Noname manuscript No.
(will be inserted by the editor)

Energy-Aware Real-Time Task Scheduling on Local and Shared Memory Systems

Gruia Calinescu · Chenchen Fu · Minming Li · Kai Wang · Chun Jason Xue

Received: date / Accepted: date

Abstract Motivated by applications in systems using Processing-In-Memory, we study four related scheduling problems, with the objective of minimizing the total energy consumption within the real-time constraints. In the preemptive Local-Shared Memory Task Scheduling problem, we are given n tasks $\tau_i(r_i, d_i, p_i, q_i)$, where integers r_i, d_i represent the release time and deadline of task τ_i, while integers p_i and q_i denote the non-negative processing time when τ_i is executing on the shared memory, and on the local memory, respectively. Each task is allocated to a separate core (n cores), and the shared memory is large enough to accommodate all the tasks. We must decide for each task whether to be scheduled on the shared or local memory, and when to turn on the shared memory (the variable g_j indicates the on-time for integer time slot j) such that for any task i assigned to the shared memory, $\sum_{j=r_i}^{d_i-1} g_j \geq p_i$. The objective is to minimize $\sum_j g_j + \sum_{i \mid \text{task } i \text{ assigned to local memory}} q_i$. For the non-preemptive version of this problem, we present a polynomial-time exact algorithm, based on dynamic programming. For the preemptive Local-Shared Memory Task Scheduling problem as well as the more general Multiple-task-per-core Local-Shared Memory Task Scheduling problem (where turning on the local memory of a core replaces q_j in the objective function), we give a 1.865-approximation algorithm based on a linear program. For both problems, we show that the integrality gap of the linear program matches exactly the approximation ratio. Finally, we prove that Multiple-tasks-per-core Local-Shared Memory Task Scheduling, with or without preemption, is NP-hard. Experimental results show that the proposed approximation algorithm performs close to the optimal solution in average.

G. Calinescu
Department of Computer Science, Illinois Institute of Technology, Chicago, USA

C. Fu, M. Li, K. Wang and C. J. Xue
Department of Computer Science, City University of Hong Kong, Hong Kong

Keywords scheduling · preemptive · dynamic program · NP-hardness · approximation algorithm · integrality gap · real-time systems · Processing-In-Memory · energy consumption · experimental results

1 Introduction

In the preemptive Single-task-per-core Local-Shared Memory Task Scheduling (or just Local-Shared Memory Task Scheduling) problem (LSMSP), we are given n tasks $\tau_i(r_i, d_i, p_i, q_i)$, where integers r_i, d_i represent the release time and deadline of task τ_i, while integers p_i and q_i denote the non-negative processing time when τ_i is executing on the shared memory, and on the local memory, respectively. Each task is allocated to a separate core (n cores), and the shared memory is large enough to accommodate all the tasks. We must decide for each task whether to be scheduled on the shared or local memory, and when to turn on the shared memory (the possibly fractional variable g_j indicates the on-time for integer time slot j) such that for any task i assigned to the shared memory, $\sum_{j=r_i}^{d_i-1} g_j \geq p_i$. The objective is to minimize $\sum_j g_j + \sum_{i \mid \text{task } i \text{ assigned to local memory}} q_i$. For the non-preemptive version of this problem, we present a polynomial-time exact algorithm, based on dynamic programming.

In the preemptive Multiple-tasks-per-core Local-Shared Memory Task Scheduling problem (MLSMSP), we are given n tasks $\tau_i(r_i, d_i, p_i, c(i))$, where integers r_i, d_i, and p_i represent the release time, deadline, and processing time on the shared memory of task τ_i, and $c(i)$ is the core to which τ_i belongs. Both the local and shared memory are large enough to accommodate all the tasks, and there is a cost $cost(c)$ for turning on the local memory of core c. We must decide which local memory to be turned on (and then all the tasks that belong to this core are assigned to local memory), and when to turn on the shared memory (the possibly fractional variable g_j indicates the on-time for integer time slot j) such that for any task i whose core memory is not turned on (and thus, the task is scheduled on the shared memory), $\sum_{j=r_i}^{d_i-1} g_j \geq p_i$. The objective is to minimize $\sum_j g_j + \sum_{c \mid \text{core } c \text{ turned on}} cost(c)$.

It can be easily seen that MLSMSP generalizes LSMSP. For preemptive MLSMSP, we give a 1.865-approximation algorithm based on a linear program. The integrality gap of the linear program matches exactly the approximation ratio, even for preemptive LSMSP. We prove that MLSMSP, with or without preemption, is NP-hard.

1.1 Motivation

Our problems model decisions that need to be made on whether tasks to execute in the shared memory or the local memory to minimize the total energy consumption within the real-time constraints.

As we are moving towards the Internet of Things (IoT), processing capabilities of sensors or smart devices are required to be faster and more powerful to better fulfill the real-time constraints or provide more satisfactory user experience. However, with the memory size increasing and the CPU frequency becoming higher, the discrepancy between computation speed (of CPU) and data transfer speed (of memory), commonly known as the memory wall [25], becomes wider, which limits the response speed of devices. To address this challenge, in current systems designers have to devote a large fraction of the transistors and area of chips to caches.

In recent years, the concept Processing-In-Memory (PIM) is proposed as a better solution to the memory wall issues and attracts wide interests. PIM, is an architecture that integrates CPU and memory into the single chip to narrow the CPU-memory performance gap, by more efficiently utilizing the internal memory bandwidth. It is reported that a 256MB memory system has at least 3000 times as much accessible memory bandwidth within chip than that to an external CPU [5]. It can achieve $5-10\times$ higher speed compared to most cache-based machines [11]. The integrated PIM chip can be used either as a conventional memory with simpler logic [4, 24] , or as a processor, which can handle data transfers in fast speed [11, 14, 17].

The main challenge of a PIM chip working as a processor is that the memory capacity is limited due to the fixed size of PIM chip. However, in the IoT system, typically the amount of data is not that large when being processed in the chips of sensors or smart devices, which makes the integration of the modest-sized memory in the PIM chip reasonable. On the other hand, to handle a large amount of data, several cores can share one off-chip large size memory. The shared memory is one of the most widely used memory architecture in the modern computing system, including the embedded systems on several kinds of smart devices [19, 27].

In the IoT system, energy efficiency is one of the most critical issues, as most of the senors or smart devices are energy harvested or battery powered [1]. Among the overall energy consumers, the static power of memory occupies a significant portion, as the memory chips are becoming denser with smaller technology scales. For example, in DRAM, which is widely used as memory, static power can be as much as 10 times of the dynamic read/write power on the memory chip using a process technology with the size smaller than 50nm [23]. Effectively reducing the static power can significantly improve the energy efficiency. One of the most widely applied methods to reduce the static power, is to switch the memory from the active state to the power-saving state, or directly turn it off when it is not accessed [2, 18].

In the memory system with local memory on chip and the off-chip shared memory, this work aims to explore the system energy efficiency by properly powering down memory. The challenges lie in how to allocate tasks into memory partitions (local or shared memory), and power down the idle memory properly while meeting the real-time constraints to minimize the energy consumption. On one hand, by allocating tasks to the local memory, shorter executing time can be achieved. In the meantime, the static power of local memory

is typically lower than the shared memory (because the local memory size is smaller). On the other hand, however, tasks scheduled in shared memory may enjoy execution-time overlap with other tasks from other cores, which in fact, brings less active time than the tasks' real execution time in shared memory. Meantime, as turning on/off the memory requires extra energy overhead, one can save the energy cost of turning on the local memory of a core if all its tasks are properly scheduled in shared memory.

Based on the integrated local memory on chip, with the off-chip memory shared among multiple cores in the IoT system, this work explores the task allocation and scheduling problem with the objective of minimizing the memory system energy consumption. Both complexity analysis and polynomial-time algorithms with constant approximation ratio are proposed. Experimental results show that the proposed scheme performs close to the optimal solution in average. The main contributions of this paper include:

- Two different task allocation models are defined based on whether the switching cost of the local memory dominates its static power of local memory;
- For the first task model, which assumes switching cost does not dominate the static power, an optimal Dynamic Programming (DP)-based algorithm within polynomial-time is proposed when tasks are non-preemptive;
- For the second task model, which assumes switching cost dominates the static power, APX-hardness is proved for both preemptive and non-preemptive versions;
- For both task models with preemption allowed for tasks, we firstly formulate the problem by Integer Linear Programming (ILP), and then propose an Linear Programming (LP)-rounding procedure. The LP-rounding procedure is proved to achieve an 1.865-approximation ratio compared to the LP solution, and this bound is then proved tight.

The rest of this paper is organized as follows. The related work is presented in Section 2. Section 3 presents the problem definitions, system models and a motivation example. In Section 4, with preemption not allowed, an optimal DP-based algorithm is proposed for the first task model. Section 5 proposes the LP-rounding procedure for both task models when preemption is permitted and proves a constant (circa about 1.865) approximation ratio. This constant is also proven to be the integrability gap of the linear program, which implies our rounding procedure is optimal (for this linear program). The APX-hardness analysis of the second task model is presented in Section 6. The experimental results are given in Section 7. Finally we conclude the paper in Section 8.

2 Related work

In this section, we introduce two groups of the most related works. First, studies based on integrating local on-chip memory into the CPU chip are

introduced. Second, the research works on minimizing the active time of shared memory among multiprocessors are presented.

There are mainly two working approaches of integrating memory and CPU on the same chip. One is using the PIM chip as the main memory with simple logic but large data, the other is using the PIM chip as a processor to perform faster processing capabilities. This work explores the energy efficiency based on the second approach, denoted as integrated memory on-chip in the following. The integrated memory architecture is firstly proposed by a group from Berkeley University [11, 20]. DRAM, as the main memory, is directly embedded in the processor chip. However, the size of DRAM is limited to 13MB [11, 12] due to the fixed chip size. More recently, the 3D stacking technology, which enables the construction of multiple layers of active silicon bonded with very dense vertical interconnects of CPUs and DRAM dies, is proposed to be applied in the PIM chip to explore larger size of memory [14, 21, 26]. Kgil et al. [14] proposed to stack 4 layers of DRAM on top of multi-core processors, the total size of which achieves 256MB. A 10%-20% performance improvement can be achieved compared to the cache-based memory system. In industry, Intel [21] demonstrated a chip, where each core has a 256KB local memory on top of the CPU, and this chip provides a bandwidth of 12GB/s for each core. Even though the memory size is limited by the chip size, in the IoT system, where typically the amount of processing data is not too large at a time, the local memory can provide fast processing capabilities for most of works. Meanwhile, this work proposed to share a larger off-chip memory among multiple cores to handle the larger task, and to explore better energy efficiency.

With the memory energy consumption becoming more and more significant, researchers have studied for several years to reduce the static power of memory by minimizing its active time. In [3], the authors explored the problem of minimizing the active time of the processor when the processor can schedule up to B tasks at a time. They pointed out that their problem is equivalent to minimizing memory active time if all tasks share the same memory. An LP-based optimal algorithm is proposed when preemption is allowed at arbitrary point. When preemption is only allowed at integral point, they proved the problem to be NP-hard when B is bounded by the number of tasks. Optimal polynomial time algorithms targeting at the same problem are proposed in [10] for the unbounded case (i.e. B is no less than the number of tasks). The Dynamic Programming (DP)-based solution to the non-preemptive version of the above problem is presented in [15]. Another similar problem is studied in [7], where each task has a unit demand, and each processor has a uniform capacity g, which implies at most g tasks can be non-preemptively scheduled on a processor simultaneously. They prove the NP-hardness of the problem when $g > 2$. In this paper, by considering the interactions between local memory and shared memory, more complex problems are studied. Novel algorithms are proposed for both the preemptive and non-preemptive cases.

3 Preliminaries

In this section, we first present the system model and the problem definitions based on two different task models, and then provide a motivation example to show the necessity of wisely scheduling tasks in the local-shared memory system.

3.1 Problem formulation and assumptions

System model: On a multi-core processor system, each core integrates the CPU with a local on-chip memory, and all cores share a large off-chip memory. The memory architecture is shown in Fig. 1. Tasks on cores have shorter latency to access the local memory, and longer latency when accessing the shared off-chip memory. We assume that a task accesses the memory during its whole execution period [28, 29]. The local memory of each core can be either turned on or off. We assume that switching on the local memory can be done instantly but requires extra energy cost. The shared memory contains multiple memory banks and can be turned to sleep state to reduce the static power when no core is accessing it. Waking up the shared memory also requires extra energy but it is negligible compared to the static power of shared memory. Actually, previous studies suggest that the schemes as well as the solutions will not differ much between considering and not considering the waking up cost of shared memory as shown in [6, 9].

Generally, the energy consumption of accessing the on-chip local memory is less than that of accessing the off-chip memory per memory access. As we assume that tasks access memory during their entire execution period, the dynamic energy is consumed as long as the memory is accessed. Hence the dynamic memory power can be roughly added to the static power of memory. In this work, as we mainly focus on the static energy of memory system optimization, we roughly assume that the static power of the shared memory $\alpha_s = \alpha_s^{orig} + \beta$, where α_s^{orig} is the original shared memory static power, and β is a parameter that represents the dynamic power increment normalized to the static power. So when tasks are executed in shared memory, both the static power and dynamic power (coefficient) of shared memory will be multiplied

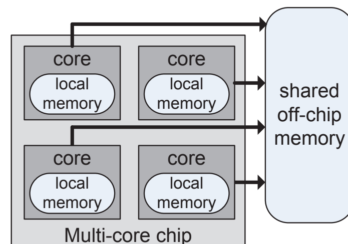


Fig. 1: The architecture of the local-shared memory system.

by the execution time to get the overall energy. The analysis of accurate and detailed dynamic energy of memory system is left as future work. Besides the memory energy, one may concern about whether the CPU energy cost might be affected due to different task execution time in different memories. Actually, the memory accessing time does not affect the CPU energy consumption, because CPU must wait until the memory accesses finish. During this waiting time, CPU is most likely to do something else, which are not related to the memory-associated tasks. Thus, no matter how long the memory is being accessed, the CPU energy will not differ much.

The memory architecture will be referred to as "local-shared memory system" in the following of this paper. For the system model, we made the following **assumptions**:

1. Tasks on different cores can access the shared memory in parallel.
2. Each task has already been assigned to a core (executed by the corresponding CPU), while it needs to be decided to either use the local memory, or not (using the shared memory). Migration is not allowed between memory partitions.
3. Each local memory is large enough to accommodate the tasks assigned to each core.
4. The shared memory is large enough to accommodate tasks from all cores. Tasks can be feasibly scheduled if all are executed in shared memory.

Note that the above assumptions do not over-idealize the problem. The reasons are presented after introducing the task models.

The static power of memory is related to the size of memory area. Considering the local memory is of relative small area, in this work we discuss the following two scenarios. First, the energy overhead of switching on/off the local memory is less than or is comparable with the static energy of local memory; second, the switch on/off energy cost is larger than the static energy in local memory. For these two scenarios, we formulate them into two system models: one-task-per-core and multiple-task-per-core models.

For the first scenario, both the active time in local memory and the switch on cost need to be considered. Even though tasks are already assigned to cores, it is basically equivalent to assuming that each task is allocated to a separate core. This is reasonable because for two tasks in the same core, the processing time when scheduled in the local memory is typically short, and it has high possibility that there is a gap between the scheduling of two tasks in local memory. Therefore we are most likely to pay for a switch on cost as long as a task is executed in the local memory. In this model, the main decision is that whether each task should be scheduled in the local memory or the shared memory. For the second scenario, it is the switch on cost that dominates the energy of local memory, especially when tasks on cores have short execution time and thus leads to smaller static energy of local memory comparing to the switch cost. It is reasonable to assume tasks on one core should well utilize the local memory once it is turned on. In this model, the main decision is which local memory to turn on, or not. For an "on" local memory, all its tasks will

be assigned to it, while for an "off" local memory, all its tasks go to the shared memory. Based on the above analysis, the features of both task models are listed as follows.
Single-task-per-core model for the first scenario:

1. Given n tasks, for each task $\tau_i(r_i, d_i, p_i, q_i)$, r_i, d_i represent the release time and deadline of task τ_i, while p_i and q_i denote the non-negative processing time when τ_i is executing on the shared memory, and on the local memory, respectively. W.l.o.g, we assume $p_i \geq q_i, \forall i$.
2. Each task is allocated to a separate core. The number of cores $C = n$. We denote $cost(i), i \in [1, n]$ as the switch cost of turning on the local memory of core i.

Power coefficients α_s and α_l, of the shared and local memory, respectively, are also given. To simplify the notation, we scale the coefficients to make $\alpha_s = 1$, and thus we reset q_i to be the $(q_i\alpha_l + cost(i))/alpha_s$, and we obtain the Single-task-per-core Local-Shared Memory Task Scheduling problem from the introduction.
Multiple-task-per-core model for the second scenario:

1. Multiple tasks can be assigned to the same core.
2. Each task τ_i does not have a q_i, but instead a core $c(i)$ to which it belongs, i.e. $\tau_i(r_i, d_i, p_i, c(i))$. The number of cores is denoted as C, $c(i) \in [1, C]$.
3. Each core c has a cost $cost(c)$ for turning on its memory.

Power coefficients α_s and α_l, of the shared and local memory, respectively, are also given. To simplify the notation, we scale the coeeficients to make $\alpha_s = 1$, and thus we reset $cost(i)$ to be the $cost(i)/alpha_s$, and we obtain the Multiple-task-per-core Local-Shared Memory task Scheduling problem from the introduction.

The system model assumptions do not over-idealize the problem. For tasks on different cores (assumption (1)), many techniques, such as bank/channel-level parallelism, the NUMA (non-uniform memory access) architecture, etc. have been proposed and applied to reduce the shared memory access conflict delay and increase the parallelism of scheduling. Therefore, even though tasks have the potential to be scheduled simultaneously to access the shared memory in the target problem, it is acceptable to assume that the access conflict delay can be ignored and tasks on different cores can access the shared memory in parallel. On the other hand, as this work mainly focuses on tasks scheduling issues, the analysis of the memory access congestion and behaviors are left as future works. For tasks that have too large data size to be allocated to the local memory (assumption (3)), they will be directly put into the shared memory. For assumption (4), we do not need to worry about that a set of tasks are not schedulable if all are executed on the shared memory for the single-task-per-core model. This is because under this model each task is allocated to a dedicated core, and hence they do not affect each other's schedulability. For the multiple-task-per-core model, if tasks on the same core are not schedulable

when all are executed on the shared memory, then it implies that the corresponding local memory should definitely be turned on for feasibility. We can move all tasks to be the local memory under this case. The problem discussed in this work ignores these trivial allocations.

Problem formulation: The objective of the target problem is to allocate tasks to memory partitions (local or shared memory), schedule them properly, turn the idle shared memory to sleep state, and turn off the idle local memory, in order to minimize the system memory energy consumption. This gives us the LSMSP for the **one-task-per-core model**, and MLSMSP for the **multiple-task-per-core model**, respectively. These problems are discussed for both the non-preemptive and preemptive cases in this work. In the following, for simplicity, by saying "turning on/off the core", it is equivalent to "turning on/off the local memory of the core".

3.2 Motivation example

To better understand the target problem, below we present a simple example based on the multiple-task-per-core model, without preemption. Three different task schedules are shown in Fig. 2. The schedule when tasks are all assigned to shared memory is based on the algorithms proposed in [10].

In Fig. 2, shadowed rectangles represent the execution time of tasks in shared memory, while tasks scheduling in local memory is represented by filling the "Core" with blue (shown in the legends). The energy consumption of shared memory E_s is calculated by multiplying the static power by total active time. For the local memory, the energy E_l is equivalent to the product of switch cost and the number of cores that are turned on. This example shows that by properly executing tasks in different memory partitions, better energy efficiency can be achieved. Intuitively, we can note that for tasks that have large overlapped execution time, like tasks τ_1, τ_2, τ_3, scheduling them in the shared memory might gain benefits by fully utilizing the active time of shared memory. For tasks like τ_4 and τ_5, which have no overlapped processing time with other tasks, it is better to schedule them on the local memory.

4 Non-preemptive case for single-task-per-core model

For the non-preemptive case of the LSMSP problem we adapt the Dynamic Programming (DP) proposed by [15], which minimizes the shared memory active time by properly scheduling tasks non-preemptively. The high-level idea of the DP algorithm for the LSMSP problem is as follows. Tasks are scheduled either in the shared memory or the local memory. In the shared memory, define an interval $[A, B]$, so that the whole time interval (e.g. $[-\infty, +\infty]$) can be divided into two sub-intervals by $[A, B]$. DP can be constructed by combining the solutions obtained from the sub-intervals.

Intuitively, select the task τ_k from T with the maximum p_k. The interval $[A, B]$ can be developed by enumerating all the possible execution intervals

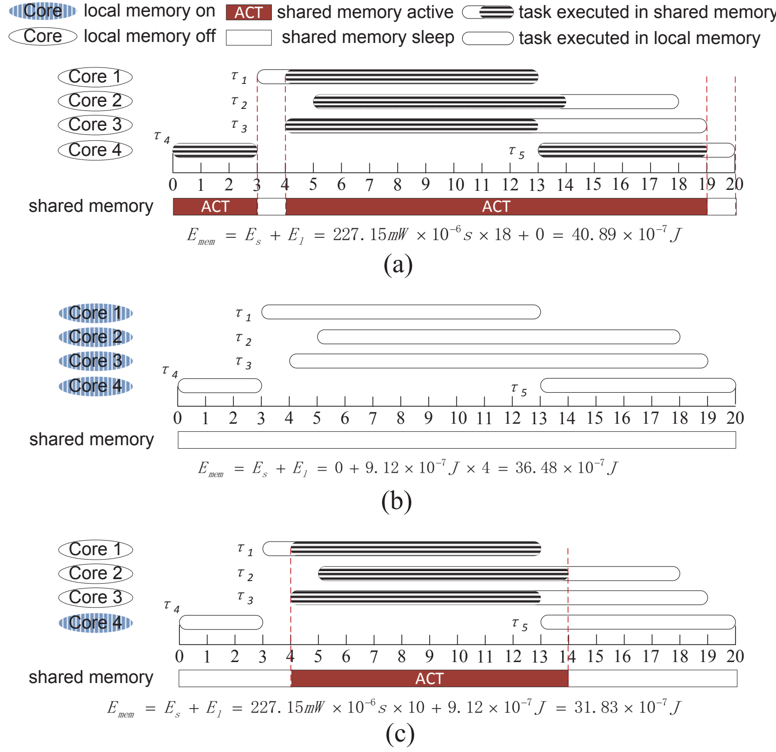


Fig. 2: Motivation example. Given a set of tasks under multiple-task-per-core model: $\tau_1(3, 13, 9, 1)$, $\tau_2(5, 18, 9, 2)$, $\tau_3(4, 19, 9, 3)$, $\tau_4(0, 3, 3, 4)$, $\tau_5(13, 20, 6, 4)$, three different schedules are shown. The static power (coefficient α_s) of shared memory is $227.15mW$, and the energy transition overhead of the local memory between active and power-down modes ($c(i)$, same for all i) is $9.12 \times 10^{-7}J$. Detailed configurations can be found in Section 7.1. Each time slot in this example represents 10^{-6}s. The energy consumption of each schedule is: (a) $40.89 \times 10^{-7}J$; (b) $36.48 \times 10^{-7}J$; (c) $31.83 \times 10^{-7}J$.

of τ_k, including $[r_k, r_k + p_k]$, and $[d_k - p_k, d_k]$ (the corresponding property is proved in [15]), $\forall k \in [1, i-1]$. We should assign as many tasks as we can to the interval $[A, B]$ to obtain the most benefit. Interval $[A, B]$ divides tasks into two subsets, T_1, which must be partially scheduled in $[-\infty, A]$, and T_2, which must be partially scheduled in $[B, +\infty]$ (based on τ_k being the longest task, it is not hard to see, and it is proved in [15]. that those tasks which are contained within $[A, B]$ do not need to be further considered in the recursive subproblems, while the tasks in T_1 do not interfere with those from T_2).

Formally, assume that the set of tasks is sorted in non-decreasing order of p_k. Let T^k be the list of the first k tasks in this order. To construct the DP, we define $OPT(L, R, k)$, for L and R in a certain set of integers, and

$0 \leq k \leq n$, to be the minimum objective that can be achieved to schedule tasks of T^k with the condition that $[-\infty, L]$ and $[R, +\infty]$ are active and cost nothing (in other words, with the objective function $\sum_{A \leq j \leq B-1} g_j + \sum_{i \mid \text{task } \tau_i \text{ of } T_k \text{ assigned to local memory}} q_i$). Note that the overall optimum is $OPT(-\infty, +\infty, n)$. The integer R in $OPT(L, R, k)$ comes from a polynomial-sized set of integers where tasks may start in an optimum solution, and L comes from a polynomial-sized set of integers where tasks may finish in an optimum solution. That these sets are polynomial-sized can be proven using arguments of [15]; without this, we can still get a pseudopolynomial algorithm.

We have as the base case $OPT(L, R, k) = 0$ if $L \geq R$ or $k = 0$. We have the recurrence relation: if $r_k + p_k \leq A$ or $d_k - p_k \geq B$, then

$$OPT(A, B, k) = OPT(A, B, k-1), \tag{1}$$

(task τ_k is schedule for free, completely outside interval (A, B)), and otherwise $OPT(A, B, k)$ is the minimum of $OPT(L, R, k-1) + q_k$, (task τ_j is scheduled in local memory), $(B - A)$ (all the remaining shared memory is used), and

$$(r_k + p_k - A) + OPT(r_k + p_k, B, k-1), \tag{2}$$

(task τ_k is scheduled in the shared memory, starting execution at r_k) and

$$(B - d_k + p_k) + OPT(A, r_k + p_k, k-1), \tag{3}$$

(task τ_k is scheduled in the shared memory, ending execution at d_k) and

$$\min_{A \leq A' \leq B - p_k} p_k + OPT(A, A', k-1) + OPT(A' + p_k, B, k-1) \tag{4}$$

(task τ_k is scheduled in the shared memory, fully in the interval (A, B)). The values A' are taken from the set of integers (mentioned above) when tasks can possibly start in an optimum solution; then $A' + p_k$ is in the set of integers (mentioned above) when tasks can possibly end in an optimum solution, and thus the subproblems to have the left and right value of the interval (A, B) of the required form.

Note that for the non-preemptive case, multiple-task-per-core model, this dynamic programming does not work. Intuitively, it can be noted that tasks executed in the local memory in the LSMSP problem are independent with each other, while those in MLSMSP are not. In MLSMSP, if a task is decided to execute in the local memory, all the tasks which share the same local memory will be forced to be scheduled in that local memory. In fact, the non-preemptive MLSMSP problem is NP-hard as shown in Section 6.

5 Approximation algorithm for both models with preemption

In this section, we firstly formulate the problem of both models by Integer Linear Programming (ILP), and then propose a LP-rounding procedure with constant and tight bound of the approximation ratio.

5.1 ILP formulation

In this section, we formulate the problem for both models by ILP. Given n tasks, we sort the release times and deadlines of all tasks, and divide the time into T time-intervals based on the sorted release times and deadlines, where $T \leq 2n$. We abuse notation to use r_i as the first time-interval during which task i can be (partially) executed, and d_i as the last time-interval during which task i can be (partially) executed. Let l_t be the length of time-interval I_t.

For both models, we assume that every task can be scheduled in the main memory, that is

$$\sum_{t=r_i}^{d_i} l_t \geq p_i, \ \forall i \in [1, n]. \tag{5}$$

Otherwise, in the single-task-per-core model, any task that does not satisfy the inequality above must be scheduled on local memory, which leads to a trivial solution for these special tasks. In the multiple-task-per-core model, if some task on a core cannot be scheduled on shared memory, we turn on the corresponding local memory and use it for all the tasks that belong to this local memory (without any decrease in the objective function). In the following, we end up solving the "reduced" instance where all tasks that cannot be scheduled in shared memory are removed from the instance.

Let the fractional variables x_t represent the active time of the shared memory in the time slot t, where $t \in [1, T]$. Note that the shared memory stays active as long as there is one core active. In other words, x_t represents the maximal interval where at least one task is scheduled during the time slot t. Let the variables z_i represent whether task τ_i is allocated to local memory (in which case $z_i = 1$) or the shared memory (in which case $z_i = 0$). The formulation for single-task-per-core model is given as follows:

$$\text{minimize} \quad \sum_{t=1}^{T} x_t + \sum_{i=1}^{n} z_i q_i, \tag{6}$$

$$\text{subject to} \quad \sum_{t=r_i}^{d_i} x_t \geq (1 - z_i) p_i, \ \forall i \in [1, n], \tag{7}$$

$$z_i \in \{0, 1\}, \tag{8}$$

$$0 \leq x_t \leq l_t, \tag{9}$$

Note that the formulation of LSMSP allows for fractional values of x_t, and we keep them so as it simplifies our discussion. As an aside, it is know that, once all z_i are integers, we get a Hitting Set problem were the sets are intervals of points, and the LP/ILP of such a Hitting Set instance is known to be Totally Unimodular, making the values x_t also integers.

For the multiple-task-per-core model, we use variable z_c to represent whether core c has its local memory turned on ($z_c = 1$) or off ($z_c = 0$). Based on this model, turning on the local memory of core c implies that all tasks in c are executed in local memory, while turning off the local memory means all tasks are

scheduled in the shared memory. The formulation for multiple-task-per-core model is given as follows:

$$\text{minimize} \quad \sum_{t=1}^{T} x_t + \sum_{c} z_c cost(c), \tag{10}$$

$$\text{subject to} \quad \sum_{t=r_i}^{d_i} x_t \geq (1 - z_{c(i)})p_i, \ \forall i \in [1, n], \tag{11}$$

$$0 \leq x_t \leq l_t, \ \forall t, \tag{12}$$

$$z_c \in \{0, 1\}, \ \forall c, \tag{13}$$

Based on the ILP formulation, the corresponding LP formulation can be developed. For LSMSP, the LP is to assume that an arbitrary task τ_i can be partially allocated to the shared memory. In other words, replace Equation (8) with

$$0 \leq z_i \leq 1. \tag{14}$$

Similarly, for MLSMSP, the corresponding LP replaces Constraint (13) to represent that the local memory of core c can be partially turned on by

$$0 \leq z_c \leq 1, \ \forall c. \tag{15}$$

In the following subsection, an LP-rounding procedure, which partitions an entire task to either shared memory or local memory based on the LP formulation solution, and its approximation ratio analysis are presented.

5.2 LP-rounding procedure

The approximation algorithm requires solving the linear programming in polynomial time, followed by a rounding procedure described in this subsection. The proposed rounding procedure works for both single-task-per-core and multiple-task-per-core models. W.l.o.g, we present the analysis based on multiple-task-per-core model as example, and the analysis toward single-task-per-core model can be done similarly.

Let δ be the rounding threshold, where $0 < \delta \leq 1$. For a given δ, the LP-rounding algorithm, which develops the scheduling results z'_c for each core c and $x'_t = x'_t(\delta)$ for each time slot t based on the LP solution is given in Algorithm 1.

As shown in Lines 1-7 in Algorithm 1, if $z_c > 1-\delta$, then we set the rounded solution $z'_c = 1$ (local memory of core c turned on, all tasks in c assigned to local memory). If $z_c \leq 1-\delta$, then we set $z'_c = 0$ (local memory of core c turned off, all tasks in c executed in shared memory). After rounding z'_c to integers, x'_t needs to be increased to satisfy Constraint (11). We firstly initialize each x'_t to be x_t (Lines 8-10), and then x'_t is updated for each time slot $t \in [1, T]$ (Lines 11-30). To update x'_t, two intermediate variables γ and γ' are defined, which are initialized to $(\frac{1}{\delta} - 1)x_t$ (Line 12). We try to add γ (γ') to x'_t in the first

Algorithm 1: LP-rounding procedure

Input: LP solution of x_t for each time slot t, z_c for each core c and rounding threshold δ
Output: LP-rounding solution x'_t and z'_c
1: **for** $c = 1$ **to** C **do**
2: **if** $z_c > 1 - \delta$ **then**
3: $z'_c = 1$
4: **else**
5: $z'_c = 0$
6: **end if**
7: **end for**
8: **for** $t = 1$ **to** T **do**
9: $x'_t = x_t$
10: **end for**
11: **for** $t = 1$ **to** T **do**
12: $\gamma' = \gamma = x_t \cdot \frac{1}{\delta} - x_t$; $j = t$
13: **while** $\gamma > 0$ and $j \leq T$ **do**
14: **if** $x'_j + \gamma > l_j$ **then**
15: $\gamma = x'_j + \gamma - l_j$; $x'_j = l_j$
16: **else**
17: $x'_j = x'_j + \gamma$; $\gamma = 0$
18: **end if**
19: $j = j + 1$
20: **end while**
21: $j = t$
22: **while** $\gamma' > 0$ and $j \geq 1$ **do**
23: **if** $x'_j + \gamma' > l_j$ **then**
24: $\gamma' = x'_j + \gamma' - l_j$; $x'_j = l_j$
25: **else**
26: $x'_j = x'_j + \gamma'$; $\gamma' = 0$
27: **end if**
28: $j = j - 1$
29: **end while**
30: **end for**

while loop (similarly, in the second **while** loop). If the updated x'_t exceeds l_t, the algorithm will add the remaining parts, i.e. $\frac{1}{\delta}x_t - x'_t$ to the next time slot $t+1$ until T ($t-1$ until 1 in the second **while** loop).

Lemma 1 *For* $\forall \delta \in (0, 1]$*, the variables* z'_c *and* x'_t *(both are functions of* δ*) output by Algorithm 1 are feasible for the integer linear programming (10).*

Proof To prove the LP-rounding solutions are feasible for the ILP, solutions z'_c and x'_t should be tested to satisfy all the constraints in ILP.

First, note that all x'_t are initialized to be at most l_t, and stay this way throughout the execution of the rounding procedure, as this will be constrained in Line 14 and 23. Thus Constraint (12) is satisfied. Next, the following analysis discusses the feasibility for Constraint (11).

Let γ_t be the value of the variable $\gamma = x_t \cdot \frac{1}{\delta} - x_t$ when executing the **for** loop for variable t. Let i be the task index. Constraint (11) can be trivially achieved if $z'_{c(i)} = 1$. On the other hand, if $z'_{c(i)} = 0$, then $1 - z_{c(i)} \geq \delta$ and we

know from Constraint (11) that

$$\sum_{t=r_i}^{d_i} x_t \geq (1 - z_{c(i)})p_i \geq \delta p_i. \tag{16}$$

For each task i, there are two cases which might appear in Algorithm 1. For the first case, every $t \in I^i$ is such that either the first **while** loop reaches $\gamma = 0$ with $j \leq d_i$ or the second **while** loop reaches $\gamma' = 0$ with $j \geq r_i$. Then under this case, when processing t in the **for** loop, at least γ_t is added to $\sum_{t=r_i}^{d_i} x'_t$, and thus overall at least $\sum_{t=r_i}^{d_i} \left(\frac{1}{\delta} - 1\right) x_t$ is added to $\sum_{t=r_i}^{d_i} x_t$. We conclude that

$$\sum_{t=r_i}^{d_i} x'_t \geq \sum_{t=r_i}^{d_i} x_t \left(1 + \frac{1}{\delta} - 1\right) = \sum_{t=r_i}^{d_i} x_t \frac{1}{\delta} \geq p_i,$$

where Equation (16) is used to obtain the last inequality. Thus Constraint (11) is satisfied for task i in terms of variables x'_t.

In the second case, there is a $t \in \{r_i, \ldots, d_i\}$ such that both **while** finish d_i and r_i with $\gamma > 0$ and $\gamma' > 0$ respectively. This means that $x'_j = l_j$ for every $j \in \{r_i, \ldots, d_i\}$. This, combined with Equation (5), implies that Constraint (11) is satisfied for task i in terms of the variables x'_t. □

5.3 Upper bound of LP-rounding procedure

In this subsection, Lemma 2 is firstly developed to bound the objective of the LP-rounding solutions for all values of δ, and then Lemma 3 is proposed to show the upper bound of the approximation ratio under proper δ.

Let $ALG = ALG(\delta)$ represent the objective of the proposed LP-rounding solution, and LP^* represent the objective of the optimal LP solution. That is,

$$LP^* = A + \sum_c z_c cost(c), \tag{17}$$

where

$$A = \sum_{t=1}^{T} x_t.$$

Lemma 2 *The objective of the LP-rounding solution (function of δ) satisfies $ALG(\delta) \leq (\frac{2}{\delta} - 1)A + \sum_{c:z_c \geq 1-\delta} cost(c)$.*

Proof Let γ_t be the value of the variable $\gamma = x_t \cdot \frac{1}{\delta} - x_t$ when executing the **for** loop for variable t. The value of $\sum_{j=1}^{T} x'_j$ increases by at most $2\gamma_t = x_t \left(\frac{2}{\delta} - 2\right)$ for each iteration in terms of t in the **for** loop. This is because there are two **while** loops, each of which will add at most γ_t to $\sum_{j=1}^{T} x'_j$, as shown in Algorithm 1 (Lines 13-20 and Lines 22-29). For $\forall t \in [1, T]$, the increase is at most $A\left(\frac{2}{\delta} - 2\right)$, and thus $\sum_{t=1}^{T} x'_t \leq A + A\left(\frac{2}{\delta} - 2\right) = (\frac{2}{\delta} - 1)A$. The lemma follows. □

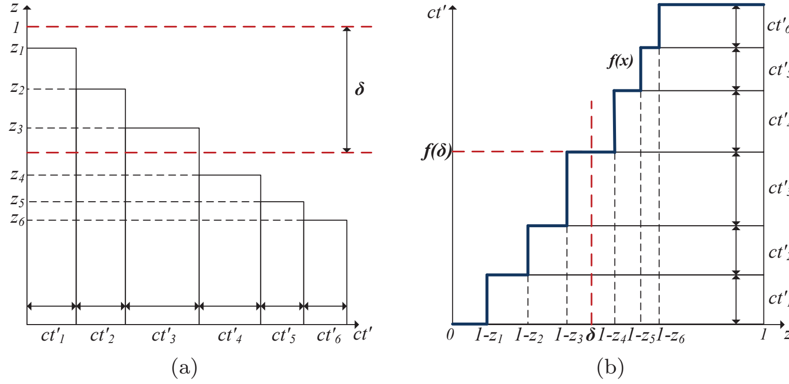


Fig. 3: Corresponding function $f(x)$ and the area of rectangles to help denote $ALG(\delta)$ and LP^* equations.

To obtain the upper bound of the approximation ratio of the proposed LP-rounding procedure, all the "interesting" values of δ should be examined, including 1 and all values of $1 - z_{c(i)}$. There are at most $n+1$ such values, and the solution of the minimum $ALG(\delta)$ can be developed.

Let ξ be the solution of the following equation

$$-2\ln(1 - \frac{1}{y}) = \frac{y+1}{y}, \tag{18}$$

with variable $y > 1$. This solution exists since the function $f(y) = \frac{y+1}{y} + 2\ln(1 - \frac{1}{y})$ has negative values if $y \to 1$, $f(2)$ is positive, and f is increasing as one can check that f's derivative is positive ($\xi \approx 1.865$). From the next lemma, it can be proved that the proposed LP-rounding procedure achieves the ξ-approximation ratio compared to the optimal LP solution (LP^*) under some proper δ for the preemptive case in both single-task-per-core and multiple-task-per-core models.

Lemma 3 *There exists $\delta \in (0, 1]$ such that $ALG(\delta) \leq \xi LP^*$.*

Proof For the sake of simplification, we eliminate the notation A from Equation (17) by setting:

$$cost'(c) = cost(c)/A, \forall c.$$

Note that if $A = 0$, then all z_c must be equal to 1 and any value of δ would work. Then, based on Equation (17) and Lemma 2, we have:

$$ALG(\delta)/A \leq \left(\frac{2}{\delta} - 1\right) + \sum_{c: z_c > 1-\delta} cost'(c) \tag{19}$$

Then we notice that once we prove that there exists a $\delta \in (0, 1]$ such that

$$\left(\frac{2}{\delta} - 1\right) + \sum_{c: z_c > 1-\delta} cost'(c) \leq \xi \left(1 + \sum_{c} z_c cost'(c)\right), \tag{20}$$

it follows that $ALG(\delta) \leq \xi LP^*$.

The part of $\sum_c z_c cost'(c)$ in the RHS of Equation (20) can be represented by an area of rectangles, as shown in Fig. 3(a). The index to z and $cost'()$ is sorted in the non-increasing order of z_c, and we use in Fig. 3(a) ct'_c to represent $cost'$ of the cth core in this sorted order. To represent the part of $\sum_{c:z_c>1-\delta} ct'_c$ in the LHS of Equation (20), we rotate Fig. 3(a) 90 degrees counterclockwise, and obtain Fig. 3(b). In the example in Fig. 3(a), it can be noted that z_1, z_2 and z_3 are larger than $1-\delta$. We use the rectangles with shadow to denote the corresponding z_c, which satisfies $z_c > 1-\delta$.

For $\delta \in (0,1]$, let $f(\delta)$ be defined as follows: for $0 < \delta \leq 1 - z_1$, $f(\delta) = 0$, for $1 - z_i < \delta \leq 1 - z_{i+1}$, $f(\delta) = \sum_{j=1}^{i} ct'_j$, where we make the convention that $z_{i_{max}+1} = 0$ (here, i_{max} is the number of cores). The function $f(x)$ is represented by the solid line of Fig. 3(b).

In Fig. 3(b), rectangles are ordered in the non-decreasing order of $1 - z_c$ from bottom to top. Therefore, $f(x)$ is non-decreasing in the range of $x \in (0,1]$. In this way, Equation (20) is equivalent to:

$$\left(\frac{2}{\delta} - 1\right) + f(\delta) \leq \xi \left(1 + \int_0^1 f(x)dx\right). \tag{21}$$

Next we show that there exists a δ satisfying Equation (21). For the sake of contradiction, we assume that $\forall \delta \in (0,1]$,

$$\begin{aligned} &\tfrac{2}{\delta} - 1 + f(\delta) > \xi \left(1 + \textstyle\int_0^1 f(x)dx\right) \\ \Longleftrightarrow\ & f(\delta) > \xi + 1 - \tfrac{2}{\delta} + \xi \textstyle\int_0^1 f(x)dx \end{aligned} \tag{22}$$

Do the integral operation to each side of the inequality.

$$\begin{aligned} &\textstyle\int_{1-\frac{1}{\xi}}^1 f(\delta)d\delta > \int_{1-\frac{1}{\xi}}^1 \left(\xi + 1 - \frac{2}{\delta} + \xi \int_0^1 f(x)dx\right) d\delta \\ \Longleftrightarrow\ &\textstyle\int_{1-\frac{1}{\xi}}^1 f(\delta)d\delta > \frac{\xi+1}{\xi} - 2\int_{1-\frac{1}{\xi}}^1 \frac{1}{\delta}d\delta + \int_0^1 f(\delta)d\delta \\ \Longleftrightarrow\ & 2\textstyle\int_{1-\frac{1}{\xi}}^1 \frac{1}{\delta}d\delta > \frac{\xi+1}{\xi} + \int_0^{1-\frac{1}{\xi}} f(\delta)d\delta \end{aligned} \tag{23}$$

Since

$$2\int_{1-\frac{1}{\xi}}^1 \frac{1}{\delta}d\delta = 2(0 - \ln(1 - \frac{1}{\xi})),$$

it can be obtained from Equation (23) that

$$-2\ln(1 - \frac{1}{\xi}) > \frac{\xi+1}{\xi} + \int_0^{1-\frac{1}{\xi}} f(x)dx,$$

which, together with $\int_0^{1-\frac{1}{\xi}} f(x)dx \geq 0$, contradicts the definition of ξ in Equation (18). $\square$

The above analysis can be similarly conducted for the single-task-per-core model by setting $cost(i)$ or $ct'_i = q_i$.

5.4 Integrality gap

In this subsection, we prove that ξ is a tight lower bound for the proposed LP-rounding solution compared to the optimal LP solution. In the above section, analysis is developed based on the multiple-task-per-core model. In the following, in order to address different features of the other model and to better understand both models, we mainly analyze the tightness of lower bound for the single-task-per-core model. Similar analysis can be extended to the multiple-tasks-per core model.

Precisely, we present and prove Theorem 1 below. Note that this theorem implies that, for any $\epsilon > 0$, no algorithm that uses the linear programming as a lower bound (in particular, no rounding procedure) can achieve an approximation ratio better than $\xi - \epsilon$ compared to the optimal solution of ILP (objective given by (6)).

Theorem 1 *For any $\epsilon > 0$, there exists an instance such that the ratio of the optimal ILP solution to the optimal linear programming solution is at least $\xi - \epsilon$.*

Proof A set of tasks is constructed as described in the following, with a (positive real) parameter a to be chosen later. Assuming that there are $2n+1$ tasks, each task $\tau_i(r_i, d_i, p_i, q_i)$ is presented below. When $i = 1$, $\tau_1(r_1, d_1, 1, 1)$, with $d_1 - r_1 = 1$; when $i \in [2, n+1]$, $\tau_i(r_1, d_1 + (i-1)\frac{a}{n}, 1 + (i-1)\frac{a}{n}, \frac{a}{n})$; when $i \in [n+2, 2n+1]$, $\tau_i(r_1 - (i-n-1)\frac{a}{n}, d_1, 1 + (i-n-1)\frac{a}{n}, \frac{a}{n})$. Note that for all tasks, we have $d_i - r_i = p_i$. Each task is represented as a rectangle in Fig. 4. The length of the rectangle represents the processing time in shared memory (i.e. p_i) of a task. Each task τ_i has $\frac{a}{n}$ longer p_i than task τ_{i-1}.

Let us look at the optimal solution, denoted as *opt* in the following. Task τ_1 has the same processing time whether allocated on the local or on the shared memory ($p_1 = q_1 = 1$), thus we do not lose anything if we assign τ_1 to the shared memory. For task τ_2, it will cost $\frac{a}{n}$ whether allocated on the local or the shared memory, thus we do not lose anything if we assign it to the shared memory. For task τ_3, similarly, it will cost $\frac{a}{n}$ whether allocated on the local or the shared memory, thus we again assign it to the shared memory. This re-assignment (if needed) can be done for tasks $4, 5, \ldots, 2n+1$ and we obtain an optimal solution with all tasks allocated on the shared memory. For this allocation, the objective (total active time) is

$$opt(n, a) = 1 + 2\sum_{i=1}^{n} \frac{a}{n} = 1 + 2a$$

Assume there is an LP solution, which satisfies the following construction. Assign $x_t = 1$ for t describing the interval $[r_1, d_1]$, and $x_t = 0$ for all the other intervals. Assign $z_1 = 0$ and $z_{i+1} = z_{n+i+1} = \frac{i(\frac{a}{n})}{1+i(\frac{a}{n})}$ for $i = 1, \ldots, n$. This solution implies that task τ_1 is assigned to the shared memory, while all the other tasks are partially scheduled within interval $[r_1, d_1]$ in the shared memory, and the remaining workload of each task is processed in the private

memory. It can be noted that this solution is a feasible solution as it can be easily checked to satisfy Constraints (7) and (9).

Denote the objective of the constructed LP instance as $LP(n,a)$. It can be deduced that

$$LP(n,a) \le h(n,a),$$

where

$$\begin{aligned} h(n,a) &= 1 + 2\sum_{i=1}^{n} \tfrac{a}{n} \tfrac{i(a/n)}{1+i(a/n)} \\ &= 1 + 2a\sum_{i=1}^{n} \tfrac{ai}{n+ai} \times \tfrac{1}{n} \\ &= 1 + 2a\sum_{i=1}^{n} \left(\tfrac{1}{n} - \tfrac{1}{n+ai}\right) \\ &= 1 + 2a - 2a\sum_{i=1}^{n} \tfrac{1}{n+ai}. \end{aligned} \tag{24}$$

Let

$$l(n,a) = \sum_{i=1}^{n} \frac{1}{1+a\frac{i}{n}} \times \frac{1}{n}.$$

For fixed a, we have

$$\lim_{n\to+\infty} l(n,a) = \int_0^1 \frac{1}{1+ax}dx = \frac{\ln(1+a)}{a} \tag{25}$$

according to the definition of integral operation.
Based on Equation (24) and (25), for n which is large enough, we have

$$\frac{opt(n,a)}{LP(n,a)} \ge \frac{1+2a}{1+2a-2\ln(1+a)} - \epsilon. \tag{26}$$

Let

$$\Omega(a) = \frac{1+2a}{1+2a-2\ln(1+a)}. \tag{27}$$

Let a_0 satisfy

$$\frac{1+2a_0}{1+a_0} - 2\ln(1+a_0) = 0. \tag{28}$$

This a_0 exists since the function $g(y) = \frac{1+2y}{1+y} - 2\ln(1+y)$ has positive values if $y = 1$, $g(10)$ is negative, and $g(y)$ is decreasing as one can check that g's derivative is negative. As an aside, Equation (28) was chosen such that the derivative of $\Omega(a)$ is 0.
Based on Equations (27) and (28), we know that

$$\frac{2\ln(1+a_0)}{1+2a_0} = \frac{1}{1+a_0} = 1 - \frac{1}{\Omega(a_0)}$$

and correspondingly,

$$\begin{aligned} 1 + \tfrac{1}{\Omega(a_0)} &= 2 - \tfrac{2\ln(1+a_0)}{1+2a_0} &&= 2 - \tfrac{1}{1+a_0} \\ &= \tfrac{1+2a_0}{1+a_0} &&= 2\ln(1+a_0) \\ &= -2\ln(\tfrac{1}{1+a_0}) &&= -2\ln(1 - \tfrac{1}{\Omega(a_0)}) \end{aligned} \tag{29}$$

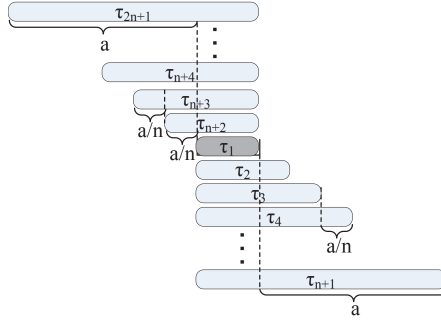


Fig. 4: The tasks constructed for lower bound analysis.

Comparing Equation (29) to Equation (18), it can be noted that

$$\Omega(a_0) = \xi,$$

and thus

$$\frac{opt}{LP^*} \geq \frac{opt(n,a)}{LP(n,a)} \geq \frac{opt(n,a_0)}{LP(n,a_0)} \geq \xi - \epsilon, \tag{30}$$

based on Equations (26) and (27), implying the statement of the theorem. □

For the multiple-task-per-core model, the above proof can be easily extended by constructing a similar task instance and *opt*/LP solutions, as briefly described below. Given $2n+1$ tasks, assume there are $2n+1$ cores, each of which is assigned a task (one can assume a core has multiple tasks as well, but it is not necessary to make that assumption). Let $cost(1) = 1$ and $cost(c) = \frac{a}{n}$ for all the other cores. The construction of *opt* solution is the same. For the LP solution, let $z_1 = 0$ and $z_{i+1} = z_{n+i+1} = \frac{i}{\frac{n}{a}+i}$ for $i = 1, \ldots, n$. And the following proof is just the same.

6 APX-hardness analysis for the multiple-task-per-core model

The section shows the complexity of MLSMSP. It works for both the preemptive and the non-preemptive model. Note that preemption only matters on the shared memory. For each core, its local memory is either on or off in this model. Whether tasks assigned to the local memory are scheduled preemptively or non-preemptively does not affect the complexity.

In the following, Theorem 2 is presented to show that the MLSMSP is APX-hard.

Theorem 2 *MLSMSP is APX-hard, that is: assuming P ≠ NP, there is an absolute constant $\epsilon_S > 0$ such that no polynomial-time algorithm has approximation ratio at most $1 + \epsilon_S$.*

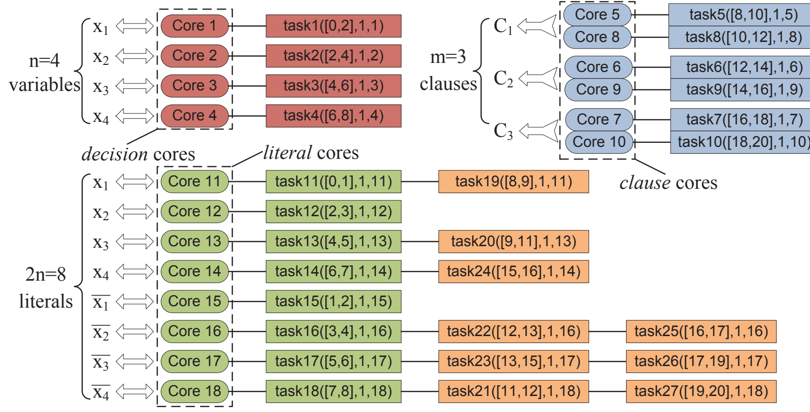


Fig. 5: The construction of a MLSMSP instance M_ϕ corresponding to $\phi = (x_1 + x_3 + \overline{x_4})(\overline{x_2} + \overline{x_3} + x_4)(\overline{x_2} + \overline{x_3} + \overline{x_4})$. Tasks are represented as ([release time, deadline], processing time, core id which it is assigned).

Our APX-hardness reduction is similar to that in [8] and is inspired by the reduction from Proposition 6.2 of [13], which uses the satisfiability problem 3-Sat.

The *maximum 3-satisfiability* problem Max-3Sat is that of finding, in a 3CNF Boolean formula (in which each clause has exactly three literals), a truth value assignment which satisfies the maximum number of clauses. For each fixed k, define Max-3Sat(k) to be the restriction of Max-3Sat to Boolean formulae in which each variable occurs at most k times. Theorem 3 below is immediate from Theorems 29.7, 29.11, and Corollary 29.8 in [22].

Theorem 3 [22] *Assuming $P \neq NP$, there is an absolute constant $\epsilon_M > 0$ such that no polynomial time approximation algorithm for* Max-3Sat(5) *which satisfies at least $(1 - \epsilon_M)m$ clauses for every formula ϕ with m clauses which is satisfiable.*

To prove the approximation hardness stated in Theorem 2, we develop two lemmas, Lemma 4 and Lemma 5 to construct the instances and the reduction from Max-3Sat(k) and to determine the scheduling objective time used when the input Boolean formula is satisfiable, respectively.

Lemma 4 *The following construction can be accomplished in polynomial time. The input to* 3-Sat *is a Boolean formula ϕ in 3CNF form. Let ϕ have n variables and m clauses. The constructed MLSMSP instance, defined as M_ϕ, will have $3n + 2m$ cores, all with unit cost to turn on the local memory, and $3n + 2m + 3m$ tasks, each with unit processing time (whether on shared or local memory). Let the cost of turning on each core's local memory be $cost(core(i)) = 1, \forall i \in [1, 3n + 2m]$.*

Proof All cores are divided into three types: *decision* cores, *clause* cores, and *literal* cores. Of these, only the literal cores are assigned with multiple tasks. The instance construction is given below.

- For *decision* cores: Each variable x_i has one core $core(i)$ with one task $\tau_i(2i-2, 2i, 1, i)$. It implies that this task is assigned to $core(i)$ with processing time 1. These n cores are referred to as *decision* cores, as one need to make the binary decision whether to use the interval $[2i-2, 2i-1]$ or $[2i-1, 2i]$ on the shared memory.
- For *clause* cores: Each clause C_j has two cores $core(n+j)$ and $core(n+m+j)$, each of which has one task: $\tau^{C_j}(2n+4j-4, 2n+4j-2, 1, n+j)$ and $\tau^{C'_j}(2n+4j-2, 2n+4j, 1, n+m+j)$, respectively. These $2m$ cores are referred as *clause* cores.
- For *literal* cores: These cores have multiple tasks assigned to them each. For literal x_i, there is one core $core(n+2m+i)$ with one task $\tau^{x_i}(2i-2, 2i-1, 1, n+2m+i)$, and more tasks to be discussed later. For literal $\overline{x_i}$, there is one core $core(2n+2m+i)$ with one task $\tau^{\overline{x_i}}(2i-1, 2i, 1, 2n+2m+i)$, and more tasks to be discussed next. These $2n$ cores are referred as the *literal* cores.

  For each C_j create three more tasks, one for each literal in this clause. Task $\tau_1^{C_j} = (2n+4j-4, 2n+4j-3, 1, q)$ has q that corresponds to the first literal of C_j (i.e if $\overline{x_3}$ is this literal, then $q = 2n+2m+3$). Task $\tau_2^{C_j} = (2n+4j-3, 2n+4j-1, 1, q)$ has q that corresponds to the second literal of C_j. Task $\tau_3^{C_j} = (2n+4j-1, 2n+4j, 1, q)$ has q that corresponds to the third literal of C_j.

An example of the instance construction is illustrated in Fig. 5 for $\phi = (x_1 + x_3 + \overline{x_4})(\overline{x_2} + \overline{x_3} + x_4)(\overline{x_2} + \overline{x_3} + \overline{x_4})$, with the number of variables $n = 4$ and the number of clauses $m = 3$. The three clauses are denoted as C_1, C_2, C_3. In Fig. 5, tasks colored as red, blue and green are those assigned to *decision*, *clause*, and *literal* cores, respectively. Tasks colored as orange are the tasks that are created for each literal in each clause on the *literal* cores. Task schedule corresponding to the instance constructed in Fig. 5 with $\phi = (x_1 + x_3 + \overline{x_4})(\overline{x_2} + \overline{x_3} + x_4)(\overline{x_2} + \overline{x_3} + \overline{x_4})$ is shown in Fig. 6. The coloring rules in Fig. 5 also work here.

On one hand, for the *decision* and *clause* cores, every optimal solution to MLSMSP can be assumed to turn off every local memory of them as their only task can be assigned to the shared memory without increasing the objective. This is because for each task on *decision* and *clause* cores, the processing time on the shared memory is unit and disjoint with each other, and the cost of turning on the local memory is unit.

For *clause* cores as an example, the canonical choices are to use either $[2n+4j-4, 2n+4j-3]$ or $[2n+4j-3, 2n+4j-2]$, and either $[2n+4j-2, 2n+4j-1]$ or $[2n+4j-1, 2n+4j]$ as the shared memory time intervals by the tasks assigned to these *clause* cores. These choices are shown in Fig. 6 as the scheduling of blue tasks. For example, for clause C_2, task6 and task9

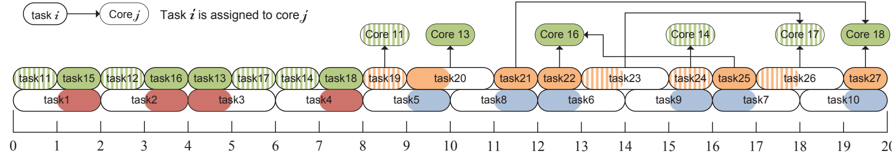


Fig. 6: Given the MLSMSP instance M_ϕ corresponding to $\phi = (x_1 + x_3 + \overline{x_4})(\overline{x_2}+\overline{x_3}+x_4)(\overline{x_2}+\overline{x_3}+\overline{x_4})$, the solution (i.e., the local memory on which core to turn on, and how to schedule tasks in the shared memory) corresponding to the truth assignment $x_1 = 1, x_2 = 1, x_3 = 0, x_4 = 1$ is shown. Turning on the local memory on core is represented by setting the literal as true ($x_1 = 1$ means core 11 is turned on and core 15 is off; $\overline{x_3} = 1$ implies core 17 is turned on and core 13 is off). The size of the rounded rectangle represents the feasible region of scheduling task. Task (or core) with shadow represents that the local memory on the corresponding core is turned on, otherwise it is turned off. For tasks with feasible region of two unit time slots, such as task1, task5, task20 etc., they are scheduled in either unit time slot (the colored slot) if assigned to shared memory. For example. task1 is scheduled in [1,2] in the shared memory.

should be scheduled in either $[12, 13]$ or $[13, 14]$, and either $[14, 15]$ or $[15, 16]$. The canonical choices of the *decision* cores are similar.

On the other hand, the choice of turning on the *literal* core corresponds to setting the literal true. As shown in Fig. 6, corresponding to the solution $x_1 = 1, x_2 = 1, x_3 = 0, x_4 = 1$, core 11. core 12, core 14, and core 17 are turned on, while task13, task15, task16 and task18 are scheduled in the shared memory within their corresponding feasible region. In the meantime, task1, task2, task3 and task4 can be scheduled in the same time slots without bringing extra active time. Orange tasks (task19-27) are scheduled similarly without bringing extra active time.

Clearly, constructing M_ϕ can be accomplished in polynomial time. We first determine the scheduling objective time used when the input Boolean formula is satisfiable. □

Lemma 5 *If ϕ is satisfiable then M_ϕ has an assignment of objective $2n+2m$.*

Proof Let γ be an assignment which satisfies ϕ. If a variable x_i is set true by γ, we turn on the literal core corresponding to x_i, i.e. core $n + 2m + i$, on which there is task $\tau^{x_i}(2i-2, 2i-1, 1, n+2m+i)$. In the meantime, use the interval $[2i-1, 2i]$ in the shared memory for task τ_i (from the decision core corresponding to variable x_i) to execute. Also, turn off the literal core corresponding to $\overline{x_i}$, i.e. core $2n + 2m + i$.

Otherwise (x_i is set false by γ), we turn on the literal core corresponding to $\overline{x_i}$, use the interval $[2i-2, 2i-1]$ in the shared memory for task τ_i, and turn off the literal core corresponding to x_i. In both cases, doing this for all the variables contributes $2n$ to the objective function. Referring to Fig.

6, scheduling task1-task4 and task11-task18 will contribute $2n = 8$ to the objective function.

For each clause C_j, if the first literal is true (according to γ), then we turn on the core where task $\tau_1^{C_j}(2n+4j-4, 2n+4j-3, 1, q)$ is assigned, and use the interval $[2n+4j-3, 2n+4j-2]$ for task τ^{C_j} in the shared memory. This is because τ^{C_j} can be scheduled in either $[2n+4j-4, 2n+4j-3]$ or $[2n+4j-3, 2n+4j-2]$. Since $\tau_1^{C_j}$ is allocated to the local memory, utilizing $[2n+4j-3, 2n+4j-2]$ in shared memory has larger potential to obtain overlap with other tasks. Meanwhile, let $\tau^{C_j'}$ execute in the interval $[2n+4j-1, 2n+4j]$ in the shared memory. Referring to Fig. 6, for C_1, x_1 is true, so we turn on Core 11, and let task5 (τ^{C_1}) execute in $[9, 10]$ while task8 ($\tau^{C_1'}$) executes in $[11, 12]$.

If the first literal is false, but the second literal is true, use for task τ^{C_j} the interval $[2n+4j-4, 2n+4j-3]$ in the shared memory, and for $\tau^{C_j'}$ the interval $[2n+4j-1, 2n+4j]$ in the shared memory. Referring to Fig. 6, for C_2, $\overline{x_2}$ is false but $\overline{x_3}$ is true, so we turn on Core 17, and let task6 execute in $[12, 13]$ while task9 executes in $[15, 16]$. Similarly, if the first two literals are false and the third literal is true, then it is used for task τ^{C_j} the interval $[2n+4j-4, 2n+4j-3]$ in the shared memory, and for $\tau^{C_j'}$ the interval $[2n+4j-2, 2n+4j-1]$ in the shared memory.

In all cases, we use an active time of 2 on the shared memory for each clause, and thus the objective of the assignment is indeed $2n+2m$. This is a feasible assignment, since whenever a local memory is turned off for a core corresponding to a literal, all the tasks assigned to this core can be scheduled on the main memory, as can be easily checked. $\square$

Note that assignments better than $2n+2m$ never exist. Firstly, the disjoint intervals of the shared memory must be used for the $2m$ clause cores and the n decision cores. Secondly, out of two literal cores corresponding to a variable, one has a task that does not fit yet on the shared memory (because there are three tasks with three unit processing time which have disjoint scheduling regions in interval $[2n+4j-4, 2n+4j]$ on a literal core), and thus an additional term of 1 must be added to the objective function. In fact, we can assume without loss of generality that the optimal solution turns on the literal core who has one such task that does not fit on the shared memory.

Based on the above lemmas, the proof to Theorem 2 can conducted and proved.

Proof (*of* **Theorem 2**) Assume that there exists a polynomial-time approximation algorithm for MLSMSP with performance ratio at most $1+\epsilon$ for some $\epsilon > 0$. The assumed algorithm gives a solution S having objective at most $(1+\epsilon)(2n+2m)$. We first transform S to S' without any increase in cost, where S' is a solution that fulfills the following two conditions: (i) S' uses exactly $2m+n$ active time on the shared memory, and (ii) In S', there is no preemption.

To achieve (i), consider the time interval $[2n+4j-4, 2n+4j]$ on the shared memory, where the tasks of the cores corresponding to clause C_j are scheduled

(we argued earlier that these two tasks are scheduled on the shared memory). There are another three tasks, which belong to the cores corresponding to the literals of C_j which must be scheduled during this interval; moreover these three tasks are time-disjoint. It can be proved below that no optimal solution uses active time more than 3 in this interval. If it uses exactly 3, change the solution to use exactly 2 and turn on the local memory of one of the three cores corresponding to the literals of C_j. One can easily verify this is feasible. And if the optimal solution uses less than 3 active time on the shared memory in this interval, then one of the three cores corresponding to the literals of C_j must have its local memory on. Thus we might as well assume exactly 2 active time is used on the shared memory during this interval.

A similar argument is used for the time interval $[2i-2, 2i]$ where the task of the core corresponding to the variable x_i must be scheduled for a time of 1. It can be assumed that an optimal solution uses exactly 1 active time on the shared memory for this interval. Thus there are exactly $n+2m$ active time on the shared memory in S' solution.

We further transform the solution so as to satisfy (*ii*). Note that all feasible regions of multiple tasks on the same core (specifically, on the literal cores) are distinct. Thus preemption does not need to be considered.

We call S' the resulting solution. Note that at most $\epsilon(2n+2m)$ variables have both cores corresponding to its literals with the local memory on. We now construct a truth value assignment τ: for each variable, if the core corresponding to its positive literal has the local memory on, set it to true; if the core corresponding to its negative literal has the local memory on, set it to false, and if both cores corresponding to its literals have the local memory on, set it arbitrarily (say, to true). The at-most $\epsilon(2n+2m)$ variables that have both literal cores with the local memory on appear in at most $5\epsilon(2n+2m)$ clauses (cf. the definition of Max-3Sat(5)). Let C be any of the remaining clauses. We claim that τ makes C true. One of the three tasks corresponding to three literals in C must force the core to which it is assigned to be turned on. Otherwise three distinct processing time slots need to be scheduled in two active shared memory time slots (because the active time of shared memory is exactly $n+2m$). As shown in Fig. 6, task19, task20, task21 need to be scheduled in two time units in shared memory that are fixed by task5 and task8. By construction, the literal is true corresponding to that the task forces the core to which it is assigned to be turned on, hence C is true.

Therefore, the number of satisfied clauses is at least $m-5\epsilon(2n+2m)$. Since we can assume that $m \geq n/3+1$, we have $m-5\epsilon(2n+2m) \geq (1-40\epsilon)m$. Setting $\epsilon_M = 40\epsilon$, the result follows from Theorem 3. That is, we can take $\epsilon_S = \epsilon_M/40$, and the proof of Theorem 2 is complete. □

7 Evaluation

This section provides the energy efficiency evaluation of the proposed LP-rounding procedure, denoted as LPR in the following. The procedure LPR is

compared with the optimal solution given by ILP, denoted as OPT, and the relaxed solution given by LP. Even though a tight lower bound ($\xi \approx 1.865$) is presented for LPR compared to LP, it remains interesting to find the average performance instead of the worst case bound. In this section, simulations are conducted for both single-task-per-core and multiple-task-per-core models to evaluate the average performance of LPR in terms of energy saving.

7.1 Simulation setup

7.1.1 Task set synthesis

To evaluate the energy efficiency of LPR compared to OPT and LP, a large set of tasks is randomly generated. The synthetic tasks are conducted and evaluated at different system utilization levels. Specifically, we generate tasks in the following manners:

- Denote the maximum number of time slots as s, which means all tasks will be generated between interval $[0, s]$.
- For each task T_j, we assume that it has 60% probability to be randomly released in $[0, \frac{s}{2}]$ and 40% probability in $(\frac{s}{2}, s]$. The probability setting is based on the consideration that the release time is more likely to be in the first half of the time interval, in order to guarantee that tasks are uniformly distributed.
- Deadline d_j is randomly generated between $(r_j, s]$.
- The processing time p_j in shared memory is set to be related with a demand ratio, denoted as $\rho \in (0.1, 0.8)$. p_j is then randomly generated with the restriction that $p_j < \rho(d_j - r_j)$.
- For single-task-per-core model, the processing time q_j in local memory is randomly generated between $[0.3p_j, 0.8p_j]$. This is because tasks may have different memory access features, and thus the ratios of tasks' access latencies in local/shared memory are not uniform.
- For multiple-task-per-core model, set the number of cores as C. For each core, randomly generate several tasks in $[0, s]$, with the restriction that tasks are schedulable on each single core.

7.1.2 System configuration

The configurations of memory are modeled based on the 50nm DRAM, and the parameters are collected using CACTI [23], where the standby static power scales with the memory size. In the following experiments, we assume the size of the shared memory as 1GB, with the standby power of 227.15mW based on CACTI, while the size of the local memory as 12MB with the static power of 2.71mW. The clock rate of the local memory is 800MHz (we do not care the clock rate of shared memory). The energy transition overhead of the local memory from power down mode to active mode is set to be $9.12 \times 10^{-7} J$ [6].

Recall that for multiple-task-per-core model, we assume that the static power of local memory is smaller than the switch cost. Based on the above configuration, it implies that $2.71mW \times S \leq 9.12 \times 10^{-7}J$, where $S \leq 3.37 \times 10^{-4}s$ represents the maximum interval between the first task's release time to the last task's deadline. In this way, according to the clock rate of 800MHz, the maximum number of time slots $s \leq \frac{3.37\times10^{-4}s}{\frac{1}{800MHz}} = 2.83 \times 10^5$. Therefore, for multiple-task-per-core model, we set $s < 2.83\times10^5$, while $s \in [2.83\times10^5, 5.66\times 10^5]$ for single-task-per-core model, where assuming the static power of local memory is larger than the switch cost. The number of slots s can be larger, but here in the simulation, we consider the limited number of slots as an example.

7.2 Simulation result

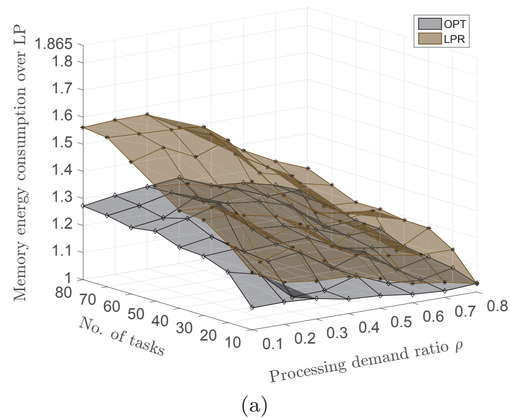
In the experiment, the LP-rounding procedure is implemented in MATLAB. The LP and ILP formulations are solved by the linear programming solver, Gurobi [16].

As shown in Table 1, in the experiment, we evaluate the OPT, LP and LPR algorithms over different parameters. For single-task-per-core model, algorithms are compared over the processing demand ρ and the number of tasks n. With the number of slots $s \in [2.83 \times 10^5, 5.66 \times 10^5]$ (i.e. 0.354-0.707ms) , the amount of the total in coming tasks are most likely smaller than 80. For multiple-task-per-core model, algorithms are compared among the processing demand ρ, and the number of cores C. Considering in most IoT systems, the number of cores is limited by chip size and low workload requirement, the simulation is conducted on at most 10 cores. To generate convincing results, for each data point, we randomly generate ten different cases, and use the average value as the final performance of each data point.

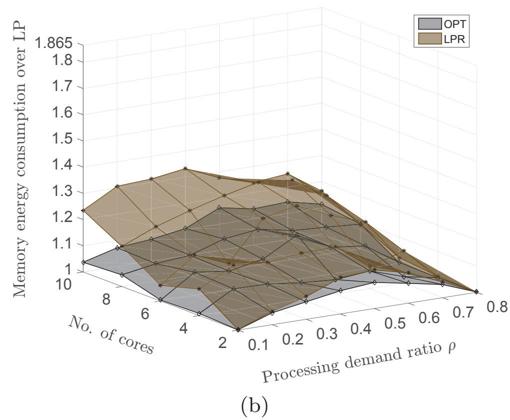
The experimental results are shown in Fig. 7(a) and 7(b), where the total memory energy consumption of the optimal solution and LP-rounding procedure are normalized to the LP solution. For single-task-per-core model, when the number of tasks is small and the processing demand is high, both OPT and LPR perform very close to the LP solution. This is because in this case, tasks have long processing time ($\rho = \frac{p_i}{d_i - r_i}$) and have large potential to occupy a long active time in shared memory. On the other hand, since the number of tasks is small, tasks will cost less if executing in local memory. In this way, all schemes will try to put each task on the local memory, which leads to little

Table 1: Parameters used for evaluating all the algorithms. "Single" is short for single-task-per-core model, while "Multiple" is short for multiple-task-per-core model.

Single	$\rho =$	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8
	$n =$	10	20	30	40	50	60	70	80
Multiple	$\rho =$	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8
	$C =$	2	4	6	8	10			



(a)



(b)

Fig. 7: The total memory energy consumption of OPT and LPR with normalized to the LP solution (a) for the single-task-per-core model and (b) for the multiple-task-per-core model.

flexibility and differences among schemes. The gap between OPT/LPR and LP solution becomes larger with the increase of tasks' amount and the decrease of the processing demand because of the increasing flexibilities or options in tasks assignment. In average, the LPR procedure is 1.299× the LP solution, while the OPT is 1.175× the LP solution. LPR consumes 10.55% more energy compared to OPT.

For multiple-task-per-core model, as shown in Fig. 7(b), similar to that in Fig. 7(a), both OPT and LPR perform very close to the LP solution when the number of cores is small and the processing demand is high. However, different from that in Fig. 7(a), it can be noted that OPT is always close to LP solution when the processing demand is low. This is because the number of total scheduling slots is small and tasks have short processing time. Hence

executing in the shared memory brings less energy consumption, comparing to the dominating energy overhead in local memory — "turning on cost" in this model. On the other hand, the LPR procedure does not perform as good as the OPT. LPR has larger gap with the LP solution with the increase of the core number. This is because even though this procedure also tends to schedule tasks in the shared memory, the active time x'_t is increased due to the rounding procedure, especially when the number of cores is increased, which implies the tasks' amount is increased as well. In average, for this model, the LPR procedure is 1.171× the LP solution, while the OPT is 1.076× the LP solution. LPR consumes 8.83% more energy compared to OPT.

8 Conclusions and future work

This paper studied task scheduling problem in the local-shared memory architecture of IoT system. Two task models are proposed based on different scenarios of energy domination in the local memory. We proposed an optimal DP-based algorithm for the single-task-per-core model with preemption forbidden in shared memory. For the preemptive case, an LP-rounding procedure with constant and tight bound of approximation ratio compared to LP solution is proposed. This LP-rounding procedure works for both task models. The integrality gap example is general enough; it will hold from the more general formulation even if $\alpha_s = \alpha_l = 1$ (same power coefficients for the shared and local memory), while the $cost(i)$ is zero.

Slightly more complicated models, i.e. if we require the time-slots to be either fully used, or not used, can be handled by the same algorithm as explained after Constraint (9).

For the multiple-task-per-core model, APX-hardness is proved and therefore there exists an $\epsilon > 0$ such that no polynomial time algorithm has approximation ratio of $1+\epsilon_S$. This hardness also holds in the more general formulation even if the power coefficient for the shared memory is 1 and is equal to the cost of turning in each local memory.

We leave open the question if the problem for the single-task-per-core model with preemption, is NP-hard or not. We also leave open the design of approximation algorithms for the non-preemptive multiple-task-per-core model.

References

1. L. Atzori, A. Iera, and G. Morabito. The internet of things: A survey. *Computer networks*, 54(15):2787–2805, 2010.
2. L. Benini, A. Bogliolo, G. A. Paleologo, and G. De Micheli. Policy optimization for dynamic power management. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, 18(6):813–833, 1999.

3. J. Chang, H. N. Gabow, and S. Khuller. A model for minimizing active processor time. In *Proceedings of the 20th Annual European Symposium on Algorithms*, ESA, pages 289–300, 2012.
4. J. Draper, J. Chame, M. Hall, C. Steele, T. Barrett, J. LaCoss, J. Granacki, J. Shin, C. Chen, C. W. Kang, et al. The architecture of the diva processing-in-memory chip. In *Proceedings of the 16th international conference on Supercomputing (ICS)*, pages 14–25. ACM, 2002.
5. D. G. Elliott, M. Stumm, W. M. Snelgrove, C. Cojocaru, and R. McKenzie. Computational ram: implementing processors in memory. *Design & Test of Computers, IEEE*, 16(1):32–41, 1999.
6. X. Fan, C. Ellis, and A. Lebeck. Memory controller policies for dram power management. In *Proceedings of the 2001 international symposium on Low power electronics and design (ISLPED)*, pages 129–134. ACM, 2001.
7. M. Flammini, G. Monaco, L. Moscardelli, H. Shachnai, M. Shalom, T. Tamir, and S. Zaks. Minimizing total busy time in parallel scheduling with application to optical networks. In *IEEE International Parallel Distributed Processing Symposium, IPDPS*, pages 1–12, 2009.
8. R. Freimer, C. Piatko, and J. S. Mitchell. On the complexity of shattering using arrangements. Technical report, Cornell University, 1991.
9. C. Fu, M. Li, and C. J. Xue. Race to idle or not: balancing the memory sleep time with dvs for energy minimization. In *Proceedings of the 2015 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 13–18. EDA Consortium, 2015.
10. C. Fu, Y. Zhao, M. Li, and C. J. Xue. Maximizing common idle time on multi-core processors with shared memory. In *Proceedings of the 2015 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 900–903. EDA Consortium, 2015.
11. B. R. Gaeke, P. Husbands, X. S. Li, L. Oliker, K. A. Yelick, and R. Biswas. Memory-intensive benchmarks: Iram vs. cache-based machines. In *Parallel and Distributed Processing Symposium., Proceedings International, IPDPS 2002, Abstracts and CD-ROM*, pages 7–pp. IEEE, 2001.
12. J. Gebis, S. Williams, D. Patterson, and C. Kozyrakis. Viram1: A media-oriented vector processor with embedded dram. *DAC04*, pages 7–11, 2004.
13. R. Hassin and N. Megiddo. Approximation algorithms for hitting objects with straight lines. *Discrete Applied Mathematics*, 30(1):29–42, 1991.
14. T. Kgil, S. D'Souza, A. Saidi, N. Binkert, R. Dreslinski, T. Mudge, S. Reinhardt, and K. Flautner. Picoserver: using 3d stacking technology to enable a compact energy efficient chip multiprocessor. *ACM SIGARCH Computer Architecture News*, 34(5):117–128, 2006.
15. R. Khandekar, B. Schieber, H. Shachnai, and T. Tamir. Minimizing busy time in multiple machine real-time scheduling. In *LIPIcs-Leibniz International Proceedings in Informatics*, volume 8. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2010.
16. G. Optimization et al. Gurobi optimizer reference manual version 5.6, 2014.

17. D. Park, S. Eachempati, R. Das, A. K. Mishra, Y. Xie, N. Vijaykrishnan, and C. R. Das. Mira: A multi-layered on-chip interconnect router architecture. In *ACM SIGARCH Computer Architecture News*, volume 36, pages 251–261. IEEE Computer Society, 2008.
18. A. Sinha and A. Chandrakasan. Dynamic power management in wireless sensor networks. *Design & Test of Computers, IEEE*, 18(2):62–74, 2001.
19. M. Uddin and T. Nadeem. A2psm: Audio assisted wi-fi power saving mechanism for smart devices. In *Proceedings of the 14th Workshop on Mobile Computing Systems and Applications*, HotMobile '13, pages 4:1–4:6. ACM, 2013.
20. B. Univ. of California. The berkeley intelligent ram (iram) project.
21. S. R. Vangal, J. Howard, G. Ruhl, S. Dighe, H. Wilson, J. Tschanz, D. Finan, A. Singh, T. Jacob, S. Jain, et al. An 80-tile sub-100-w teraflops processor in 65-nm cmos. *IEEE Journal of Solid-State Circuits*, 43(1):29–41, 2008.
22. V. V. Vazirani. *Approximation Algorithms*. Springer-Verlag, 2001.
23. S. J. E. Wilton and N. Jouppi. Cacti: an enhanced cache access and cycle time model. *IEEE Journal of Solid-State Circuits*, 31(5):677–688, May 1996.
24. D. H. Woo, N. H. Seong, D. L. Lewis, and H. H. S. Lee. An optimized 3d-stacked memory architecture by exploiting excessive, high-density tsv bandwidth. In *2010 The 16th International Symposium on High Performance Computer Architecture (HPCA)*, pages 1–12. IEEE, 2010.
25. W. A. Wulf and S. A. McKee. Hitting the memory wall: implications of the obvious. *ACM SIGARCH computer architecture news*, 23(1):20–24, 1995.
26. Y. Xie. Processor architecture design using 3d integration technology. In *VLSID'10. In 2010 23rd International Conference on VLSI Design VLSI Design*, pages 446–451. IEEE, 2010.
27. H. Yun, G. Yao, R. Pellizzoni, M. Caccamo, and L. Sha. Memguard: Memory bandwidth reservation system for efficient performance isolation in multi-core platforms. In *Real-Time and Embedded Technology and Applications Symposium (RTAS), 2013 IEEE 19th*, pages 55–64. IEEE, 2013.
28. X. Zhong and C.-Z. Xu. Frequency-aware energy optimization for real-time periodic and aperiodic tasks. LCTES, pages 21–30, 2007.
29. X. Zhong and C.-Z. Xu. System-wide energy minimization for real-time tasks: Lower bound and approximation. *ACM Trans. Embed. Comput. Syst.*, 7(3):28:1–28:24, May 2008.