Research Issues in Heterogeneous Multicore Systems: A Survey

Shaurya Choudhary[1], Yokesh Babu[2]

Vellore Institute of Technology, Vellore, Tamil Nadu, India

[1]shaurya.choudhary.2018@vitstudent.ac.in
[2] yokeshbabu.s@vit.ac.in

[1] 18BCE2113

ABSTRACT. In this technology driven era, the need for computation power is increasing day by day. There’s been a rapid development in the field of semiconductors which led to the formation of Heterogeneous Multicore Systems. Now, these processors are becoming a go-to solutions for satisfying these needs as they deliver much improved performance and at the same time are quite power efficient. But in order to keep up and provide better results over time, tons of researches are going on in all possible aspects of these systems in order to deliver better results. From designing hardware to implementing optimal scheduling algorithms, there’s plenty of stuff to tweak. In this survey paper we would be discussing few of these challenges and existing models to tackle them.

INTRODUCTION

A heterogeneous multicore processor architecture comprises of several CPU cores and a special purpose processor which provides more flexibility while working with advanced embedded systems. As the concept is comparatively new, there’s still a ton of challenges on which research is still going on. I have selected 5 research journals on some of these problems being faced in HMPs. However, my main focus would be on Dynamic Scheduling of Real-Time Tasks in Heterogeneous Multicore Systems.

Future heterogeneous single-ISA multicore processors will have an edge in potential performance per watt over comparable homogeneous processors. To fully tap into that potential, the OS scheduler needs to be heterogeneity-aware, so it can match jobs to cores according to characteristics of both. The shift from homogeneous to heterogeneous multicore introduces challenges to assign processes to appropriate core while maintaining the time deadline.

A model is proposed to handle scheduling issues which increases the system utilization by accommodating almost all the tasks into appropriate cores. Dispatcher is also used to allocate rejected works from small cores into the big cores which in turn improves system performance.

MIMO control theory poses several additional challenges in HMPs, a MIMO-based resource manager is proposed which will manage complexity of modeling the system dynamics. The presented design tackles the challenges like: (a) system decomposition and identification; (b) selection of suitable sensor and actuator granularity; and (c) appropriate system modeling to make the system identifiable as well as controllable. This provides a robust MIMO controller with rapid response and formal guarantees for coordinated management of HMPs.

In multichannel low-bit-rate vocoder applications, like ones mainly used in military and security fields, the vocoders provide a platform to perform on embedded devices efficiently. A multicore Application Specific Instruction Set Processor for multichannel low-bit-rate vocoders with real-time performance, HAVA, is proposed to provide efficiency along with flexibility. It integrates two types of processing cores and a shared-memory core on a 2-D-mesh on-chip network and cuts down the performance requirement of vocoders by almost half.

To tackle the energy management problem in HMPs, an algorithm is proposed (H-EARtH) which optimizes CPU platform energy by scheduling each workload according to its most advantageous core and managing voltage and frequency.

PROBLEM DESCRIPTION

There is plethora of issues faced in this domain. Heterogeneous Multicore systems have researches going in all kinds of aspect ranging from design methodology to scheduling algorithms. In this survey, we tried to get a broad idea about this vast range of problems by analyzing research journals addressing distinct issues.

To apply MIMO control theory with robust and responsive results, it requires complex design modeling of the dynamics of HMPs. This is one of the major issues in design segment of heterogeneous multicore systems. These processors are frequently used for special purposes like in military and security fields, to serve the requirements of vocoding, multiple channels of low-bit-rate vocoders are required. These vocoders perform efficiently on embedded devices but pose a critical problem to achieve that state.

Like everything else, HMPs also have some performance bounds. These performance scaling properties prevent the system from giving full throughput of its potential. This restriction is exploited by massive parallelism using Amdahl's Law.

Scheduling plays a very vital role in multicore systems. No matter how good the hardware is, the performance wouldn't be up to the mark without proper and efficient process scheduling. Scheduling affects both performance and power management. Different approaches are discussed to tackle performance by shifting from homogeneous to heterogenous multicores to maintain the time deadline and managing the frequency and voltage of CPU while scheduling workload to optimize platform energy.

We will further discuss ways to overcome the mentioned issues in the domain of Heterogeneous Multicore Systems.

DEFINITION

Cluster: In context of this paper, a set of cores of the same type in a Heterogeneous Multicore Platform is called cluster.

Dual Property: The dual property of a task represents that a task can execute on a cluster exactly when it is idle on the other cluster, and vice versa.

Vocoder: It is a program to analyze and synthesize the human voice signals to audio data for audio data compression, encryption, etc. [10]

Amdahl's Law: It gives the theoretical speedup in latency of task execution at fixed workload which can be observed from a system after improving its resources. [11]

RELATED WORKS ON THE DOMAIN

1. Design Methodology for MIMO

Controller Configuration: 2 Control-based resource managers are designed – (a) Two individual 2x2 MIMOs, one for each cluster. (b) A single systemwide 4x2 MIMO with a single IPS and power reference for the whole system. In the first manager, each controller tracks an IPS and power reference for its cluster. The controlled inputs are the cluster clock frequency and the number. The MIMO controllers are designed using MATLAB System Identification Toolbox. Accuracy is achieved using fourth-order model which is efficient for runtime invocation. The first manager meets all the design conditions whereas the second manager fails to meet the uniformity condition.

Controller Training: A custom micro-benchmark for system identification test waveforms is used which consists of a sequence of independently multiply-accumulate operations performed over both sequential and random memory locations, yielding varied instruction-level and memory-level parallelism. Test waveform is generated by running numerous instances of the benchmark in each cluster and varying control inputs in the format of a staircase test, both with single-input variation and all-input variation. Due to lack of memory sharing between the multiple instances, it enables to identify the system under very high variations in the system outputs given changes in the controllable inputs.[1]

Controller Robustness: A Kalman filter is used as an estimator for robust stability analysis whose role is to guess the state information by looking at the system outputs and inputs. An optimal tracking controller is designed and is liked with the estimator.[1] The tracker uses the state estimate from the estimator along with output tracking errors to generate the system inputs. It's ensured that the controller is stable for all the uncertainties. The maximum sustained impact of the mentioned uncertainties is bounded by a designer-specific margin.

Implementation: The controllers are implemented as Linux user space processes that execute in parallel with the applications. A lightweight kernel module collects the Instruction and cycle counters for computing IPS which access ARM's Performance Monitor Unit (PMU) on each core. Power is calculated per-cluster using the on-board current and voltage sensors present on the ODROID board. Power measurements are made in the same time increments as performance for each cluster. IPS/power measurements and controller invocation are performed periodically every 50 ms.

The proposed methodology of robust and predictable Multiple-Input-Multiple-Output control of Heterogeneous Multicore CPUs considers the non-uniform nature of the sensors and actuators in the system. It outlines the steps for proper system decomposition and system identification prior to the classical MIMO controller design process. The methodology ensures system stability and satisfies the objectives of the design. But the proposed designing approach suffers from exponential growth due to the i/o sizes and infeasibility of Dynamic System Model identification for larger MIMO systems.

2. Real-time Dynamic Scheduling

The proposed heterogeneous architecture model consists of 2 modules:

- *Module 1:* Consisting of simple small cores with low power. The cores are homogeneous in nature. This module is dedicated for executing low power tasks and the module is operational up to a certain low power level.
- *Module 2:* Consisting of complex big cores with high power and is responsible for executing high power tasks and the module is operational from a certain high-power level.

The architecture of module 1 constituted of numerous small low power cores (similar) along with an incorporated run time dispatcher which allowed:

  - To dispatch the matching jobs into small cores.
  - To migrate the rejected jobs from small cores into big cores for execution.

On the other hand, the architecture of module 2 has been incorporated where a few big complex cores are placed for high serial performance. These cores are designed for highly specialized application-specific task with high power.

In module 1, Searching to time-core map table is done, based on the pseudo-release time of new job of task in the Global queue, and accordingly forwarded to local queue of the matching cores through the run-time dispatcher. Subsequently the job has been allocated from local queue to small cores maintaining the constraints. The role of the run-time dispatcher is to forward the matching jobs to the local queues of the cores and migrate the rejected jobs to the Ready Queue of the module 2.

In module 2, partition scheduling is adopted to fulfill the objective of scheduling tasks into big complex cores. The partition scheduling maps tasks from the Ready Queue into suitable big cores.

A feasible matrix for task allocation into big cores as output is obtained in the form of solution.

The proposed model for dynamic scheduling of jobs in multi-core real time systems is divided into two modules: (a) for small simple cores designed for high parallel performance with low power, and (b) for big complex cores designed for highly specialized application specific task with high power. The working of first module is enhanced by modifying the run-time dispatcher while the second module is enhanced by the allocation of rejected jobs to the appropriate cores. The tasks are accommodated into cores according to their type. The proposed model, however, doesn’t address issues like resource sharing, load balancing, and other performance parameters.[2]

3. Vocoder Applications

The proposed model HAVA is a multicore ASIP used for low-bit-rate vocoders. It is based on 40nm CMOS technology and has a chip size of 12mm^2 which can run at a maximal frequency of 450MHz. It can perform multiple channels of 200bit/s vocoders in various scenarios with higher flexibility and lower power consumption. Different HPCs are compared by applying various VLIW issue widths for design exploration.

HPC adopting a four-way VLIW pipeline achieves the best tradeoff between performance improvement and hardware cost among different design alternatives. The MLPe 300bit/s encoder can run on two HPCs in a pipeline fashion.[3] After comparison of numerous combinations on the basis of performance gain and speedup, it's found that one little and one big HPCs are optimal for the encoder applications.

Maximal data throughput for all working HPCs is achieved, by the help of HMC along with DMA engines and SCs for efficient communication among the cores, when they acquire different codebook data for encoding or decoding tasks simultaneously. The sharing of codebook data between multiple channels leads to insignificant performance degradation. The proper scheduled requests to the HMC avoids almost all the access conflicts in the codebook. To reduce the performance requirements, frame level four-way parallelization is used. It reduces the performance requirements by around 42% of an encoder baseline with 2 HPCs.

A big HPC saves more than 45% cycles of other platforms when performing selected benchmarks. The benchmarks result also depict that HAVA is highly efficient for performing audio and speech processing.[3]

A multicore ASIP HAVA is proposed, for real-time processing of multichannel vocoders at low bit rates, which is based upin 3x3 mesh NoC. It consumes 149mW at 100MHz for four channels of MELPe encoders. The proposed model improves the performance of vocoders by 40% for multichannel processing with negligible efficiency loss. It also provides high scalability which allows it to substantially reduce (42 to 75 percent) the processing time of single-channel vocoders after frame-level parallelization.

4. Energy Management

A model is proposed for energy management of Heterogeneous Multicore Platform. For experimentation purposes, Cycle Accurate simulator is tested with two 3rd gen Intel Core processors as the big cores and four Intel Atom processors as the small cores sharing the Intel Core processor interconnect. The power and performance of multithreaded SPEC components at the eight frequencies is obtained using the simulator. The proposed H-EARtH algorithm is applied offline to find the frequency that optimally minimized the entire platform's energy consumption for each workload and for each core type independently.

For the entire workload, a fixed core type is selected. Simulator is again used to get the CPU power and SCA. SCA is calculated using the internal architectural memory stall counters and using the CPU power, the CPR (the platform power, p1, is characterized offline once and is stored in nonvolatile memory). The P1 for the rest of the platform is adopted from the real-system study.

The cross-prediction k ratio is extracted from a 100 percent scalable application (SPEC CPU 2000 gzip).

The obtained graph sorts the energy savings of the heterogeneous CPU versus the homogeneous CPU for all 37 workloads in ascending order. The left-most percent of the applications achieve the lowest energy by using the big core (yielding no energy savings on the heterogeneous CPU). The remaining 91 percent benefit from the heterogeneous architecture; 31 percent achieve the maximum 33 percent energy savings by using the small cores at RtH frequency. The heterogeneous CPU saves an average 21 percent of energy compared to the big core CPU.[4]

After the implication of H-EARtH algorithm on several systems, it's been observed that MHPs can perform computational tasks at lower platform energies than CPUs with only big cores. An average of 21% energy saving is obtained on all workloads. The algorithm allows scheduling a workload to the most advantageous core while managing the core's voltage and frequency. Due to limitations and compatibility with existing OSs, the model cannot utilize all the available cores.

5. Performance Enhancement

In heterogeneous CPU-GPU multicore, some cores can be more powerful than the others. The more powerful cores are built as the CPU cores while the comparatively weaker cores are built as the GPU cores. The computational performance is calculated with the assumption that the serial performance of data preparation is proportional to the amount of resources on the chip. A speedup equation is further obtained which is used to draw results. After calculations, the highest performance heterogeneous CPU-GPU architecture is composed of one CMP with 4 CPU cores, where each CPU core consists of 32 Base-Core Equivalents.[5] The remaining 128 BCEs are built are 128 GPU cores.

The Heterogeneous Dynamic CPU-GPU Multicore, in addition, can also schedule all the computation resources, and is also capable of executing base cores in parallel with either CPUs or GPUs cores. The speedup is further increased in this model, where assumption is made that the cost of data preparation is same as in the previous case.

Conclusions are drawn on the basis of quantitative analysis performed on theoretical and general model. When Overhead Data Preparation is taken into account [5], the speedup in HMPs do not grow linearly with increase in the number of cores. The obtained comparison curves of speedup distribution depict that a heterogeneous CPU-GPU architecture would be more beneficial than a homogeneous architecture.

ISSUE FOCUSED

The scheduling aspect of HMPs is mainly focused for further analysis and finding better approach to the problem. The scheduling on Identical Multicore Systems is significantly easier when compared to HMPs as in these systems the processing doesn't only depend on the core type, but also on the task executed. This raises a situation where a call has to be made to clarify which type of core will execute a task over time.

The most common and simple example of HMP is the ARM's big. LITTLE. It is a two-type heterogeneous multicore chip with high performance "big" cores and energy efficient "Little" cores. This processor is being widely used in the world of smartphones. But in order to further improve the performance of these devices, a fully-migrative optimal scheduling framework should be implemented. This is the goal of our proposed work. Despite of the differences in the type of cores, HMPs have usually same Instruction Set Architecture which allows task to be able to execute on any core and even migrate between them over time.

There are three types of migration: non-migrative, intra-migrative, and fully migrative. Non-migrative scheduling doesn't allow any shift and thus the task executes only on the core it is particularly assigned to. Tasks are allowed to migrate from one core to another of same type in intra-migrative scheduling while fully-migrative scheduling allow tasks to migrate between different cores of different types.

PROPOSED WORK

Optimal Real-Time Scheduling on Two-Type Heterogeneous Multicore Platforms. The proposed scheduling algorithm focuses on proving optimal real-time fully-migrative method on Two-Type HMPs. Fully-migrative scheduling algorithms are considered as a global approach towards the problem. An efficient way of achieving our requirements is proposed which divides the complete scheduling process into two stages: *workload assignment* and *schedule generation.* For workload assignment a per-cluster algorithm "Hetero-Split" is used which assigns fractions of workload of each task to both the clusters without losing the feasibility. Then, another algorithm is designed which decides how to schedule these workloads on the heterogeneous cores, and this algorithm is called as "Hetero-Fair". By tightly coupling these two proposed algorithms together we get our optimal Scheduling Algorithm, "Hetero-Wrap", for Two-type Heterogeneous multicore platforms. The finally obtained algorithm has time complexity of O(n), where n is the number of tasks to be scheduled.

The main HMP considered is ARM's big.LITTLE as along with having same ISA on both types of core, it also packs a specially designed interconnected bus that enables seamless data transfer among the cores. This design inclusion allows each task to migrate between the cores at much more reasonable cost.

MAIN CHALLENGE: The main challenge on heterogeneous scheduling arises from the fact that the execution time of a task varies depending on how much portion of its workload is assigned to each different cluster, since each task may have a different execution rate on a different cluster.[6]

In first stage, Workload Assignment, we determine the amount of partial workload on each cluster for each individual task. To increase the efficiency, No Parallel Execution rule is followed which restricts a particular task to execute on both the clusters at the same time. This restriction provides us a with a lot of simplicity as now we can address this issue by solving successive identical multicore scheduling. This restriction is taken care of by the dual property of tasks.

In second stage, Schedule Generation, we use simple guidelines to extend already existing identical multicore algorithms for two-type heterogeneous platforms. Using this an optimal scheduler is designed – “Hetero-Fair”.

In third stage, we combine our previous work and tightly couple the above two solutions to get our complete scheduling algorithm - “Hetero-Wrap”.

We further distribute the task migration into two kinds: intra-cluster migration and intra cluster migration in the fourth stage. This helps in deriving the first upper-bounds in the number of different types of migrations in the HMP.

In final stage, we run the simulation to quantitively evaluate the proposed solution and compare it against the classical scheduling approaches.

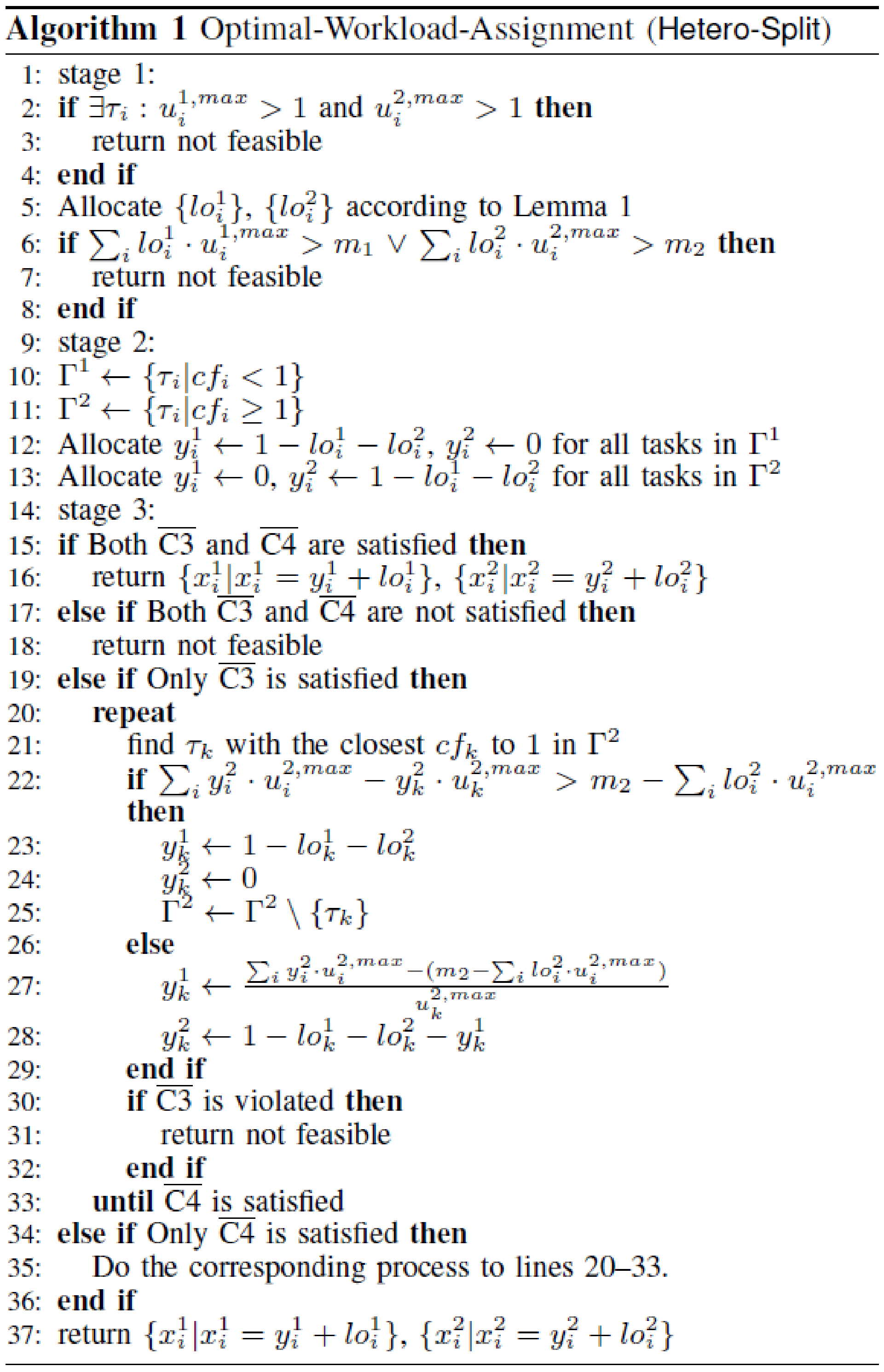
System model: We consider a two-type heterogeneous platform π consists of two clusters $\pi 1$ and $\pi 2$, where each cluster πk comprises mk identical cores of type-k (k = 1, 2). Each task $\tau i \in \tau$ is characterized by a period Ti, an execution requirement Ci, and a pair of execution rates, $r1i$ and $r2i$. Such a task τi is assumed to generate a potentially infinite sequence of jobs every Ti time-units, with each job needing to complete Ci units of work within a relative deadline of Ti time-units.

Important parameters to describe the execution behavior of τi on heterogeneous platforms are its execution rates, $r1i$ and $r2i$. The value of rki denotes the rate (or speed) at which work of task τi is completed on a core of type-k, indicating that executing τi on a core of type-k completes rki units of work per unit of time. We assume that type-1 and type-2 cores are unrelated in the sense that $r1i$ and $r2i$ can be assigned any arbitrary positive real values for all tasks $\tau i \in \tau$.

We assume that tasks are independent and preemptible, and migrate from one core to another within or across the cluster boundary during execution. Although our work takes both intra- and inter-cluster migrations into consideration, we employ the standard (incorrect) assumption that these occur at no cost or penalty. In actual systems, measured preemption and migration overheads can be accommodated by adjusting worst case execution time requirements when upper-bounds on the numbers of preemptions and migrations are available. We also assume that a single job cannot be executed simultaneously upon more than one core, regardless of core type.[6]

Hetero-Fair

The given algorithm is used to generate solution for workload assignment of the first stage. The algorithm keeps on checking if at any stage the process becomes infeasible. This feasibility is kept on check by using five constraints (C1-C5) which must be satisfied by the workload for assignment. These constraints are deadline constraint(C2) and capacity constraint(C3, C4).



Algorithm 1 Optimal-Workload-Assignment (Hetero-Split)

```
 1: stage 1:
 2: if $\exists \tau_i : u_i^{1,max} > 1$ and $u_i^{2,max} > 1$ then
 3:     return not feasible
 4: end if
 5: Allocate $\{lo_i^1\}$, $\{lo_i^2\}$ according to Lemma 1
 6: if $\sum_i lo_i^1 \cdot u_i^{1,max} > m_1 \vee \sum_i lo_i^2 \cdot u_i^{2,max} > m_2$ then
 7:     return not feasible
 8: end if
 9: stage 2:
10: $\Gamma^1 \leftarrow \{\tau_i | cf_i < 1\}$
11: $\Gamma^2 \leftarrow \{\tau_i | cf_i \geq 1\}$
12: Allocate $y_i^1 \leftarrow 1 - lo_i^1 - lo_i^2$, $y_i^2 \leftarrow 0$ for all tasks in $\Gamma^1$
13: Allocate $y_i^1 \leftarrow 0$, $y_i^2 \leftarrow 1 - lo_i^1 - lo_i^2$ for all tasks in $\Gamma^2$
14: stage 3:
15: if Both $\overline{C3}$ and $\overline{C4}$ are satisfied then
16:     return $\{x_i^1 | x_i^1 = y_i^1 + lo_i^1\}$, $\{x_i^2 | x_i^2 = y_i^2 + lo_i^2\}$
17: else if Both $\overline{C3}$ and $\overline{C4}$ are not satisfied then
18:     return not feasible
19: else if Only $\overline{C3}$ is satisfied then
20:     repeat
21:         find $\tau_k$ with the closest $cf_k$ to 1 in $\Gamma^2$
22:         if $\sum_i y_i^2 \cdot u_i^{2,max} - y_k^2 \cdot u_k^{2,max} > m_2 - \sum_i lo_i^2 \cdot u_i^{2,max}$
            then
23:             $y_k^1 \leftarrow 1 - lo_k^1 - lo_k^2$
24:             $y_k^2 \leftarrow 0$
25:             $\Gamma^2 \leftarrow \Gamma^2 \setminus \{\tau_k\}$
26:         else
27:             $y_k^1 \leftarrow \frac{\sum_i y_i^2 \cdot u_i^{2,max} - (m_2 - \sum_i lo_i^2 \cdot u_i^{2,max})}{u_k^{2,max}}$
28:             $y_k^2 \leftarrow 1 - lo_k^1 - lo_k^2 - y_k^1$
29:         end if
30:         if $\overline{C3}$ is violated then
31:             return not feasible
32:         end if
33:     until $\overline{C4}$ is satisfied
34: else if Only $\overline{C4}$ is satisfied then
35:     Do the corresponding process to lines 20–33.
36: end if
37: return $\{x_i^1 | x_i^1 = y_i^1 + lo_i^1\}$, $\{x_i^2 | x_i^2 = y_i^2 + lo_i^2\}$
```

Fig. Hetero-Fair Algorithm for workload assignment.[6]

The time complexity of this Hetero-Split algorithm is O (n log n).

Deadline Constraint: As a first step, we calculate the minimum fractions of workload of τi on type-1 and type-2 clusters in order to satisfy constraint C2. By the deadline constraint C2, if there exists a task τi such that u1,maxi > 1 and u2,max i > 1 hold, the task cannot satisfy the constraint, which leads to infeasibility. However, if u2,max i > 1 and u1,max I ≤ 1 hold, we may find a feasible solution by assigning some workload of τi to type-1 cluster. In this case, there exists the minimum fraction of workload of τi that should be assigned to the type-1 cluster so as to satisfy constraint C2. Conversely, if u2,max i ≤ 1 and u1,max i > 1 hold, there exists the minimum fraction of workload of τi that should be assigned to the type-2 cluster with the same reasoning. If u2,max i ≤ 1 and u1,max i ≤ 1 hold, a task τi always satisfies constraint C2. [6]

Capacity Constraint: Since we reduce the problem by allocating lo1i and lo2i amount of fractions of workload to type-1 and type-2 clusters, respectively, the remaining step is to determine {y1 i } and {y2 i } such that C1–C5 is satisfied.

Each task has different execution behavior between clusters. A task τi consumes the capacity of u1,max i if fully allocated on the type-1 cluster, and u2,max i on the type-2 cluster. We define cfi as τi's capacity efficiency ratio of type-1 cluster to type-2 cluster, expressed as

$$cf_i = \frac{u_i^{1,max}}{u_i^{2,max}}.$$

If cfi > 1, executing τi on the type-2 cluster is more capacity efficient than the type-1 cluster; on the contrary, if cfi < 1, the converse holds. Thereby, if there is no capacity limit for each cluster, allocating all of the remaining workload of τi to its capacity-efficient cluster consumes the least capacity.

However, each cluster has its capacity limit as shown in constraints C3 and C4, so it might be impossible to allocate all τi with cfi > 1 on the type-2 cluster (or all τi with cfi < 1 on the type-1 cluster). Consequently, we need to rearrange each task workload allocation in order to satisfy cluster capacity limits.

Equations:

$$\begin{aligned}
&\overline{\text{C1}}\text{: } \forall \tau_i \in \tau, y_i^1 + y_i^2 = 1 - lo_i^1 - lo_i^2,\\
&\overline{\text{C3}}\text{: } \sum_{\tau_i \in \tau} y_i^1 \cdot u_i^{1,max} \le m_1 - \sum_i lo_i^1 \cdot u_i^{1,max},\\
&\overline{\text{C4}}\text{: } \sum_{\tau_i \in \tau} y_i^2 \cdot u_i^{2,max} \le m_2 - \sum_i lo_i^2 \cdot u_i^{2,max},\\
&\overline{\text{C5}}\text{: } \forall \tau_i \in \tau, 0 \le y_i^1, y_i^2 \le 1 - lo_i^1 - lo_i^2.
\end{aligned}$$

The constraints are calculated using the above equations and are accordingly implemented in the proposed algorithm.

Hetero-Fair

In this stage, simple guidelines must be derived to extend the DP-fair guidelines [9] which provides optimal solution for identical multicore scheduling.

This works on the concept of proportionate fairness, that each task should execute proportionally to its utilization. In DB-fair guidelines it has been shown that by applying this requirement even only at job deadlines give optimal scheduling.

Deadline partitioning is done which slices the time based on the deadline of the task.

Rule 1: Always run all the jobs with zero task-level local laxity (i.e., Li(t) = 0);

Rule 2: Never run a job with no workload remaining on both clusters in the slice (i.e., R1i (t) = 0 and R2i (t) = 0);

Rule 3: Always allocate m1 jobs on the type-1 cores at time t if its cluster-level laxity is zero (i.e., L1(t) = 0), and allocate m2 jobs on the type-2 cores at time t if its cluster-level laxity is zero (i.e., L2(t) = 0);

Some more hidden rules are applied in order to extend the algorithm for our requirements.

Hetero-Wrap

Then the above 2 stages are tightly coupled to compile our final scheduling method, which provides the fully-migrative dynamic scheduling of tasks in HMP. The algorithm also holds the dual property and performs more optimally.

Using some theorems it has been shown that even after obtaining the final algorithm, it satisfies each and every rule of Hetero-Fair.

EVALUATION

For the evaluation, 3 input parameters are taken – number of cores on each platform (m1, m2), number of tasks (n), and individual task parameters ($u^{1,max}_i$, $u^{2,max}_i$). Here u denotes the utilization of each cluster by an individual task.

A big task set of 80,000 feasible tasks is generated to test the proposed scheduler (OUR). Then, the obtained results are compared against intra-migrative algorithm (SA) and non-migrative algorithm (SA-P).

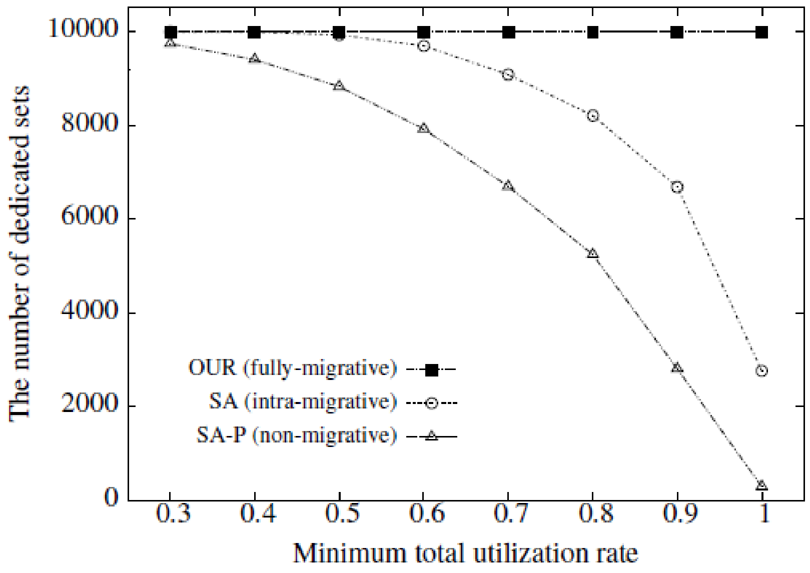


Fig. The number of schedulable tasks sets by OUR, SA, and SA-P. [6]

The obtained graph after simulation depicts the number of tasks that deemed schedulable by the respective algorithms. The proposed algorithm OUR shows 100% result by being able to schedule all the tasks while the classical algorithms SA and SA-P were only able to schedule 17% and 36% less tasks than OUR, respectively. This confirms that the proposed work promises full protection against process starvation while in the other 2 algorithms, several tasks suffer from starvation and never get to execute.

As a result, OUR outperforms both intra-migrative and non-migrative approaches while retaining the same time-complexity.

RESULT

We addressed the optimal two-type heterogeneous multicore fully-migrative scheduling problem. We proposed Hetero- Split as a feasibility-optimal per-cluster workload assignment algorithm and introduced the Hetero-Fair scheduling guidelines to correctly schedule all tasks without any deadline miss on a two-type heterogeneous multicore platform. We then presented the Hetero-wrap optimal two-type heterogeneous multicore scheduling algorithm that implements Hetero-Fair by exploiting McNaughton’s wrap-around rule based on a property (i.e., a dual property) derived in Hetero-Split. Finally, we provided the first upper-bounds on the numbers of intra- and inter-cluster migrations under two-type heterogeneous multicore scheduling, respectively.

In this paper, we viewed fully-migrative scheduling on a two-type heterogeneous multicore platform from the perspective of a collection of identical multicore scheduling for each cluster while cooperatively handling the NPE restriction. As a first attempt to explore such a perspective, we extended one of the simplest optimal scheduling algorithms for identical multicore platforms toward two-type heterogeneous scheduling. Recently, new optimal scheduling algorithms, such as

Bfair [8], proposed for identical multicore platforms with the aim of reducing the number of preemptions and migrations. As a future work, we will extend those advanced scheduling techniques to two-type heterogeneous scheduling with the additional consideration on two types of migration showing different costs, aiming at reducing the overall preemption and migration overheads.[6]

CONCLUSION

This paper gives a survey of different research issues and proposed methods to counter them in the domain of Heterogenous Multicore Systems. The issues discussed in this paper consist of different aspects of the domain, consisting of design, scheduling, energy management, etc. After getting a rough idea about the subject, we further focus on the domain of Dynamic Scheduling of HMP and evaluate is against already existing classical algorithms. We find that, the proposed work delivers better result and gives us a good overview about how the system works. Although the works discussed in this paper deliver impressive results, but there’s still plenty of room for further development.

After slightly analyzing the vast aspects of the domain, few shortcomings can be answered in upcoming time to enhance performance and make most of the resources. Some suggested future work is –

1. Energy-aware scheduling algorithms can be used on HMPs. Token based scheduling could be used for task mapping to obtain better performance and energy efficiency.
2. A custom open-source OS environment can be developed which will support the existing H-EARtH algorithm with extended compatibility, so all the available CPU cores can be utilized for better energy management.
3. The general digraph task model (modelling arbitrary directed graphs for sequential job releases) can be extended for real-time applications to attain intra-task parallelism. This provides an energy-efficient real-time HM Systems.

The technology has already achieved such heights that in day today life of a common man, we notice approximately no to very little delay in execution of programs in our devices. It would be interesting to see how far the development in this field goes and not only makes our lives easier but contribute in the overall technical development.

REFERENCES

1. *Mück, T., Donyanavard, B., Moazzemi, K., Rahmani, A. M., Jantsch, A., & Dutt, N. (2018). Design Methodology for Responsive and Rrobust MIMO Control of Heterogeneous Multicores. IEEE Transactions on Multi-Scale Computing Systems, 4(4), 944-951.*

2. *Baital, K., & Chakrabarti, A. (2018). Dynamic Scheduling of Real-Time Tasks in Heterogeneous Multicore Systems. IEEE Embedded Systems Letters, 11(1), 29-32.*
3. *Wei, Z., Liu, P., Sun, R., Dai, J., Zhou, Z., Geng, X., & Ying, R. (2016). HAVA: Heterogeneous Multicore ASIP for Multichannel Low-Bit-Rate Vocoder Applications. IEEE Transactions on Very Large Scale Integration (VLSI) Systems, 24(7), 2593-2597.*
4. *Rotem, E., Weiser, U. C., Mendelson, A., Ginosar, R., Weissmann, E., & Aizik, Y. (2016). H-earth: Heterogeneous multicore platform energy management. Computer, 49(10), 47-55.*
5. *Pei, S., Kim, M. S., & Gaudiot, J. L. (2016). Extending Amdahl's law for heterogeneous multicore processor with consideration of the overhead of data preparation. IEEE Embedded Systems Letters, 8(1), 26-29.*
6. *Chwa, H. S., Seo, J., Lee, J., & Shin, I. (2015, December). Optimal real-time scheduling on two-type heterogeneous multicore platforms. In 2015 IEEE Real-Time Systems Symposium (pp. 119-129). IEEE.*
7. *ARM, "big.little technology: The future of mobile," 2013. [Online]. Available: http://www.arm.com/files/pdf/big-LITTLE-Technology-the-Futue-of-Mobile.pdf*
8. *D. Zhu, X. Qi, D. Mosse, and R. Melhem, "An optimal boundary fair scheduling algorithm for multiprocessor real-time systems," Journal of Parallel and Distributed Computing, vol. 7, pp. 1411–1425, 2011.*
9. *G. Levin, F. Shelby, S. Caitlin, P. Ian, and B. Scott, "DP-Fair: A simple model for understanding optimal multiprocessor scheduling," in ECRTS, 2010.*
10. *https://en.wikipedia.org/wiki/Vocoder*
11. *https://en.wikipedia.org/wiki/Amdahl%27s_law*