound to G_j. This worst-case is depicted in Figure 3; τ_i is released at $t = 0$ and this is the scenario that maximizes interference exerted by any higher-priority process τ_j. Note that this worst case scenario does not occur for synchronous process releases as it requires the release of each higher-priority process τ_j to precede by G_j the release of the process in consideration.

It can be argued that instead of the limited parallel model, a uniprocessor model may be used with remote execution treated as blocking; we explain why that is problematic:

If remote execution is modelled as blocking on some external event (in this case the completion of the remote operation) then it has to be introduced as a blocking term of length G_i; then $R_i = X_i + I_i + G_i$. Then there are two approaches: i) to consider that blocking (due to remote execution) contributes to interference on lower-priority processes, or ii) that

it doesn't. The WCRT equations for those two approaches then respectively become:

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

$$R_i = X_i + \sum_{j \in hp(i)} \left\lceil \frac{R_i}{T_j} \right\rceil X_j + G_i.$$

The first approach is clearly pessimistic, as gaps are treated as contributing to interference even though they do not. As for the second one, it actually does not cover the worst case which is captured by the limited parallel model and results from the variability in the placement of local execution inside a process invocation. This worst-case placement, reflected in Equation 1 as a "jitter" of G_j in the periodicity of local execution, is depicted in Figure 3.

2.1 A remaining source of pessimism

Some pessimism persists in the original limited parallel analysis [3, 4] 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 best described by this example:

In a two-process set, the higher-priority process (τ_j) consists of a sequence of interleaved LBs and gaps, while the other one (τ_i) only executes locally. We do not know the relative offset of the releases of τ_j, τ_i but wish to find an upper bound for R_i. The worst-case as per the original limited parallel analysis [3, 4] would have τ_i be preempted for the sum of length of the LBs of τ_j before it ever gets to execute. This is pessimistic, as in reality, whatever the release offset of the process, τ_i cannot suffer an initial preemption longer than the longest individual LB of τ_j.

3 Process Structure Exploited

For the ensuing analysis we require invocations of the same process to be characterised by 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 local and remote blocks e.g. $xgxgx$ or $gxgx$ (x for LBs, g for gaps).

3.1 Considerations on the notion of a worst-case phasing

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 the following example: In a two-process system, the higher-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 an interval of length 7 between the termination of one invocation of the process and the release of the next one). To identify the worst-case phasing in terms of interference suffered by the lower-priority process we thus examine all 3 candidate phasings. We do this for two scenaria: A) the lower-priority process having an execution requirement of 2 or B) 3 respectively, as depicted in Figure 4.

The example shows that **in the general case there is no phasing that always provides the worst-case**; which phasing yields the worst-case depends on the execution requirement of the process suffering the interference. Moreover for multiple higher-priority processes, if $n(\tau_j)$ phasings from each higher-priority process τ_j would have to be considered in combination for an exhaustive answer, this might not be tractable for large systems.

3.2 Notation and Transformations

If LBs (x) and gaps (g) are indexed by activation order, 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 x_{j_k}. G_{j_k}, $\hat{G}_{j_k}$ denote the maximum / minimum length of gap g_{j_k}. When specific values are assigned to this execution pattern we obtain the exact distribution for the respective process activation.

The above pattern actually starts with a LB and ends with a gap, thus having an equal number of remote and local blocks. This may not always be the case. For convenience in analysis 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 to the left if required) to start with a LB. Since our approach covers the worst-case under all possible release phasings for the process set, this is also a valid transformation. Adjacent blocks mapped both local or remote are then merged. Then the number of both LBs and gaps is the same, denoted $n(\tau_j)$ - this facilitates the analysis. An example of such a normalisation transformation is shown in Figure 5: letters denote block lengths; gaps appear shaded.

The concept of a notional gap (as will become evident upon complete formulation of this analysis) allows for transforming a given distribution by "shift-rotation" of its constituent blocks, an operation used throughout our approach in order to derive a bound on the worst case that covers all possible relative release phasings.

We introduce 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 $big_g(\tau_j, m)$, $small_g(\tau_j, m)$

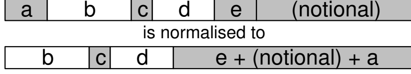

Figure 5: Distribution Normalisation

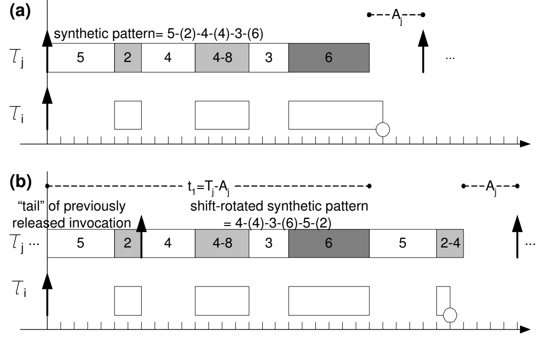

Figure 6: Jitter-like Effects

for gaps.

3.3 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. This function provides tighter bounds on the exact worst case than the existing analysis [3, 4].

The idea behind our approach is to transform process distributions by reordering the constituent blocks so that there exists one phasing that always provides the worst-case in terms of interference on lower-priority processes regardless of their execution requirement. The worst-case interference under these *synthetic* distributions is shown to be an upper bound for the worst-case under the actual distributions.

To come up with a tractable approach, we start off by expressing the problem in terms of the established uniprocessor analysis [10, 9]. 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 $u(\bullet)$ is the unit step function ($u(t)$=1$\forall t \geq 0$; else $u(t)$=0). The term $u(R_i - O_{j_k})$ is added to the equation so as to describe 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 summation operator 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 classic uniprocessor analysis [5]. 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.

This transformation allows for the computation of interference from given process execution distributions and phasings. However what is missing is the identification of the exact worst-case. To derive that we rely on the following 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 until it coincides with a release of a LB of τ_j does not decrease the interference suffered.
- Then if all LBs acquire the respective maximum length, interference can only increase.
- If all gaps acquire their respective minimum lengths interference can only increase.
- Then if in each invocation of τ_j, the first LB exchanges length with the longest one, interference can only increase. Similarly then with the second LB and the longest remaining LB and so on, until we run out of LBs.
- Then if in each invocation of τ_j, the first gap exchanges length with the shortest one, interference can only increase. Similarly then for the remaining gaps, as with LBs.

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

Suppose that there is another execution distribution that yields greater interference. By subjecting it to the transformation just described, the interference can only increase. But the product of the transformation would again be the synthetic worst-case distribution, which contradicts the initial supposition. Thus the synthetic worst-case distribution gives an upper bound for the actual worst-case interference.

□

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 if we consider the releases to be coincident, with the 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 the same process are characterised by 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 6):

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 6(a) shows the synchronous release scenario and the interference suffered by lower-priority process τ_i. In Figure 6(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 6(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 $I_{j\rightarrow i}$ exerted on τ_i by τ_j:

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

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

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 $A_j = G_j - \hat{G}_j$.

With multiple higher priority processes things are more complicated. However, we will show that even then, 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

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

3.4 Response Time Equations

The WCRT equations according to this newer analysis take the form

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

where $O_{j_k} = \sum_{m=1}^{k-1} (X_{j_m} + \hat{G}_{j_m})$, $A_j = G_j - \hat{G}_j$.

The equation is solved by forming a recurrence relationship as in classical analysis [5].

By juxtaposition with Equation 1 (which gives WCRTs under the Original Limited Parallel model) we observe that we can expect reduced pessimism due to:

- 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
- 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 [4]. 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.

In case a specific process is not amenable to the synthetic analysis (because 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 [3].

Comparing the complexity of the synthetic analysis with that of the original limited parallel model, the increase comes from two factors:

- The increased number of addends in the WCRT equations (one for each of the $n(\tau_j)$ LBs of each interfering process τ_j compared to one per process).
- The increased average number of iterations for the recurrence relationship to converge.

Regarding the first factor, the cost scales linearly with the number of addends for each iteration of the recurrence relation. Thus the respective complexity is at most $max_j\{n(\tau_j)\}$ times that for the original limited parallel (and the uniprocessor) model.

The effect of the second factor has not been quantified as it is highly input-dependent but it is deemed negligible.

4 The Effects Of Blocking

Resource control protocols can be implemented on top of limited parallel systems with no change of semantics, as demonstrated in [4] with the Priority Ceiling Protocol (PCP) [14]. Remote co-processors (where gaps execute) can also be treated in the same way as regular resources, if accessed through a protected object [4]. What changes in the context of this paper up to this point is not the class of the systems covered but rather the approach to timing analysis. Thus the PCP still applies and the effects of blocking can be modelled in the same way as previously. Namely a blocking term B_i is added to the WCRT equation: $R_i = C_i + I_i + B_i$. According to the PCP:

$$B_i = max_{u=1}^{U} usage(u,i)C(u)$$

where $usage(u,i) = 1$ if resource-u is used by at least one process with priority less than that of τ_i and at least one process with priority greater than or equal to that of τ_i (else it is 0). $C(u)$ denotes the worst-case length of the respective critical section. Note that if critical sections transcend block boundaries, their lengths are established according to the actual distribution, not the synthetic one.

4.1 Evaluation Of The Analysis

We evaluate the Synthetic analysis against the existing one for an arbitrary process set. The process set attributes are presented in Table 1 whereas the actual distributions and the synthetic ones derived from them are shown in Figure 7. Response times are juxtaposed in Table 1 with those derived under the original analysis for limited parallel systems [3, 4].

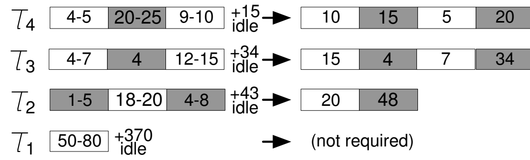

Figure 7: Distributions: actual → synthetic

Any improvement is input-dependent but our example shows that it can be noticeable. We were not able to use a real-world process set for the evaluation at this point, as the codesign environment into which the analysis is to be integrated has not yet been developed.

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

5 Priority Assignment

WCRT analysis is of limited usefulness if an arbitrary or suboptimal process set is assumed as systems which would otherwise have been fesible for a different priority assignment may appear infeasible. In the absence of a tractable optimal priority assignment algorithm all individual priority orderings would have to be examined to determine feasibility, at the cost of exponential complexity. We thus formulate an optimal branch-and-bound algorithm for priority assignment which uses WCRT analysis for the feasibility test and discuss its characteristics. The algorithm is general; it applies to either the uniprocessor model or the variants of the limited parallel models in the presence of blocking.

5.1 Insight

Audsley formulated a generalised priority asignment algorithm [2] that is optimal in the absence of shared resources. The algorithm uses the familiar WCRT equations as a feasibility test. These have the form

$$R_i = C_i + I_i$$

where R_i is the derived bound on the WCRT of process τ_i and $_i$ is the worst case interference suffered by τ_i (due to the execution of higher-priority processes). A brief description of that algorithm [2] follows:

1. Starting with the lowest priority, processes are tested for feasibility at that proprity level.
2. If none is found feasible then neither is the process set. Else, any of the processes found feasible is assigned this priority level.
3. The previous steps are repeated with the remaining processess and priority levels.

The algorithm works in the absence of shared resources because then the response time of a process can only go down (up) if its priority increases (decreases) and the relative priority ordering of the remaining processes is retained. This is because process WCETs are invariant and the interference suffered by a process exhibits the same monotonicity (and $R = C + I$). However with shared resources we have $R = C + I + B$; the blocking term is not a decreasing function of the priority, given a relative priority ordering of the remaining processes. Thus the monotonicity of the response time can no more be asserted. In practice, this means that a process feasible at priority i may cease to so be if exchanging priorities with the next-highest-priority process. Thus Audsley's algorithm, if applied, may fail to produce a feasible priority assignment when one does in fact exist. We later provide such an example.

We overcome this complication by noting that in the presence of shared resources, the WCRT (and thus feasibility) of a process depends on the set of higher-priority processes (which of course also determines the set of lower-priority processes and vice versa) but not their relative priority ordering. This has the following interesting corollary:

Corollary 1: *If a process is not feasible given a priority ordering, it will not be feasible either for any priority ordering with the same lower-priority process set (and thus also the higher-priority process set) for the process under consideration as in the original ordering.*

In our approach, this property is used to eliminate groups of orderings with a single test, thus reducing the overall complexity.

5.2 The Branch-And-Bound Algorithm

We proceed to formulate the algorithm:

1. Starting with the lowest priority level, all processes are tested for feasibility at this level.
2. For each of those processes found feasible at that priority, the procedure continues recursively, after assigning that priority to the process, with the remaining processes and priority levels. If at this stage no more priorities and processes are left, a feasible ordering has been identified.

The algorithm always terminates and will find all feasible (in the offset-agnostic sense) orderings. If any of those suffices, it can be modified to terminate on the first one found.

The algorithm essentially constructs an implicit permutation tree for the n processes, like that of Figure 8. Each path from a leaf to the root of the tree corresponds to a priority ordering. Each node represents a process, its depth i corresponding to its assigned priority,

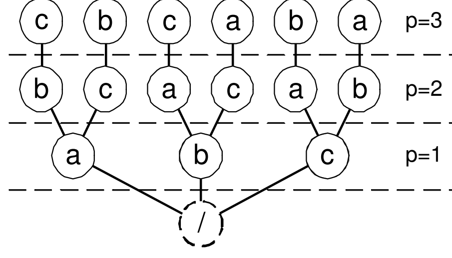

Figure 8: Permutation tree for $\{\tau_a,\, \tau_b,\, \tau_c\}$

and is the parent of $n-i$ distinct subtrees. Our algorithm traverses the tree depth-first and conducts a process feasibility test for each node. If the test returns $infeasible$ the traversal of the subtree is aborted, as all paths (orderings) sharing the specific node would be infeasible. A feasible ordering is identified as soon as a leaf is reached for which the test returns $feasible$. Our algorithm is thus classified as *branch-and-bound*.

In the worst-case (which occurs a full traversal of the tree) the number of process feasibility tests (equals to the number of nodes) is:
$n+n(n-1)+n(n-1)(n-2)+...+n(n-1)...1=\sum_{i=1}^{n}\frac{n!}{(n-i)!}=n!\sum_{u=0}^{n-1}\frac{1}{u!}<n!\sum_{u=0}^{\infty}\frac{1}{u!}=en!$
Thus the complexity in scheduling a process set is $O(n!)$ times the complexity of the process feasibility test. Alternatively we note that the complexity of a process feasibility test is roughly proportional to the number of the terms in the response time equation (thus, of higher-priority processes). Then, if ξ is the complexity of the feasibility test for a single higher-priority process, the overall scheduling complexity is

$$\sum_{i=1}^{n}\frac{n!}{(n-i)!}(n-i)\xi = \sum_{i=1}^{n}\frac{n!}{(n-i-1)!}\xi = n!\sum_{v=0}^{n-2}\frac{1}{v!}\xi < en!\xi = O(n!)\xi$$

However, as the algorithm is branch-and bound, we expect an average complexity orders of magnitude lower. By comparison, the worst-case complexity of Audsley's algorithm is

$$\sum_{i=1}^{n} i(i-1)\xi = O(n^3)\xi$$

and the complexity for exhaustively testing all $n!$ possible priority orderings is

$$n!\sum_{i=1}^{n}(n-i)\xi = n!\frac{n^2+n}{2}\xi = O((n+2)!)\xi.$$

A demonstration of the algorithm follows.

5.3 Demonstration

We illustrate the priority assignment algorithm for an artificial process set; in the process we also highlight why Audsley's algorithm is not optimal for this class of systems. The process set parameters are provided in Table 2. For convenience, the timing analysis is according to the original limited parallel model [3, 4]

G, X denote worst-case figures for execution time spent in hardware/software respectively and for our example $C = X + G$ for all processes. The column with the heading b displays the length of the critical section guarding access to the single shared resource. We use the following notation:

The string XYZ denotes the assignment of priorities 1, 2, 3 to processes τ_X, τ_Y, τ_Z respectively. The expression $R_{Y|XYZ}$ denotes the worst-case response time of process τ_Y for the given priority assignment XYZ. The wildcard $*$ can be used to represent partial assignments, for example $A{*}{*}$ denotes that τ_A has a priority of 1 but the remaining priorities (2 and 3) may be assigned in any way to the remaining processes (τ_B, τ_C). However, as the set of higher-priority processes is defined for τ_A for the partial priority assignment $A**$, we can calculate the worst-case response time of τ_A and $R_{A|A**} = R_{A|ABC} = R_{A|ACB}$.

τ	T	D	X	G	C	b
τ_A	60	40	5	15	20	20
τ_B	60	60	15	5	20	20
τ_C	60	40	15	5	20	20

Table 2: Process set parameters

For convenience only, we provide in Table 3 the blocking terms for each process for each of the possible priority orderings according to the ceiling protocol [14]. Onwards with the demonstration:

τ	ABC	ACB	BAC	BCA	CAB	CBA
τ_A	0	0	20	20	20	20
τ_B	20	20	0	0	20	20
τ_C	20	20	20	20	0	0

Table 3: Blocking term lookup table

Processes are tested in turn for feasibility at the lowest priority (of 1):

- As $R_{A|A**} = 50 > D_A$ the whole set of orderings $A{*}{*}$ is disqualified as infeasible.
- We next calculate $R_{B|B**}$ which is found to be $40 < D_B$ thus τ_B is feasible. Thus we recursively test the remaining priorities at the remaining levels: But
  - $R_{A|BA*} = 55 > D_A$ and
  - $R_{C|BC*} = 60 > D_C$

  thus τ_A, τ_C are both infeasible so we disqualify the set of orderings $B**$ and backtrack. Audsley's algorithm does not backtrack and would terminate at this point without producing a feasible assignment.

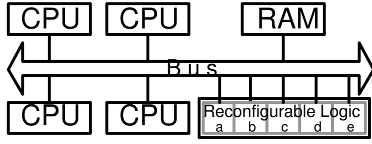

Figure 9: Multiprocessor Architecture Extension

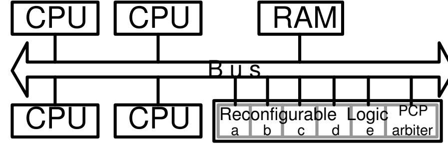

Figure 10: Implementation Of A PCP Arbiter

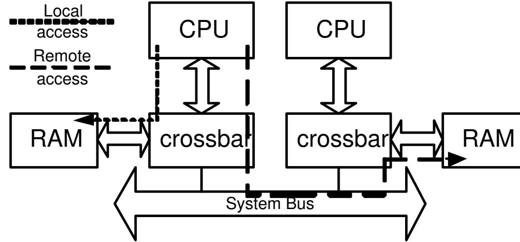

Figure 11: A typical NUMA architecture

- However, we find that $R_{C|C**} = 40 < D_C$ and recursively:
  - $R_{A|CA*} = 55 > D_A$ (thus $CA*$ is disqualified) but
  - $R_{B|CB*} = 60 = D_A$ (feasible) and recursively
    - $R_{A|CBA} = 40 = D_A$

thus the process set is feasible under the priority ordering CBA.

6 Multiprocessor Extension

We generalise the analysis to also cover limited parallel systems with co-processors where instead of a single instruction set processor there exists an array thereof: these processors are identical, sharing the same address space and any process can run on any processor and even migrate during execution. Figure 9 depicts such an architecture. The N highest-priority processes among those competing for the processor are executing on any given instant.

We require system resources to be global and accessed symmetrically by all processing elements. One could argue for local memories attached to specific processors, in what would effectively constitute a Non Uniform Memory Access (NUMA) architecture. Such an example is depicted in Figure 11. Local memory is accessed through the local I/O controller (crossbar). Remote Memory is accessed through two additional hops, accross the system bus and through the crossbar of the remote processsor; this incurs greater access latencies. We explain why the NUMA approach would be problematic.

First of all, as processes are not attached to a specific processor (and can even migrate during execution), there is no incentive to have any resource be local to any particular processor. Secondly, having process state reside in local memory would introduce long delays

(due to data tranfer) on process migration. By having all memory resources be global and symmetrically accessed we can assume migration costs to be negligible. And thirdly, it would be impossible to determine during offline WCET whether a memory access would be local or remote, so the worst would have to be assumed.

The issue then of access to the shared system bus arises. To cope with that 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. The resulting increased memory latencies are an aspect of the architecture/implementation and, as such, are factored in during the WCET analysis for the code (which exceeds the scope of this paper).

6.1 The Synthetic Analysis Revisited

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 upper bounds on WCRTs for such systems. For the ensuing analysis we again require that no resources be shared; such considerations are dealt with later.

For a process τ_i the term W_i is an upper bound on the time spent executing locally to the processor array (ie. in software) by all higher-priority processes:

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

W_i is measured in processor-ticks. If there is only $N = 1$ processors in the system, then $W_i = I_i$. However, with $N > 1$ processors, a process competing for the a processor can only be denied a processor to execute on (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 given process $\tau + i$ over a time window of length equal to its WCRT thus is an upper bound for the worst-case interference suffered by it. We will show that such a bound can be derived as:

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

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 processor-ticks) for the whole array would have been at least

$$N\check{I}_i = N(I_i + a) = Na + N\left\lfloor \frac{W_i}{N} \right\rfloor \geq Na + N\frac{W_i}{N} - 1 = Na - 1 + W_i > W_i$$

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 [13]) offer utilisation-based feasibility tests only, which do not cover systems where deadlines are less than the process periods.

Process WCETs calculated under our approach for multiprocessor architectures are a decreasing function of the process priority (for a given relative priority ordering of the remaining processes). Thus we note the **absence of any scheduling anomalies** like those described by Graham [7] under non-preemptive EDF.

The next section introduces the effects of shared resource control.

6.2 Resource Control and Priority Assignment For Multiprocessor Limited Parallel Systems

The Priority Ceiling Protocol (PCP) [14] as formulated for uniprocessor systems has been shown to have the same properties (liveness, bounded blocking times, no chained blocking) for limited parallel systems if the hardware co-processors themselves are treated as shared resources [4]. Bounds on blocking times are then derived by using the same equations as in the uniprocessor case.

We will show that same protocol can be applied to the multiprocessor extension of the limited parallel model in the same manner (and that the same properties and equations hold) if requests for resource locking originating from processes on different processors are serialised. We then propose a mechanism enforcing this serialisation.

Rajkumar [12] discusses two variants of the PCP for multiple processor systems with/without shared memory respectively. Local resources are accessed under uniprocessor PCP on each processor; global resources are accessed according to the Distributed/Multiprocessor PCP respectively. In either case, global critical sections reside on dedicated *synchronization processors*, which arbitrate access requests by processes from other (application) processors. Then,

because all global critical sections are on a single synchronisation processor, the execution of global critical sections is serialised (a single processor not allowing parallel execution).

The multiprocessor model we use in this paper 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. [12] assumes a static assignment of processes to processors). Thus, uniprocessor PCP suffices, an arbiter is only needed for the avoidance of race conditions, with the execution of the global critical section occuring on the processor of the calling process (and not the synchronisation processor). This provides a significant reduction in blocking overheads – many global critical sections can be executed in parallel (by processes on different processors), with only the access requests themselves serialised by the arbiter.

One possible solution for implementing the serialisation of locking operations originating from different processors is by means of a dedicated hardware arbiter. This resource is then to be accessed as a shared location in the common address space. In fact if the locking/unlocking requests can be implemented as atomic single instruction read/write operations, the issue of controlling access to the arbiter is resolved (by the memory bus itself).

Consider the following protocol:

Assuming that shared memory blocks are placed at 2^{β} byte boundaries and a data bus width of λ bits, a control word can be used to encode such requests where the $\lambda - \beta$ highest-order bits signify the memory location, the remaining β bits serving as opcode. A process requesting to lock a memory location (signifying the start of an area of shared memory) writes such a control word to the memory address of the arbiter. When it receives an ACK, it can then access it. When it has finished, it sends another control message to signal this.

Since FPGAs are the platform of choice for implementing limited parallel systems, they also provide the flexibility to implement the proposed arbiter in reconfigurable logic. Moreover the use of co-processors by itself suggests a design paradigm where the platform itself is tailor-made for the application. The introduction of the PCP arbiter into the hardware fits well with this paradigm so that trying to adapt the application and the protocol to the limitations of a rigid architecture is not an attractive option.

Regarding optimal priority assignment in the presence of blocking, the algorithm detailed in [6] is applicable. It is an exponential algorithm ($(O(n-1)!)$ where n is the number of processes), but due to its branch-and-bound nature, we expect the average complexity to be orders of magnitude better.

7 Conclusion

The analysis formulated provides tighter bounds on the interference exerted by processes in the worst-case, compared to existing analysis for the the limited parallel model [3, 4]. However processes 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 worst-case response times are respective upper bounds for all possible relative release offset combinations for the processes. We deem the analysis suitable for use in hardware / software codesign, aiming at *correctness by design* regarding timing requirements. We introduce a multiprocessor extension to the model and a WCRT analysis free of scheduling anomalies for multiprocessor systems under FPS, with or without co-processors.

References

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

[2] N. C. Audsley. On Priority Assignment in Fixed Priority Scheduling. *Information Processing Letters*, 79(1):39–44, 2001.

[3] 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.

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

[5] 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.

[6] K. Bletsas. A priority assignment algorithm in the presence of blocking. Technical Report YCS-385, Department of Computer Science, University of York, 2005.

[7] R. L. Graham. Bounds on multiprocessor timing anomalies. *SIAM Journal on Applied Mathematics*, 17(2):416–429, Mar. 1969.

[8] 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 Conference on Real-Time Systems*, 1997.

[9] M. Joseph and P. Pandya. Finding response times in a real-time system. *The Computer Journal (British Computer Society)*, 29(5):390–395, October 1986.

[10] 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.

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

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

[13] 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.

[14] L. Sha, R. Rajkumar, and J. P. Lehoczky. Priority inheritance protocols: An approach to real-time synchronisation. *IEEE Transactions on Computers*, 39(9):1175–85, Sept. 1990.

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

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

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