Scheduling Constrained-Deadline Sporadic Parallel Tasks Considering Memory Contention

Björn Andersson, Dionisio de Niz, Hyoseung Kim, Mark Klein, Ragunathan Rajkumar
Carnegie Mellon University

ABSTRACT

Consider global-Earliest-Deadline-First scheduling on a multiprocessor assuming (i) constrained-deadline sporadic tasks: a task generates a sequence of jobs and the deadline of a job is at most the minimum inter-arrival time of the task generating the job; (ii) stage-parallelism: a task comprises at least one stage, a stage comprises at least one segment, and a segment is allowed to execute only if all segments of the previous stage have finished; and (iii) contention for shared resources in the memory system — cache eviction, reordering in memory controller (MC), memory bus contention. We present an algorithm that (i) performs schedulability testing; (ii) configures virtual-to-physical address translation (VPAT) so a cache block fetched to the last-level cache by one task is not evicted by another; (iii) configures VPAT to reduce the reordering effect (REE) in the MC; and (iv) considers contention for the memory bus. Our algorithm solves a Mixed-Integer Linear Program (MILP); we have implemented a tool and tested it. Across all of our experiments, we found that the maximum time it takes to finish is 18h and the median time is 2.5h. We have also done preliminary testing on a real computer platform.

1. INTRODUCTION

Multicore processors are the norm today. The trend is that the number of processors on a chip increases exponentially while the clock frequency stays constant. And software practitioners are under pressure to deliver improved functionality which requires more CPU cycles. This trend makes it increasingly common that a job has execution requirement so large that executing it sequentially causes a deadline miss; hence, the only way for a job to meet its deadline is to perform some execution in parallel. This brings the challenge:

> **C1.** Schedule software where some parts can execute in parallel so that all deadlines are met; also prove before run-time that deadlines are met.

The timing of software executing on a COTS multicore processor depends not only on the processor scheduler but also on contention for shared resources in the memory system. This includes (i) the last-level cache (LLC) shared between processors, (ii) the row buffer in each memory bank (MB) storing the most recently accessed row, and (iii) the memory bus (the bus between the memory controller (MC) and DRAM modules). A cache memory is typically organized as a set of cache sets where certain bits of the physical address (PA) of a memory access determine which cache set the memory access should use. Hence, if the virtual-to-physical address translation (VPAT) is set up such that no two memory accesses of different tasks use the same cache set, then it is guaranteed that a cache block fetched into the cache by one task cannot be evicted by another task. Also, DRAM modules are typically organized as a set of MBs with each MB having multiple rows and each MB having one row buffer which stores the data of the most recently accessed row. When a memory access experiences a miss in the LLC, it (i) precharges the MB, that is, the data in the row buffer of the MB is written back to its row in the MB and then (ii) activates the MB, that is, load a row in the MB (given by the address of the PA) to the row buffer of the MB and then (iii) reads data from this buffer and transfers the data to the processor (if the memory access is a read) or writes data to this row buffer (if the memory access is a write). If the row needed for a memory access is already loaded in the row buffer, then precharge and activate are not performed and hence execution is faster. Hence, MCs reorder memory accesses so memory accesses to the row that is in the row buffer get ahead in certain queues in the MC. Thus, a memory access can be delayed because other memory accesses, of other tasks, get ahead in the queue — reordering effect (REE). Hence, if VPAT is set up so that a MB is accessed by at most one task then it is guaranteed that no task can suffer from this REE. Also, a memory access can be delayed because other accesses use the memory bus. This brings the challenges:

Solution	Addresses challenges			
	C1	C2	C3	C4
[17, 27, 13, 10, 9, 22, 4, 7, 2, 16, 8, 18, 24]	Yes	No	No	No
[21, 29]	No	Yes	No	No
[19, 25]	No	No	Yes	No
[28]	No	Yes	Yes	No
[26, 11, 20, 23, 31, 30, 14]	No	No	No	Yes
This paper	Yes	Yes	Yes	Yes

Table 1: Summary of the state of art.

> **C2.** Configure VPAT so that a cache block fetched to the LLC by one task cannot be evicted by another.

> **C3.** Configure VPAT so that reordering of memory accesses from different tasks are avoided and if they occur, then the schedulability test computes an upper bound on extra execution time due to reordering.

> **C4.** Compute an upper bound on extra execution time caused by processors sharing the memory bus.

Unfortunately, the literature offers no solution for *all* of these challenges — see Table 1.

Therefore, this paper presents a solution for *all* of these challenges. We assume global-EDF (gEDF) and reformulate a previously-known schedulability test as a Mixed-Integer Linear Program (MILP) and extend this formulation to (i) configure VPAT so a cache block fetched to the LLC by one task is not evicted by another, (ii) configure VPAT to try to eliminate extra execution time caused by the REE in the MC; if not possible, the REE is considered in the schedulability test, and (iii) consider contention for the memory bus. We also present a tool[1] that implements this theory and evaluate it.

[1]See http://www.andrew.cmu.edu/user/banderss/projects.html

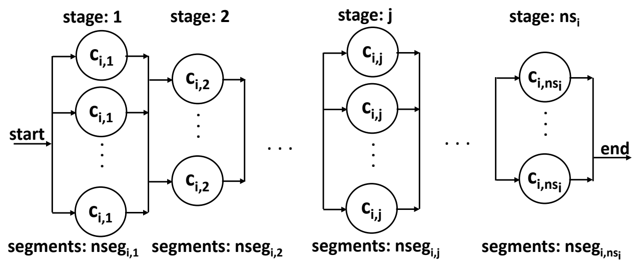


Figure 1: The parallel task model used in this paper.

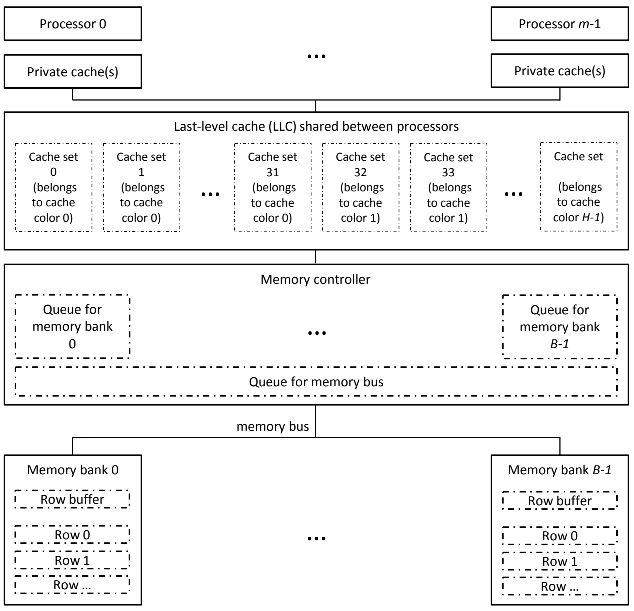
The remainder of this paper is organized as follows. Section 2 presents the system model. Section 3 adapts a previously known schedulability test to a MILP. Section 4 presents constraints that express an upper bound on the execution time of a segment due to memory contention and how it depends on VPAT, and also expresses other constraints. Section 5 puts it all together as a solution for the four challenges. Discussions, evaluations, and conclusions follow.

2. SYSTEM MODEL

Fig. 1 shows the parallel task model we use and Fig. 2 shows the hardware model we use. We consider a system with m processors of speed s and a taskset τ. A task τ_i in τ is characterized by T_i, D_i, $\mathrm{ns_i}$, $\mathrm{nseg_{i,j}}$, and $C_{i,j}$. The interpretation of these parameters is that (i) τ_i generates a sequence of jobs with arrival times of two consecutive jobs of τ_i separated by at least T_i; (ii) a job of τ_i needs to finish execution by its absolute deadline (the absolute deadline of a job of τ_i is D_i time units after its arrival); and (iii) job execution is described with stages where $\mathrm{ns_i}$ denotes the number of stages of a job of τ_i and $\mathrm{nseg_{i,j}}$ denotes the number of segments in stage j of a job of τ_i. A segment executing contiguously for Δ time units performs $\Delta \times s$ units of execution. A segment of a job finishes when it has performed a number of units of execution equal to its execution requirement. For a segment of a job, if the segment is in stage j of τ_i then its execution requirement is at most $C_{i,j}$ assuming that it does not experience memory contention; if it experiences memory contention then its execution requirement may be larger (explained later). When a job of τ_i arrives, all the $\mathrm{nseg_{i,1}}$ segments of stage 1 of τ_i become eligible for execution. For each $j \geq 2$, at the time when all the $\mathrm{nseg_{i,j-1}}$ segments of stage $j-1$ of τ_i have finished, all the $\mathrm{nseg_{i,j}}$ segments of stage j of τ_i become eligible for execution. A segment becomes non-eligible when it has finished execution. A job of τ_i finishes when all the $\mathrm{nseg_{i,ns_i}}$ segments of stage $\mathrm{ns_i}$ of this job have finished. We assume $\forall \tau_i \in \tau : D_i \leq T_i$ — such tasksets are called *constrained-deadline sporadic tasksets.*

gEDF works as follows: (i) jobs are assigned priorities such that if a job has higher priority than another job then the absolute deadline of the former is no later than the absolute deadline of the latter, (ii) a segment inherits the priority of the job it belongs to, and (iii) at each instant, if at most m segments are eligible for execution at this instant, then all of them execute at this instant; if $m+1$ or more segments are eligible for execution, then the m highest priority segments at this instant are selected for execution at this instant. A taskset τ is *gEDF-schedulable* on a computer with m processors of speed s if, for each jobset that τ can generate, for each schedule that gEDF can generate for this jobset, it holds that each job finishes no later than its absolute deadline.

Each segment has a virtual address (VA) space. The VA space is organized into pages of size PAGESIZE bytes. (For example, PAGESIZE=4096 bytes.) The memory footprint of a segment of stage j of τ_i is at most $\mathrm{np}_{i,j}$ pages. Each page is associated with a range of VA and a page is associ-



Processor 0
Processor *m*-1
Private cache(s)
Private cache(s)
Last-level cache (LLC) shared between processors
Cache set 0 (belongs to cache color 0)
Cache set 1 (belongs to cache color 0)
Cache set 31 (belongs to cache color 0)
Cache set 32 (belongs to cache color 1)
Cache set 33 (belongs to cache color 1)
Cache set (belongs to cache color *H-1*)
Memory controller
Queue for memory bank 0
Queue for memory bank *B-1*
Queue for memory bus
memory bus
Memory bank 0
Memory bank *B-1*
Row buffer
Row 0
Row 1
Row ...

Figure 2: The hardware model we use.

ated with a frame of physical memory (of size PAGESIZE bytes) and this frame is associated with a range of PAs. The $\log_2$ PAGESIZE least significant bits of the PA are called *frame offset.* The other bits of the PA are called *frame index.* The $\log_2$ PAGESIZE least significant bits of the VA are called *page offset.* The other bits of the VA are called *page index.* A VA is mapped to a PA as follows: (i) the frame offset is identical to the page offset and (ii) the frame index is obtained through VPAT from the page index (typically through a page table in an operating system). For convenience, $\mathrm{seg}(i,j,g)$ denotes the g^{th} segment of stage j of τ_i and $\mathrm{page}(i,j,g,p)$ denotes page p of $\mathrm{seg}(i,j,g)$.

In order to allow segments to share data, we introduce shfr that specifies the requirements that this imposes on VPAT. shfr is a set containing 8-tuples with the interpretation that for each 8-tuple $\langle i,j,g,p,i',j',g',p' \rangle$ it is required that $\mathrm{page}(i,j,g,p)$ and $\mathrm{page}(i',j',g',p')$ are mapped to the same frame.

In previous work [14], we presented and validated a model of the memory system of typical COTS multicore processor based systems. In this paper, we extend this model to a more fine-grained description of memory accesses. Our model is as follows. The LLC (see LLC in Fig. 2) is shared between processors. This cache is organized as a set of cache sets where certain bits in the PA determine which cache set a memory access is associated with. Some of these bits are part of the frame index and some are part of the frame offset. When a memory access experiences a miss in LLC, the memory access is passed on to the MC and identifies which MB the memory access is associated with (certain bits in the frame index determine this) and which row in this MB it is associated with (other bits in the frame index determine this) and it is inserted in a queue for memory accesses to this MB. The queuing discipline First-Ready-First-Come-First-Served (FR-FCFS) is used. With this queuing discipline, FCFS is used but with the following exceptions (i) a memory access can be prevented (to be explained later) from being performed at certain instants because there are DRAM timing parameters which state that a certain part of a DRAM access must wait until a certain timing requirement (based on previous memory accesses) is satisfied and (ii) elements in the queue of a MB can be reordered so that a memory access gets to the head of the queue when this memory access is associated with the row that is currently loaded in the row buffer. When a memory access gets to the head of the queue of the

MB, it contends for the memory bus with memory accesses of other MB. When a memory access is granted the memory bus, the memory access *precharges* its associated MB (that is, the data in the row buffer is written back to its row in the MB) and then the memory access *activates* its associated row in its associated MB (that is, the data in this row is loaded to the row buffer) and finally it transfers data (from the row buffer of the MB to the MC if the memory access is a read; the other direction if it is a write). If the row associated with the memory access is already in the row buffer then precharge and activate are not performed.

In many of today's processors, the bits in the PA from which the cache set index of LLC is obtained overlap with the bits that determine the frame index (see [28]). Also, in many of today's processors, the bits in the PA from which the MB index is obtained overlap with the bits that determine the frame index (see [28]). Therefore, one can use a technique called *cache coloring* [21, 29, 28, 15] which partitions memory frames of physical memory into cache colors so if two memory accesses belong to different frames and these two memory frames belong to two different cache colors, then one memory access cannot evict a cache block fetched to LLC by the another memory access. Also, one can use a technique called *bank coloring* [28] which partitions memory frames of physical memory into MB colors so if two memory accesses belong to different frames and these two memory frames belong to two different MB colors, then one memory access cannot evict a row in a MB that another memory access has loaded. Typically an MB color is a single MB. But a cache color typically consists of multiple cache sets. Let H denote the number of cache colors and let B denote the number of MB colors. MEMCAP denotes the amount of physical memory in the computer, measured in the number of frames. Let $\text{CAP} = \text{MEMCAP}/(H \times B)$. (For example, in a typical personal computer: $\text{MEMCAP} = 2^{31}/4096 = 2^{19}$,$H = 32$,$B = 16$, and this yields $\text{CAP} = 2^{19}/(32 \times 16) = 2^{10}$). Note that in some modern multicore chips, the number of cache colors is affected by a hash function within the LLC [15].

One factor that determine the execution time of a program is its self-eviction of blocks in its cache which, in turn, depends on VPAT. Hence, if map is the VPAT of all tasks in the system then $C_{i,j}(\text{map})$ is an upper bound on the exe<u>c</u>ution requirement of a segment in stage j of τ_i for the case that this segment does not experience contention for resources in the memory system from other segments. Also, $\text{MA}_{i,j,p}(\text{map})$ is an upper bound on the number of <u>m</u>emory <u>a</u>ccesses reaching the MC from page p of a segment in stage j of τ_i for the case that this segment does not experience contention for resources in the memory system from other segments. Let $C_{i,j}$ be a value such that $\forall \text{map} : C_{i,j}(\text{map}) \leq C_{i,j}$. Let $\text{MA}_{i,j,p}$ be a value such that $\forall \text{map} : \text{MA}_{i,j,p}(\text{map}) \leq \text{MA}_{i,j,p}$. In practice, if the VPAT map is known, then it is possible to obtain $C_{i,j}(\text{map})$ and $\text{MA}_{i,j,p}(\text{map})$ (e.g. using a worst-case execution-time analysis tool) but obtaining $C_{i,j}$ and $\text{MA}_{i,j,p}$ is very expensive because they describe behavior of the software for *all possible VPAT* of the system. Even if $C_{i,j}$ and $\text{MA}_{i,j,p}$ are obtained, it can happen that we choose a VPAT map such that $C_{i,j}$ is much higher than $C_{i,j}(\text{map})$ (and analogously for $\text{MA}_{i,j,p}(\text{map})$). This would result in large pessimism. Also, note that in real systems, there are private caches. Typically, the cache set in the private cache that is used by a memory access is detemined by the VA of the memory access. Thus, cache coloring cannot control the eviction in private caches. We defer discuss of these issues to Section 6. For now, assume $C_{i,j}$ and $\text{MA}_{i,j,p}$ are known.

Let MBCF denote the memory bus clock frequency and let $t_{CK} = 1/\text{MBCF}$. We assume (as do many previous studies [30, 23, 20, 11]) that a processor is stalled when it waits for memory. Let $L^{\text{PRE}}_{\text{inter}}$ denote the time required for precharge; L^{ACT}_{inter} is the time required for activate; $L^{\text{RW}}_{\text{inter}}$ is the time for data transfer. Then, using [14], these can be computed from parameters available in DRAM datasheets [12] as follows:

$$\begin{aligned}
L^{\text{PRE}}_{\text{inter}} =& t_{\text{CK}} \\
L^{ACT}_{\text{inter}} =& \max(t_{\text{RRD}}, t_{\text{FAW}} - 3 \times t_{\text{RRD}}) \times t_{\text{CK}} \\
L^{\text{RW}}_{\text{inter}} =& \max(\text{WL} + \frac{\text{BL}}{2} + t_{\text{WTR}}, \text{CL} + \frac{\text{BL}}{2} + 2 - \text{WL}) \times t_{CK} \\
L_{\text{inter}} =& L^{\text{PRE}}_{\text{inter}} + L^{ACT}_{\text{inter}} + L^{\text{RW}}_{\text{inter}}
\end{aligned}$$

We will also use parameters which describe how one memory access can experience interference from other memory accesses. L_{conf} denotes row-conflict service time. $L_{\text{conhit}}(x)$ is a function which describes the time it takes to serve x consecutive memory accesses to the same row in the same MB if the row was already activated. These can be computed [14] from parameters available in DRAM datasheets [12] as:

$$\begin{aligned}
L_{\text{conf}} =& (t_{\text{RP}} + t_{\text{RCD}} + \max(\text{CL} + \frac{\text{BL}}{2} + 2, \\
& \text{WL} + \frac{\text{BL}}{2} + \max(t_{\text{WTR}}, t_{\text{WR}}))) \times t_{\text{CK}} \\
L_{\text{conhit}}(x) =& (\lceil \frac{x}{2} \rceil \times (\text{WL} + \frac{\text{BL}}{2} + t_{\text{WTR}}) + \\
& \lfloor \frac{x}{2} \rfloor \times \text{CL} + \max(t_{\text{WR}} - t_{\text{WTR}})) \times t_{\text{CK}}
\end{aligned}$$

For convenience, we use the following notation. P denotes the least common multiple of T_i values. $\text{DMAX} = \max_{\tau_i \in \tau} D_i$. N_{re} is a hardware limit on reordering (to be discussed later). $\{a..b\}$ denotes the set of integers greater than or equal to a and less than or equal to b. Let s.t. mean *such that* and : mean *it holds that.* $(\forall t \geq 0 : x)$ means the predicate $(\forall t \text{ s.t. } t \geq 0 : x)$. lhs means left-hand side and rhs means right-hand side. We define $\text{UBNOMR} = \sum_{\tau_i \in \tau} (\max_{j \in \{1..\text{ns}_i\}} \text{nseg}_{i,j})$ meaning <u>u</u>pper <u>b</u>ound on the <u>n</u>umber of <u>o</u>utstanding <u>m</u>emory <u>r</u>equests and define $\text{LL} = \min(m - 1, \text{UBNOMR} - 1)$ and $\text{LH} = \text{LL} + N_{\text{re}}$.

Tasks typically perform execution and access memory in an initialization phase which does not have real-time requirements. This execution and memory accesses are not considered as a job but the pages accessed need to be mapped to memory frames. Therefore, INO indicates the number of pages accessed during initialization.

3. SCHEDULABILITY ANALYSIS FOR GEDF OF PARALLEL TASKS WITHOUT MEMORY CONTENTION

The literature offers many sufficient schedulability tests for gEDF for tasks that are not parallel — see for example [3, 6, 5]. Of particular interest is [5] which offers a schedulability test with speedup factor two. The key to its good performance is the use of ffdbf — <u>f</u>orced-<u>f</u>orward <u>d</u>emand-<u>b</u>ound <u>f</u>unction — for describing the maximum amount of execution a task can demand in a time interval, rather than using the traditional dbf — <u>d</u>emand-<u>b</u>ound <u>f</u>unction. This schedulability test [5] states that if there exists a σ such that σ is at least as large as the density of each task and for each value of t the sum of ffdbf of tasks is at most a certain value then the taskset is gEDF-schedulable. (If such a σ exists then it is called a *witness.*) Later work [2] extended this for parallel tasks by defining ffdbf for parallel tasks. Fig. 3 shows it. In Fig. 3, $\text{f}(\tau, m, s)$ is a schedulability test. It takes as input a taskset τ, m, and s and outputs a boolean such that if this boolean is true then τ is gEDF-schedulable on m processors of speed s. $\text{f}(\tau, m, s)$ is evaluated by checking if there exists a σ such that σ is at

$$C_i \overset{\text{def}}{=} \sum_{j=1}^{\text{ns}_i} \left(\text{nseg}_{\text{i,j}} \times C_{i,j}\right) \qquad \eta_i \overset{\text{def}}{=} \sum_{j=1}^{\text{ns}_i} \left(\lceil \frac{\text{nseg}_{\text{i,j}}}{m} \rceil \times C_{i,j}\right)$$

$$\text{bsp}_{i,j} \overset{\text{def}}{=} \frac{C_{i,j}}{s} \times \lfloor \frac{\text{nseg}_{i,j}}{m} \rfloor \qquad \text{sp}_{i,j} \overset{\text{def}}{=} \frac{C_{i,j}}{s} \times \lceil \frac{\text{nseg}_{i,j}}{m} \rceil$$

$$\text{WJ}(\tau_i, t, s) \overset{\text{def}}{=} \begin{cases} 0 & \text{if t} < 0 \\ \text{WJS}(i, t, 1, s) & \text{if } 0 \leq \text{t} < \frac{\eta_i}{s} \\ C_i & \text{if } \frac{\eta_i}{s} \leq \text{t} \end{cases}$$

$$\text{WJS}(i, t, j, s) \overset{\text{def}}{=} \begin{cases} t \times m \times s & \text{if } 0 \leq t < \text{bsp}_{i,j} \\ \text{bsp}_{i,j} \times m \times s + (t - \text{bsp}_{i,j}) \times (\text{nseg}_{i,j} \bmod m) \times s & \text{if } \text{bsp}_{i,j} \leq t < \text{sp}_{i,j} \\ C_{i,j} \times \text{nseg}_{i,j} + \text{WJS}(i, t - \text{sp}_{i,j}, j+1, s) & \text{if } \text{sp}_{i,j} \leq \text{t} \end{cases}$$

$$\text{ffdbf}(\tau_i, t, v, s) \overset{\text{def}}{=} \lfloor \frac{t}{T_i} \rfloor \times C_i + C_i - \text{WJ}(\tau_i, (D_i - (t \bmod T_i)) \times v, s)$$

$$\text{h}(\tau, m, s, \sigma, t) \overset{\text{def}}{=} (\sum_{\tau_i \in \tau} \text{ffdbf}(\tau_i, t, \frac{\sigma}{s}, s) \leq ((m - (m-1) \times \frac{\sigma}{s}) \times t \times s))$$

$$\text{f}(\tau, m, s) \overset{\text{def}}{=} \left(\exists \sigma \text{ s.t. } (\sigma \geq \max_{\tau_i \in \tau} \frac{\eta_i}{D_i}) \wedge (\forall t \geq 0 : \text{h}(\tau, m, s, \sigma, t))\right)$$

$\text{f}(\tau, m, s) \Rightarrow \tau$ is gEDF schedulable on m processors of speed s if tasks do not experience memory contention

Figure 3: Previously known schedulability analysis for gEDF scheduling of parallel tasks [2].

least as large as the density of each task and for each value of t the sum of ffdbf of tasks is at most $(m-(m-1) \times \frac{\sigma}{s}) \times t \times s$. Intuitively, $\text{ffdbf}(\tau_i, t, v, s)$ expresses an upper bound on the amount of time that a task τ_i can perform in a time interval of duration t if a deadline miss occurred at the end of this time interval and the parameter v is used to describe the amount of idleness in the system. Hence, $\text{ffdbf}(\tau_i, t, v, s)$ comprises two parts: execution of jobs of τ_i that arrived before the beginning of this time interval and execution of jobs of τ_i that arrived after this time interval. An upper bound on the former is $C_i - \text{WJ}(\tau_i, (D_i - (t \bmod T_i)) \times v, s)$. An upper bound on the latter is $\lfloor \frac{t}{T_i} \rfloor \times C_i$. Together, this yields $\text{ffdbf}(\tau_i, t, v, s)$ as expressed in Fig. 3. See [5] for a discussion about $\text{ffdbf}(\tau_i, t, v, s)$ and [2] for WJ for parallel tasks

We will now discuss how to modify this schedulability test and then rewrite it as MILP. Note that evaluating $\text{f}(\tau, m, s)$ in Fig. 3 is non-trivial because it requires that one finds a σ such that a certain condition is true and here σ is a real number. If instead, we change this expression so that we only check a finite number of possible values of σ then we get a schedulability test with a slight increase in pessimism but it comes with the benefit of being easier to express as MILP. We choose a value of K that is a positive integer (e.g. $K = 20$) and check only those σ such that there is a $k \in \{1..K\}$ such that $\sigma = (k/K) \times s$. This yields:

$$\text{h}^*(\tau, m, s, k, K, t) \overset{\text{def}}{=} (\sum_{\tau_i \in \tau} \text{ffdbf}(\tau_i, t, \frac{k}{K}, s) \leq (m - (m-1) \times \frac{k}{K}) \times t \times s)$$

$$\text{f}^*(\tau, m, s, K) \overset{\text{def}}{=} \left(\exists k \in \{1..K\} \text{ s.t. } (\frac{k \times s}{K} \geq \max_{\tau_i \in \tau} \frac{\eta_i}{D_i}) \wedge (\forall t \geq 0 : \text{h}^*(\tau, m, s, k, K, t))\right)$$

$\text{f}^*(\tau, m, s, K) \Rightarrow \tau$ is gEDF schedulable on m processors of speed s if tasks do not experience memory contention

Clearly, $t = \lfloor t/P \rfloor \times P + t \bmod P$. Thus $\sum_{\tau_i \in \tau} \text{ffdbf}(\tau_i, t, \frac{k}{K}, s) = \lfloor t/P \rfloor \times P \times (\sum_{\tau_i \in \tau} C_i/T_i) + \sum_{\tau_i \in \tau} \text{ffdbf}(\tau_i, t \bmod P, \frac{k}{K}, s)$. Thus, $(\forall t \geq 0 : \text{h}^*(\tau, m, s, k, K, t))$ can be evaluated by only considering $t \leq P$. Hence, $\text{f}^*(\tau, m, s, K)$ is true if and only if the following is satisfiable: $\sum_{k=1}^{K} \text{wi}_k \geq 1$ and

$$\forall k \in \{1..K\} : \text{wi}_k \in \{0, 1\}$$
$$\forall \langle i, k \rangle \text{ s.t. } (\tau_i \in \tau) \wedge (k \in \{1..K\}) : (\text{wi}_k = 1) \Rightarrow (\eta_i \leq \frac{s \times k \times D_i}{K})$$
$$\forall k \in \{1..K\} : (\text{wi}_k = 1) \Rightarrow (\forall t \in [0, P] : \text{h}^*(\tau, m, s, k, K, t))$$

(In the above expression, wi_k should be read as witness.) Observe that lhs of the inequality defining $\text{h}^*(\tau, m, s, k, K, t)$ is a piecewise linear function of t and rhs of the inequality defining $\text{h}^*(\tau, m, s, k, K, t)$ is a linear function of t. Hence, when evaluating $(\forall t \in [0, P] : \text{h}^*(\tau, m, s, k, K, t))$ it is only necessary to evaluate $\text{h}^*(\tau, m, s, k, K, t)$ for the following values of t: (i) values of t such that the derivative of the piecewise linear function changes, (ii) $t = P$, and (iii) $t = 0$. With respect to (iii), note that $\text{h}^*(\tau, m, s, k, K, 0)$ is true and hence it does not need to be checked. With respect to (ii), note that $\text{h}^*(\tau, m, s, k, K, P)$ can be rewritten as $((\sum_{\tau_i \in \tau} C_i/T_i) \leq (m-(m-1) \times (k/K)) \times s)$. With respect to (i), note that for t such that there is a positive integer q' and a task $\tau_{i'} \in \tau$ such that $t = (q'-1) \times T_{i'} + D_{i'} - (\eta_{i'}/s) \times (K/k)$, the above mentioned derivative changes but this t is dominated by another t in the condition and hence, this t does not need to be checked. Hence, $\text{f}^*(\tau, m, s, K)$ is true if and only if the following is satisfiable: $\sum_{k=1}^{K} \text{wi}_k \geq 1$ and

$$\forall k \in \{1..K\} : \text{wi}_k \in \{0, 1\}$$
$$\forall \langle i, k \rangle \text{ s.t. } (\tau_i \in \tau) \wedge (k \in \{1..K\}) : (\text{wi}_k = 1) \Rightarrow (\eta_i \leq \frac{s \times k \times D_i}{K})$$
$$\forall \langle i', q', j', f', k \rangle \text{ s.t. } (\tau_{i'} \in \tau) \wedge (q' \in \{1..\frac{P}{T_{i'}}\}) \wedge (j' \in \{1..\text{ns}_{i'}\}) \wedge (f' \in \{0, 1\}) \wedge (k \in \{1..K\}) : t_{i',q',j',f',k} = (q'-1) \times T_{i'} + D_{i'} - ((\sum_{j'' \in \{1..j'-1\}} \text{sp}_{i',j''}) + f' \times \text{bsp}_{i',j'}) \times \frac{K}{k}$$
$$\forall \langle i', q', j', f', k \rangle \text{ s.t. } (\tau_{i'} \in \tau) \wedge (q' \in \{1..\frac{P}{T_{i'}}\}) \wedge (j' \in \{1..\text{ns}_{i'}\}) \wedge (f' \in \{0, 1\}) \wedge (k \in \{1..K\}) : (\text{wi}_k = 1) \Rightarrow (\sum_{\tau_i \in \tau} \text{ffdbf}(\tau_i, t_{i',q',j',f',k}, \frac{k}{K}, s) \leq (m - (m-1) \times \frac{k}{K}) \times t_{i',q',j',f',k} \times s)$$

Note that in the expressions above, we use primed variables to indicate variables that are used for computing t. For example, $i = 3$ means that we are computing ffdbf for task τ_3. But $i' = 3$ means that we are computing a t and it is approximately three times T_3.

We will now rewrite $\text{ffdbf}(\tau_i, t_{i',q',j',f',k}, \frac{k}{K}, s)$ to a form closer to MILP. Define $I_{i,q,i',q',j',f',k}$ so that $I_{i,q,i',q',j',f',k} = 1$ if $\lfloor t_{i',q',j',f',k}/T_i \rfloor = q$; otherwise $I_{i,q,i',q',j',f',k} = 0$. Define $r_{i,i',q',j',f',k} = t_{i',q',j',f',k} \bmod T_i$. Define $\text{aj}_{i,i',q',j',f',k}$ so that $(I_{i,q,i',q',j',f',k} = 1) \Rightarrow (\text{aj}_{i,i',q',j',f',k} = (q+1) \times C_i)$. (Intuitively, $I_{i,q,i',q',j',f',k}$ can be read as an integer num-

$$\mathrm{cu}_i = \sum_{j=1}^{\mathrm{ns}_i} \left(\mathrm{nseg_{i,j}} \times \mathrm{cu}_{i,j}\right) \qquad \mathrm{etau}_i = \sum_{j=1}^{\mathrm{ns}_i} \left(\lceil \frac{\mathrm{nseg_{i,j}}}{m} \rceil \times \mathrm{cu}_{i,j}\right) \qquad \mathrm{bspu}_{i,j} = \frac{\mathrm{cu}_{i,j}}{s} \times \lfloor \frac{\mathrm{nseg}_{i,j}}{m} \rfloor \qquad \mathrm{spu}_{i,j} = \frac{\mathrm{cu}_{i,j}}{s} \times \lceil \frac{\mathrm{nseg}_{i,j}}{m} \rceil \tag{1}$$

$$\forall \langle i', q', j', f', k \rangle : \ (\mathrm{wi}_k = 1) \Rightarrow (t_{i',q',j',f',k} = (q'-1) \times T_{i'} + D_{i'} - ((\sum_{j'' \in \{1..j'-1\}} \mathrm{spu}_{i',j''}) + f' \times \mathrm{bspu}_{i',j'}) \times (K/k)) \tag{2}$$

$$\forall \langle i, i', q', j', f', k \rangle : \ t_{i',q',j',f',k} = (\sum_{q \in \{1..P/T_i\}} I_{i,q,i',q',j',f',k} \times q \times T_i) + r_{i,i',q',j',f',k} \tag{3}$$

$$\forall \langle i, i', q', j', f', k \rangle : \ \sum_{q \in \{0..P/T_i\}} I_{i,q,i',q',j',f',k} = 1 \tag{4}$$

$$\forall \langle i, i', q', j', f', k \rangle : \ r_{i,i',q',j',f',k} \leq T_i \tag{5}$$

$$\forall \langle i, q, i', q', j', f', k \rangle : \ (I_{i,q,i',q',j',f',k} = 1) \Rightarrow (\mathrm{aj}_{i,i',q',j',f',k} = (q+1) \times \mathrm{cu}_i) \tag{6}$$

$$\forall \langle i, i', q', j', f', k \rangle : \ \mathrm{fi}_{i,i',q',j',f',k} + (\sum_{j \in \{1..\mathrm{ns}_i\}} \mathrm{sf}_{i,j,i',q',j',f',k}) + (\sum_{j \in \{1..\mathrm{ns}_i\}} \mathrm{ss}_{i,j,i',q',j',f',k}) + \mathrm{th}_{i,i',q',j',f',k} = 1 \tag{7}$$

$$\forall \langle i, i', q', j', f', k \rangle : \ (\mathrm{fi}_{i,i',q',j',f',k} = 1) \Rightarrow ((D_i - r_{i,i',q',j',f',k}) \times (k/K) \leq 0) \tag{8}$$

$$\forall \langle i, j, i', q', j', f', k \rangle : \ (\mathrm{sf}_{i,j,i',q',j',f',k} = 1) \Rightarrow (\sum_{j'' \in \{1..j-1\}} \mathrm{spu}_{i,j''} \leq (D_i - r_{i,i',q',j',f',k}) \times \frac{k}{K} \leq (\sum_{j'' \in \{1..j-1\}} \mathrm{spu}_{i,j''}) + \mathrm{bspu}_{i,j}) \tag{9}$$

$$\forall \langle i, j, i', q', j', f', k \rangle : \ (\mathrm{ss}_{i,j,i',q',j',f',k} = 1) \Rightarrow ((\sum_{j'' \in \{1..j-1\}} \mathrm{spu}_{i,j''}) + \mathrm{bspu}_{i,j} \leq (D_i - r_{i,i',q',j',f',k}) \times \frac{k}{K} \leq \sum_{j'' \in \{1..j\}} \mathrm{spu}_{i,j''}) \tag{10}$$

$$\forall \langle i, i', q', j', f', k \rangle : \ (\mathrm{th}_{i,i',q',j',f',k} = 1) \Rightarrow (\sum_{j'' \in \{1..\mathrm{ns}_i\}} \mathrm{spu}_{i,j''} \leq (D_i - r_{i,i',q',j',f',k}) \times (k/K)) \tag{11}$$

$$\forall \langle i, i', q', j', f', k \rangle : \ (\mathrm{fi}_{i,i',q',j',f',k} = 1) \Rightarrow (\mathrm{w}_{i,i',q',j',f',k} = 0) \tag{12}$$

$$\forall \langle i, j, i', q', j', f', k \rangle : \ (\mathrm{sf}_{i,j,i',q',j',f',k} = 1) \Rightarrow (\mathrm{w}_{i,i',q',j',f',k} = (\sum_{j'' \in \{1..j-1\}} \mathrm{nseg}_{i,j''} \times \mathrm{cu}_{i,j''}) +$$

$$((D_i - r_{i,i',q',j',f',k}) \times (k/K) - (\sum_{j'' \in \{1..j-1\}} \mathrm{spu}_{i,j''})) \times m \times s) \tag{13}$$

$$\forall \langle i, j, i', q', j', f', k \rangle : \ (\mathrm{ss}_{i,j,i',q',j',f',k} = 1) \Rightarrow (\mathrm{w}_{i,i',q',j',f',k} = (\sum_{j'' \in \{1..j-1\}} \mathrm{nseg}_{i,j''} \times \mathrm{cu}_{i,j''}) +$$

$$\mathrm{bspu}_{i,j} \times m \times s + ((D_i - r_{i,i',q',j',f',k}) \times (k/K) - (\sum_{j'' \in \{1..j-1\}} \mathrm{spu}_{i,j''}) - \mathrm{bspu}_{i,j}) \times (\mathrm{nseg}_{i,j} \bmod m) \times s) \tag{14}$$

$$\forall \langle i, i', q', j', f', k : (\mathrm{th}_{i,i',q',j',f',k} = 1) \Rightarrow (\mathrm{w}_{i,i',q',j',f',k} = \mathrm{cu}_i) \tag{15}$$

$$\forall \langle i', q', j', f', k \rangle : \ (\mathrm{wi}_k = 1) \Rightarrow ((\sum_{\tau_i \in \tau} (\mathrm{aj}_{i,i',q',j',f',k} - w_{i,i',q',j',f',k})) \leq (m - (m-1) \times (k/K)) \times t_{i',q',j',f',k} \times s) \tag{16}$$

$$\forall k : (\mathrm{wi}_k = 1) \Rightarrow ((\sum_{\tau_i \in \tau} \mathrm{cu}_i / T_i) \leq (m - (m-1) \times (k/K)) \times s) \tag{17}$$

$$\forall \langle i, k \rangle : \ (\mathrm{wi}_k = 1) \Rightarrow (\mathrm{etau}_i \leq (k/K) \times s \times D_i) \tag{18}$$

$$\sum_{k=1}^{K} \mathrm{wi}_k \geq 1 \tag{19}$$

Figure 4: Schedulability analysis for gEDF scheduling of parallel tasks formulated as a MILP.

$$\mathrm{cu}_{i,j} \in \mathbb{R}_{\geq 0}, \mathrm{cu}_i \in \mathbb{R}_{\geq 0}, \mathrm{etau}_i \in \mathbb{R}_{\geq 0}, \mathrm{bspu}_{i,j} \in \mathbb{R}_{\geq 0}, \mathrm{spu}_{i,j} \in \mathbb{R}_{\geq 0}, t_{i',q',j',f',k} \in \mathbb{R}_{\geq 0}, I_{i,q,i',q',j',f',k} \in \{0,1\}, r_{i,i',q',j',f',k} \in \mathbb{R}_{\geq 0},$$
$$\mathrm{aj}_{i,i',q',j',f',k} \in \mathbb{R}_{\geq 0}, \mathrm{fi}_{i,i',q',j',f',k} \in \{0,1\}, \mathrm{sf}_{i,j,i',q',j',f',k} \in \{0,1\}, \mathrm{ss}_{i,j,i',q',j',f',k} \in \{0,1\}, \mathrm{th}_{i,i',q',j',f',k} \in \{0,1\}, \mathrm{w}_{i,i',q',j',f',k} \in \mathbb{R}_{\geq 0},$$
$$\mathrm{wi}_k \in \{0,1\}, \mathrm{mb}_{i,j,g,b} \in \mathbb{Z}_{\geq 0}, \mathrm{mmbo}_{i,j,g,b} \in \mathbb{Z}_{\geq 0}, \mathrm{o}_{i,j,g,p,h,b} \in \{0,1\}, \mathrm{ino}_{h,b} \in \mathbb{Z}_{\geq 0}, \mathrm{x}_{i,j,g,h} \in \{0,1\}, \mathrm{cm}_{i,j,g} \in \mathbb{R}_{\geq 0}, \mathrm{se}_{i,j,g} \in \{0,1\},$$
$$\mathrm{b1}_{i,j,g,b} \in \{0,1\}, \mathrm{b2}_{i,j,g,b} \in \{0,1\}, \mathrm{b3}_{i,j,g,b} \in \{0,1\}, \mathrm{b4}_{i,j,g,b} \in \{0,1\}, \mathrm{bu}_{i,j,g,b} \in \{0,1\}, \mathrm{coat}_{i,j,g,b} \in \mathbb{R}_{\geq 0}, \mathrm{oao}_{i,j,g,b} \in \mathbb{Z}_{\geq 0}, \mathrm{oat}_{i,j,g,b} \in \mathbb{Z}_{\geq 0}$$
$$\tau_i \in \tau, j \in \{1..\mathrm{ns}_i\}, g \in \{1..\mathrm{nseg}_{i,j}\}, p \in \{0..\mathrm{np}_{i,j}-1\}, q \in \{1..P/T_i\}, \tau_{i'} \in \tau, j' \in \{1..\mathrm{ns}_{i'}\}, g' \in \{1..\mathrm{nseg}_{i',j'}\},$$
$$p' \in \{0..\mathrm{np}_{i',j'}-1\}, q' \in \{1..P/T_{i'}\}, f' \in \{0,1\}, k \in \{1..K\}, h \in \{0..H-1\}, b \in \{0..B-1\}$$

Figure 5: Domains of variables and domains of indices. Here we also show domains of variables that will be used in later sections.

ber of jobs of task τ_i for a time interval of duration t. The symbol $r_{i,i',q',j',f',k}$ can be read as $\underline{r}$emainder of time. And $\text{aj}_{i,i',q',j',f',k}$ can be read as execution from $\underline{a}$ll integer $\underline{j}$obs of task τ_i.) Then, rewrite $\text{ffdbf}(\tau_i, t_{i',q',j',f',k}, \frac{k}{K}, s)$ as:

$$\text{aj}_{i,i',q',j',f',k} - \text{WJ}(\tau_i, (D_i - r_{i,i',q',j',f',k}) \times \frac{k}{K}, s)$$

Consider $\text{WJ}(\tau_i, (D_i - r_{i,i',q',j',f',k}) \times \frac{k}{K}, s)$ in the expression above. We will now discuss how to rewrite it so that it is on a form closer to a MILP. One can observe (from Fig. 3) that WJ is computed differently for three different cases. Also, for the second case, it is necessary to compute WJS and this is done through recursion; hence we need to know for which value of j this recursion terminates. Therefore, we introduce fi,sf,ss, and th as variables that indicate if a certain condition is true. If the condition is true then the variable is 1; otherwise the variable is 0. $\text{fi}_{i,i',q',j',f',k}$ indicates that when $\text{WJ}(\tau_i, (D_i - r_{i,i',q',j',f',k}) \times \frac{k}{K}, s)$ is called, the $\underline{f}$irst case in the definition of WJ is taken. $\text{sf}_{i,j,i',q',j',f',k}$ indicates that when $\text{WJ}(\tau_i, (D_i - r_{i,i',q',j',f',k}) \times \frac{k}{K}, s)$ is called, the $\underline{s}$econd case in the definition of WJ is taken and WJS terminates with 3rd parameter taking the value j and the $\underline{f}$irst case in definition of WJS is taken. $\text{ss}_{i,j,i',q',j',f',k}$ indicates that when $\text{WJ}(\tau_i, (D_i - r_{i,i',q',j',f',k}) \times \frac{k}{K}, s)$ is called, the $\underline{s}$econd case in the definition of WJ is taken and WJS terminates with 3rd parameter taking the value j and the $\underline{s}$econd case in definition of WJS is taken. $\text{th}_{i,i',q',j',f',k}$ indicates that when $\text{WJ}(\tau_i, (D_i - r_{i,i',q',j',f',k}) \times \frac{k}{K}, s)$ is called, the $\underline{t}$hird case in the definition of WJ is taken. Since exactly one of them is true, it holds that:

$$\text{fi}_{i,i',q',j',f',k} + (\sum_{j \in \{1..\text{ns}_i\}} \text{sf}_{i,j,i',q',j',f',k}) +$$
$$(\sum_{j \in \{1..\text{ns}_i\}} \text{ss}_{i,j,i',q',j',f',k}) + \text{th}_{i,i',q',j',f',k} = 1$$

For convenience, we can also introduce the variable $w_{i,i',q',j',f',k}$ to mean $\text{WJ}(\tau_i, (D_i - r_{i,i',q',j',f',k}) \times \frac{k}{K}, s)$. With this notation, we obtain:

$$(\text{fi}_{i,i',q',j',f',k} = 1) \Rightarrow ((D_i - r_{i,i',q',j',f',k}) \times \frac{k}{K} \le 0)$$
$$(\text{fi}_{i,i',q',j',f',k} = 1) \Rightarrow (w_{i,i',q',j',f',k} = 0)$$

We obtain similar expressions for sf,ss, and th.

The above reasoning yields that $\text{f}^*(\tau, m, s, K)$ is true if and only if there is an assignment of values to variables such that the constraints in Fig. 4 are satisfied and the domains of the variables are as given by Fig. 5. In Fig. 3, $C_{i,j}$ denotes the upper bound on the execution requirement of a segment in stage j of τ_i but in Fig. 4 $\text{cu}_{i,j}$ denotes this. ($\text{cu}_{i,j}$ means exe$\underline{c}$ution requirement that we will $\underline{u}$se.)

4. MEMORY CONTENTION

Previous work [14] offers a method for computing an upper bound on the response time of a task considering contention for resources in the memory system. That method assumes fixed-priority preemptive non-migrative scheduling and integrates memory contention analysis in the schedulability test. In this section, we will adapt this memory contention analysis (i) to compute an upper bound on the extra execution of a segment of a single job of a task without assuming any specific processor scheduler, (ii) using our more advanced model for discussing memory accesses (page-level), and (iii) expressing it on a form easily translatable to MILP.

$\text{cm}_{i,j,g}$ denotes an upper bound on the exe$\underline{c}$ution requirement of $\text{seg}(i,j,g)$ considering contention for resources in the $\underline{m}$emory system (the extra execution of this contention is considered to be part of the execution requirement). Also, recall that $\text{cu}_{i,j}$ was defined in Section 3. We will now redefine it. $\text{cu}_{i,j}$ denotes an upper bound on the execution requirement of a segment of stage j of τ_i considering contention for resources in the memory system (the extra execution of this contention is considered to be part of the execution requirement). Hence:

$$\text{cu}_{i,j} = \max_{g \in \{1..\text{nseg}_{i,j}\}} \text{cm}_{i,j,g} \qquad (20)$$

(30) in Fig. 7 expresses (20).

Let $\text{o}_{i,j,g,p,h,b} = 1$ indicate that $\text{page}(i,j,g,p)$ is mapped to a memory frame with cache color h and MB color b; otherwise $\text{o}_{i,j,g,p,h,b} = 0$. Clearly, a page can only be mapped to one frame and one frame belongs to exactly one cache color and one MB color. Hence, each page belongs to exactly one combination of cache and MB color. This yields (24). Also, if a cache and MB color is given then the number of pages that can be mapped to this combination of cache and MB color cannot exceed its amount of physical memory. For the special case that there are no shared frames and there was no memory required during initialization, considering those pages that are mapped to frames of cache color h and bank color h, we can express the limited memory capacity as:

$$\sum_{\tau_i \in \tau} \sum_{j \in \{1..\text{ns}_i\}} \sum_{g \in \{1..\text{nseg}_{i,j}\}} \sum_{p \in \{0..\text{np}_{i,j}-1\}} \text{o}_{i,j,g,p,h,b} \le \text{CAP}$$

Let us now discuss how to express the constraint of limited memory capacity for the general case. Let $\text{GS}_{i,j,g,p}$ be a constant that indicates how many pages maps to the same frame as $\text{page}(i,j,g,p)$ maps to. For normal pages, it holds that $\text{GS}_{i,j,g,p} = 1$ but if a page maps to a shared frame then $\text{GS}_{i,j,g,p}$ is larger. $\text{GS}_{i,j,g,p}$ can be computed as follows. Form a graph, with one vertex for each each $\langle i,j,g,p \rangle$ and there is an edge between two vertices $\langle i,j,g,p \rangle$ and $\langle i',j',g',p' \rangle$ if $\langle i,j,g,p,i',j',g',p' \rangle \in \text{shfr}$. Compute the connected components of the graph. Then, for $\langle i,j,g,p \rangle$, let $\text{GSVS}_{i,j,g,p}$ denote the set of vertices in the connected component to which the vertex corresponding to $\langle i,j,g,p \rangle$ belong. Let $\text{GS}_{i,j,g,p}$ denote the cardinality of $\text{GSVS}_{i,j,g,p}$. Then, if we consider a page $\text{page}(i,j,g,p)$ and all other pages that map to the same frame then we know that all of them only consume a single frame. We can model that as if each of them consumed $\frac{1}{\text{GS}_{i,j,g,p}}$ frame. With this observation and letting $\text{ino}_{h,b}$ indicates the number of pages accessed during initialization that maps to frames of which belong to cache color h and MB color b, we obtain that the limited memory capacity can be expressed by (25). In addition, the requirement on shared frames, expressed by the set shfr, yields (31).

For a pair of segments that could possibly execute in parallel, we require that the VPAT is set up so that one segment cannot evict a cache block that another segment has fetched to the cache. We can express it as follows: $x_{i,j,g,h} = 1$ indicates that $\text{seg}(i,j,g)$ uses cache color h; otherwise $x_{i,j,g,h} = 0$. Then, if $\text{seg}(i,j,g)$ can execute in parallel with $\text{seg}(i',j',g')$, cache coloring requires: $\text{x}_{i,j,g,h} + \text{x}_{i',j',g',h} \le 1$, where $(x_{i,j,g,h} = 1) \Leftrightarrow (\sum_{p \in \{0..\text{np}_{i,j}-1\}} \sum_{b \in \{0..B-1\}} \text{o}_{i,j,g,p,h,b} \ge 1)$. Rewriting these to a form close to MILP yields (27),(28), and (29).

Let $\text{mb}_{i,j,g,b}$ be an upper bound on the number of $\underline{m}$emory accesses from $\text{seg}(i,j,g)$ to memory $\underline{b}$ank b. This yields (22). For a segment $\text{seg}(i,j,g)$, the symbol $\text{mmbo}_{i,i',j',g',b}$ is an upper bound on the number of $\underline{m}$emory accesses on memory $\underline{b}$ank b from $\underline{m}$ultiple jobs of all $\underline{o}$ther segments than $\text{seg}(i,j,g)$ such that these memory accesses can impact a job of $\text{seg}(i,j,g)$. (23) expresses it. In the proof of Theorem 1, we will show that it is an upper bound.

Now consider memory contention. Look at the queues inside the MC in Fig. 2. A single memory access accessing MB b can be delayed by the following:

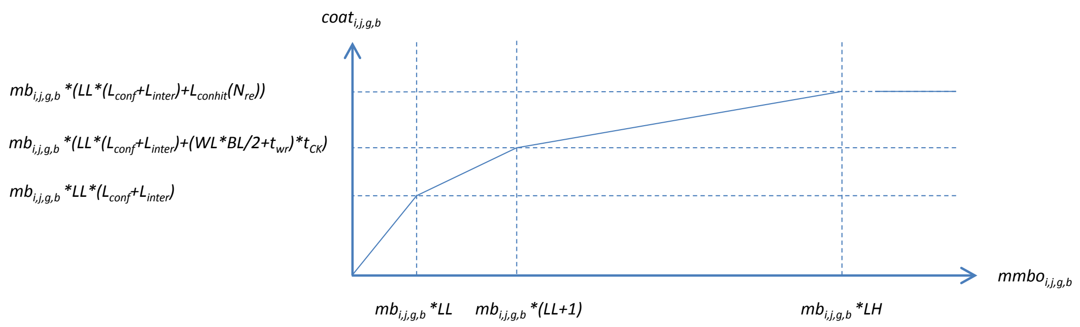


Figure 6: Contention on MB queue. The vertical axis shows an upper bound on the delay that the $\mathrm{mb}_{i,j,g,b}$ memory accesses from $\mathrm{seg}(i,j,g)$ can suffer from because the other $\mathrm{mmbo}_{i,j,g,b}$ memory accesses from other segments access memory of MB b and hence contend on the queue for MB b.

1. There are already memory accesses in the queue for MB b when this single memory access is inserted in the queue for MB b and because of FCFS queuing, these other memory accesses are served first.
2. After this single memory access is enqueued in the queue for MB b, there are other memory accesses enqueued in the queue for this MB and these other memory accesses' row is currently loaded in the row buffer and hence they get ahead in the queue for this MB (reordering).
3. When one of the other memory accesses mentioned in 1) or 2) reaches the head of the queue of MB b, it is not served immediately; instead it has to wait for the memory bus being granted and this takes time because other memory accesses in the queues to other MBs than MB b use the memory bus.

Consider $\mathrm{seg}(i,j,g)$ and its (at most $\mathrm{mb}_{i,j,g,b}$) memory accesses that it performs on memory locations of MB b and how the three effects increase the execution time of $\mathrm{seg}(i,j,g)$. Let $\mathrm{coat}_{i,j,g,b}$ denote an upper bound on the extra exe<u>c</u>ution time of this segment because <u>o</u>ther memory <u>a</u>ccesses (from other segments) access <u>t</u>his MB (MB b). (This includes 1. and 2. above.) Let $\mathrm{oao}_{i,j,g,b}$ denote an upper bound on the number of other memory accesses that causes extra execution time of this segment because <u>o</u>ther memory <u>a</u>ccesses (from other segments) access <u>o</u>ther MBs (than MB b). (This includes 3. above.) Each of those $\mathrm{oao}_{i,j,g,b}$ accesses can delay the execution of $\mathrm{seg}(i,j,g)$ for at most L_{inter} time units. Hence, the extra execution of $\mathrm{seg}(i,j,g)$ because of memory contention is at most $\mathrm{coat}_{i,j,g,b} + L_{\mathrm{inter}} \times \mathrm{oao}_{i,j,g,b}$ This can be added up for all memory banks. Then multiplying by s yields an upper bound on the number of units of extra execution that $\mathrm{seg}(i,j,g)$ needs because of memory contention. Hence:

$$\mathrm{cm}_{i,j,g} = \mathrm{C}_{i,j} + s \times (\sum_{b=0}^{B-1} (\mathrm{coat}_{i,j,g,b} + L_{\mathrm{inter}} \times \mathrm{oao}_{i,j,g,b})) \tag{21}$$

About 1) and 2) We will now discuss $\mathrm{coat}_{i,j,g,b}$. Since we assume a processor stalls until its memory access has been completed, it follows that from each processor, there can be at most one outstanding memory access and hence there are at most LL memory accesses of 1) above. The hardware places a limit on the number of reorderings that can happen. In previous work [14], we introduced the parameter to indicate an upper bound on the number of those reorderings that a single memory access can experience. In this paper, we let N_{re} denote this parameter; a typical value [14] is $N_{\mathrm{re}} = 12$. Consequently, the $\mathrm{mb}_{i,j,g,b}$ memory accesses from $\mathrm{seg}(i,j,g)$ performing on MB b has to wait for at most $\mathrm{mb}_{i,j,g,b} \times (\mathrm{LL} + N_{\mathrm{re}}) = \mathrm{mb}_{i,j,g,b} \times \mathrm{LH}$ other memory accesses performing on MB b (because of 1) and 2) above). Using mmbo yields that: The $\mathrm{mb}_{i,j,g,b}$ memory accesses from $\mathrm{seg}(i,j,g)$ performing on MB b has to wait for at most

$$\min(\mathrm{mb}_{i,j,g,b} \times \mathrm{LH}, \mathrm{mmbo}_{i,j,g,b}) \tag{47}$$

other memory accesses performing on MB b (because of 1) and 2) above). Let $\mathrm{oat}_{i,j,g,b}$ be the expression in (47). (It means <u>o</u>ther <u>a</u>ccesses to <u>t</u>his MB.) By inspecting $L_{\mathrm{conhit}}(x)$ and the parameters in Section 2, one can see that these memory accesses have different effects; the memory accesses that are in the queue before a memory access has arrived to the queue cause more interference than the ones that arrive later that cause reordering. Fig. 6 shows an upper bound. It gives us $\mathrm{coat}_{i,j,g,b}$.

About 3) We will now discuss $\mathrm{oao}_{i,j,g,b}$. A memory access related to MB b is inserted in the queue for the memory bus only if (i) this memory access is at the head of the queue of the MB b and (ii) there is no memory access related to MB b already in the queue of the memory bus. Hence, a memory access that has reached the head of the queue of its MB needs to wait for at most B-1 other memory accesses until it is granted the memory bus. Consequently, the $\mathrm{mb}_{i,j,g,b}$ memory accesses from $\mathrm{seg}(i,j,g)$ performing on MB b has to wait for at most $(\mathrm{mb}_{i,j,g,b} + \mathrm{oat}_{i,j,g,b}) \times (B-1)$ other memory accesses performing on MB b (because of 3)). Using mmbo yields that: The $\mathrm{mb}_{i,j,g,b}$ memory accesses from $\mathrm{seg}(i,j,g)$ performing on MB b has to wait for at most

$$\min\big((\mathrm{mb}_{i,j,g,b} + \mathrm{oat}_{i,j,g,b}) \times (B-1), \sum_{b'' \in \{0..B-1\} \wedge (b'' \neq b)} \mathrm{mmbo}_{i,j,g,b''}\big) \tag{48}$$

other memory accesses performing on other MBs than MB b (because of 3) above). (48) expresses $\mathrm{oao}_{i,j,g,b}$.

This reasoning yields an upper bound on the execution requirement on a form close to MILP — see Fig. 7.

5. THE MILP FORMULATION

Let Π denote the computer platform (the parameters m, s, H, B and the parameters describing the memory system). $\mathrm{fmem}(\tau, \Pi, K)$ is a function which returns the tuple $\langle \mathrm{flag}, o \rangle$ where flag is a boolean and o is a multi-dimensional array. If there exists an assignment of values to the variables so that the constraints in Fig. 4 and Fig. 7 are satisfied then flag is true and o is the values of the o-variables; otherwise flag is false and o is undefined.

$$\forall\langle i,j,g,b\rangle:\ \mathrm{mb}_{i,j,g,b} = \sum_{p\in\{0..\mathrm{np}_{i,j}-1\}} \sum_{h\in\{0..H-1\}} \mathrm{MA}_{i,j,p} \times \mathrm{o}_{i,j,g,p,h,b} \tag{22}$$

$$\forall\langle i,j,g,b\rangle:\ \mathrm{mmbo}_{i,j,g,b} = \sum_{\tau_{i'}\in\tau} \sum_{j'\in\{1..\mathrm{ns}_{i'}\}} \sum_{g'\in\{1..\mathrm{nseg}_{i',j'}\}\wedge(((i'=i)\wedge(j'=j)\wedge(g'\neq g))\vee(i'\neq i))} (\lceil\frac{D_i}{T'_i}\rceil + 1) \times \mathrm{mb}_{i',j',g',b} \tag{23}$$

$$\forall\langle i,j,g,p\rangle:\ \sum_{h\in\{0..H-1\}} \sum_{b\in\{0..B-1\}} \mathrm{o}_{i,j,g,p,h,b} = 1 \tag{24}$$

$$\forall\langle h,b\rangle:\ (\sum_{\tau_i\in\tau} \sum_{j\in\{1..\mathrm{ns}_i\}} \sum_{g\in\{1..\mathrm{nseg}_{i,j}\}} \sum_{p\in\{0..\mathrm{np}_{i,j}-1\}} (\frac{1}{\mathrm{GS}_{i,j,g,p}} \times \mathrm{o}_{i,j,g,p,h,b})) + \mathrm{ino}_{h,b} \leq \mathrm{CAP} \tag{25}$$

$$\sum_{h\in\{0..H-1\}} \sum_{b\in\{0..B-1\}} \mathrm{ino}_{h,b} = \mathrm{INO} \tag{26}$$

$$\forall\langle i,j,g,h\rangle:\ (\mathrm{x}_{i,j,g,h} = 1) \Rightarrow (\sum_{p\in\{0..\mathrm{np}_{i,j}-1\}} \sum_{b\in\{0..B-1\}} \mathrm{o}_{i,j,g,p,h,b} \geq 1) \tag{27}$$

$$\forall\langle i,j,g,h\rangle:\ (\mathrm{x}_{i,j,g,h} = 0) \Rightarrow (\sum_{p\in\{0..\mathrm{np}_{i,j}-1\}} \sum_{b\in\{0..B-1\}} \mathrm{o}_{i,j,g,p,h,b} \leq 0) \tag{28}$$

$$\forall\langle i,j,g,i',j',g',h\rangle\ \text{ s.t. } (((i=i')\wedge(j=j')\wedge(g<g'))\vee(i<i')) : \mathrm{x}_{i,j,g,h} + \mathrm{x}_{i',j',g',h} \leq 1 \tag{29}$$

$$\forall\langle i,j,g\rangle:\ \mathrm{cm}_{i,j,g} \leq \mathrm{cu}_{i,j} \qquad \forall\langle i,j\rangle:\ \sum_{g=1}^{\mathrm{nseg}_{i,j}} \mathrm{se}_{i,j,g} = 1 \qquad \forall\langle i,j,g\rangle:\ (\mathrm{se}_{i,j,g} = 1) \Rightarrow (\mathrm{cm}_{i,j,g} \geq \mathrm{cu}_{i,j}) \tag{30}$$

$$\forall\langle i,j,g,p,i',j',g',p',h,b\rangle\ \text{ s.t. } \langle i,j,g,p,i',j',g',p'\rangle \in \mathrm{shfr} : o_{i,j,g,p,h,b} = o_{i',j',g',p',h,b} \tag{31}$$

$$\forall\langle i,j,g\rangle:\ \mathrm{cm}_{i,j,g} = \mathrm{C}_{i,j} + s \times (\sum_{b\in\{0..B-1\}} (\mathrm{coat}_{i,j,g,b} + L_{\mathrm{inter}} \times \mathrm{oao}_{i,j,g,b})) \tag{32}$$

$$\forall\langle i,j,g,b\rangle:\ \mathrm{bc1}_{i,j,g,b} + \mathrm{bc2}_{i,j,g,b} + \mathrm{bc3}_{i,j,g,b} + \mathrm{bc4}_{i,j,g,b} = 1 \tag{33}$$

$$\forall\langle i,j,g,b\rangle:\ (\mathrm{bc1}_{i,j,g,b} = 1) \Rightarrow (\mathrm{mmbo}_{i,j,g,b} \leq \mathrm{mb}_{i,j,g,b} \times \mathrm{LL}) \tag{34}$$

$$\forall\langle i,j,g,b\rangle:\ (\mathrm{bc2}_{i,j,g,b} = 1) \Rightarrow (\mathrm{mb}_{i,j,g,b} \times \mathrm{LL} \leq \mathrm{mmbo}_{i,j,g,b} \leq \mathrm{mb}_{i,j,g,b} \times (\mathrm{LL}+1)) \tag{35}$$

$$\forall\langle i,j,g,b\rangle:\ (\mathrm{bc3}_{i,j,g,b} = 1) \Rightarrow (\mathrm{mb}_{i,j,g,b} \times (\mathrm{LL}+1) \leq \mathrm{mmbo}_{i,j,g,b} \leq \mathrm{mb}_{i,j,g,b} \times \mathrm{LH}) \tag{36}$$

$$\forall\langle i,j,g,b\rangle:\ (\mathrm{bc4}_{i,j,g,b} = 1) \Rightarrow (\mathrm{mb}_{i,j,g,b} \times \mathrm{LH} \leq \mathrm{mmbo}_{i,j,g,b}) \tag{37}$$

$$\forall\langle i,j,g,b\rangle:\ (\mathrm{bc1}_{i,j,g,b} = 1) \Rightarrow (\mathrm{coat}_{i,j,g,b} = \mathrm{mmbo}_{i,j,g,b} \times (L_{\mathrm{conf}} + L_{\mathrm{inter}})) \tag{38}$$

$$\begin{aligned}\forall\langle i,j,g,b\rangle:\ (\mathrm{bc2}_{i,j,g,b} = 1) \Rightarrow (\mathrm{coat}_{i,j,g,b} = \mathrm{mb}_{i,j,g,b} \times \mathrm{LL} \times (L_{\mathrm{conf}} + L_{\mathrm{inter}}) + \\ (\mathrm{mmbo}_{i,j,g,b} - \mathrm{mb}_{i,j,g,b} \times \mathrm{LL}) \times (\mathrm{WL} + \mathrm{BL}/2 + t_{\mathrm{WR}}) \times t_{CK})\end{aligned} \tag{39}$$

$$\begin{aligned}\forall\langle i,j,g,b\rangle:\ (\mathrm{bc3}_{i,j,g,b} = 1) \Rightarrow (\mathrm{coat}_{i,j,g,b} = \mathrm{mb}_{i,j,g,b} \times \mathrm{LL} \times (L_{\mathrm{conf}} + L_{\mathrm{inter}}) + \mathrm{mb}_{i,j,g,b} \times (\mathrm{WL} + \mathrm{BL}/2 + t_{\mathrm{WR}}) \times t_{CK} + \\ (\mathrm{mmbo}_{i,j,g,b} - \mathrm{mb}_{i,j,g,b} \times (\mathrm{LL}+1)) \times (\mathrm{WL} + \mathrm{BL}/2 + t_{\mathrm{WR}} + \mathrm{CL}) \times (1/2) \times t_{CK})\end{aligned} \tag{40}$$

$$\forall\langle i,j,g,b\rangle:\ (\mathrm{bc4}_{i,j,g,b} = 1) \Rightarrow (\mathrm{coat}_{i,j,g,b} = \mathrm{mb}_{i,j,g,b} \times (\mathrm{LL} \times (L_{\mathrm{conf}} + L_{\mathrm{inter}}) + L_{\mathrm{conhit}}(N_{\mathrm{re}}))) \tag{41}$$

$$\forall\langle i,j,g,b\rangle:\ (\mathrm{bc4}_{i,j,g,b} = 0) \Rightarrow (\mathrm{oat}_{i,j,g,b} = \mathrm{mmbo}_{i,j,g,b}) \qquad \forall\langle i,j,g,b\rangle:\ (\mathrm{bc4}_{i,j,g,b} = 1) \Rightarrow (\mathrm{oat}_{i,j,g,b} = \mathrm{mb}_{i,j,g,b} \times \mathrm{LH}) \tag{42}$$

$$\forall\langle i,j,g,b\rangle:\ (\mathrm{bu}_{i,j,g,b} = 0) \Rightarrow (\sum_{b''\in\{0..B-1\}\wedge(b''\neq b)} \mathrm{mmbo}_{i,j,g,b''} \leq (\mathrm{mb}_{i,j,g,b} + \mathrm{oat}_{i,j,g,b}) \times (B-1)) \tag{43}$$

$$\forall\langle i,j,g,b\rangle:\ (\mathrm{bu}_{i,j,g,b} = 1) \Rightarrow (\sum_{b''\in\{0..B-1\}\wedge(b''\neq b)} \mathrm{mmbo}_{i,j,g,b''} \geq (\mathrm{mb}_{i,j,g,b} + \mathrm{oat}_{i,j,g,b}) \times (B-1)) \tag{44}$$

$$\forall\langle i,j,g,b\rangle:\ (\mathrm{bu}_{i,j,g,b} = 0) \Rightarrow (\mathrm{oao}_{i,j,g,b} = (\sum_{b''\in\{0..B-1\}\wedge(b''\neq b)} \mathrm{mmbo}_{i,j,g,b''})) \tag{45}$$

$$\forall\langle i,j,g,b\rangle:\ (\mathrm{bu}_{i,j,g,b} = 1) \Rightarrow (\mathrm{oao}_{i,j,g,b} = (\mathrm{mb}_{i,j,g,b} + \mathrm{oat}_{i,j,g,b}) \times (B-1)) \tag{46}$$

Figure 7: Expressing increased execution time.

THEOREM 1.

$((\langle \text{flag}, o\rangle = \text{fmem}(\tau, \Pi, K)) \wedge (\text{flag} = \text{true})) \Rightarrow$

τ is gEDF schedulable on m processors of speed s for the case that tasks experience memory contention and the VPAT conforms to o

PROOF. See Appendix of [1]. □

Some of the constraints mentioned are not MILP — they have binary variables and logical operators. We will now discuss how to convert them to MILP. A constraint of the form $(x = 1) \Rightarrow (a = b)$ can be rewritten as: $((x = 1) \Rightarrow (a \leq b)) \wedge ((x = 1) \Rightarrow (a \geq b))$. Note that if x is a variable with the domain $\{0, 1\}$ and a and b are non-negative real variables and BIG is a constant selected so that $a \leq$ BIG and $b \leq$ BIG, then a constraint $(x = 1) \Rightarrow (a \leq b)$ can be rewritten as

$$a - b + \text{BIG} \times x \leq \text{BIG} \tag{49}$$

Note that in (17), we can use BIG $= m$ and in (28), we can use BIG $= \text{np}_{i,j} \times B$ and (27) can be rewritten without BIG. Let us now discuss the other constraints. In a feasible solution to Fig. 4 and Fig. 7, the variables $\text{mb}_{i,j,g,b}$ and $\text{mmbo}_{i,j,g,b}$ are at most

$$\max_{\tau_i \in \tau}\left(\sum_{\tau_i' \in \tau}\left(\left(\left\lceil \frac{D_i}{T_i'}\right\rceil + 1\right) \times \right.\right.$$

$$\left.\left.\left(\sum_{j' \in \{1..\text{ns}_{i'}\}} \sum_{g' \in \{1..\text{nseg}_{i',j'}\}} \sum_{p' \in \{0..\text{np}_{i',j'}-1\}} \text{MA}_{i',j',p'}\right)\right)\right) \tag{50}$$

Hence, the lhs of the constraints (33)-(46) is at most

$$\max\Big(1, L_{\text{conf}} + L_{\text{inter}}\Big) \times ((50)) \tag{51}$$

Also, for each of the other constraints, the lhs is at most

$$(P + \text{DMAX}) \times m \times \max(1, s) \tag{52}$$

Applying the rewriting expressed by (49) (and minor variants of it), with BIG $= \max((51), (52))$, yields that all of our constraints can be converted to a MILP.

6. DISCUSSION

$\text{fmem}(\tau, \Pi, K)$ and the model it is based on has three shortcomings: (i) it outputs a mapping from a page to a cache color and MB color but we actually need a mapping from a page to a memory frame in PA, (ii) it assumes that $\text{MA}_{i,j,p}$ and $C_{i,j}$ are given and known, and (iii) it ignores the effect of eviction of cache blocks in a private cache. We can deal with (i) by simply solving the MILP and if it returns a tuple with the 1st parameter being true then a VPTA can be obtained from o (the 2nd parameter in the tuple) as follows:

```
1. for each ⟨$i, j, g, p, h, b$⟩ s.t. $o_{i,j,g,p,h,b} = 1$ do
2.  if ($\exists\langle i', j', g', p'\rangle$ s.t. ($\langle i, j, g, p, i', j', g', p'\rangle \in$ shfr)$\vee$
3.   ($\langle i',' j', g', p', i, j, g, p\rangle \in$ shfr)) and (for this
4.   $\langle i', j', g', p'\rangle$, it holds that page($i', j', g', p'$) has already
5.   been mapped to a frame) then
6.    map page($i, j, g, p$) to the same frame as
7.     page($i', j', g', p'$) is mapped to
8.  else
9.    find an unused memory frame in PA with cache
10.    color $h$ and MB $b$ and then map page($i, j, g, p$) to it
11.  end if
12. end for
```

	m=4	m=8
min	420	999
q1	7623	8199
median	7655	8253
q3	11251	15446
max	25792	62238

Table 2: Five number summary of the time required, in our experiments, for performing schedulability analysis and configuring memory (in seconds).

We can deal with (ii) by making an initial guess of the values of $\text{MA}_{i,j,p}$ and $C_{i,j}$ and then check if the guess is correct; if it is not, then refine the guess with data obtained from the checking procedure. Specifically, do it as follows. Guess values of $\text{MA}_{i,j,p}$ and $C_{i,j}$ and then call the function $\text{fmem}(\tau, \Pi, K)$ and then obtain a new o and then obtain an VPTA map from this o and then obtain $C_{i,j}(\text{map})$ and $\text{MA}_{i,j,p}(\text{map})$ and check if $C_{i,j}(\text{map}) \leq C_{i,j}$ and check if $\text{MA}_{i,j,p}(\text{map}) \leq \text{MA}_{i,j,p}$; if this check fails then use map and obtain the execution requirement and number of memory accesses and use that as a new guess of $\text{MA}_{i,j,p}$ and $C_{i,j}$. An algorithm based on these ideas is shown below:

```
1. Choose a value of $K$ (for example $K = 20$)
2. Choose a value of maxiter (for example maxiter = 3)
3. Choose one $o'$
4. for iter := 1 to maxiter do
5.  choose a VPAT map$'$ that conforms to $o'$
6.  $\forall i, j$ do
7.   obtain $C_{i,j}(\text{map}')$
8.   assign $C^{\text{guess}}_{i,j} := C_{i,j}(\text{map}')$
9.  $\forall i, j, p$ do
10.   obtain $\text{MA}_{i,j,p}(\text{map}')$
11.   assign $\text{MA}^{\text{guess}}_{i,j,p} := \text{MA}_{i,j,p}(\text{map}')$
12.  $\langle \text{flag}, o\rangle = \text{fmem}(\tau, \Pi, K)$; in this call, assume that
13.    $\forall i, j : C_{i,j} = C^{\text{guess}}_{i,j}$; $\forall i, j, p : \text{MA}_{i,j,p} = \text{MA}^{\text{guess}}_{i,j,p}$
14.  if flag then
15.    choose a VPAT map that conforms to $o$
16.    if ($\forall i, j : C_{i,j}(\text{map}) \leq C^{\text{guess}}_{i,j}$) and
17.      ($\forall i, j, p : \text{MA}_{i,j,p}(\text{map}) \leq \text{MA}^{\text{guess}}_{i,j,p}$) then
18.      declare SUCCESS
19.    else o' := o end if
20.  else
21.    choose an $o'$ that has not been tried before
22.  end if
23. end for
24. declare FAILURE
```

We can deal with (iii) by modifying the pseudo code above so that on line 7, it not only runs a WCET tool but also computes an upper bound on the cost of a preemption (e.g. by considering that the entire private cache needs to get reloaded) and also computes an upper bound on the number of preemptions that a job can experience. Line 10 can be changed analogously to line 7.

Hence, by using these modifications, our solution can be used in practice.

7. EVALUATION SUMMARY

We have implemented a tool based on this theory and tested it on systems with 4 and 8 processors. Table 2 offers a summary of results — for details, see Appendix in [1]. It can be seen that the maximum time it takes to finish is 18h and the median time is 2.5h. We performed a preliminary evaluation of the guarantee provided by this tool as follows: We developed a plugin for AADL based on Valgrind[2]; this tool obtains an upper bound on the number of memory accesses, from each page, that reaches the memory controller and then outputs a model of the software system. We applied this plugin on a taskset with synthetic benchmark programs (matrix multiply) and obtained the parameters of our model. We then ran it on a gEDF implementation in the Linux kernel and used our previously developed Linux implementation of coordinated cache and bank coloring [28] configured as specified by our tool. We ran the software system for 8h and observed no deadline misses.

8. CONCLUSIONS

Using COTS multicore processors in hard real-time systems is challenging because (i) taking full advantage of them for meeting tight deadlines requires parallelization and (ii) the contention for shared resources in the memory system makes execution times hard to predict. In this paper, we have developed a solution that addresses these issues. Our main idea is to formulate a MILP that configures the memory mapping and performs schedulability analysis.

Acknowledgment

Copyright 2015 Carnegie Mellon University This material is based upon work funded and supported by the Department of Defense under Contract No. FA8721-05-C-0003 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the United States Department of Defense. NO WARRANTY. THIS CARNEGIE MELLON UNIVERSITY AND SOFTWARE ENGINEERING INSTITUTE MATERIAL IS FURNISHED ON AN "AS-IS" BASIS. CARNEGIE MELLON UNIVERSITY MAKES NO WARRANTIES OF ANY KIND, EITHER EXPRESSED OR IMPLIED, AS TO ANY MATTER INCLUDING, BUT NOT LIMITED TO, WARRANTY OF FITNESS FOR PURPOSE OR MERCHANTABILITY, EXCLUSIVITY, OR RESULTS OBTAINED FROM USE OF THE MATERIAL. CARNEGIE MELLON UNIVERSITY DOES NOT MAKE ANY WARRANTY OF ANY KIND WITH RESPECT TO FREEDOM FROM PATENT, TRADEMARK, OR COPYRIGHT INFRINGEMENT. This material has been approved for public release and unlimited distribution. DM-0002723

9. REFERENCES

[1] www.andrew.cmu.edu/user/banderss/papers/m/g.pdf.

[2] B. Andersson and D. de Niz. Analyzing Global-EDF for multiprocessor scheduling of parallel tasks. In *OPODIS*, 2012.

[3] T. P. Baker. Multiprocessor EDF and deadline monotonic schedulability analysis. In *RTSS*, 2003.

[4] S. Baruah, V. Bonifaci, A. Marchetti-Spaccamela, L. Stougie, and A. Wiese. A generalized parallel task model for recurrent real-time processes. In *RTSS*, 2012.

[5] S. K. Baruah, V. Bonifaci, A. Marchetti-Spaccamela, and S. Stiller. Implementation of a speedup-optimal global EDF schedulability test. In *ECRTS*, 2009.

[6] M. Bertogna, M. Cirinei, and G. Lipari. Improved schedulability analysis of EDF on multiprocessor platforms. In *ECRTS*, 2005.

[7] V. Bonifaci, A. Marchetti-Spaccamela, S. Stiller, and A. Wiese. Feasibility analysis in the sporadic DAG model. In *ECRTS*, 2013.

[8] H. S. Chwa, J. Lee, K.-M. Phan, A. Easwaran, and I. Shin. Global EDF schedulability analysis for synchronous parallel tasks on multicore platforms. In *ECRTS*, 2013.

[9] S. Collette, L. Cucu, and J. Goossens. Integrating job parallelism in real-time scheduling theory. *IPL*, 2008.

[10] L. Cong and J. H. Anderson. Supporting soft real-time dag-based systems on multiprocessors with no utilization loss. In *RTSS*, 2010.

[11] D. Dasari, B. Andersson, V. Nélis, S. M. Petters, A. Easwaran, and J. Lee. Response Time Analysis of COTS-Based Multicores Considering the Contention on the Shared Memory Bus. In *ICESS*, 2011.

[12] JEDEC. DDR3 SDRAM Standard. http://www.jedec.org.

[13] S. Kato and Y. Ishikawa. Gang EDF scheduling of parallel task systems. In *RTSS*, 2009.

[14] H. Kim, D. de Niz, B. Andersson, M. Klein, O. Mutlu, and R. Rajkumar. Bounding Memory Interference Delay in COTS-based Multi-Core Systems. In *RTAS*, 2014.

[15] H. Kim, A. Kandhalu, and R. Rajkumar. A Coordinated Approach for Practical OS-Level Cache Management in Multi-Core Real-Time Systems. In *ECRTS*, 2013.

[16] J. Kim, H. Kim, K. Lakshmanan, and R. R. Rajkumar. Parallel scheduling for cyber physical systems: Analysis and case study on a self-driving car. In *RTAS*, 2013.

[17] K. Lakshmanan, S. Kato, and R. Rajkumar. Scheduling parallel real-time tasks on multi-core processors. In *RTSS*, 2010.

[18] J. Li, K. Agrawal, C. Lu, and C. Gill. Analysis of global EDF for parallel tasks. In *ECRTS*, 2013.

[19] L. Liu, Z. Cui, M. Xing, Y. Bao, M. Chen, and C. Wu. A software memory partition approach for eliminating bank-level interference in multicore systems. PACT'12.

[20] M. Lv, G. Nan, W. Yi, and G. Yu. Combining Abstract Interpretation with Model Checking for Timing Analysis of Multicore Software. In *RTSS*, 2010.

[21] R. Mancuso, R. Dudko, E. Betti, M. Cesati, M. Caccamo, and R. Pellizzoni. Real-Time Cache Management Framework for Multi-core Architectures. In *RTAS*, 2013.

[22] L. Nogueira and L. P. Pinho. Server-based scheduling of parallel real-time tasks. In *EMSOFT*, 2012.

[23] R. Pellizzoni, A. Schranzhofer, J.-J. Chen, M. Caccamo, and L. Thiele. Worst case delay analysis for memory interference in multicore systems. In *DATE'10*.

[24] M. Qamhieh, F. Fauberteau, G. Laurent, and S. Midonnet. Global EDF scheduling of directed acyclic graphs on multiprocessor systems. In *RTNS*, 2013.

[25] J. Reineke, I. Liu, H. D. Patel, S. Kim, and E. A. Lee. PRET DRAM Controller: Bank Privatization for Predictability and Temporal Isolation. In *CODES+ISSS'11*.

[26] J. Rosén, A. Andrei, P. Eles, and Z. Peng. Bus Access Optimization for Predictable Implementation of Real-Time Applications on Multiprocessor Systems-on-Chip. In *RTSS*, 2007.

[27] A. Saifullah, K. Agrawal, C. Lu, and C. Gill. Multi-core real-time scheduling for generalized parallel tasks models. In *RTSS*, 2011.

[28] N. Suzuki, H. Kim, D. de Niz, B. Andersson, L. Wrage, M. Klein, and R. Rajkumar. Coordinated bank and cache coloring for temporal protection of memory accesses. In *ICESS*, 2013.

[29] B. Ward, J. Herman, C. Kenna, and J. Anderson. Making Shared Caches More Predictable on Multicore Platforms. In *ECRTS*, 2013.

[30] Z. P. Wu, Y. Krish, and R. Pellizzoni. Worst Case Analysis of DRAM Latency in Multi-Requestor Systems. In *RTSS*, 2013.

[2]Available at http://www.andrew.cmu.edu/user/dionisio/cachegrind-pager-patch.tgz

[31] H. Yun, G. Yao, R. Pellizzoni, M. Caccamo, and L. Sha. Memory Access Control in Multiprocessor for Real-time Systems with Mixed Criticality. In *ECRTS*, 2012.

APPENDIX

Proof

THEOREM 1.

$$((\langle \text{flag}, o \rangle = \text{fmem}(\tau, \Pi, K)) \wedge (\text{flag} = \text{true})) \Rightarrow$$

τ is gEDF schedulable on m processors of speed s for the case that tasks experience memory contention and the VPAT conforms to o

PROOF. If the theorem is false then there exists a τ, m, s, K and an assignment of the number of jobs that each task generates and an assignment of arrival time to jobs and execution requirement of segments and a schedule such that the following two statements are true:

1. $(\langle \text{flag}, o \rangle = \text{fmem}(\tau, \Pi, K)) \wedge (\text{flag} = \text{true})$
2. for the jobset generated by τ with the aforementioned assignment, it holds that gEDF can generate the aforementioned schedule and there is at least one job that misses its deadline in this schedule.

For this schedule, let t_0 denote the earliest time when a deadline miss occurs. Remove all jobs with arrival time $\geq t_0$. There is still a deadline miss at time t_0. Let us now reason as follow: For each job with absolute deadline $> t_0$ such that it performs execution after time t_0, do the following: identify the latest stage of this job such that there is a segment of this stage that performs execution after t_0. Then reduce the execution of this segment. Repeated application of this yields that no job with absolute deadline $> t_0$ performs execution after time t_0. Hence, it holds that: (i) 1) and 2) above are true, (ii) one or many jobs with absolute deadline at t_0 misses deadlines, (iii) each job with absolute deadline $< t_0$ meets its deadline, (iv) all jobs have arrival times $< t_0$, and (v) no job with absolute deadline $> t_0$ performs execution after time t_0.

For each job with absolute deadline $< t_0$, we can reason as follows: Let τ_i denote the task that generates the job. Let A denote the arrival time of this job and consider the time interval $[A, A+D_i)$ and consider a task i' which is not the task that generated the job of τ_i. Because of (iii) and (iv), there can be at most one job of task i' such that this job arrives before A and it has execution that overlaps with $[A, A+D_i)$. Also, because of (iii), there can be at most $\lceil D_i/T_{i'} \rceil$ jobs of task i' such that this job arrives at or after A and it has execution that that overlaps with $[A, A+D_i)$.

For each job with absolute deadline $\geq t_0$, we can reason as follows: Let τ_i denote the task that generates the job. Let A denote the arrival time of this job and consider the time interval $[A, A+D_i)$ and consider a task i' which is not the task that generated the job of τ_i. Because of (iii) and (iv), there can be at most one job of task i' such that this job arrives before A and it has execution that overlaps with $[A, A+D_i)$. Also, because of (v), there can be at most $\lceil D_i/T_{i'} \rceil$ jobs of task i' such that this job arrives at or after A and it has execution that that overlaps with $[A, A+D_i)$.

Consequently, for each of these cases, there are at most $(\lceil \frac{D_i}{T_i'} \rceil + 1) \times \text{mb}_{i',j',g',b}$ memory accesses on MB b of jobs of $\text{seg}(i', j', g')$ that overlaps with $[A, A+D_i)$. Putting it together yields that there are at most $\text{mmbo}_{i,j,g,b}$ memory accesses that can be issued in parallel with $\text{seg}(i,j,g)$ for MB b. Since we know the values of $\text{mmbo}_{i,j,g,b}$, using Fig. 7 yields $\text{cm}_{i,j,g}$. This yields $\text{cu}_{i,j}$ which provides an upper bound on the execution requirement. Since $\text{cu}_{i,j}$ is an upper bound on execution requirement we can treat the system as if there was no contention for resources in the memory system and execution requirements were given by $\text{cu}_{i,j}$. Since the constraints in Fig. 4 are satisfied, all deadlines are met. This contradicts 2) above. Hence, the theorem is correct. □

Solving MILP

Finding a solution to the MILP expressed by Fig. 4 and Fig. 7 is challenging because (i) the number of variables and constraints is large and (ii) BIG is much larger than the other constants causing numerical issues. Therefore, we will rewrite the MILP to avoid numerical issues. We will also present different methods for solving the MILP; they differ in (i) the amount of time to finish and (ii) whether a solution is guaranteed to be found if a solution exists. They all have in common, however, that they return a tuple $\langle \text{flag}, o\rangle$ such that if flag is true, then the MILP is feasible. It can be seen in Fig. 7, that changing the domain of $\text{mb}_{i,j,g,b}$ from non-negative integer to non-negative real does not change the feasiblity of the MILP. The same applies to $\text{mmbo}_{i,j,g,b}$, $\text{oat}_{i,j,g,b}$, and $\text{oao}_{i,j,g,b}$. We will now rewrite the constraints without changing feasibility but so that numerical issues are avoided. Let SCALINGFACTORNACCESSES be an integer that we choose (e.g. SCALINGFACTORNACCESSES = 2^{23}). It can be seen that multiplying the right hand sides of (38),(39),(40),(41) by MBCF and replacing $\text{coat}_{i,j,g,b}$ in (32) by $\frac{1}{\text{MBCF}} \times \text{coat}_{i,j,g,b}$ does not change feasibility. It can also be seen that dividing the right-hand side of (22) by SCALINGFACTORNACCESSES and replacing $\frac{1}{\text{MBCF}} \times \text{coat}_{i,j,g,b}$ in (32) by $\frac{\text{SCALINGFACTORNACCESSES}}{\text{MBCF}} \times \text{coat}_{i,j,g,b}$ does not change feasibility. Figure 8 shows these rewritten expressions. This leaves us with discussion on how to choose SCALINGFACTORNACCESSES. We do it as follows.

```
1. SCALINGFACTORNACCESSES := 1
2. if (50)> 0 then
3.   SCALINGFACTORNACCESSES := smallest number
4.     ≥ (51)/(52) such that it is equal to two raised
5.     to some integer.
6. end if
```

In this way, the parameter BIG is kept small. We will now present the methods.

Method 1

Method 1 is guaranteed to output a solution if a solution exists. Method 1 is to simply take the constraints in Fig. 4 and Fig. 7 and solve the MILP. If there exists an assignment of values to the variables so that the constraints in Fig. 4 and Fig. 7 are satisfied then flag is true and o is the values of the o-variables; otherwise flag is false and o is undefined.

Method 2

Method 2 is guaranteed to output a solution if a solution exists. We can reason as follows: If there is a feasible solution, then it holds that for each cache color, the pages that are mapped to frames of this cache color all belong to the same task (otherwise (29) would be violated). Let $\text{occupiescachecolor}_{i,h}$ be 1 if τ_i occupies cache color h; otherwise 0. If, for this solution, it holds that there is a task τ_i and a task $\tau_{i'}$ and a cache color h and a cache color h' such that $i < i'$ and $h > h'$ and $\text{occupiescachecolor}_{i,h} = 1$ and $\text{occupiescachecolor}_{i',h'} = 1$, then we can change the o-values of the solution so that each page of τ_i that was mapped to h is mapped to h' and each page of τ_i' that was mapped to h' is mapped to h. Also update the x-values accordingly. This gives us a new feasible solution such that $i < i'$ and $h < h'$ and $\text{occupiescachecolor}_{i,h} = 1$ and $\text{occupiescachecolor}_{i',h'} = 1$. Repeating this argument yields that for each τ_i, tasks with lower index than τ_i only occupies cache colors of lower index and tasks with higher index than τ_i only occupies cache colors of higher index. If there is a cache color h that is not occupied by any task, then we can identify all tasks that occupies cache colors of index greater than h and let each of their memory allocation use a cache color that has index 1 less. Also update the x-values accordingly.

For this reason, we can, without loss of generality, add the following constraint:

$$\forall \langle i,j,g,h,i',j',g',h'\rangle \text{ s.t. } (i<i') \land (h \geq h') : \quad \text{x}_{i,j,g,h} + \text{x}_{i',j',g',h'} \leq 1$$

Method 2 is like Method 1 but with the constraint above.

Method 3

Method 3 is not guaranteed to output a solution if a solution exists. Method 3 is defined as follows.

1. Let the following variables be non-negative real numbers: loadfactorofcells, utilconsideringcont, $\text{loadofdeadline}_\text{i}$, myobj.
2. Let utilconsideringcont, $\text{loadofdeadline}_\text{i}$ be defined as follows: utilconsideringcont $= (\sum_{\tau_{i'} \in \tau} \frac{\text{cu}_{i'}}{T_i})/(m \times s)$ and $\text{loadofdeadline}_\text{i} = (\sum_{\tau_{i'} \in \tau} \max(\lfloor \frac{D_i - D_{i'}}{T_{i'}} \rfloor + 1, 0) \times \text{cu}_{i'})/(m \times s \times D_i)$.
3. Solve the following problem: minimize myobj subject to $\text{cu}_i = \sum_{j=1}^{\text{ns}_i} (\text{nseg}_{\text{i,j}} \times \text{cu}_{i,j})$ and the constraints in Fig. 7 and $\forall \langle h,b\rangle$ s.t. $(h \in [0, H-1]) \land (b \in \{0..B-1\})$: $(\sum_{\tau_i \in \tau} \sum_{j \in \{1..\text{ns}_i\}} \sum_{g \in \{1..\text{nseg}_{i,j}\}} \sum_{p \in \{0..\text{np}_{i,j}-1\}} (\frac{1}{\text{GS}_{i,j,g,p}} \times \text{o}_{i,j,g,p,h,b})) \leq \text{CAP} \times$ loadfactorofcells and loadfactorofcells $\leq$ myobj and utilconsideringcont $\leq$ myobj and $\forall \tau_i \in \tau$: $\text{loadofdeadline}_\text{i} \leq$ myobj.
4. Consider the optimization problem of $\text{fmem}(\tau, \Pi, K)$ where the o-values must be equal to the values obtained in step 3 above. Solve this optimization problem.
5. If the optimization problem in step 4 is feasible then return $\langle \text{true}, o\rangle$ where o is the o-values obtained in step 3 above.
6. If the optimization problem in step 4 is infeasible then return $\langle \text{false}, o\rangle$ where o is undefined.

We solve the optimization problem in step 3 as follows. Keep running the solver for 3600 seconds and then check the status and then run the solver for additional 3600 seconds and then check the status and then run the solver for additional 3600 seconds and check the status and so so. We finish this process if one of the following conditions are true (i) an optimal solution has been found or (ii) when checking, a feasibile solution has been found and at the preceding checking, a feasible solution was found as well and the MIP-gap (the gap between the objective function of the current solution as compared to the best bound) has not changed between these checks. As a result, the solution we obtain from step 3 is either optimal or it is such that it did not change during the recent 3600 seconds when the solver ran.

The intuition behind the optimization problem of Step 3 is that we would like to find a memory allocation such that with this memory allocation, the MILP of Step 4 will be feasible if the MILP in Method 2 is feasible and the way we do it is to make sure that the capacity of the memory cell which is utilized at most is not too high (i.e. loadfactorofcells $\leq$ 1) and that conditions that approximate the schedulability test in the MILP of Method 2 are satisfied. Specifically, when $t = \infty$, it holds that whether the ffdbf is at most the supply is equivalent to checking utilconsideringcont $\leq$ 1. But there are many other durations for which we need to check whether the ffdbf is at most the supply. We only explore those durations that are equal to a deadline; hence we check $\forall \tau_i \in \tau$: $\text{loadofdeadline}_\text{i} \leq$ myobj.

$$\forall\langle i,j,g,b\rangle : \ \mathrm{mb}_{i,j,g,b} = \sum_{p\in\{0..\mathrm{np}_{i,j}-1\}} \sum_{h\in\{0..H-1\}} \frac{\mathrm{MA}_{i,j,p}}{\mathrm{SCALINGFACTORNACCESSES}} \times \mathrm{o}_{i,j,g,p,h,b} \tag{53}$$

$$\forall\langle i,j,g\rangle : \ \mathrm{cm}_{i,j,g} = \mathrm{C}_{i,j} + s \times (\sum_{b\in\{0..B-1\}} (\frac{\mathrm{SCALINGFACTORNACCESSES}}{\mathrm{MBCF}} \times \mathrm{coat}_{i,j,g,b} + L_{\mathrm{inter}} \times \mathrm{oao}_{i,j,g,b})) \tag{54}$$

$$\forall\langle i,j,g,b\rangle : \ (\mathrm{bc1}_{i,j,g,b} = 1) \Rightarrow (\mathrm{coat}_{i,j,g,b} = \mathrm{mmbo}_{i,j,g,b} \times (L_{\mathrm{conf}} + L_{\mathrm{inter}}) \times \mathrm{MBCF}) \tag{55}$$

$$\begin{aligned} \forall\langle i,j,g,b\rangle : \ (\mathrm{bc2}_{i,j,g,b} = 1) \Rightarrow (\mathrm{coat}_{i,j,g,b} = \mathrm{mb}_{i,j,g,b} \times \mathrm{LL} \times (L_{\mathrm{conf}} + L_{\mathrm{inter}}) \times \mathrm{MBCF} + \\ (\mathrm{mmbo}_{i,j,g,b} - \mathrm{mb}_{i,j,g,b} \times \mathrm{LL}) \times (\mathrm{WL} + \mathrm{BL}/2 + t_{\mathrm{WR}}) \times t_{CK} \times \mathrm{MBCF}) \end{aligned} \tag{56}$$

$$\begin{aligned} \forall\langle i,j,g,b\rangle : \ (\mathrm{bc3}_{i,j,g,b} = 1) \Rightarrow \\ (\mathrm{coat}_{i,j,g,b} = \mathrm{mb}_{i,j,g,b} \times \mathrm{LL} \times (L_{\mathrm{conf}} + L_{\mathrm{inter}}) \times \mathrm{MBCF} + \mathrm{mb}_{i,j,g,b} \times (\mathrm{WL} + \mathrm{BL}/2 + t_{\mathrm{WR}}) \times t_{CK} \times \mathrm{MBCF} + \\ (\mathrm{mmbo}_{i,j,g,b} - \mathrm{mb}_{i,j,g,b} \times (\mathrm{LL}+1)) \times (\mathrm{WL} + \mathrm{BL}/2 + t_{\mathrm{WR}} + \mathrm{CL}) \times (1/2) \times t_{CK} \times \mathrm{MBCF}) \end{aligned} \tag{57}$$

$$\forall\langle i,j,g,b\rangle : \ (\mathrm{bc4}_{i,j,g,b} = 1) \Rightarrow (\mathrm{coat}_{i,j,g,b} = \mathrm{mb}_{i,j,g,b} \times (\mathrm{LL} \times (L_{\mathrm{conf}} + L_{\mathrm{inter}}) + L_{\mathrm{conhit}}(N_{\mathrm{re}})) \times \mathrm{MBCF}) \tag{58}$$

Figure 8: Rewriting some expressions. (22) is rewritten as (53). (32) is rewritten as (54). (38) is rewritten as (55). (39) is rewritten as (56). (40) is rewritten as (57). (41) is rewritten as (58). After this rewriting, feasibility has not changed.

Evaluation

In this section, we address the following questions: (i) how long time does it take to perform the schedulability test (solve the MILP) and (ii) how pessimistic is our schedulability test. We will use Method 3. We will solve the MILP with Gurobi 6.0 — a state-of-the-art solver.

Consider the system in Fig. 9. It models a hypothetical autonomous system with 4 processors and task τ_1 performing sensor fusion (it first reads the sensors in its 1^{st} stage and then performs parallel processing in its 2^{nd} stage and then merges the results in its 3^{rd} stage) and task τ_2 is a mission controller task (it takes high-level decisions about the mission, e.g, whether the mission should be aborted) and task τ_3 recomputes the current plans when a certain critical event occurs (its 2^{nd} stage performs computations in parallel). We will run our evaluation by varying parameters of this system. Specifically, we will vary $C_{1,2}$ and $\mathrm{MA}_{i,j,p}$. We will vary $C_{1,2}$ by simply setting it to a new value. We will vary $\mathrm{MA}_{i,j,p}$ of all segments by multiplying them by mult. For example, mult = 0 means that all segments perform no memory accesses. mult = 1 means that the upper bounds on the memory accesses are the same as in Fig. 9. Table 3 shows the outcome of our evaluation for $m = 4$ and Table 4 shows the outcome of our evaluation for $m = 8$.

The first column shows the value of $C_{1,2}$. The second column shows the number of memory accesses to a page relative to the number of memory accesses stated in Fig. 9. If the value in the column is 1 then the number of memory accesess to a page is equal to the number of memory accesses stated in Fig. 9. The third column indicates the amount of time it takes to perform the schedulability analysis (with the MILP). The fourth column indicates whether that schedulability analysis provides a guarantee that the taskset is schedulable.

Look at the cases $C_{1,2} = 0.005$ in Table 3. It can be seen that mult = 10 results in not-schedulable and mult = 4 results in schedulable. The reason for this is as follows. For the case mult = 10, in the first phase (when we obtain o), we obtain that the objective function is 0.78 and the left-hand of the inequality in (17) is $0.78 \times 4 = 3.12$ (because there is a very large number of memory accesses). This requires that we choose $k/K \geq 0.78$. Since $K = 20$, we obtain that $k \geq 16$ and hence $k/K \geq 0.80$ and this makes the right-hand side of (17) the value $4 - (4-1) \times 0.8 = 1.6$. Hence, the left-hand of the inequality in (17) is larger than its right-hand side and consequently this constraint is violated. With larger k, we obtain the same conclusion: the constraint is violated. Hence, this MILP is infeasible.

For the case mult = 4, we obtain another outcome though. in the first phase (when we obtain o), we obtain that the objective function is 0.302 and the left-hand of the inequality in (17) is $0.302 \times 4 = 1.208$ (because there are fewer memory accesses). This requires that we choose $k/K \geq 0.302$. Since $K = 20$, we obtain that $k \geq 7$ and hence $k/K \geq 0.35$ and this makes the right-hand side of (17) the value $4-(4-1)\times 0.35 = 2.95$. Hence, the left-hand of the inequality in (17) is less than its right-hand side and consequently this constraint is satisfied. It turns out that there is a solution (we got the solution for k=18).

We have rerun some of the experiments and found that for a given setting, the time required can vary. This is because the MILP solver solves multiple LPs and does this concurrently and hence the MILP solver is non-deterministic; for a given setup, the progress that it makes within one hour can vary. Hence, when using Method 3 for a given setting, the time required can be different for different runs. Note that this non-determinism is only about the running time; the output (true/false) of the algorithm is deterministic and hence it does not lead to unsafe results.

$m = 4$	s=1	$\tau = \{\tau_1, \tau_2, \tau_3\}$			
T_1=0.100	D_1=0.100	ns_1=3	$\mathrm{nseg}_{1,1}$=1	$\mathrm{nseg}_{1,2} = 4$	$\mathrm{nseg}_{1,3}$=1
			$C_{1,1} = 0.001$	$C_{1,2} = 0.030$	$C_{1,3} = 0.001$
			$\mathrm{np}_{1,1} = 17$	$\mathrm{np}_{1,2} = 17$	$\mathrm{np}_{1,3} = 17$
			$\mathrm{MA}_{1,1,0} = 100$	$\mathrm{MA}_{1,2,0} = 1000$	$\mathrm{MA}_{1,3,0} = 100$
			$\mathrm{MA}_{1,1,1} = 100$	$\mathrm{MA}_{1,2,1} = 1000$	$\mathrm{MA}_{1,3,1} = 100$
			$\mathrm{MA}_{1,1,2} = 100$	$\mathrm{MA}_{1,2,2} = 1000$	$\mathrm{MA}_{1,3,2} = 100$
			$\mathrm{MA}_{1,1,3} = 100$	$\mathrm{MA}_{1,2,3} = 1000$	$\mathrm{MA}_{1,3,3} = 100$
			$\mathrm{MA}_{1,1,4} = 100$	$\mathrm{MA}_{1,2,4} = 1000$	$\mathrm{MA}_{1,3,4} = 100$
			$\mathrm{MA}_{1,1,5} = 100$	$\mathrm{MA}_{1,2,5} = 1000$	$\mathrm{MA}_{1,3,5} = 100$
			$\mathrm{MA}_{1,1,6} = 100$	$\mathrm{MA}_{1,2,6} = 1000$	$\mathrm{MA}_{1,3,6} = 100$
			$\mathrm{MA}_{1,1,7} = 100$	$\mathrm{MA}_{1,2,7} = 1000$	$\mathrm{MA}_{1,3,7} = 100$
			$\mathrm{MA}_{1,1,8} = 100$	$\mathrm{MA}_{1,2,8} = 1000$	$\mathrm{MA}_{1,3,8} = 100$
			$\mathrm{MA}_{1,1,9} = 100$	$\mathrm{MA}_{1,2,9} = 1000$	$\mathrm{MA}_{1,3,9} = 100$
			$\mathrm{MA}_{1,1,10} = 100$	$\mathrm{MA}_{1,2,10} = 1000$	$\mathrm{MA}_{1,3,10} = 100$
			$\mathrm{MA}_{1,1,11} = 100$	$\mathrm{MA}_{1,2,11} = 1000$	$\mathrm{MA}_{1,3,11} = 100$
			$\mathrm{MA}_{1,1,12} = 100$	$\mathrm{MA}_{1,2,12} = 1000$	$\mathrm{MA}_{1,3,12} = 100$
			$\mathrm{MA}_{1,1,13} = 100$	$\mathrm{MA}_{1,2,13} = 1000$	$\mathrm{MA}_{1,3,13} = 100$
			$\mathrm{MA}_{1,1,14} = 100$	$\mathrm{MA}_{1,2,14} = 1000$	$\mathrm{MA}_{1,3,14} = 100$
			$\mathrm{MA}_{1,1,15} = 100$	$\mathrm{MA}_{1,2,15} = 1000$	$\mathrm{MA}_{1,3,15} = 100$
			$\mathrm{MA}_{1,1,16} = 100$	$\mathrm{MA}_{1,2,16} = 1000$	$\mathrm{MA}_{1,3,16} = 100$
			$\mathrm{MA}_{1,1,17} = 100$	$\mathrm{MA}_{1,2,17} = 1000$	$\mathrm{MA}_{1,3,17} = 100$
T_2=0.010	D_2=0.010	ns_2=1	$\mathrm{nseg}_{2,1}$=1		
			$C_{2,1} = 0.002$		
			$\mathrm{np}_{2,1} = 17$		
			$\mathrm{MA}_{2,1,0} = 100$		
			$\mathrm{MA}_{2,1,1} = 100$		
			$\mathrm{MA}_{2,1,2} = 100$		
			$\mathrm{MA}_{2,1,3} = 100$		
			$\mathrm{MA}_{2,1,4} = 100$		
			$\mathrm{MA}_{2,1,5} = 100$		
			$\mathrm{MA}_{2,1,6} = 100$		
			$\mathrm{MA}_{2,1,7} = 100$		
			$\mathrm{MA}_{2,1,8} = 100$		
			$\mathrm{MA}_{2,1,9} = 100$		
			$\mathrm{MA}_{2,1,10} = 100$		
			$\mathrm{MA}_{2,1,11} = 100$		
			$\mathrm{MA}_{2,1,12} = 100$		
			$\mathrm{MA}_{2,1,13} = 100$		
			$\mathrm{MA}_{2,1,14} = 100$		
			$\mathrm{MA}_{2,1,15} = 100$		
			$\mathrm{MA}_{2,1,16} = 100$		
			$\mathrm{MA}_{2,1,17} = 100$		
T_3=0.100	D_3=0.044	ns_3=3	$\mathrm{nseg}_{3,1}$=1	$\mathrm{nseg}_{3,2}$=2	$\mathrm{nseg}_{3,3}$=1
			$C_{3,1} = 0.002$	$C_{3,2} = 0.006$	$C_{3,3} = 0.002$
			$\mathrm{np}_{3,1} = 17$	$\mathrm{np}_{3,2} = 17$	$\mathrm{np}_{3,3} = 17$
			$\mathrm{MA}_{3,1,0} = 100$	$\mathrm{MA}_{3,2,0} = 100$	$\mathrm{MA}_{3,3,0} = 100$
			$\mathrm{MA}_{3,1,1} = 100$	$\mathrm{MA}_{3,2,1} = 100$	$\mathrm{MA}_{3,3,1} = 100$
			$\mathrm{MA}_{3,1,2} = 100$	$\mathrm{MA}_{3,2,2} = 100$	$\mathrm{MA}_{3,3,2} = 100$
			$\mathrm{MA}_{3,1,3} = 100$	$\mathrm{MA}_{3,2,3} = 100$	$\mathrm{MA}_{3,3,3} = 100$
			$\mathrm{MA}_{3,1,4} = 100$	$\mathrm{MA}_{3,2,4} = 100$	$\mathrm{MA}_{3,3,4} = 100$
			$\mathrm{MA}_{3,1,5} = 100$	$\mathrm{MA}_{3,2,5} = 100$	$\mathrm{MA}_{3,3,5} = 100$
			$\mathrm{MA}_{3,1,6} = 100$	$\mathrm{MA}_{3,2,6} = 100$	$\mathrm{MA}_{3,3,6} = 100$
			$\mathrm{MA}_{3,1,7} = 100$	$\mathrm{MA}_{3,2,7} = 100$	$\mathrm{MA}_{3,3,7} = 100$
			$\mathrm{MA}_{3,1,8} = 100$	$\mathrm{MA}_{3,2,8} = 100$	$\mathrm{MA}_{3,3,8} = 100$
			$\mathrm{MA}_{3,1,9} = 100$	$\mathrm{MA}_{3,2,9} = 100$	$\mathrm{MA}_{3,3,9} = 100$
			$\mathrm{MA}_{3,1,10} = 100$	$\mathrm{MA}_{3,2,10} = 100$	$\mathrm{MA}_{3,3,10} = 100$
			$\mathrm{MA}_{3,1,11} = 100$	$\mathrm{MA}_{3,2,11} = 100$	$\mathrm{MA}_{3,3,11} = 100$
			$\mathrm{MA}_{3,1,12} = 100$	$\mathrm{MA}_{3,2,12} = 100$	$\mathrm{MA}_{3,3,12} = 100$
			$\mathrm{MA}_{3,1,13} = 100$	$\mathrm{MA}_{3,2,13} = 100$	$\mathrm{MA}_{3,3,13} = 100$
			$\mathrm{MA}_{3,1,14} = 100$	$\mathrm{MA}_{3,2,14} = 100$	$\mathrm{MA}_{3,3,14} = 100$
			$\mathrm{MA}_{3,1,15} = 100$	$\mathrm{MA}_{3,2,15} = 100$	$\mathrm{MA}_{3,3,15} = 100$
			$\mathrm{MA}_{3,1,16} = 100$	$\mathrm{MA}_{3,2,16} = 100$	$\mathrm{MA}_{3,3,16} = 100$
			$\mathrm{MA}_{3,1,17} = 100$	$\mathrm{MA}_{3,2,17} = 100$	$\mathrm{MA}_{3,3,17} = 100$
$K = 20$	$\mathrm{MEMCAP} = 2^{21}$	$H = 32$	$B = 16$	$\mathrm{HWSHARE} = 1/4$	$\mathrm{CAP} = 2^{10}$

INO = 970

$\mathrm{shfr} = \{\langle 1,1,1,0,1,2,1,0\rangle, \langle 1,1,1,0,1,2,1,0\rangle, \langle 1,1,1,2,1,2,3,0\rangle, \langle 1,1,1,3,1,2,4,0\rangle, \langle 1,3,1,0,1,2,1,1\rangle, \langle 1,3,1,1,1,2,2,1\rangle, \langle 1,3,1,3,1,2,4,1\rangle, \langle 3,1,1,0,3,2,1,0\rangle, \langle 3,1,1,1,3,2,2,0\rangle, \langle 3,3,1,0,3,2,1,1\rangle, \langle 3,3,1,1,3,2,2,1\rangle\}$

$\mathrm{MBCF} = 1.5 \times 10^9$,trrd = 4,tfaw = 20,wl = 7,bl = 8,twtr = 5,cl = 9,trp = 9,trcd = 9,twr = 10

Figure 9: One of the systems used in our evaluation.

Acknowledgment

Copyright 2015 Carnegie Mellon University This material is based upon work funded and supported by the Department of Defense under Contract No. FA8721-05-C-0003 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the United States Department of Defense. NO WARRANTY. THIS CARNEGIE MELLON UNIVERSITY AND SOFTWARE ENGINEERING INSTITUTE MATERIAL IS FURNISHED ON AN "AS-IS" BASIS. CARNEGIE MELLON UNIVERSITY MAKES NO WARRANTIES OF ANY KIND, EITHER EXPRESSED OR IMPLIED, AS TO ANY MATTER INCLUDING, BUT NOT LIMITED TO, WARRANTY OF FITNESS FOR PURPOSE OR MERCHANTABILITY, EXCLUSIVITY, OR RESULTS OBTAINED FROM USE OF THE MATERIAL. CARNEGIE MELLON UNIVERSITY DOES NOT MAKE ANY WARRANTY OF ANY KIND WITH RESPECT TO FREEDOM FROM PATENT, TRADEMARK, OR COPYRIGHT INFRINGEMENT. This material has been approved for public release and unlimited distribution. DM-0002122

$C_{1,2}$ (seconds)	mult $MA_{i,j,p}$	Time (seconds)	Schedulable
0.005	0.000	452.6	yes
0.005	0.010	7659.3	yes
0.005	0.100	25792.5	yes
0.005	0.250	7650.3	yes
0.005	0.500	11253.5	yes
0.005	1.000	7648.4	yes
0.005	2.000	7649.4	yes
0.005	4.000	7649.9	yes
0.005	10.000	11220.3	no
0.010	0.000	451.5	yes
0.010	0.010	7654.9	yes
0.010	0.100	11250.8	yes
0.010	0.250	7650.3	yes
0.010	0.500	7650.5	yes
0.010	1.000	14854.6	yes
0.010	2.000	7649.7	yes
0.010	4.000	7650.8	yes
0.010	10.000	11220.9	no
0.015	0.000	451.0	yes
0.015	0.010	22061.2	yes
0.015	0.100	14859.0	yes
0.015	0.250	7651.5	yes
0.015	0.500	11261.7	yes
0.015	1.000	11248.3	yes
0.015	2.000	7647.7	yes
0.015	4.000	7650.0	yes
0.015	10.000	7622.6	no
0.020	0.000	451.8	yes
0.020	0.010	11248.7	yes
0.020	0.100	11249.6	yes
0.020	0.250	14853.6	yes
0.020	0.500	7655.2	yes
0.020	1.000	11250.5	yes
0.020	2.000	7649.8	yes
0.020	4.000	7650.4	yes
0.020	10.000	7620.4	no
0.025	0.000	452.3	yes
0.025	0.010	7649.7	yes
0.025	0.100	14857.8	yes
0.025	0.250	7650.7	yes
0.025	0.500	7655.1	yes
0.025	1.000	7650.7	yes
0.025	2.000	18469.1	yes
0.025	4.000	18470.3	yes
0.025	10.000	7619.7	no
0.030	0.000	452.5	yes
0.030	0.010	7654.6	yes
0.030	0.100	7650.3	yes
0.030	0.250	18459.4	yes
0.030	0.500	7649.6	yes
0.030	1.000	7649.9	yes
0.030	2.000	11251.7	yes
0.030	4.000	14851.0	yes
0.030	10.000	11224.2	no
0.035	0.000	451.3	yes
0.035	0.010	11257.1	yes
0.035	0.100	7650.3	yes
0.035	0.250	7651.4	yes
0.035	0.500	11249.0	yes
0.035	1.000	11251.8	yes
0.035	2.000	11250.6	yes
0.035	4.000	7620.8	no
0.035	10.000	7619.3	no
0.040	0.000	452.2	yes
0.040	0.010	14864.5	yes
0.040	0.100	11250.7	yes
0.040	0.250	18469.6	yes
0.040	0.500	11249.4	yes
0.040	1.000	7649.2	yes
0.040	2.000	7620.6	no
0.040	4.000	7621.3	no
0.040	10.000	7619.2	no
0.045	0.000	422.1	no
0.045	0.010	18438.2	no
0.045	0.100	11221.7	no
0.045	0.250	18433.1	no
0.045	0.500	7621.2	no
0.045	1.000	14821.3	no
0.045	2.000	7620.6	no
0.045	4.000	7620.1	no
0.045	10.000	7621.2	no

Table 3: Results from evaluation ($m = 4$).

$C_{1,2}$	mult	Time	
(seconds)	$\mathrm{MA}_{i,j,p}$	(seconds)	Schedulable
0.005	0.000	1053.4	yes
0.005	0.010	8326.8	yes
0.005	0.100	11850.3	yes
0.005	0.250	8249.8	yes
0.005	0.500	8243.0	yes
0.005	1.000	26282.0	yes
0.005	2.000	8247.9	yes
0.005	4.000	8198.8	no
0.005	10.000	11804.9	no
0.010	0.000	1050.3	yes
0.010	0.010	8253.4	yes
0.010	0.100	11848.9	yes
0.010	0.250	8246.6	yes
0.010	0.500	15446.5	yes
0.010	1.000	8252.3	yes
0.010	2.000	11846.2	yes
0.010	4.000	8192.8	no
0.010	10.000	11792.1	no
0.015	0.000	1043.0	yes
0.015	0.010	8252.6	yes
0.015	0.100	19059.0	yes
0.015	0.250	40689.4	yes
0.015	0.500	8243.8	yes
0.015	1.000	8245.1	yes
0.015	2.000	8240.5	yes
0.015	4.000	8193.0	no
0.015	10.000	8204.6	no
0.020	0.000	1047.4	yes
0.020	0.010	8248.1	yes
0.020	0.100	8253.8	yes
0.020	0.250	11848.0	yes
0.020	0.500	8247.2	yes
0.020	1.000	8193.7	no
0.020	2.000	8197.8	no
0.020	4.000	8193.1	no
0.020	10.000	11809.2	no
0.025	0.000	1057.3	yes
0.025	0.010	8261.3	yes
0.025	0.100	15450.2	yes
0.025	0.250	15446.6	yes
0.025	0.500	8253.3	yes
0.025	1.000	8244.3	yes
0.025	2.000	8240.8	yes
0.025	4.000	8192.3	no
0.025	10.000	11794.2	no
0.030	0.000	1054.0	yes
0.030	0.010	8260.0	yes
0.030	0.100	8245.3	yes
0.030	0.250	8246.1	yes
0.030	0.500	8245.9	yes
0.030	1.000	15452.1	yes
0.030	2.000	22666.2	yes
0.030	4.000	19014.3	no
0.030	10.000	26217.4	no
0.035	0.000	1044.5	yes
0.035	0.010	8249.1	yes
0.035	0.100	8333.5	yes
0.035	0.250	8314.2	yes
0.035	0.500	33487.0	yes
0.035	1.000	29875.2	yes
0.035	2.000	8195.5	no
0.035	4.000	11798.2	no
0.035	10.000	22602.0	no
0.040	0.000	1046.8	yes
0.040	0.010	51503.8	yes
0.040	0.100	8322.6	yes
0.040	0.250	26269.1	yes
0.040	0.500	22664.7	yes
0.040	1.000	15450.8	yes
0.040	2.000	19001.0	no
0.040	4.000	8195.0	no
0.040	10.000	11797.1	no
0.045	0.000	1048.0	yes
0.045	0.010	8256.5	yes
0.045	0.100	50665.2	yes
0.045	0.250	37235.9	yes
0.045	0.500	8200.9	no
0.045	1.000	8202.2	no
0.045	2.000	8196.3	no
0.045	4.000	8193.3	no
0.045	10.000	19003.5	no

Table 4: Results from evaluation ($m = 8$).