Multi–task Scheduling for PIM-based Heterogeneous Computing System

Dawen Xu
xudawen@hfut.edu.cn
Hefei University of Technology &
Institute of Computing Technology,
Chinese Academy of Sciences
Hefei, China

Cheng Chu
chucheng@mail.hfut.edu.cn
Hefei University of Technology &
Institute of Computing Technology,
Chinese Academy of Sciences
Hefei, China

Cheng Liu*
liucheng@ict.ac.cn
Institute of Computing Technology,
Chinese Academy of Sciences
Beijing, China

Ying Wang
wangying2009@ict.ac.cn
Institute of Computing Technology,
Chinese Academy of Sciences
Beijing, China

Xianzhong Zhou
zhouxzh@gdut.edu.cn
Guangdong University of Technology
Guangzhou, China

Lei Zhang
zlei@ict.ac.cn
Institute of Computing Technology,
Chinese Academy of Sciences
Beijing, China

Huaguo Liang
huagulg@hfut.edu.cn
Hefei University of Technology
Hefei, China

Huawei Li
lihuawei@ict.ac.cn
Institute of Computing Technology,
Chinese Academy of Sciences
Beijing, China

ABSTRACT

Processing-in-Memory (PIM) or Near-Data Processing has been recognized as the most potential solution to resolve the ever-aggravating memory wall especially as the thrive of memory-intensive scale-out workloads such as graph computing and data analytics. However, when the future computing system becomes more and more likely to adopt PIM architectures as a type of the storage and processing component, there is a lack of literature and research work on the general scheduling framework with the emerging heterogeneous system except for some ad-hoc task partitioning methods with specialized PIM designs. This work is the first to propose a formalized model to quantitatively describe the multi-task scheduling problem in PIM+CPU platform without loss of generality, and also an optimized task mapping-and-scheduling algorithm to boost the hardware utility for these novel heterogeneous systems. The proposed scheduling framework is fully aware of the data access bandwidth and processing capability distinction between the CPU and PIM devices, and also the implications of task mapping on the bandwidth contention, data communication intensity and hardware utility for the concurrent workloads. Experimental results show that, compared to the traditional scheduling algorithm for heterogeneous system, the proposed method is able to improve the system performance by over 10% and the energy efficiency by almost 10% for multi-core scale-out applications.

GLSVLSI '20, September 7–9, 2020, Virtual Event, China

ACM ISBN 978-1-4503-7944-1/20/09…$15.00
https://doi.org/10.1145/3386263.3406946

CCS CONCEPTS

• **Computer systems organization** → **Other architectures**; • **Hardware** → *Analysis and design of emerging devices and systems*; • **Software and its engineering** → Process management.

KEYWORDS

Process in memory, Heterogeneous (hybrid) systems, Task scheduling

ACM Reference Format:
Dawen Xu, Cheng Chu, Cheng Liu, Ying Wang, Xianzhong Zhou, Lei Zhang, Huaguo Liang, and Huawei Li. 2020. Multi–task Scheduling for PIM-based Heterogeneous Computing System. In *Proceedings of the Great Lakes Symposium on VLSI 2020 (GLSVLSI '20), September 7–9, 2020, Virtual Event, China.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3386263.3406946

1 INTRODUCTION

The popularity of emerging big-data workloads such as in-memory database and data analysis imposes a big challenge to the state-of-art memory system. The increasingly critical memory bottleneck is further stressed by the growing computation power enhanced by the multi-core/many-core and heterogeneous architectures including GPU and DSPs. A promising solution is to conduct computation at the place where data resides to detour the memory wall and eliminate the performance and power penalty caused by the intensive data transfer between processors and memory devices. In this context, PIM is returning to alleviate the memory bottleneck. In general, PIM is prompted due to the demand of memory-intensive workloads and also the development of enabling technologies like 3D stacking ICs and emerging memory. Because of the advantages

of PIM, there is a myriad of literature PIM research in various backgrounds, which are mostly focused on accelerating a specific application or a domain of workloads or elevating their system-level energy efficiency[1], such as graph processing and binary CNN, etc. Prior work also reveals one important design aspect of PIM architecture that the task partitioning and mapping in PIM-enabled system actually plays an essential role in the system efficiency. To our observation, it is found that a unreasonable task mapping between conventional CPU and PIM engines will make a big difference and even lead to a system performance drop compared to the system without PIM, which is further illustrated in latter analysis.

Compared to prior related work on PIM and PIM task scheduling, our work has three distinct features: 1. This is the first work that formalizes the multi-task scheduling problem in the emerging CPU+PIM based heterogeneous multi-core architecture. 2. Instead of studying the task mapping for a specific PIM design as prior work, our design is expected to work for various PIM-integrated systems without loss of generality. Ahn[2] proposed a PIM architecture with self-defined instruction to identify the data locality of the codes and decide whether the codes should be executed on PEI engine. [7][12][11] developed new mechanisms to enable computation offloading to 3D-stacked memory. However, they relies on the empirical clues, such as data locality of the application, to decide the execution on CPU or PIM but does not solve the problem from the perspective of formal analysis. 3. Instead of dealing with specific applications as prior work, we consider the complexity of general PIM architectures and the influence of concurrent multi-programmed workloads in realistic multi-core processor, where independent programs in the parallel workloads will interact and influence each other by sharing the same PIM engines as well as the memory bandwidth, which may have profound impacts on the task scheduling results in general CPU+PIM systems.

There are plenty of literatures on heterogeneous multi-core scheduler worth referring to, in consideration that the PIM+CPU systems can be viewed as a novel heterogeneous system. HEFT and CPOP[13] are well-known list scheduling heuristics for heterogeneous computing environments. [3] combines a dual approximation scheme with a fast integer linear program(ILP). MRCRG [15] solves the problem of resource consumption cost minimization for a reliable parallel application. Push-Pull [8] starts to search with the best task schedule found by a fast deterministic task scheduling algorithm. DUECM[16] miminizes energy consumption for real-time parallel applications. [6] optimizes the robustness. [10] adopts statistical analysis measures and machine learning to predict the makespan. However, these scheduling algorithms cannot be directly applied to PIM+CPU system for the following two reasons: firstly, as PIM operates directly on the memory-coherent DRAM, tasks executed on PIM have nonnegligible memory interference on those executed on the multi-core CPU. This is quite different from the conventional hybrid computing architectures. Secondly, communication cost between dependent tasks varies in a wide range depending on the location of the data to be transferred while most of the existing scheduling algorithms adopt a linear model to measure the communication cost.

Therefore, it requires a formalized and accurate model to profile the execution model on heterogeneous PIM+CPU system and also an accompanied mapping-and-scheduling strategy for such a system. This paper makes the following major contributions:

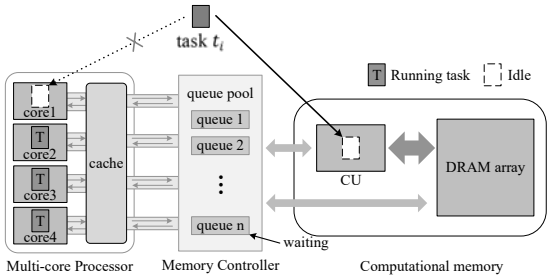

Figure 1: PIM+mutli-core CPU system

- We generalize and formalize the PIM-based multi-task execution model for task scheduling and mapping, which fully considers the impacts of inter-task memory interference and the communication overhead in either PIM or CPU execution.
- The proposed scheduling framework is fully aware of the data access bandwidth and processing capability distinction between the CPU and PIM devices, and also the implications of task mapping on the bandwidth contention, data communication intensity and hardware utility for the concurrent workloads.
- A CPU+HMC PIM simulation environment is built to evaluate the model and scheduler in the context of multiple independent and concurrent workloads. It is shown in evaluation that with randomly generated concurrent big-data workloads, the proposed scheduler can improve the system throughput by 18.0% on average and increase the energy efficiency by 12.2%.

2 TARGET SYSTEM MODEL

2.1 System Description

A typical hybrid PIM+CPU system is shown in Fig1. The illustrative general PIM device has a logic die (computing unit) with DRAM dies stacked on top and it can access the DRAM arrays through a high bandwidth connection of through-silicon-vias (TSVs). The cache, as the intermediate buffer between CPU and memory, stores the pre-fetched data of CPU and the computation results generated by CPU. The heterogeneous PIM and CPU system have PIM logics and CPU share the same DRAM space. However, PIM and CPU have completely distinct ways to access the memory and the independent tasks executed on them will barely conflict on memory access if they access the different areas in memory, which makes the most important difference in the multi-task scheduler for PIM-based systems in contrast to conventional heterogeneous architectures.

Task scheduling on PIM + multi-core CPU system not only maps PIM-friendly task to PIM but also balances workload considering with the task communication and cross-task memory interference to achieve a good performance for the entire system. As shown in Fig1, t_i, which represents the ith task, is going to be run on the PIM-enabled multi-core CPU system, core-1 of the CPU and PIM is available. The additional assumption is that t_i is a non PIM-friendly

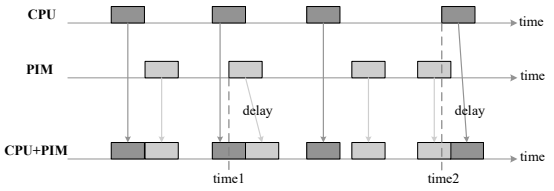

Figure 2: FIFO model for memory conflict between CPU and PIM

task, which spends more runtime on PIM than CPU. According to PEI[2] strategy, t_i will be assigned to CPU for the shortest runtime of t_i itself. However, it will lead to much more memory interference since there are already three tasks running on the other three cores of CPU. So, if the runtime of t_i running on PIM is not too much longer than on CPU, scheduling t_i running on PIM will reduce the memory interference of other three tasks and make the entire application complete earlier. This is the key difference of the proposed scheduling strategy from all earlier PIM off-loading mechanisms or heterogeneous scheduling algorithms.

2.2 Formalization of scheduling problem on PIM+CPU Heterogeneous Systems

2.2.1 Memory Interference Model. In PIM+CPU architecture, memory inference generally includes two scenarios i.e. parallel tasks are executed on different CPU cores or they are executed on both CPU and PIM. Accordingly, the memory inference latency includes two aspects. 1) memory request queueing latency when multiple memory requests are issued to the memory controller from different cores of the CPU, 2) memory conflict latency when memory requests from CPU and PIM are contending for the same memory bank. Queueing latency for parallel memory requests in multi-core CPU is essentially a queueing problem. We use an open and closed queuing model given by [4]. Due to space limitations, no detailed explanation will be given here.

For the memory conflict between PIM and CPU, we adopt a first-in-first-out (FIFO) model as shown in Fig.2. When the tasks on CPU and tasks on PIM are executed in parallel on CPU+PIM architecture, task issued earlier will be handled first while the latter one will be delayed accordingly. For instance, the task on CPU has been issued at time1, thus the task issued at time1 from PIM will be delayed. Similarly, task issued at time2 on CPU will be delayed as well. Detailed delay model will be detailed in the following paragraphs.

The memory conflict latency is calculated with formula 1 and depends on three factors including the number of memory requests for the corresponding processing unit in this task N_{arch}, the expected delay when a memory request from one architecture is suspended by a memory request from another architecture i.e. ED, and the probability of the memory conflict when the task on PIM and tasks on CPU access the overlapped memory location simultaneously i.e. $P_{conflict}$.

$$delay = N_{arch} \times ED \times P_{conflict} \tag{1}$$

Suppose D_{mem} stands for the average memory access delay. When memory conflict happens, the memory request waiting time may range from 0 to D_{mem} in uniform a distribution. Therefore, the expected memory waiting time (ED) can be expressed as $\frac{1}{2}D_{mem}$. $P_{conflict}$ is relative complex. Basically memory conflict happens when both tasks from CPU and PIM issues memory request to the overlapped memory space at the same time. It can be obtained through formula 2.

$$P_{conflict} = \frac{N \times D_{mem}}{T_{total}} \times P_{cpu} \times P_{pim} \tag{2}$$

P_{cpu} and P_{pim} stand for the probability that tasks on the CPU and PIM access overlapping memory segments, respectively. N is total number of memory instructions in the task while T_{total} represents the total execution time of the task. The number of memory requests can be estimated by the product of the total number of instructions and the cache miss rate executed, that is, $Inst_{num} \times P_{cache-miss}$. Then $P_{conflict}$ can be further transformed as follows:

$$\begin{aligned} P_{conflict} &= \frac{Inst_{num}}{T_{total}} \times P_{cache-miss} \times D_{mem} \times P_{cpu} \times P_{pim} \\ &= IPC \times Freq \times P_{cache-miss} \times D_{mem} \times P_{cpu} \times P_{pim} \end{aligned} \tag{3}$$

IPC refers to instructions per cycle, $Freq$ stands for the clock frequency of the processing unit, and $P_{cache-miss}$ represents the cache miss rate of the execution. When the task is put on PIM, $P_{cache-miss}$ equals to be 1. IPC and $Freq$ are also different for the tasks on CPU and PIM, so the $P_{conflict}$ should also be changed accordingly when applying the model tasks executed on different processing units. We get these parameters from profiling. For regular applications, the parameters can also be estimated via static program analysis.

Suppose there are C CPU cores and a single PIM core in the hybrid CPU-PIM system and task p_i (i=1,2,...,K) is assigned to processing unit t_j (j=1,2,...,C,C+1). K stands for the total amount of tasks in the task graph. t_{C+1} refers to the PIM. When the task is assigned to PIM, the delay is mainly the memory conflict. When the task is assigned to CPU, the delay consists of both the queueing delay and the conflict delay. In this case, the memory access delay of the task t_i executed on processing unit p_j can be calculated as follows:

$$delay(t_i, p_j) = \begin{cases} delay_{pim}(t_i) & (j = C+1) \\ D \times Inst_{num} \times P_{cache-miss} + delay_{cpu}(t_i) & (others) \end{cases} \tag{4}$$

2.2.2 Communication Costs. Communication between tasks are highly related to the task allocation, so the communication latency model must be aware of the task location. As the dependent tasks may either be assigned to CPU or PIM, there are four possible combinations and the communication latency model is essentially a precise function with four segments as shown in formula 5.

Suppose $C_{m,i}$ represents the communication cost between t_m and t_i depends on t_m without loss of generality. When both tasks are on CPU, the communication data has parts in cache and the rest in memory. The data in cache can be transferred immediately and the communication latency in this case is basically induced by the in-memory data to be transferred to cache, which is thus the amount of in-memory data divided by the memory bandwidth which can be obtained through formula **??**. When task t_m is in CPU and t_i is in PIM, the data in cache needs to be written to memory first while the in-memory part can be transferred to PIM through the high bandwidth cache interface. For the rest two task location scenarios, the communication cost is completely induced by the

memory data movement and can be easily calculated through the amount of data transfer and communication bandwidth as detailed in the formula 5.

$$delay(t_m, t_i) = \begin{cases} T_{init} + \frac{cache_{m,i}}{B_{actual}(t_m)} & (t_m \text{ in } CPU, t_i \text{ in } PIM) \\ T_{init} + \frac{data_{m,i}}{B_{actual}(t_i)} & (t_m \text{ in } PIM, t_i \text{ in } CPU) \\ T_{init} + \frac{data_{m,i} - cache_{m,i}}{B_{actual}(t_i)} & (t_m \text{ and } t_i \text{ in } CPU) \\ T_{init} + \frac{data_{m,i}}{B_{pim}(t_i)} & (t_m \text{ and } t_i \text{ in } PIM) \end{cases} \quad (5)$$

T_{init} is the startup time of memory access.$B_{actual}(t_i)$ is the actual bandwidth of t_i, referring to formula**??**. $cache_{m,i}$ is the communication data between t_m and t_i cached in cache. $data_{m,i}$ is the communication data between t_m and t_i cached in memory.

2.2.3 Task graph and task Execution Time. An application is represented by a directed acyclic graph, $G = (V, E)$ where V is the set of tasks and E is the set of edges, each edge $(i, j) \in E$ represents that t_i should complete its execution before t_j starts. $data(i, j)$, the weight of each edge represents the amount of data that is transferred from t_i to t_j. In a given task graph, a task without any parent is called an entry task and a task without any child is called an exit task. $avail\left[p_j\right]$ is the earliest time that processor p_j is ready for task execution. We define the EST and EFT attributes, which represent the earliest execution start time and the earliest execution finish time of t_i on processor p_j. For the entry task t_{entry}, $EST\left(t_{entry}, p_j\right) = 0$. For the other tasks in the graph, the EFT and EST values are computed recursively,

$$EST\left(t_i, p_j\right) = max\left\{avail\left[p_j\right], \max_{t_m \in pred(t_i)}\left(AFT\left(t_m\right) + C_{m,i}\right)\right\} \quad (6)$$

$$EFT\left(t_i, p_j\right) = EST\left(t_i, p_j\right) + w_{i,j} + delay\left(t_i, p_j\right) \quad (7)$$

Where $AFT\left(t_k\right)$ represents the actual finish time of t_k while it has already been scheduled onto a process unit, $C_{m,i}$ is the communication cost of the entry task and $w_{i,j}$ is the execution time of task t_i on processor p_j when no other tasks is executed on the system during task t_i running, $delay\left(t_i, p_j\right)$ is the memory interference delay time when t_i scheduled to processor p_j.

The finished time of all the applications depends on the last task, so it equals to $MAX\left\{AFT\left(t_{exit}\right)\right\}$, where t_{exit} represents each exit tasks.

2.3 Task Scheduling for PIM-enabled system

2.3.1 Task Priority Computation. Some tasks are located on the critical paths of the task graph execution. If they are delayed by other tasks, the finish time of the entire application will be longer. Therefore, we tend to assign the tasks on the critical path with higher priority and the task with higher priority will be executed earlier. The priority of each task is defined as follows:

$$priority\left(t_i\right) = \overline{w_i} + \max_{t_j \in succ(t_i)}\left(\overline{C_{i,j}} + priority\left(t_j\right)\right) \quad (8)$$

Where t_j is the successor node of the t_i, $\overline{w_i}$ is the average execution time of t_i when t_i located on all the process unit, $\overline{C_{i,j}}$ is the average communication time of task t_i and t_j which is equal to the average of the four cases in the formula5. $succ\left(t_i\right)$ represents all the immediate successor nodes of the node t_i.

Algorithm 1 The PIM-FAST Algorithm

Require: tasks $t_1, t_2, \cdots, t_n$;
1: Calculates the average execution time of all the tasks $(t_1, t_2, \cdots, t_n)$;
2: Compute weight of each node;
3: Generate a task queue by sorting the weight of the tasks;
4: **while** task queue is not empty **do**
5: Select the first task t in the task queue;
6: **for** all processor unit p_j **do**
7: Compute EFT and delay;
8: **end for**
9: Select the task with minimum EFT, and assign the task to the processor;
10: **end while**

Table 1: SYSTEM PARAMETERS USED IN SIMULATION

Component	parameters
CPU	Architecture: x86; Core number: 4; Frequency: 2GHz; L1cache: 64KB; L2 cache: 2MB
HMC	**Memory Dies**:Frequency: 66MHz; Page Size: 4KB; Memory Size: 4GB; Number of Banks: 8; Bandwidth: 320GB/s; **Logic Dies:** Frequency: 1333MHz

2.3.2 Task Assignment. We employ greedy strategy to assign a task to a processing unit which could finish the task in the earliest time, we select the processing unit for each task using the following steps: First, based the memory interference delay when t_i is assigned to processor p_j, task communication and the earliest execution finish time; second, according to the task priority from high to low, we assign each task to the processor unit with the earliest execution finish time. The complete PIM-FAST algorithm is as algorithm1. The task queue in line3 is a priority queue, and each weight of the tasks is calculating by Formula8. Line5 to 9 select the best processor for the task by calculating the earliest finish time of each processor.

The time complexity of the PIM-FAST algorithm is $O\left(e \times m\right)$, where e is the number of edges in the task graph and m is the number of processing units. For sparse graphs, the number of edges is proportional to the number of n (n is the number of tasks) and the time complexity is $O\left(n \times m\right)$. For dense graphs, the number of edges is proportional to n^2 and the time complexity is $O\left(n^2 \times m\right)$.

3 EVALUATION

3.1 Simulation Configuration

We choose Multi2Sim[14] which is a simulator of CPUs, GPUs and heavily modify it to simulate multicore CPU and PIM architectures. Additionally, HMC[11] is used as our computing memory model. McPAT[9] is used to evaluate energy consumption of the system. The parameters of the simulation system are listed in Table.1.

3.2 Workloads

A set of big-data workloads[5] selected from BigDataBench and linear algebra solvers are used as the benchmark. Six workloads are PageRank (PR), Breath-first Search (BFS), K-Means (KM), Sparse Matrix-Vector Multiply (SPMV), Connected Component (CC) and

Table 2: INPUT DATA SIZE

APPs	Small	Medium	Large
PageRank	2M	16M	118M
Breadth First Search	1M	4M	16M
K-means	1M	8M	50M
Sparse Matrix-Vextor Multiply	11M	44M	136M
Connected Component	1M	5M	31M
Topological Sorting	2M	10M	43M

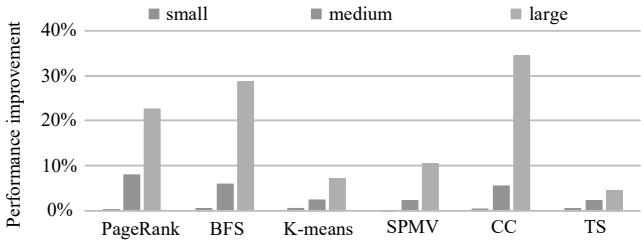

Figure 3: Performance improvement using PIM-FAST algorithm

Topological Sorting (TS). Table.2 lists the 6 workloads with different input data size ranging from small, medium to large. We parallelize the applications and abstract task graphs from them based on the parallelization strategy. Then the task graphs are used for the scheduling experiments.

3.3 Performance Evaluation

We measured the performance of the applications with different data sets on a 4-core CPU+PIM architecture using both PIM-FAST and HEFT scheduling algorithms. The performance comparison is shown in Fig. 4. It can be seen that PIM-FAST scheduling outperforms HEFT scheduling on all the applications of different data sets. Particularly, the performance of PageRank, BFS and CC on a large data set gets most significant improvement. And the performance of CC with large data set improves by 34.4%. The reason is that these graph applications are memory-bound and involve many irregular memory accesses which can benefit most from the PIM architecture.

To gain insight of the scheduling algorithms, we count the memory conflict between parallel tasks in the system. According to the statistic presented in Fig. 5, task execution using PIM-FAST scheduling demonstrates much lower memory conflict rate compared to that using HEFT algorithm. For task graphs with larger data set, PIM-FAST algorithm shows more significant memory conflict reduction as PIM-FAST is aware of the PIM's larger memory bandwidth and takes advantage of the bandwidth for more efficient memory access.

The amount of data transfer from memory to CPU on the benchmark are also estimated and listed in Fig.6. When the data set is small and it can be cached, data transfer between CPU and memory can be reduced. Hereby, the benefit using PIM-FAST is less significant. On the contrast, when the data set is large and cache cannot accommodate the data set, data needs to be frequently transferred. In this case, the PIM-FAST will offload data intensive tasks to PIM

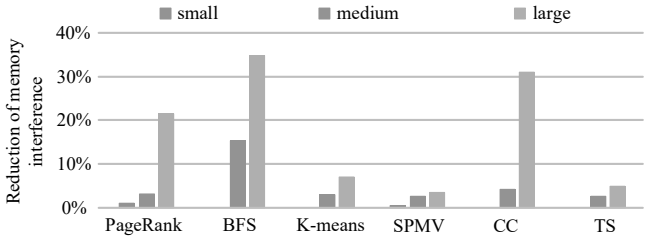

Figure 4: Reduction of memory conflict rate using PIM-FAST algorithm

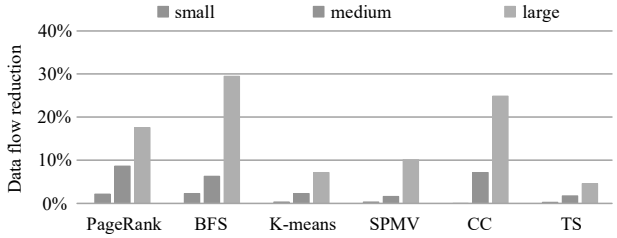

Figure 5: Data transfer reduction between CPU and memory using PIM-FAST algorithm

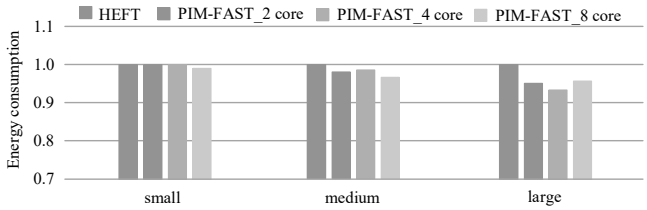

Figure 6: Energy consumption of PIM-FAST over HEFT

and saves considerable data transfer compared to the HEFT. According to the experiments, data transfer reduces up to 27.3% and 15.7% on average.

Fig.6 shows the normalized energy consumption. McPAT is used to calculate the energy consumption of the whole system including HMC, CPU and the in-memory computational unit. With large data set, tasks execution using PIM-FAST save 16.7% energy than that using HEFT on average. With the small data set, the energy consumption of the task execution scheduled using PIM-FAST and HEFT is on a par, because small data set fits in the cache and there is little memory access via the off-chip interconnects.

Fig.7 shows the reduction of memory conflict using PIM-FAST under a different number of bank and different number of cores. In case of 4 cores, with 8 banks, tasks execution using PIM-FAST reduce 17.1% memory conflict on average. However, for 32 banks, tasks execution using PIM-FAST only reduce 2.0% memory conflict on average, because the more the number of banks, the higher the parallelism of memory access. But, in case of 8 cores, with 8 banks, tasks execution using PIM-FAST reduce 27.6% memory conflict on average, and for 32 banks, tasks execution using PIM-FAST reduce 5.3% memory conflict on average. State of the art, the mainly memory always contains 8 banks and PIM-FAST can significantly reduce 27.6% memory conflict for the hardware platform of 8-banks.

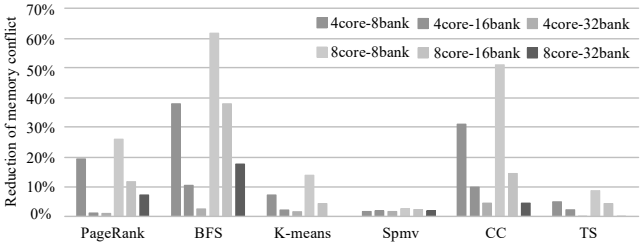

Figure 7: Reduction of memory conflict rate under different number of bank and core

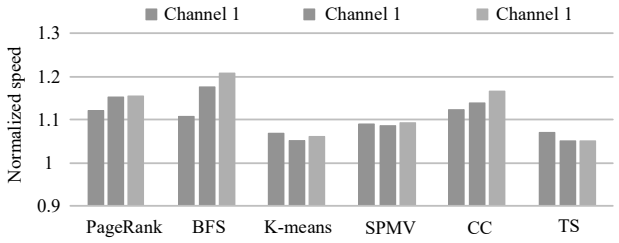

Figure 8: Performance speedup over HEFT on system with different memory channels and PIM cores

While the cores of CPU increase to 8 and the banks increase to 16 or 32, PIM-FAST can also reduce 12.6% or 5.3% memory conflict.

Fig.8 shows the performance improvement of PIM-FAST over HEFT when the number of memory channels as well as the PIM cores scale from 1 to 4. For applications with less memory intensive operations such as TS and K-means, multiple channel memory with higher memory bandwidth squeezes the chance of offloading memory accesses to PIM cores. The performance speedup is relatively lower accordingly. For memory intensive applications, PIM-FAST scheduling achieves higher performance speedup given more memory channels. This reason comes from two aspects. On the one hand, each channel has one PIM core included and multiple bank memory has more PIM cores allowing more tasks to be offloaded. On the other hand, the memory conflict is alleviated, which is also beneficial to the overall performance. In general, the number of memory channel has both positive and negative influence on the proposed scheduling algorithm. It affects the resulting scheduling performance in combination with the application memory access patterns.

4 CONCLUSION

In this paper, we formalized the task scheduling problem on general PIM-CPU architecture and proposed a PIM-FAST scheduling algorithm to adapt to the PIM-CPU architecture. With dedicated analytical models, the scheduling algorithm is fully aware of the PIM processing features, memory access contention and data communication on concurrent workloads. Experiments show that the proposed scheduling algorithm outperforms the state-of-art scheduling algorithm for conventional hybrid computing architectures by 18.0% on average and up to 34.4% for larger data sets.

5 ACKNOWLEDGEMENT

This work is supported by National Natural Science Foundation of China under Grant No.61874124, No.61674048, No.61704032, No.61834006 and No.61902375. The corresponding author is Cheng Liu.

REFERENCES

[1] Junwhan Ahn, Sungpack Hong, Sungjoo Yoo, Onur Mutlu, and Kiyoung Choi. 2015. A scalable processing-in-memory accelerator for parallel graph processing. In *Proceedings of the 42nd Annual International Symposium on Computer Architecture, Portland, OR, USA, June 13-17, 2015*, Deborah T. Marr and David H. Albonesi (Eds.). ACM, 105–117. https://doi.org/10.1145/2749469.2750386

[2] Junwhan Ahn, Sungjoo Yoo, Onur Mutlu, and Kiyoung Choi. 2015. PIM-enabled instructions: a low-overhead, locality-aware processing-in-memory architecture. In *Proceedings of the 42nd Annual International Symposium on Computer Architecture, Portland, OR, USA, June 13-17, 2015*, Deborah T. Marr and David H. Albonesi (Eds.). ACM, 336–348. https://doi.org/10.1145/2749469.2750385

[3] Raphaël Bleuse, Sascha Hunold, Safia Kedad-Sidhoum, Florence Monna, Gregory Mounie, and Denis Trystram. 2017. Scheduling Independent Moldable Tasks on Multi-Cores with GPUs. *IEEE Trans. Parallel Distrib. Syst.* 28, 9 (2017), 2689–2702. https://doi.org/10.1109/TPDS.2017.2675891

[4] Ingrid Y Bucher and Donald A Calahan. 1990. Access conflicts in multiprocessor memories queueing models and simulation studies. *SIGARCH* 18, 3b (1990), 428–438.

[5] Ingrid Y. Bucher and Donald A. Calahan. 1990. Access conflicts in multiprocessor memories queueing models and simulation studies. In *Proceedings of the 4th international conference on Supercomputing, ICS 1990, Amsterdam, The Netherlands, June 11-15, 1990*, Ahmed H. Sameh and Henk A. van der Vorst (Eds.). ACM, 428–438. https://doi.org/10.1145/77726.255184

[6] Louis-Claude Canon and Emmanuel Jeannot. 2010. Evaluation and Optimization of the Robustness of DAG Schedules in Heterogeneous Environments. *IEEE Trans. Parallel Distrib. Syst.* 21, 4 (2010), 532–546. https://doi.org/10.1109/TPDS.2009.84

[7] Kevin Hsieh, Eiman Ebrahimi, and Gwangsun Kim. 2016. Transparent Offloading and Mapping (TOM): Enabling Programmer-Transparent Near-Data Processing in GPU Systems. In *43rd ACM/IEEE Annual International Symposium on Computer Architecture, ISCA 2016, Seoul, South Korea, June 18-22, 2016*. IEEE Computer Society, 204–216. https://doi.org/10.1109/ISCA.2016.27

[8] Sang Cheol Kim and Sunggu Lee. 2007. Push-Pull: Deterministic Search-Based DAG Scheduling for Heterogeneous Cluster Systems. *IEEE Trans. Parallel Distrib. Syst.* 18, 11 (2007), 1489–1502. https://doi.org/10.1109/TPDS.2007.1106

[9] Sheng Li, Jung Ho Ahn, Richard D. Strong, Jay B. Brockman, and Dean M. Tullsen. 2009. McPAT: an integrated power, area, and timing modeling framework for multicore and manycore architectures. In *42st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-42 2009), December 12-16, 2009, New York, New York, USA*, David H. Albonesi, Margaret Martonosi, David I. August, and José F. Martínez (Eds.). ACM, 469–480. https://doi.org/10.1145/1669112.1669172

[10] Sadegh Mirshekarian and Dusan N. Sormaz. 2016. Correlation of job-shop scheduling problem features with scheduling efficiency. *Expert Syst. Appl.* 62 (2016), 131–147. https://doi.org/10.1016/j.eswa.2016.06.014

[11] Lifeng Nai and Hyesoon Kim. 2015. Instruction Offloading with HMC 2.0 Standard: A Case Study for Graph Traversals. In *Proceedings of the 2015 International Symposium on Memory Systems, MEMSYS 2015, Washington DC, DC, USA, October 5-8, 2015*, Bruce Jacob (Ed.). ACM, 258–261. https://doi.org/10.1145/2818950.2818982

[12] Ashutosh Pattnaik, Xulong Tang, Adwait Jog, Onur Kayiran, Asit K. Mishra, Mahmut T. Kandemir, Onur Mutlu, and Chita R. Das. 2016. Scheduling Techniques for GPU Architectures with Processing-In-Memory Capabilities. In *Proceedings of the 2016 International Conference on Parallel Architectures and Compilation, PACT 2016, Haifa, Israel, September 11-15, 2016*, Ayal Zaks, Bilha Mendelson, Lawrence Rauchwerger, and Wen-mei W. Hwu (Eds.). ACM, 31–44. https://doi.org/10.1145/2967938.2967940

[13] Haluk Topcuoglu, Salim Hariri, and Min-You Wu. 2002. Performance-Effective and Low-Complexity Task Scheduling for Heterogeneous Computing. *IEEE Trans. Parallel Distrib. Syst.* 13, 3 (2002), 260–274. https://doi.org/10.1109/71.993206

[14] Rafael Ubal, Julio Sahuquillo, Salvador Petit, and Pedro López. 2007. Multi2Sim: A Simulation Framework to Evaluate Multicore-Multithreaded Processors. In *19th Symposium on Computer Architecture and High Performance Computing (SBAC-PAD 2007), 24-27 October 2007, Gramado, RS, Brazil*. IEEE Computer Society, 62–68. https://doi.org/10.1109/SBAC-PAD.2007.30

[15] Guoqi Xie, Yuekun Chen, Yan Liu, Yehua Wei, Renfa Li, and Keqin Li. 2017. Resource Consumption Cost Minimization of Reliable Parallel Applications on Heterogeneous Embedded Systems. *IEEE Trans. Ind. Informatics* 13, 4 (2017), 1629–1640. https://doi.org/10.1109/TII.2016.2641473

[16] Guoqi Xie, Junqiang Jiang, Yan Liu, Renfa Li, and Keqin Li. 2017. Minimizing Energy Consumption of Real-Time Parallel Applications Using Downward and Upward Approaches on Heterogeneous Systems. *IEEE Trans. Ind. Informatics* 13, 3 (2017), 1068–1078. https://doi.org/10.1109/TII.2017.2676183