Technical Report

Provably Good Scheduling of Sporadic Tasks with Resource Sharing on a Two-type Heterogeneous Multiprocessor Platform

Gurulingesh Raravi

Björn Andersson

Konstantinos Bletsas

HURRAY-TR-110904

Version:

Date: 9/23/2011

Provably Good Scheduling of Sporadic Tasks with Resource Sharing on a Two-type Heterogeneous Multiprocessor Platform

Gurulingesh Raravi, Björn Andersson, Konstantinos Bletsas

IPP-HURRAY!

Polytechnic Institute of Porto (ISEP-IPP)

Rua Dr. António Bernardino de Almeida, 431

4200-072 Porto

Portugal

Tel.: +351.22.8340509, Fax: +351.22.8340509

E-mail: ghri@isep.ipp.pt, baa@isep.ipp.pt, ksbs@isep.ipp.pt

http://www.hurray.isep.ipp.pt

Abstract

Consider the problem of scheduling a set of implicit-deadline sporadic tasks to meet all deadlines on a two-type heterogeneous multiprocessor platform where a task may request at most one of |R| shared resources. There are m1 processors of type-1 and m2 processors of type-2. Tasks may migrate only when requesting or releasing resources. We present a new algorithm, FF-3C-vpr, which offers a guarantee that if a task set is schedulable to meet deadlines by an optimal task assignment scheme that only allows tasks to migrate when requesting or releasing a resource, then FF-3C-vpr also meets deadlines if given processors 4+6*ceil(|R|/min(m1,m2)) times as fast. As far as we know, it is the first result for resource sharing on heterogeneous platforms with provable performance.

Provably Good Scheduling of Sporadic Tasks with Resource Sharing on a Two-type Heterogeneous Multiprocessor Platform

Gurulingesh Raravi[1], Björn Andersson[2 1], and Konstantinos Bletsas[1]

[1] CISTER-ISEP Research Center, Polytechnic Institute of Porto, Portugal.
[2] Software Engineering Institute, Carnegie Mellon University, Pittsburgh, USA.
{ghri, baa, ksbs}@isep.ipp.pt; baandersson@sei.cmu.edu

Abstract. Consider the problem of scheduling a set of implicit-deadline sporadic tasks to meet all deadlines on a two-type heterogeneous multiprocessor platform where a task may request at most one of $|R|$ shared resources. There are m_1 processors of type-1 and m_2 processors of type-2. Tasks may migrate only when requesting or releasing resources. We present a new algorithm, FF-3C-vpr, which offers a guarantee that if a task set is schedulable to meet deadlines by an optimal task assignment scheme that only allows tasks to migrate when requesting or releasing a resource, then FF-3C-vpr also meets deadlines if given processors $4+6\cdot\left\lceil\frac{|R|}{\min(m_1,m_2)}\right\rceil$ times as fast. As far as we know, it is the first result for resource sharing on heterogeneous platforms with provable performance.

Keywords: heterogeneous multiprocessor systems, real-time scheduling, resource sharing

1 Introduction

In heterogeneous multiprocessor platforms (i) not all processors are of the same type and (ii) task execution times depends on the processor type. Many manufacturers offer chips combining different types of processors [1, 13–15, 18]. Clearly, such chips are key components in heterogeneous systems, and such systems are increasingly used in practice. Yet, despite this trend, the state-of-art in real-time scheduling theory for heterogeneous multiprocessors is under-developed. The reasons include (i) processors typically sharing low-level hardware resources (e.g. caches, interconnects), which makes task execution times interdependent and (ii) dispatching limitations (e.g. some processors depend on another processor for dispatching [12]). Such idiosyncratic challenges must be addressed on a case-by-case basis, accounting for the particularities of the architecture. The state-of-art does offer some general ideas on analyzing shared low-level hardware resources [3, 16, 17] and scheduling co-processors [9, 11]. Ultimately though, the dependency of the task execution time on the processor-type is what inherently complicates the design of scheduling algorithms for heterogeneous platforms.

The problem of scheduling independent *implicit-deadline sporadic tasks* (i.e., for each task, its deadline is equal to its minimum inter-arrival time) on heterogeneous multiprocessors has been studied in the past, both for generic [5, 7, 6] and for two-type [4] platforms but without considering the case when tasks share resources. One might partition tasks to processors and apply a resource-sharing protocol conceived for identical multiprocessors (e.g. D-PCP [19]). However, protocols such as D-PCP are not as effective in minimizing *priority inversion* when used in heterogeneous multiprocessors. For example, a task holding a shared resource may be executing on a processor where it runs slowly – causing large priority inversion to other tasks and poor schedulability. Therefore, a resource-sharing protocol for heterogeneous platforms ought to be cognizant of the execution speed of each task on each processor. It should also provide a finite bound on how much worse it performs, compared to an optimal scheme.

This paper introduces an algorithm, FF-3C-vpr, for scheduling tasks that share resources on a two-type heterogeneous multiprocessor. It offers a guarantee that if a task set can be scheduled to meet deadlines by an optimal scheme that allows a task to migrate only when requesting or releasing a resource then FF-3C-vpr also meets deadlines if given processors $2 + 3 \cdot \left\lceil \frac{|R|}{\min(m_1, m_2)} \right\rceil$ times as fast. Notably this is the first result with provably good performance for resource sharing on heterogeneous multiprocessors – which are increasingly relevant.

In this paper, Section 2 briefs the system model and assumptions. Section 3 gives the main idea of FF-3C-vpr. Section 4 lists notations and results used later. Section 5 discusses virtual processors – integral to our algorithm, presented in Section 6 along with the proof of its performance. Section 7 concludes.

2 System Model and Assumptions

We consider the problem of scheduling implicit-deadline sporadic tasks that share resources on a two-type heterogeneous multiprocessor platform with *restricted migration* (defined later). The system is specified as follows:

- **Processors (Π):** The platform consists of m processors of which $m_1 \geq 1$ processors are of type-1 and $m_2 \geq 1$ processors are of type-2.
- **Shared Resources (R):** A set R of $|R|$ resources that tasks share.
- **Task set (τ):** There are n *implicit-deadline sporadic tasks* – for each task τ_i, its deadline is equal to its minimum inter-arrival time, denoted as T_i.
- **Execution Time and Utilization:** The worst-case execution time of τ_i on a type-z processor ($z \in \{1, 2\}$) is denoted by C_i^z and its utilization by U_i^z.

We make the following assumptions:

- **Sharing the resources:** Each task may request at most one resource from R (known at design time) and at most once by each job of that task.
- **Virtual processors:** *Virtual processors* are logical constructs, used as task assignment targets by our algorithm. A virtual processor vp_i acts equivalent to a (physical) processor of the same type with (scaled) speed $\frac{1}{f}$ – and we

assume that it can be "emulated" on a physical processor of the same type (of speed 1), using no more than $\frac{1}{f}$ of its processing capacity[3].

– **Restricted migration:** A job of a task may only migrate to another processor during execution when it requests a resource; it must then migrate back to the original processor upon releasing the resource. We call this model *restricted migration*. Migrations between processors of any type are allowed.

3 Overview of our approach

The key to our approach is to distinguish between three *phases* in the execution of a task and make different scheduling provisions for each of them (Figure 1):

– **Phase-A** of a task spans from its arrival until it requests a shared resource.
– In its **Phase-B**, the task is holding (or waiting for) the shared resource.
– In its **Phase C**, the task has released the resource.

The main structure of our approach is as follows:

1. Split the task execution into phases A, B and C – in essence creating three subtasks out of it. The phase-B and phase-C subtasks of a task "arrive" (i.e. first become ready to execute) at a (respective) fixed time offset to the arrival of the respective phase-A subtask. This ensures that subtasks "inherit" the inter-arrival time of the original task and exhibit no arrival jitter.
2. Use m physical processors to create a set VP of virtual processors, formed by disjoint sets $\mathrm{VP_{AC}}$ and $\mathrm{VP_B}$ (i.e. $\mathrm{VP}=\mathrm{VP_{AC}}\cup\mathrm{VP_B}$ and $\mathrm{VP_{AC}}\cap\mathrm{VP_B}=\emptyset$).
3. Phases A and C of a task are assigned (both) to a virtual processor $vp_j\in\mathrm{VP_{AC}}$. Phase-B of the same task is assigned to a virtual processor $vp_k\in\mathrm{VP_B}$.
4. The phase-A and phase-C subtasks of a task are scheduled using preemptive EDF on their assigned virtual processor in $\mathrm{VP_{AC}}$; the phase-B subtask is scheduled on its assigned virtual processor in $\mathrm{VP_B}$ using non-preemptive EDF – as a way of serializing accesses to shared resources[4].

[3] One intuitive way of achieving this is by dividing time to short slots of length S and using $\frac{1}{f}\cdot S$ time units in each slot to serve the workload of vp_i. By selecting S, we can then make the speed of the emulated processor arbitrarily close to $\frac{1}{f}$ (and in practice, S need rarely be impractically short) [10]. In strict terms, a sufficient condition for emulating m_1 type-1 virtual processors from $\mathrm{VP_{AC}}$ onto m_1 type-1 physical processors is: $\sum_{\substack{vp_i\in VP_{AC}\\ vp_i\ is\ type-1}} V_i < m_1$, where V_i is the speed of virtual processor vp_i (and similarly for type-2 processors in $\mathrm{VP_{AC}}$ and for $\mathrm{VP_B}$ processors). For more details (including how to tradeoff spare processing capacity for longer S), see [10].

[4] Observe that implementing multiple virtual processors on the same physical processor might in practice involve frequent "context-switching" between those. Yet, whenever a physical processor "context-switches" between a phase-B virtual processor and some other virtual processor mapped to it, this does not violate the semantics of non-preemptive scheduling on the phase-B virtual processor because we are only interested (for the purposes of resource access serialization) in ensuring that phase-B subtasks never preempt each other – and this property is not violated.

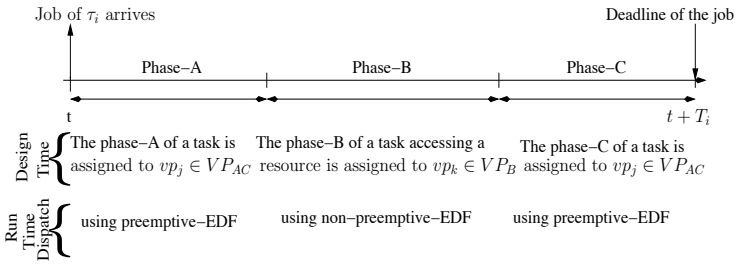

Fig. 1. Three execution phases of a job along with the design time (task assignment) and run time (task dispatching) decisions of FF-3C-vpr.

Steps 1-3 are performed at *design time*; step 4 is carried out at *run time*. Despite using virtual processors, our algorithm by-construction ensures that the "restricted migration" assumption is not violated – discussed in Section 5 and 6. Subtasks corresponding to task phases are assigned *constrained* deadlines, i.e. not exceeding their inter-arrival time (inherited from the original task).

4 Few Notations and Useful Results

4.1 Notations

Let $\Pi(m_1, m_2)$ denote a two-type heterogeneous multiprocessor platform having m_1 processors of type-1 and m_2 processors of type-2. Let $\Pi(m_1, m_2) \cdot \langle s_1, s_2 \rangle$ denote a platform in which the speed of a type-1 and type-2 processor is respectively, s_1 and s_2 times the speed of a type-1 and type-2 processor in $\Pi(m_1, m_2)$ platform (where s_1 and s_2 are positive real-numbers, i.e. $s_1 > 0$ and $s_2 > 0$).

Let the predicate $sched(A, \tau, \Pi(m_1, m_2) \cdot \langle s_1, s_2 \rangle)$ signify that a task set τ *meets all its deadlines* if scheduled by an algorithm A on a platform $\Pi(m_1, m_2) \cdot \langle s_1, s_2 \rangle$. The term *meets all its deadlines* in this and other predicates means 'meets deadlines for every possible valid arrival of jobs of tasks in τ'.

We use $sched(\text{nmo}, \tau, \Pi(m_1, m_2) \cdot \langle s_1, s_2 \rangle)$ to signify that there exists a *non-migrative-offline* preemptive schedule which meets all deadlines for the specified system. Here, *non-migrative* schedule refers to a schedule in which all the jobs of a task execute on the same processor to which the task is assigned. In this predicate (and others), the term *offline* means that the schedule (i) can contain inserted idle times and (ii) can be generated using knowledge of future task arrival times (irrespective of whether such knowledge is available in practice).

The predicate $sched(\text{rmo}, \tau, R, \Pi(m_1, m_2) \cdot \langle s_1, s_2 \rangle)$ signifies that there exists a *restricted-migration-offline* preemptive schedule which meets all deadlines for the specified system when tasks share resources from R. As mentioned in Section 2, each task requests at most one resource from R and each job of that task may request that resource at most once during its execution. The term "restricted migration" has the same meaning as discussed in Section 2.

Similarly, $sched(A, \tau, R, \Pi(m_1, m_2) \cdot \langle s_1, s_2 \rangle)$ signifies that τ "sharing the resources" (see Section 2) from R *meets all its deadlines* when scheduled by an algorithm A on $\Pi(m_1, m_2) \cdot \langle s_1, s_2 \rangle$ with "restricted migration" (see Section 2).

Finally, in the above predicates, the suffix -δ (where applicable, i.e. in (sub-) task-partitioned schemes) to a scheduling algorithm (or algorithm class) implies that the schedulability of τ (other than just being established via some exact test) must additionally be ascertainable via a (potentially pessimistic) *density-based* uniprocessor schedulability test. This means that for the sub-set τ' of (sub-)tasks assigned on every type-z processor of speed V, it has to hold that $\sum_{i \in \tau'} \delta_i^z \leq V$, where $\delta_i^z = \frac{C_i^z}{D_i^z}$ is the *density*, C_i^z is the execution time (w.r.t. a processor of speed 1) and D_i^z is the deadline of a task τ_i on a type-z processor.

On a type-z processor: Let $C_{i,1}^z$ denote the execution time of a task τ_i before requesting a resource, i.e. in its phase-A. Let $C_{i,2(k)}^z$ denote the execution time of τ_i while holding resource R^k (where k is the index of the resource used by τ_i), i.e. in its phase-B. Let $C_{i,3}^z$ denote the execution time of a task τ_i after releasing the resource, i.e. in its phase-C. Note that $\forall \tau_i \in \tau$: $C_{i,1}^z + C_{i,2(k)}^z + C_{i,3}^z = C_i^z$.

We derive three new *constrained-deadline* (denoted by D_i^z) *sporadic task* sets (i.e., for each task, its deadline is less than or equal to its minimum inter-arrival time) namely, $TD_A(\tau)$, $TD_{B,R^k}(\tau)$ and $TD_C(\tau)$ from implicit-deadline sporadic task set τ by modifying the parameters of the tasks in τ. Intuitively, (i) a task $\tau_{i(A)} \in TD_A(\tau)$ represents phase-A execution of $\tau_i \in \tau$, (ii) a task $\tau_{i(B)} \in TD_{B,R^k}(\tau)$ represents phase-B execution of $\tau_i \in \tau$, accessing the resource R^k and (iii) a task $\tau_{i(C)} \in TD_C(\tau)$ represents phase-C execution of $\tau_i \in \tau$.

$TD_A(\tau)$, $TD_{B,R^k}(\tau)$ and $TD_C(\tau)$ are defined as follows – for each task $\tau_i \in \tau$:

$$\tau_{i(A)} = \{T_{i(A)} = T_i, \quad D_{i(A)}^z = \frac{C_{i,1}^z}{C_i^z} \cdot \frac{T_i}{2}, \quad C_{i(A)}^z = C_{i,1}^z\}$$

$$\tau_{i(B)} = \{T_{i(B)} = T_i, \quad D_{i(B)}^z = \frac{T_i}{2}, \quad C_{i(B)}^z = C_{i,2(k)}^z\}$$

$$\tau_{i(C)} = \{T_{i(C)} = T_i, \quad D_{i(C)}^z = \frac{C_{i,3}^z}{C_i^z} \cdot \frac{T_i}{2}, \quad C_{i(C)}^z = C_{i,3}^z\}$$

Note that $D_i^A + D_i^B + D_i^C \leq T_i$. This is essential as it ensures that if $\tau_{i(A)}, \tau_{i(B)}$ and $\tau_{i(C)}$ derived from τ_i meet their deadlines then τ_i meets its deadline as well. Also, observe that $TD_A(\tau)$ and $TD_C(\tau)$ are derived such that the densities of $\tau_{i(A)}$ and $\tau_{i(C)}$ are twice the utilization of $\tau_i \in \tau$. For example,

$$\forall \tau_{i(A)} \in TD_A(\tau): \quad \delta_{i(A)}^z = \frac{C_{i(A)}^z}{D_{i(A)}^z} = \frac{C_{i,1}^z}{\frac{C_{i,1}^z}{C_i^z} \cdot \frac{T_i}{2}} = \frac{2C_i^z}{T_i} = 2U_i^z \text{of } \tau_i \in \tau \tag{1}$$

4.2 Useful Results

Lemma 1 and Lemma 2 (re-)state the speed competitive ratios of FF-3C (which is 2 – see Th. 1 in [4]) and of uniprocessor non-preemptive EDF (at most 3 –see Lem. 1 in [2]). FF-3C is a *non-migrative* scheduling scheme for implicit-deadline sporadic tasks (without resource sharing) on a two-type heterogeneous platform.

Lemma 1. *sched*$(nmo, \tau, \Pi(m_1, m_2)) \Rightarrow$ *sched*$(FF\text{-}3C,\ \tau, \Pi(m_1, m_2) \cdot \langle 2, 2\rangle)$

Lemma 2. *sched*$(nmo\text{-}np, \tau, \Pi(1, 0)) \Rightarrow$ *sched*$(nm\text{-}np\text{-}EDF, \tau, \Pi(1, 0) \cdot \langle 3, 3\rangle)$

The heterogeneous multiprocessor in Lemma 2 (with only one processor of type-1) is (trivially) a uniprocessor. (Lemma 2 also holds for $\Pi(0, 1)$ platform.)

Lemma 3 states that if a task is non-preemptive EDF-schedulable on a uniprocessor, it is also non-preemptive non-migrative (i.e. partitioned) EDF-schedulable on a platform with one more processor.

Lemma 3. *sched*$(nm\text{-}np\text{-}EDF,\tau,\Pi(1,0)\cdot\langle 3,3\rangle) \Rightarrow$ *sched*$(nm\text{-}np\text{-}EDF,\tau,\Pi(1,1)\cdot\langle 3,3\rangle)$

The intuition behind Lemma 3 is that if the additional (type-2) processor is ignored, τ is schedulable on the original (type-1) processor. (The lemma also holds for platform $\Pi(0, 1) \cdot \langle 3, 3\rangle$ in left-hand side predicate.)

Lemma 4. *(Combining Lemma 2 and Lemma 3)*
sched$(nmo\text{-}np, \tau, \Pi(1, 0)) \Rightarrow$ *sched*$(nm\text{-}np\text{-}EDF, \tau, \Pi(1, 1) \cdot \langle 3, 3\rangle)$

The following lemma states that if implicit-deadline task set τ is non-migrative offline schedulable on $\Pi(m_1, m_2)$ then constrained-deadline sporadic task set $TD_A(\tau)$ derived from τ (as described in Section 4.1) is also non-migrative schedulable (e.g. under partitioned preemptive EDF) on $\Pi(m_1, m_2)\cdot\langle 2, 2\rangle$ *and additionally* this can be established via use of a (potentially pessimistic) *density-based* schedulability test. It is easy to see that the claim holds since the density of a task $\tau_{i(A)}$ in $TD_A(\tau)$ is always twice the utilization of the corresponding task τ_i in τ.

Lemma 5. *sched*$(nmo, \tau, \Pi(m_1, m_2)) \Rightarrow$ *sched*$(nmo\text{-}\delta, TD_A(\tau), \Pi(m_1, m_2) \cdot \langle 2, 2\rangle)$

Proof. Let us assume that a non-migrative-offline feasible schedule exists for τ on $\Pi(m_1, m_2)$. So, there must exist a schedule in which the following holds:

$$\forall p \in \Pi(m_1, m_2) : \sum_{\tau_i \in \tau[p]} U_i^z \leq 1 \tag{2}$$

where $\tau[p]$ denotes the set of tasks assigned to processor p. Now, we show that there also exists a non-migrative-offline feasible schedule for $TD_A(\tau)$ on $\Pi(m_1, m_2) \cdot \langle 2, 2\rangle$.

We know that for every task $\tau_i \in \tau$ there exists a task $\tau_{i(A)} \in TD_A(\tau)$. We also know from Expression (1) that $\forall \tau_{i(A)} \in TD_A(\tau) : \delta^z_{i(A)} = 2U_i^z$ of $\tau_i \in \tau$. Let us assign $TD_A(\tau)$ to $\Pi(m_1, m_2)\cdot\langle 2, 2\rangle$ as follows: if $\tau_i \in \tau$ is assigned to processor $p \in \Pi(m_1, m_2)$ then assign $\tau_{i(A)} \in TD_A(\tau)$ to $p \in \Pi(m_1, m_2) \cdot \langle 2, 2\rangle$. From the fact that this assignment of $TD_A(\tau)$ (which is identical to the assignment of τ) is made on a platform twice faster (on which the densities of tasks will be halved) and from Expressions (1) and (2), we get:

$$\forall p \in \Pi(m_1, m_2) \cdot \langle 2, 2\rangle : \sum_{\tau_{i(A)} \in TD_A(\tau)[p]} \delta^z_{i(A)} \leq 1 \tag{3}$$

The above inequality corresponds to density-based schedulability test, on every processor $p \in \Pi(m_1, m_2)\cdot\langle 2, 2\rangle$, for partitioned preemptive EDF (which is a non-migrative algorithm). Thus, $TD_A(\tau)$ is also non-migrative-offline schedulable on $\Pi(m_1, m_2) \cdot \langle 2, 2\rangle$.

The following lemma largely follows from Lemma 1 — obtained by applying density-based schedulability test and on faster platforms and using the reasoning provided in Lemma 5.

Lemma 6. *(From Lemma 1 and Lemma 5)*
sched(*nmo-δ*, $TD_A(\tau), \Pi(m_1, m_2)\cdot\langle 2,2\rangle$) ⇒ *sched*(*FF-3C-δ*, $TD_A(\tau), \Pi(m_1, m_2)\cdot\langle 4,4\rangle$)

Proof. Assume that predicate *sched*(nmo-$\delta, TD_A(\tau), \Pi(m_1, m_2) \cdot \langle 2,2\rangle$) holds. Then, since the density of every (sub-)task in $TD_A(\tau)$ is twice the utilization of the corresponding (original) task in τ, (from the reasoning similar to the one provided in the previous lemma,) predicate *sched*(nmo, $\tau, \Pi(m_1, m_2)$) holds as well. In that case, we know from Lemma 1 that *sched*(FF-3C, $\tau, \Pi(m_1, m_2) \cdot \langle 2,2\rangle$) holds. But then, since the density of every (sub-)task in $TD_A(\tau)$ is twice the utilization of the corresponding (original) task in τ, it follows from similar reasoning provided in previous lemma that: *sched*(FF-3C-$\delta, TD_A(\tau), \Pi(m_1, m_2)\cdot \langle 4,4\rangle$).

Finally, a lemma that will be relied upon for assigning phase-C subtasks:

Lemma 7. *If, for a set $TD_A(\tau)[p]$ of phase-A subtasks,*

$$\delta_{TD_A(\tau)[p]} \stackrel{def}{=} \sum_{\tau_{i(A)} \in TD_A(\tau)[p]} \frac{C^z_{i(A)}}{D^z_{i(A)}} \leq V$$

then $TD_A(\tau)[p] \cup TD_C(\tau)[p]$ (where $TD_C(\tau)[p]$ is the set of the respective phase-C subtasks) is preemptive-EDF schedulable on a type-z (virtual) processor vp_p of speed V.

Proof. That $\delta_{TD_A(\tau)[p]} \leq V$ means that $TD_A(\tau)[p]$ is schedulable under preemptive EDF on vp_p. We now show that the *demand-bound function*[5], dbf(τ', t), of a task set $\tau' = TD_A(\tau)[p] \cup TD_C(\tau)[p]$ is upper bounded at every instant t by $\delta_{TD_A(\tau)[p]} \cdot t$ and hence is *also* schedulable on vp_p under preemptive EDF. Note that, for every phase-A subtask $\tau_{i(A)} \in TD_A(\tau)$ (and respective phase-C subtask $\tau_{i(C)} \in TD_C(\tau)$):

$$\text{dbf}(\{\tau_{i(A)}, \tau_{i(C)}\}, t) \leq \delta^z_{i(A)} \cdot t = \frac{C^z_{i(A)} \cdot t}{D^z_{i(A)}} \tag{4}$$

This is easy to verify because, the maximum "slope" to any point in the graph (Figure 2) of dbf($\{\tau_{i(A)}, \tau_{i(C)}\}, t$) from the origin is $\delta^z_{i(A)} = \frac{C^z_{i(A)}}{D^z_{i(A)}}$ (which is equal to $2U^z_i$ of $\tau_i \in \tau$, as per our choice of $D^z_{i(A)}$), at abscissa $t = D^z_{i(A)}$. Summation of Equation (4) over all $\tau_{i(A)} \in TD_A(\tau)[p]$ (and respective $\tau_{i(C)} \in TD_C(\tau)[p]$) yields:

$$\text{dbf}(TD_A(\tau)[p] \cup TD_C(\tau)[p], t) \leq t \cdot \sum_{\tau_{i(A)} \in TD_A(\tau)[p]} \delta^z_{i(A)} = t \cdot \delta_{TD_A(\tau)[p]}$$

[5] The *demand bound function* of a task τ_i, $dbf(\tau_i, t)$, is the maximum possible computation demand by jobs of τ_i, that have both release and deadline within any interval of length t. The demand bound function of a task set τ is defined as: $dbf(\tau, t) = \sum_{\tau_i \in \tau} dbf(\tau_i, t)$ [8].

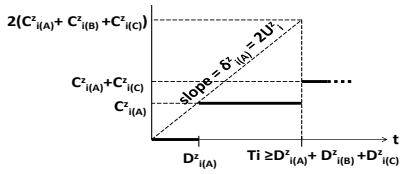

Fig. 2. Assigning phase-C sub-tasks to the same virtual processor as the respective phase-A sub-tasks (earlier assigned using a density-based test) preserves schedulability.

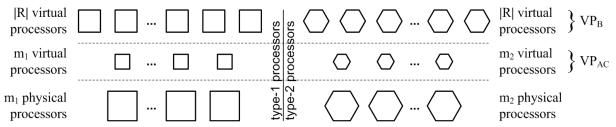

Fig. 3. $m + 2\,|R|$ virtual processors created from m physical processors on a two-type heterogeneous multiprocessor platform ($m = m_1 + m_2$).

5 Creating Virtual Processors on A Two-type Heterogeneous Multiprocessor Platform

We create $m + 2\,|R|$ virtual processors from m physical processors on a two-type heterogeneous multiprocessor platform as shown in Figure 3. The main idea is as follows. We treat physical processors of each type as an identical multiprocessor platform and create a certain number of virtual processors of the corresponding type from this platform. To be precise, m_1 physical processors of type-1 are treated as an identical multiprocessor platform and $m_1 + |R|$ virtual processors (of type-1) are created from them and ordered as shown in the left half of Figure 3 (i.e. left side of the vertical *solid line*). Analogously, m_2 physical processors of type-2 are treated as an identical multiprocessor platform and $m_2 + |R|$ virtual processors (of type-2) are created from them and ordered as shown in the right half of Figure 3 (i.e. right side of the vertical *solid line*). Now, if we look at each row in Figure 3 (separated by *horizontal lines*), it represents a two-type heterogeneous multiprocessor platform (for example, the second row represents a two-type heterogeneous multiprocessor platform with m_1 virtual processors of type-1 and m_2 virtual processors of type-2). Thus, $m + 2\,|R|$ virtual processors are created from m physical processors on a two-type heterogeneous platform. Precisely, we create the virtual processors with following specifications:

- m **virtual processors (denoted as** $\mathrm{VP_{AC}}$**):** m_1 virtual processors of type-1 each of speed $\frac{2}{2+3\left\lceil \frac{|R|}{m_1} \right\rceil}$ times the speed of a physical processor of type-1 and m_2 virtual processors of type-2 each of speed $\frac{2}{2+3\left\lceil \frac{|R|}{m_2} \right\rceil}$ times the speed of a physical processor of type-2. They are used to schedule phase-A and phase-C of a task execution and are referred to as '*virtual processors in* $\mathrm{VP_{AC}}$'.

Algorithm 1: FF-3C-vpr($\tau, \Pi^2(m_1, m_2), R$): for scheduling tasks that share resources on a two-type heterogeneous multiprocessor platform

```
   // Lines 1-17 execute offline; line 18 executes at run-time.
 1 Create TD_A(τ), TD_{B,R^k}(τ) and TD_C(τ) from τ as described in Section 4.1
 2 {VP_AC, VP_B} := VP_Create(Π^2(m_1, m_2), R) // Create VP_AC and VP_B virtual
     processors and store them in arrays of structures
 3 for i = 1 to |R| do //Form |R| pairs from 2|R| virtual processors in VP_B
 4 |   Pair_B[i] := ⟨VP_B[i], VP_B[|R| + i]⟩
 5 end
 6 Assign TD_A(τ) to virtual processors in VP_AC using FF-3C
 7 for i = 1 to n do
 8 |   if τ_i requests a resource then
 9 |   |   let k denote the resource that task τ_i requests
10 |   |   if (C^1_{i(B)} ≤ C^2_{i(B)}) then
11 |   |   |   assign τ_{i(B)} to VP_B[k]
12 |   |   else
13 |   |   |   assign τ_{i(B)} to VP_B[|R| + k]
14 |   |   end
15 |   end
16 end
17 Assign TD_C(τ) to virtual processors in VP_AC using the assignment made by FF-3C for
     phase-A of tasks on line 6, i.e. if τ_{i(A)} of TD_A(τ) was assigned to VP_AC[j] processor then
     assign τ_{i(C)} of TD_C(τ) to VP_AC[j] processor
18 Dispatch tasks in (i) TD_A(τ) with preemptive EDF on VP_AC, (ii) TD_B(τ) with
     non-preemptive EDF on VP_B and (iii) TD_C(τ) with preemptive EDF on VP_AC
```

– $2|R|$ **virtual processors (denoted as** $\mathrm{VP_B}$**):** $|R|$ virtual processors of type-1 each of speed $\frac{3}{2+3\left\lceil \frac{|R|}{m_1} \right\rceil}$ times the speed of a physical processor of type-1 and $|R|$ virtual processors of type-2 each of speed $\frac{3}{2+3\left\lceil \frac{|R|}{m_2} \right\rceil}$ times the speed of a physical processor of type-2. They are used to schedule phase-B of task execution and are referred to as '*virtual processors in* $\mathrm{VP_B}$'.

We ensure that no virtual processor is created using two or more physical processors, i.e., the capacity of a virtual processor comes from a single physical processor alone. The pseudo-code for creating virtual processors, referred to as VP_Create in the rest of the paper, can be found in Appendix (Section 8.1). Since VP_Create creates a virtual processor out of the processing capacity of a single respective physical processor, within each of its phases, any job executes on only one physical processor (i.e. does not migrate between different physical processors). However, it can migrate to a different physical processor at the boundaries separating (i) its phase-A and phase-B and (ii) its phase-B and phase-C executions. FF-3C-vpr adheres to the "restricted migration" model by assigning phase-A and phase-C of a task to the same physical processor.

6 FF-3C-vpr and its Speed Competitive Ratio

6.1 The FF-3C-vpr Algorithm

The pseudo-code of FF-3C-vpr is listed in Algorithm 1. The algorithm works as

follows. On line 1, it creates three subsets of tasks, i.e. $TD_A(\tau)$, $TD_{B,R^k}(\tau)$ and $TD_C(\tau)$ from the given task set τ. On line 2, it creates $m + 2|R|$ virtual processors specified in Section 5 from m physical processors. On lines 3-5, it groups $2|R|$ phase-B virtual processors into $|R|$ *pairs of processors*, each pair containing one processor of each type, i.e. one processor of type-1 and one processor of type-2. Each pair of processors, $Pair_B[k]$ where $k = \{1, \cdots, |R|\}$, is used for scheduling phase-B of tasks that access the resource R^k. At any time instant, only one processor from each heterogeneous pair is used for executing the tasks: this is, in each case, the processor of the type on which the given task executes fastest (termed the *favorite* processor type for that task); the other processor is kept idle during the execution of the task. This technique ensures mutual exclusion for accessing each resource. Moreover, it effectively creates, out of each pair, the equivalent of a hypothetical single virtual processor whereupon every task would execute as fast as on its (respective) favorite processor type. This design choice aims at minimizing blocking times related to resource sharing. On line 6, the algorithm assigns phase-A of a task (in $TD_A(\tau)$) to virtual processors in $\mathrm{VP_{AC}}$ using FF-3C [4]. On lines 7-16, it assigns phase-B of a task (in $TD_{B,R^k}(\tau)$) accessing resource R^k to that virtual processor in $Pair_B[k]$ which is of its *favorite* processor type in phase-B. On line 17, it assigns phase-C of a task (in $TD_C(\tau)$) to a virtual processor in $\mathrm{VP_{AC}}$ in the same manner as that of assignment of a task in $TD_A(\tau)$ to a virtual processor in $\mathrm{VP_{AC}}$ by FF-3C (on line 6). Instead of running FF-3C again on $TD_C(\tau)$ task set, the algorithm makes use of the output of FF-3C (that was run on line 6 to assign tasks in $TD_A(\tau)$ on $\mathrm{VP_{AC}}$) to assign $TD_C(\tau)$. Line 17 ensures that phase-C of a task is assigned to that virtual processor in $\mathrm{VP_{AC}}$ to which phase-A of the same task has been assigned. Assigning phase-C subtasks on the same virtual processor as its corresponding phase-A subtask (i) does not endanger the schedulability of a previously schedulable virtual processor; intuitively, this is because these two subtasks have precedence constraints – Lemma 7 provides formal proof and (ii) ensures that the "restricted migration" assumption is not violated. On line 18, FF-3C-vpr schedules tasks executing in their phase-A onto $\mathrm{VP_{AC}}$ using preemptive EDF, tasks in their phase-B onto $\mathrm{VP_B}$ using non-preemptive EDF and tasks in their phase-C onto $\mathrm{VP_{AC}}$ using preemptive EDF. Lines 1-17 can be performed at design time and only line 18 has to be performed at run time.

6.2 Time complexity of FF-3C-vpr

We now show that the time-complexity of FF-3C-vpr is a polynomial function of the number of tasks (n), processors (m) and/or resources ($|R|$). From FF-3C-vpr pseudo-code (Algorithm 1), we can observe that the time-complexity for:

- creating $TD_A(\tau)$, $TD_{B,R^k}(\tau)$ and $TD_C(\tau)$ subsets (on line 1) is $O(n)$.
- creating the virtual processor subsets, $\mathrm{VP_{AC}}$ and $\mathrm{VP_B}$ (on line 2) is $O(m)$.
- forming the virtual processor pairs (on lines 3-5) is $O(|R|)$.
- assigning $TD_A(\tau)$ on $\mathrm{VP_{AC}}$ using FF-3C (on line 6) is $O(n \cdot \max(m, \log n))$ [4].
- assigning $TD_{B,R^k}(\tau)$ on $\mathrm{VP_B}$ (on lines 7-16) is $O(n)$.

– assigning $TD_C(\tau)$ on $\mathrm{VP}_{\mathrm{AC}}$ (on line 17) is $O(n)$.

Thus the time-complexity of FF-3C-vpr is at most

$$\Big(\underbrace{O(n)}_{\substack{\text{create}\\ \text{subtasks}}} + \underbrace{O(m)}_{\substack{\text{create virtual}\\ \text{processors}}} + \underbrace{O(|R|)}_{\substack{\text{form virtual}\\ \text{processor pairs}}} + \underbrace{O(n \cdot \max(m, \log n)}_{\text{assign } TD_A(\tau)} + \underbrace{O(n)}_{\text{assign } TD_{B,R^k}(\tau)} + \underbrace{O(n)}_{\text{assign } TD_C(\tau)} \Big) = O(\max(n \cdot \max(m, \log n)), |R|) = O(n \cdot \max(m, \log n))$$

6.3 The Speed Competitive Ratio of FF-3C-vpr Algorithm

Theorem 1. *The speed competitive ratio of FF-3C-vpr is* $4 + 6 \cdot \left\lceil \frac{|R|}{\min(m_1, m_2)} \right\rceil$.

Proof. The proof considers separately the scheduling of each of the three phases and then combines the results. Let us look at phase-A first. Combining Lemma 5 and Lemma 6 and applying the result to virtual processors in $\mathrm{VP}_{\mathrm{AC}}$ yields:

$$sched(\text{nmo}, \tau, \Pi(m_1, m_2)) \Rightarrow sched(\text{FF-3C-}\delta, TD_A(\tau), \Pi(m_1, m_2) \cdot \langle 4, 4 \rangle) \quad (5)$$

Now consider phase-C. Since a task in its phase-A cannot be in its phase-C simultaneously (and vice versa), the respective sub-tasks are not independent. Treating them as such would be potentially pessimistic; conversely, accounting for these precedence constraints during (sub-)task assignment could improve performance. Indeed, our algorithm assigns each phase-C sub-task to the same virtual processor as its respective phase-A sub-task (Algorithm 1, line 17.).

For convenience, let us introduce a notation say, FF-3C-δ+cp for this (sub-) task assignment strategy (using FF-3C-δ to assign phase-A subtasks and "copying" the assignment for respective phase-C subtasks, as done by FF-3C-vpr on line 6). Then, applying Lemma 7 to Equation (5) yields:

$$sched(\text{nmo}, \tau, \Pi(m_1, m_2)) \Rightarrow sched(\text{FF-3C-}\delta\text{+cp}, TD_A(\tau) \cup TD_C(\tau), \Pi(m_1, m_2) \cdot \langle 4, 4 \rangle) \quad (6)$$

Now, let us consider phase-B. If tasks in τ sharing resource r_k are non-migrative-offline, non-preemptive schedulable on $\Pi(m_1, m_2)$ then $TD_{B,R^k}(\tau)$ is also non-migrative-offline, non-preemptive schedulable on $\Pi(m_1, m_2) \cdot \langle 2, 2 \rangle$. This speedup factor of 2 comes from the fact that we have halved the deadlines of tasks in $TD_{B,R^k}(\tau)$ compared to the deadlines of corresponding tasks in τ. Hence, we can write: $\forall R^k \in R$:

$$sched(\text{nmo-np}, \tau, \Pi(m_1, m_2)) \Rightarrow sched(\text{nmo-np}, TD_{B,R^k}(\tau), \Pi(m_1, m_2) \cdot \langle 2, 2 \rangle) \quad (7)$$

For each resource r_k, since r_k is accessed in a mutually exclusive way, all the tasks that access r_k must execute sequentially. So, if $TD_{B,R^k}(\tau)$ in which tasks share a single resource R^k is non-migrative-offline non-preemptive schedulable on $\Pi(m_1, m_2) \cdot \langle 2, 2 \rangle$ then the same task set is also non-migrative-offline non-preemptive schedulable on $\Pi(1, 1) \cdot \langle 2, 2 \rangle$. the intuition is that the tasks are

always executed on their 'favorite' processor type and in a sequential manner as they are accessing a mutually exclusive resource. $\forall R^k \in R$:

$$sched(\text{nmo-np}, TD_{B,R^k}(\tau), \Pi(m_1, m_2) \cdot \langle 2, 2\rangle) \Rightarrow sched(\text{nmo-np}, TD_{B,R^k}(\tau), \Pi(1, 1) \cdot \langle 2, 2\rangle) \quad (8)$$

Hence, combining Equations (7) and (8) gives: $\forall R^k \in R$:

$$sched(\text{nmo-np}, \tau, \Pi(m_1, m_2)) \Rightarrow sched(\text{nmo-np}, TD_{B,R^k}(\tau), \Pi(1, 1) \cdot \langle 2, 2\rangle) \quad (9)$$

Without loss of generality, Lemma 4 can be rewritten as:

$$sched(\text{nmo-np}, \tau, \Pi(1, 1)) \Rightarrow sched(\text{nm-np-EDF}, \tau, \Pi(1, 1) \cdot \langle 3, 3\rangle) \quad (10)$$

The intuition behind this generalization of Lemma 4 to Expression (10) is that the extra processor added to the left-hand side predicate (of Lemma 4 to obtain the Expression (10)) is ignored while scheduling.

Applying Equation (10) to a task set $TD_{B,R^k}(\tau)$ and multiplying the processor speeds by 2 on both left-hand and right-hand side platforms gives: $\forall R^k \in R$:

$$sched(\text{nmo-np}, TD_{B,R^k}(\tau), \Pi(1, 1) \cdot \langle 2, 2\rangle) \Rightarrow sched(\text{nm-np-EDF}, TD_{B,R^k}(\tau), \Pi(1, 1) \cdot \langle 6, 6\rangle) \quad (11)$$

Combining Equation (9) and (11) and applying the result to VP_B virtual processors: $\forall R^k \in R$,

$$sched(\text{nmo-np}, \tau, \Pi(m_1, m_2)) \Rightarrow sched(\text{nm-np-EDF}, TD_{B,R^k}(\tau), \Pi(1, 1) \cdot \langle 6, 6\rangle) \quad (12)$$

Combining the above intermediate results: dividing type-1 and type-2 processor speeds by, respectively, $4 + 6\left\lceil \frac{|R|}{m_1} \right\rceil$ and $4 + 6\left\lceil \frac{|R|}{m_2} \right\rceil$ in Equations (6) and (12) gives:

$$sched(\text{nmo}, \tau, \Pi(m_1, m_2) \cdot \left\langle \frac{1}{4 + 6\left\lceil \frac{|R|}{m_1} \right\rceil}, \frac{1}{4 + 6\left\lceil \frac{|R|}{m_2} \right\rceil} \right\rangle) \Rightarrow sched(\text{FF-3C-}\delta\text{+cp}, TD_A(\tau) \cup TD_C(\tau), \Pi(m_1, m_2) \cdot \left\langle \frac{2}{2 + 3\left\lceil \frac{|R|}{m_1} \right\rceil}, \frac{2}{2 + 3\left\lceil \frac{|R|}{m_2} \right\rceil} \right\rangle) \quad (13)$$

$$\forall R^k \in R : sched(\text{nmo-np}, \tau, \Pi(m_1, m_2) \cdot \left\langle \frac{1}{4 + 6\left\lceil \frac{|R|}{m_1} \right\rceil}, \frac{1}{4 + 6\left\lceil \frac{|R|}{m_2} \right\rceil} \right\rangle) \Rightarrow sched(\text{nm-np-EDF}, TD_{B,R^k}(\tau), \Pi(1, 1) \cdot \left\langle \frac{3}{2 + 3\left\lceil \frac{|R|}{m_1} \right\rceil}, \frac{3}{2 + 3\left\lceil \frac{|R|}{m_2} \right\rceil} \right\rangle) \quad (14)$$

In the right-hand sides of Equations (13) and (14), the processor specifications match those created by FF-3C-vpr. Note also that under FF-3C-vpr (which only allows "restricted migration"), phase-A and phase-C sub-tasks are assigned to virtual processors in VP_{AC} and phase-B sub-tasks are assigned to virtual

processors in VP_B (and $VP_{AC} \cap VP_B = \emptyset$). Hence by combining Equations (13) and (14) we get:

$$sched(\text{rmo}, \tau, R, \Pi(m_1, m_2) \cdot \left\langle \frac{1}{4 + 6\left\lceil \frac{|R|}{m_1} \right\rceil}, \frac{1}{4 + 6\left\lceil \frac{|R|}{m_2} \right\rceil} \right\rangle) \Rightarrow$$
$$sched\,(\text{FF-3C-vpr}, \tau, R, \Pi(m_1, m_2)) \tag{15}$$

We know that higher speed processors do not jeopardize the schedulability of a task set. Hence, we can write:

$$sched(\text{rmo}, \tau, R, \Pi(m_1, m_2) \cdot \langle \min(s_1, s_2), \min(s_1, s_2) \rangle) \Rightarrow$$
$$sched(\text{rmo}, \tau, R, \Pi(m_1, m_2) \cdot \langle s_1, s_2 \rangle)$$

Substituting $s_1 = \frac{1}{4+6\left\lceil \frac{|R|}{m_1} \right\rceil}$ and $s_2 = \frac{1}{4+6\left\lceil \frac{|R|}{m_2} \right\rceil}$ in the above equation and combining with Equation (15) and rewriting gives:

$$sched(\text{rmo}, \tau, R, \Pi(m_1, m_2) \cdot \left\langle \frac{1}{4 + 6 \cdot \max\left(\left\lceil \frac{|R|}{m_1} \right\rceil, \left\lceil \frac{|R|}{m_2} \right\rceil\right)}, \right.$$
$$\left. \frac{1}{4 + 6 \cdot \max\left(\left\lceil \frac{|R|}{m_1} \right\rceil, \left\lceil \frac{|R|}{m_2} \right\rceil\right)} \right\rangle) \Rightarrow sched(\text{FF-3C-vpr}, \tau, R, \Pi(m_1, m_2)) \tag{16}$$

Multiplying processor speeds in Equation (16) by $4 + 6 \cdot \max\left(\left\lceil \frac{|R|}{m_1} \right\rceil, \left\lceil \frac{|R|}{m_2} \right\rceil\right)$:

$$sched(\text{rmo}, \tau, R, \Pi(m_1, m_2)) \Rightarrow sched(\text{FF-3C-vpr}, \tau, R, \Pi(m_1, m_2) \cdot$$
$$\left\langle 4 + 6 \cdot \max\left(\left\lceil \frac{|R|}{m_1} \right\rceil, \left\lceil \frac{|R|}{m_2} \right\rceil\right), 4 + 6 \cdot \max\left(\left\lceil \frac{|R|}{m_1} \right\rceil, \left\lceil \frac{|R|}{m_2} \right\rceil\right) \right\rangle) \tag{17}$$

By rewriting the RHS of the above equation, we get:

$$sched(\text{rmo}, \tau, R, \Pi(m_1, m_2)) \Rightarrow sched(\text{FF-3C-vpr}, \tau, R, \Pi(m_1, m_2) \cdot$$
$$\left\langle 4 + 6 \cdot \left\lceil \frac{|R|}{\min(m_1, m_2)} \right\rceil, 4 + 6 \cdot \left\lceil \frac{|R|}{\min(m_1, m_2)} \right\rceil \right\rangle)$$

7 Conclusions

We proposed a new algorithm, FF-3C-vpr, for scheduling implicit-deadline sporadic tasks with restricted migration to meet all the deadlines on a two-type heterogeneous multiprocessor platform where each task can access at most one shared resource. We showed that FF-3C-vpr has a speed competitive ratio of $4 + 6 \cdot \left\lceil \frac{|R|}{\min(m_1, m_2)} \right\rceil$.

Acknowledgments

This work was partially supported by the REHEAT project, ref. FCOMP-01-0124-FEDER-010045, funded by FEDER funds through COMPETE (POFC – Operational Programme Thematic Factors of Competitiveness), National Funds through FCT – Portuguese Foundation for Science and Technology and REJOIN project of FLAD (Luso-American Development Foundation).

References

1. AMD Inc.: The AMD Fusion Family of APUs. http://sites.amd.com/us/fusion/apu/Pages/fusion.aspx
2. Andersson, B., Easwaran, A.: Provably good multiprocessor scheduling with resource sharing. Real-Time System 46(2), 153–159 (2010)
3. Andersson, B., Easwaran, A., Lee, J.: Finding an Upper Bound on the Increase in Execution Time Due to Contention on the Memory Bus in COTS-Based Multicore Systems. In: WiP of 30th IEEE Real-Time Systems Symposium (2009)
4. Andersson, B., Raravi, G., Bletsas, K.: Assigning real-time tasks on heterogeneous multiprocessors with two unrelated types of processors. In: 31st IEEE Real-Time Systems Symposium. pp. 239–248 (2010)
5. Baruah, S.: Task partitioning upon heterogeneous multiprocessor platforms. In: Proceedings of the 10th IEEE International Real-Time and Embedded Technology and Applications Symposium. pp. 536–543 (2004)
6. Baruah, S.: Feasibility analysis of preemptive real-time systems upon heterogeneous multiprocessor platforms. In: 25th IEEE Real-Time Systems Symposium (2004)
7. Baruah, S.: Partitioning real-time tasks among heterogeneous multiprocessors. In: Proc. of the 33^{rd} International Conference on Parallel Processing (2004)
8. Baruah, S., Mok, A., Rosier, L.: Preemptively scheduling hard-real-time sporadic tasks on one processor. In: IEEE Real-Time Systems Symposium (1990)
9. Bletsas, K.: Worst-case and Best-case Timing Analysis for Real-time Embedded Systems with Limited Parallelism. Ph.D. thesis, The University of York (2007)
10. Bletsas, K., Andersson, B.: Notional Processors: An Approach for Multiprocessor Scheduling. In: Proceedings of the 15th IEEE International Real-Time and Embedded Technology and Applications Symposium. pp. 3–12 (2009)
11. Gai, P., Abeni, L., Buttazzo, G.C.: Multiprocessor DSP scheduling in system-on-a-chip architectures. In: 14th Euromicro Conference on Real-Time Systems (ECRTS 2002). pp. 231–238. Vienna, Austria (Jun 2002)
12. Gschwind, M., Hofstee, H.P., Flachs, B., Hopkins, M., Watanabe, Y., Yamazaki, T.: Synergistic Processing in Cell's Multicore Architecture. IEEE Micro 26(2) (2006)
13. IBM Corp.: The Cell Project. http://www.research.ibm.com/cell/
14. IEEE Spectrum: With Denver Project NVIDIA and ARM Join CPU-GPU Integration Race. http://spectrum.ieee.org/tech-talk/semiconductors/processors/with-denver-project-nvidia-and-arm-join-cpugpu-integration-race
15. Intel Corporation: The 2nd generation Intel Core processor family. http://www.intel.com/en_IN/consumer/products/processors/core-family.htm
16. Li, Y., Suhendra, V., Liang, Y., Mitra, T., Roychoudhury, A.: Timing Analysis of Concurrent Programs Running on Shared Cache Multi-Cores. In: Proceedings of the 30th IEEE Real-Time Systems Symposium. pp. 57–67 (2009)
17. Lv, M., Guan, N., Yi, W., Yu, G.: Combining Abstract Interpretation with Model Checking for Timing Analysis of Multicore Software. In: Proceedings of the 31st IEEE Real-Time Systems Symposium. pp. 339–349 (2010)
18. NVIDIA: Dell and NVIDIA Workstation Solutions. http://www.nvidia.com/object/IO_16084.html
19. Rajkumar, R., Sha, L., Lehoczky, J.: Real-Time Synchronization Protocols for Multiprocessors. In: 9th IEEE Real-Time Systems Symposium. pp. 259 –269 (1988)

8 Appendix

8.1 Algorithm for Creating Virtual Processors

In our notation, PP denotes the set of physical processors, pp_i denotes the i^{th} physical processor and vp_i denotes the i^{th} virtual processor. The VP_Create pseudo-code for creating the specified virtual processors (in Section 5) is listed in Algorithm 2. It, in turn, uses the sub-routine VP_{ABC}_Create (Algorithm 3).

The VP_Create function on line 2 calls sub-routine VP_{ABC}_Create to create $m_1 + |R|$ virtual processors of type-1 from m_1 physical processors of type-1. The sub-routine first creates m_1 virtual processors (see lines 1-5 in Algorithm 3) from m_1 physical processors and then creates $|R|$ virtual processors (see lines 6-20 in Algorithm 3) from the remaining capacity of type-1 processors. Observe that no virtual processor is created using two physical processors, i.e. the capacity of a virtual processor comes from a single physical processor alone. Similarly, VP_Create() on line 3 creates $m_2 + |R|$ virtual processors of type-2 from m_2 physical processors of type-2.

Since VP_{ABC}_Create creates a virtual processor out of the processing capacity of a single respective physical processor, within each of its phases, any job executes on only one physical processor (i.e. does not migrate between different physical processors). However, it can migrate to a different physical processor at the boundaries separating (i) its phase-A and phase-B and (ii) its phase-B and phase-C. FF-3C-vpr adheres to the "restricted migration" model by assigning phase-A and phase-C of a task to the same physical processor.

The following observations can be made regarding our specification and creation of virtual processors. After creating one VP_{AC} virtual processor (for phase-A and phase-C) from every physical processor (lines 1-5 in the sub-routine shown in Algorithm 3), let us see (i) how much capacity remains in each of the physical processors and (ii) how many phase-B virtual processors (i.e. virtual processors in VP_{B}) can be created from that capacity. For ease of explanation, consider the case of type-1 processors. After creating one virtual processor, i.e. one in VP_{AC} (for phase-A and phase-C) of speed $\frac{2}{2+3\left\lceil \frac{|R|}{m_1} \right\rceil}$ (times the speed of a physical processor of type-1) from each physical processor, every physical pro-

Algorithm 2: VP_$Create$(PP, $|R|$): for creating virtual processors from a two-type heterogeneous platform

Input : PP, $|R|$
Output: VP_{AC}, VP_{B}
// PP denotes the set of physical processors
// $|R|$ denotes the number of shared resources
1 $\text{VP}_{\text{AC}}[1, \cdots, m] := \{0, \cdots, 0\}$ $\text{VP}_{\text{B}}[1, \cdots, 2\,|R|] := \{0, \cdots, 0\}$
2 VP_{ABC}_$Create(PP, \text{VP}_{\text{AC}}, \text{VP}_{\text{B}}, 0, 0, 1)$
3 VP_{ABC}_$Create(PP, \text{VP}_{\text{AC}}, \text{VP}_{\text{B}}, m_1, |R|, 2)$
4 **return** $\text{VP}_{\text{AC}}, \text{VP}_{\text{B}}$

Algorithm 3: $\text{VP}_{\text{ABC}}\_Create(\text{PP}, \text{VP}_{\text{AC}}, \text{VP}_{\text{B}}, lb, si, z)$: for creating phase-AC and phase-B virtual processors

Input : $\text{PP}, \text{VP}_{\text{AC}}, \text{VP}_{\text{B}}, lb, si, z$
Output: VP_{AC}, VP_{B}
// lb denotes the starting index for array VP_{AC}
// si denotes the starting index for array VP_B
// z denotes the processor type
$\text{VP}_{\text{AC}}[lb+1, \cdots, lb+m_z] := \{0, \cdots, 0\}$ // initialize the relevant elements in VP_{AC} to zero
for $i = 1$ **to** m_z **do**
  Create a virtual processor say, vp_i^{ACz} from pp_i of speed $\frac{2}{2+3\left\lceil \frac{|R|}{m_z} \right\rceil}$ times the speed of pp_i
  $\text{VP}_{\text{AC}}[lb+i] := vp_i^{ACz}$
end
$cnt := 1, flag := 0$
for $i = 1$ **to** m_z **do**
  for $j = 1$ *to* $\left\lceil \frac{|R|}{m_z} \right\rceil$ **do**
    create a virtual processor say, vp_{cnt}^{Bz} from pp_i of speed $\frac{3}{2+3\left\lceil \frac{|R|}{m_z} \right\rceil}$ times the speed of pp_i
    $\text{VP}_{\text{B}}[si+cnt] := vp_{cnt}^{Bz}$
    if $(cnt = |R|)$ **then**
      $flag := 1$
      break
    end
    $cnt := cnt + 1$
  end
  if $(flag = 1)$ **then**
    break
  end
end

cessor is left with a capacity: $1 - \frac{2}{2+3\left\lceil \frac{|R|}{m_1} \right\rceil} = \frac{3\left\lceil \frac{|R|}{m_1} \right\rceil}{2+3\left\lceil \frac{|R|}{m_1} \right\rceil}$. As per our specification (in Section 5), the phase-B virtual processor must have $\frac{3}{2+3\left\lceil \frac{|R|}{m_1} \right\rceil}$ times the speed of a physical processor of type-1. Hence, it is possible to create:

$$\left\lfloor \frac{\frac{3\left\lceil \frac{|R|}{m_1} \right\rceil}{2+3\left\lceil \frac{|R|}{m_1} \right\rceil}}{\frac{3}{2+3\left\lceil \frac{|R|}{m_1} \right\rceil}} \right\rfloor = \left\lfloor \left\lceil \frac{|R|}{m_1} \right\rceil \right\rfloor = \left\lceil \frac{|R|}{m_1} \right\rceil \geq 1$$

phase-B virtual processors from the remaining capacity of every physical processor of type-1. This allows us to successfully create $|R|$ phase-B virtual processors from the remaining capacity of m_1 processors of type-1. Analogous reasoning holds for type-2 processors as well.