EFFICIENT TASK ASSIGNMENT AND SCHEDULING FOR MPSOC DSPS WITH VS-SPM CONSIDERING CONCURRENT ACCESSES THROUGH DATA ALLOCATION

Shouzhen Gu[†] *Qingfeng Zhuge*[†] *Jingtong Hu*[⋆] *Juan Yi*[†] *Edwin H.M. Sha*[⋆†]

[†]College of Computer Science, Chongqing University, Chongqing, China.
[⋆]Department of Computer Science, University of Texas at Dallas, Richardson, TX, USA.

ABSTRACT

Virtually Shared Scratch-Pad Memory (VS-SPM) with multiple memory banks can be used as on-chip memory on multiprocessor systems-on-chips (MPSoCs) to close the speed gap between fast processors and slow memories. By exploring the parallelism of computation tasks on processors and concurrent data accesses on each SPM, the results of task assignment and data allocation can significantly affect the overall performance of a schedule. In this paper, we propose ILP formulations for solving the problem of task assignment and scheduling on MPSoCs with multi-bank VS-SPM. We also propose a polynomial-time algorithm, *the Potential Remote Access Prediction* (PRAP) algorithm, to generate near-optimal results efficiently. The experimental results demonstrate the effectiveness of our technique.

Index Terms— Task Assignment, Data Allocation, Scheduling, MPSoC, Virtually Shared SPM

1. INTRODUCTION

Many modern embedded systems, including high-performance digital signal processing systems, are designed as multiprocessor System-on-Chip (MPSoC). Scratch-Pad Memory (SPM), a Software-controlled on-chip memory, has been considered as an alternative to cache due to its small die area and energy efficiency [1]. In SPM, data transfer between SPMs and off-chip memory is explicitly managed by software [2], which gives more control of memory accesses to software. Examples of MPSoCs using SPMs as their on-chip memories include Texas Instruments TMS370Cx [3] and Motorola 68HC12 [4]. To further improving the overall system performance, many MPSoCs are equipped multiple SPM banks. All the SPM banks can be accessed in parallel. Providing the increasing parallelism on multiple SPM banks, the process of task assignment and data allocation becomes very critical in fully utilizing the advantage of this kind of architecture and generating compact schedules.

In this paper, we study the problem of task assignment and scheduling along with data allocation on MPSoC with virtually shared SPM [5]. In our target architecture, each processor is attached to a local on-chip SPM with multiple banks that can be accessed concurrently. Each processor is able to access all the SPMs in MPSoC.

There have been many research efforts on solving the problem of data allocation and task assignment for performance improvement [6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20]. However, most of their target architectures are different from this paper. Zhang et al. [21] proposed two variable partitioning heuristics based on an initial schedule and presented a loop pipeline scheduling algorithm to improve overall throughput. The Virtually Shared Scratch-Pad Memory (VS-SPM) architecture in their paper has a single bank SPM on each core. Hence, concurrent data accesses are not allowed on one SPM. Zhuge et al. [22] explored variable partitioning and scheduling on multiple-memory-module architectures and proposed a variable independence graph (VIG) model for approaching the variable partitioning problem. Their research work focused on instruction-level scheduling. Zhuge et al. [23] proposed an optimal data allocation for scalar variables on a single CPU with multiple memory types. Zhang et al. [24] proposed a dynamic programming approach to allocate the scalar and array variables for embedded systems with multiple types of memory units. Their approaches are not applicable to the architecture which allows concurrent data accesses. In this paper, we study the problem of minimizing execution time during task assignment and task-level scheduling while taking advantage of parallel memory accesses on VS-SPM through data allocation.

In this paper, we first propose Integer Linear Programming (ILP) formulations for task assignment and scheduling on MPSoC with multi-bank VS-SPM. The ILP method is able to obtain the optimal solution with minimum schedule length. We also propose an efficient algorithm, *the Potential Remote Access Prediction* (PRAP) algorithm for solving the task assignment and scheduling problem in polynomial time. The PRAP algorithm generates task assignment and data allocation for MPSoC with multi-bank VS-SPM so that the concurrency of memory accesses can be fully utilized during scheduling. We evaluate the effectiveness of the PRAP algorithm by comparing schedule lengths generated by three different techniques: the High Access Frequency First (HAFF) algorithm proposed in [21], the ILP method and our PRAP algorithm. Experimental results show that the PRAP algorithm achieves an improvement of 25.9% on average for all the benchmarks compared with the HAFF algorithm.

The main contributions of this paper include:

- We develop ILP formulations to produce the optimal solution for the task assignment and scheduling problem on MPSoCs with multi-bank VS-SPM.
- We design a polynomial-time algorithm, the PRAP algorithm, to explore the opportunity of concurrent memory accesses among SPMs and generate near-optimal schedule lengths efficiently.

This work is partially supported by NSF CNS-1015802, Texas NHARP 009741-0020-2009, HK GRF 123609, NSFC 61173014, National 863 Program 2013AA013202, China Thousand-Talent Program.

The rest of our paper is organized as follows: hardware and software models are introduced in Section 2. A motivational example is presented in Section 3 to illustrate the basic ideas. The ILP formulations are presented in Section 4. The PRAP algorithm is presented in Section 5. The experimental results are presented in Section 6. Section 7 concludes the whole paper.

2. BASIC MODEL

In this section, we first introduce the *MPSoC with multi-bank VS-SPM* architecture. Then, the *Memory-access Date Flow Graph (MDFG)* model and concepts are presented.

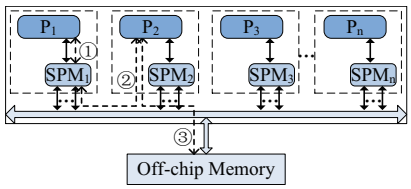


Fig. 1. Architectural model MPSoC with multi-bank VS-SPM.

The target MPSoC architecture is shown in Fig. 1. There are multiple processors on a single chip. Each processor is equipped with an SPM, which serves as local on-chip memory. Each processor can access its local SPM and all the other SPMs. If a processor accesses its own SPM, we call it a Local Access. The access latency on a local SPM is quite small. If a processor accesses an SPM in other cores, we call it a Remote Access. It incurs a larger latency than local access latency. If a processor accesses off-chip memory, we call it an Off-chip Access. It causes the largest access latency. In this work, we assume that off-chip access latency can be hidden by data prefetching and reasonable regional program fetching. Please note that the SPMs used in our architectural model are multi-bank SPMs. Hence, multiple data accesses can be performed in parallel on each SPM.

The target Architecture MPSoC *T* is a set of *M* cores $\{C_1, C_2, \ldots, C_m\}$. Each core is composed of a SPM and a processor. The concurrent access number MA denotes the number of data accesses that can be processed concurrently on a SPM.

In this paper, we use Memory-access Data Flow Graph (MDFG) to model tasks and memory accesses. In our MDFG model, there are two types of nodes. One type of nodes represents computation tasks. The other type represents memory access operations. Each task needs to read data before execution and write data to SPMs after execution. The MDFG is defined as follows:

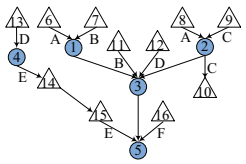
Definition 2.1. *A MDFG* $G = < V_1, V_2, E, t, var >$ *is a node-weighted directed graph, where* V_1 *represents a set of computation tasks,* V_2 *represents a set of memory access operations,* $E \subseteq V \times V$*, where* $V = V_1 \bigcup V_2$*, is an edge set. Each edge* $(e : u \rightarrow v) \in E$ *represents precedence relation between nodes* $u, v \in V$*,* $t(u)$ *is a function represents the computation time of a task* $u \in V_1$*,* $var(u)$ *represents a variable accessed by a memory operation* $u \in V_2$*.*

Given a MDFG G and target architecture configurations, the goal of task assignment on MPSoC is to find appropriate allocation for both tasks and data on a set of cores and SPMs such that the schedule length can be minimized. To achieve this, our proposed methods need to solve the following problems:

- *Task Assignment*. A legal assignment for computation tasks should obey both precedence relations and resource constraints.
- *Data Allocation*. Data are allocated to SPMs so that memory access cost can be minimized. We only keep one copy of data in the system.
- *Scheduling*. A schedule determines the start time of computation tasks and memory access operations.

3. MOTIVATIONAL EXAMPLE

In this section, we illustrate the main technique proposed in this paper with a motivational example.



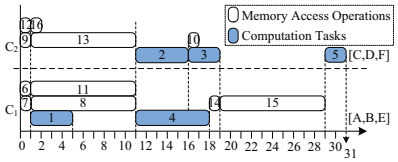
(a) The MDFG.

u	Latency(*u*)	R(*u*)	W(*u*)
1	4	A, B	
2	5	A, C	C
3	3	B, D	
4	7	D	E
5	2	E, F	

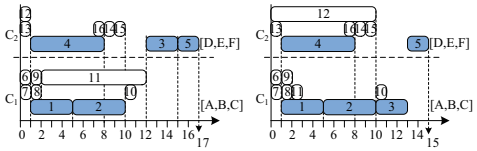
(b) Execution time and variable accesses for each computation task.

Fig. 2. Motivational example.

Fig. 2(a) shows an example MDFG. The computation tasks are donated by shaded circles. The memory access operations are denoted by triangle nodes. Fig. 2(b) shows the execution time and variable accesses for each computation task. For example, task 2 reads variables "A" and "C" before execution. It writes "C" back to SPM after the execution. In this example, we assume that the computation latency of task 2 is 5 clock cycles. We also assume that a local SPM access takes 1 clock cycle and a remote access takes 10 clock cycles. The target architecture for this example has two cores. Each core has one on-chip SPM of size 3 data units. The number of concurrent accesses allowed on each SPM is 2.



(a) The original schedule.



(b) The schedule generated by PRAP algorithm.

(c) The schedule generated by ILP.

Fig. 3. Compare schedules generated by various methods.

Fig. 3(a) shows a schedule generated by list scheduling algorithm without considering task assignment and data allocation. The memory access operations are denoted by whites bars. The computation tasks are denoted by shaded bars. Schedules on two cores are separated by dotted line. Data allocated on two cores are shown at the end of each schedule. The schedule length is 31 clock cycles.

Fig. 3(b) shows an improved schedule generated by the proposed PRAP algorithm. In the PRAP algorithm, we try to maximize the parallelism of computation tasks and data accesses on all the SPMs. At the same time, the algorithm minimizes the number of remote accesses. By assigning computation tasks 1 and 2 and variables A, B and C on core C_1 , a remote access of variable "A" for task 2 can be eliminated. By assigning computation tasks 3, 4 and 5 and variables D, E and F on core C_2 , remote accesses of variable "D" for task 4 and variable "E" for task 5 can be eliminated. As a result, the schedule length is reduced from 31 to 17 clock cycles.

Fig. 3(c) shows the optimal schedule generated by ILP formulations. It generates a schedule length of 15 clock cycles which is very close to the schedule length obtained by the proposed PRAP algorithm.

4. ILP MODEL

In this section, we propose ILP formulations for the task assignment and scheduling problem on MPSoC. First, we introduce notations necessary for constructing the ILP formulations as shown in Table 1. Then, we formally define the ILP formulations.

Table 1. Notations used in the ILP formulations.

Notation	Definition
$x_{u,j,k}$	$= 1$ if and only if a computation task or memory operation u starts to execute on core k at step j.
$y'_{u,j,k}$	$= 1$ if and only if there is a memory access operation u is being executed on SPM k at step j.
$z'_{u,j,k}$	$= 1$ if and only if there is a computation task u is being executed on processor k at step j.
$m_{h,k}$	$= 1$ if and only if data variable h is allocated to SPM k.
N_1	Number of computation tasks
N_2	Number of memory access operations
S	Upper bound of execution time which can be represented by steps.
M	Number of processors or SPMs
N_d	Number of data in G
$MSize_k$	Size of SPM k
$Size_h$	Size of data h

Task Mapping Constraint Each node in a given MDFG is executed exactly once at any time, as shown in equation (1).

$$\sum_{j=1}^{S}\sum_{k=1}^{M} x_{u,j,k} = 1, \forall u \in V_1 \cup V_2. \tag{1}$$

Data Allocation Constraints During data allocation, we need to consider three constraints. First, each data can be allocated to one SPM, as shown in equation (2).

$$\sum_{k=1}^{M} m_{h,k} = 1, \forall h \in [1, N_d]. \tag{2}$$

Second, the total size of data allocated to an SPM should not exceed the size of SPM.

$$\sum_{h=1}^{N_d} Size_h \times m_{h,k} \leq MSize_k, \forall k \in [1, M]. \tag{3}$$

Third, data should be allocated to the same core for corresponding memory access operations.

$$\sum_{k=1}^{M} k \times m_{var(u),k} = \sum_{j=1}^{S}\sum_{k=1}^{M} k \times x_{u,j,k}, \forall u \in V_2. \tag{4}$$

Dependency Constraint A dependency edge $e(u, v) \in E$ indicates that node v cannot be executed until node u is finished, where $u, v \in V_1 \cup V_2$.

$$\sum_{j=1}^{S}\sum_{k=1}^{M} j \times x_{u,j,k} + t(u) \leq \sum_{j=1}^{S}\sum_{k=1}^{M} j \times x_{v,j,k}, \quad \forall e(u, v) \in E, \forall u, v \in V_1 \cup V_2. \tag{5}$$

In equation (5), $t(u)$ represents the execution time of a node u. For a computation task, the execution latency is given in MDFG. For a data access operation, the execution time depends on the distance of data location. If a computation task u and a data h accessed by u are assigned to the same processor, the corresponding data access is a local access. Otherwise, it is a remote access.

Resource Constraint There is only one computation task can be executed on a processor k at any time, as shown in equation (6).

$$\sum_{u=1}^{N_1} z'_{u,j,k} \leq 1, \forall j \in [1, S], \forall k \in [1, M]. \tag{6}$$

There are up to MA concurrent data accesses can be executed on an SPM at any time.

$$\sum_{u=1}^{N_2} y'_{u,j,k} \leq MA, \forall j \in [1, S], \forall k \in [1, M]. \tag{7}$$

Objective Function. The objective function can be formulated as follows:

$$\min \sum_{j=1}^{S}\sum_{k=1}^{M} j \times x_{u_0,j,k}. \tag{8}$$

where u_0 is a dummy node added to MDFG to compute the finish time of the whole graph. We also added one outgoing edge from each node with no child to u_0.

5. THE POTENTIAL REMOTE ACCESS PREDICTION (PRAP) ALGORITHM

In this section, we propose the *Potential Remote Access Prediction*(PRAP) algorithm. It computes task assignment and data allocation for a given MDFG by exploring opportunity of parallelization of data accesses and tasks to generate a compact schedule in polynomial time.

Algorithm 5.1 shows the procedure of PRAP algorithm. Initially, we assume that all the variables are located in main memory and there is no task assigned to any core. We build a priority queue of computation tasks in a given MDFG using topological sort.An array $Available(n)$ is used to record the available time slots on core n for task assignment. A matrix $Freq[n][d]$ is used to record data access frequencies for each variable d allocated on core n.

The PRAP algorithm tries to maximize the parallelism of computation tasks and data accesses while considering the cost of remote data accesses. In each iteration from line 8 to line 17, the algorithm tries to assign a computation task q_α to a core n. Meanwhile, it tries to allocate variables accessed by task q_α. If a variable is already located in core n, task q_α can access the variable on a local SPM. Otherwise, the algorithm tries to allocate the variable to the local SPM whenever the number of concurrent data accesses on an SPM is allowed by the architecture. Line 13 to line 16 compare the finish

Algorithm 5.1 Potential Remote Access Prediction (PRAP)

Require: (1) A graph model MDFG $G =< V_1, V_2, E, t, var >$; (2) Data access frequency for each task; (3) Architectural model T.

Ensure: Data allocation results for all the variables. A schedule of tasks and data accesses on all the cores.

1: Build a priority queue Q of computation tasks.
2: Initialize $Available(n) \leftarrow 0$.
3: Initialize a $M \times N_d$ matrix $Freq[n][d] \leftarrow 0$.
4: **while** (Q) **do**
5: $q_\alpha \leftarrow$ Dequeue(Q);
6: $Est \leftarrow$ The earliest executable time of q_α;
7: $Core\_id \leftarrow$ ID of a randomly picked core, $1 \leq Core\_id \leq n$;
8: **for** core n **do**
9: $Min \leftarrow \infty$;
10: $Num(n) \leftarrow$ The number of data variables that have been located on core n for task q_α before it is assigned to core n.
11: $R_p(n) \leftarrow$ The total number of remote accesses for task q_α assuming q_α is assigned to core n.
12: $St(n) \leftarrow$ The earliest start time of q_α on core n;
13: **if** $((Min > St(n) + L_r \times R_p(n))or(Min = St(n) + L_r \times R_p(n)$ and $Num(n) > Num(Core\_id)))$ **then**
14: $Min \leftarrow St(n) + L_r \times R_p(n)$;
15: $Core\_id \leftarrow n$;
16: **end if**
17: **end for**
18: $ass(q_\alpha) \leftarrow Core\_id$;
19: Update array $Available$ and matrix $Freq$.
20: **end while**
21: Decide data allocation on each SPM for variables with the highest access frequency in $Freq$ considering the size of SPM.
22: $Sch \leftarrow$ Conduct list Scheduling.
23: **return** Sch and a map of data allocation;

time of task q_α on each core and update the value of Min with the earliest finish time. Line 18 assigns computation task q_α to a core where it can be finished at the earliest time. Line 21 decides variable location based on the size of each SPM. If there's no enough space on SPM to store all the variables, the algorithm allocates the variables with the lowest access frequency to main memory.

6. EXPERIMENTS

The effectiveness of PRAP algorithm is evaluated by a set of DSP-stone benchmarks. The experiments are conducted on two simulated MPSoCs with multi-bank SPMs allowing concurrent accesses. Each simulated MPSoC has two cores. One of them allows 2 concurrent accesses on SPM. The other allows 4 concurrent accesses on SPM. We run our simulator on a PC with a Intel(R) Core(TM) i5-2400 CPU which runs at 3.10 GHz and has 4GB of main memory. We implement the ILP formulations in Lingo to generate optimal results. However, the ILP cannot finish within reasonable time for some benchmarks such as "elf" filter.

Table 2 shows the experimental results of various schedule lengths produced by three different methods on the architecture with 2 cores. Each core is equipped with a 2-bank SPM allowing 2 concurrent data accesses. They are the HAFF algorithm proposed in [21], the ILP method, and our PRAP algorithm. The "SL" columns show the number of clock cycles of a schedule generated by different methods. The "Imp" columns show the reduction rate of schedule length obtained by ILP or the PRAP algorithm compared with the HAFF algorithm. ILP cannot find optimal solutions for most of the benchmarks in several hours. The PRAP algorithm is able to generate schedules efficiently for all the benchmarks. It reduces schedule length by 25.9% on average. For the "rls-lat" filter, which has 19 tasks and 11 data items, the PRAP algorithm reduces the schedule length by 36.1%.

Table 2. Schedule lengths on a 2-core MPSoC with 2-bank SPMs.

	HAFF	ILP		PRAP	
Benchmark	SL	SL	Imp	SL	Imp
motiv	45	30	33.3%	30	33.3%
iir	23	12	47.8%	15	34.8%
deq	57	45	21.1%	49	14.0%
diff-clt	40	15	62.5%	30	25.0%
allpole	30	17	43.3%	22	26.7%
rls-lat	61	-	-	39	36.1%
volt	84	-	-	60	28.6%
elf	198	-	-	181	8.6%
Ave. Imp	-	-	41.6%	-	25.9%

Table 3. Schedule lengths on a 2-core MPSoC with 4-bank SPMs.

	HAFF	ILP		PRAP	
Benchmark	SL	SL	Imp	SL	Imp
motiv	53	29	45.3%	29	45.3%
iir	42	12	71.4%	15	64.3%
deq	57	41	28.1%	49	14.0%
diff-clt	30	15	50.0%	26	13.3%
allpole	30	17	43.3%	22	26.7%
rls-lat	49	-	-	39	20.4%
volt	64	-	-	56	12.5%
elf	195	-	-	158	19.0%
Ave. Imp	-	-	47.6%	-	26.9%

Table 3 shows the experimental results of various schedule lengths produced by three different methods on the architecture with 2 cores and 4-bank SPMs providing 4 concurrent data accesses. On this architecture, the PRAP algorithm is able to utilize 4 parallel data accesses as possible and generate compact schedules. It achieves a 26.9% reduction of schedule length on average over all the benchmarks.

Compared with the HAFF algorithm, our PRAP algorithm is more effective on task assignment and scheduling for MPSoCs with multi-bank SPMs providing concurrent data accesses on each core. The reason is that the PRAP algorithm considers effects of both task assignment and data allocation at the same time. It tries to maximize the parallelism of tasks and data accesses while considering the cost of data accesses on remote SPM.

7. CONCLUSION

In this paper, we study the problem of task assignment and scheduling on MPSoC with virtually shared multi-bank SPMs allowing concurrent data accesses for each core. We build ILP formulations to obtain the optimal schedule length. We also propose an efficient algorithm, the *Potential Remote Access Prediction*(PRAP) algorithm, to generate near-optimal results in polynomial time. The experimental results show that the PRAP algorithm is more effective than the HAFF algorithm in exploring parallelism in simulated MPSoC architectures and generating compact schedules.

8. REFERENCES

[1] R. Banakar, S. Steinke, B.-S. Lee, M. Balakrishnan, and P. Marwedel, "Scratchpad memory: A design alternative for cache on-chip memory in embedded systems," in *CODES '02*, 2002, pp. 73–78.

[2] R. Preeti, D. Nikil, and N. Alexandru, "On-chip vs. off-chip memory: The data partitioning problem in embedded processor-based systems," *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 5, no. 682-704, July 2000.

[3] Texas Instruments (1997), "*Tms370cx7x 8-bit microcontroller*," `http://www-s.ti.com/sc/psheets/spns034c/spns034c.pdf`.

[4] Motorola Corporation (2000), "*Cpu12 reference manual*," `http://e-www.motorola.com/brdata/PDFDB/MICROCONTROLLERS/16BIT/68HC12FAMILY/REFMAT/CPU12RM.pdf`.

[5] M. Kandemir, J. Ramanujam, and A. Choudhary, "Exploiting shared scratch pad memory space in embedded multiprocessor systems," in *DAC'02*, 2002, pp. 214–224.

[6] R. Leupers and D. Kotte, "Variable partitioning for dual memory bank dsps," *ICASSP '01*, vol. 2, no. 1121-1124, 2001.

[7] Yibo Guo, Qingfeng Zhuge, Jingtong Hu, and Edwin H.-M. Sha, "Optimal data placement for memory architectures with scratch-pad memories," in *The 8th IEEE International Conference on Embedded Software and Systems (ICESS 2011)*, 2011, pp. 1045 – 1050.

[8] Prashant Agrawal, Kanishk Sugand, Martin Palkovic, Praveen Raghavan, Liesbet Van der Perre, and Francky Catthoor, "Partitioning and assignment exploration for multiple modes of ieee 802.11n modem on heterogeneous mpsoc platforms," in *15th Euromicro Conference on Digital System Design (DSD)*, 2012, pp. 608–615.

[9] Yibo Guo, Qingfeng Zhuge, Jingtong Hu, Meikang Qiu, and Edwin H.-M. Sha, "Optimal data allocation for scratch-pad memory on embedded multi-core systems," in *The 40th International Conference on Parallel Processing (ICPP 2011)*, 2011, pp. 401–410.

[10] Jingtong Hu, Qingfeng Zhuge, Chun Jason Xue, Wei-Che Tseng, and Edwin H.-M. Sha, "Management and optimization for non-volatile memory based hybrid scratchpad memory on multi-core embedded processors," *ACM Transactions on Embedded Computing Systems (TECS)*.

[11] O. Avissar, R. Barua, and D. Stewart, "An optimal memory allocation scheme for scratch-pad-based embedded systems," *ACM Transaction on Embedded Computing System*, vol. 1, no. 6-26, 2002.

[12] Chun Jason Xue, Tiantian Liu, Zili Shao, Jingtong Hu, Zhiping Jia, Weijia Jia, and Edwin H.-M. Sha, "Address assignment sensitive variable partitioning and scheduling for dsps with multiple memory banks," in *ICASSP*, 2008, pp. 1453 – 1456.

[13] Jingtong Hu, Chun Jason Xue, Wei-Che Tseng, Meikang Qiu, Yingchao Zhao, and Edwin H.-M. Sha, "Minimizing memory access schedule for memories," in *Proceedings of the 15th International Conference on Parallel and Distributed Systems (ICPADS)*, 2009, pp. 401–410.

[14] Yibo Guo, Qingfeng Zhuge, Jingtong Hu, Meikang Qiu, and Edwin H.-M. Sha, "Optimal data placement and duplication for embedded multi-core systems with scratch pad memory," *IEEE Transactions on COMPUTER-AIDED DESIGN of Integrated Circuits and Systems (TCAD)*.

[15] Jingtong Hu, Chun Jason Xue, Wei-Che Tseng, Qingfeng Zhuge, Yingchao Zhao, and Edwin H.-M. Sha, "Memory access schedule minimization for embedded systems," *Journal of Systems Architecture (JSA)*, vol. 58, no. 48-59, 2012.

[16] Zili Shao, Qingfeng Zhuge, Chun Xue, and Edwin H.-M. Sha, "Efficient assignment and scheduling for heterogeneous dsp systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 16, no. 516-525, 2005.

[17] Chun Xue, Zili Shao, Meilin Liu, Meikane Qiu, and E.H.-M. Sha, "Loop scheduling with complete memory latency hiding on multi-core architecture," in *12th International Conference on Parallel and Distributed Systems*, 2006.

[18] Meikang Qiu and Lei Zhangand Edwin H.-M. Sha, "Ilp optimal scheduling for multi-module memory," in *Proceedings of the 7th IEEE/ACM international conference on Hardware/software codesign and system synthesis*, 2009, pp. 277–286.

[19] Meikang Qiu, Minyi Guo, Meiqin Liu, Chun Jason Xue, Laurence T. Yang, and Edwin H.-M. Sha, "Loop scheduling and bank type assignment for heterogeneous multi-bank memory," *Journal of Parallel and Distributed Computing*, vol. 69, 2009.

[20] Jingtong Hu, Chun Jason Xue, Qingfeng Zhuge, Wei-Che Tseng, and Edwin H.-M. Sha, "Data allocation optimization for hybrid scratch pad memory with sram and non-volatile memory," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems (TVLSI)*.

[21] Lei Zhang, Meikang Qiu, Wei-Che Tseng, and Edwin H.-M. Sha, "Variable partitioning and scheduling for mpsoc with virtually shared scratch pad memory," *Journal of Signal Processing System*, vol. 58, no. 247-265, 2010.

[22] Qingfeng Zhuge, Edwin H.-M. Sha, Bin Xiao, and Chantana Chantrapornchai, "Efficient variable partitioning and scheduling for dsp processors with multiple memory modules," *IEEE Transactions on Signal Processing (TSP)*, vol. 52, no. 1090-1099, 2004.

[23] Qingfeng Zhuge, Yibo Guo, Jingtong Hu, Wei-Che Tseng, Chun Jason Xue, and Edwin H.-M. Sha, "Minimizing access cost for multiple types of memory units in embedded systems through data allocation and scheduling," *IEEE Transactions on Signal Processing (TSP)*, vol. 60, no. 3253-3263, 2012.

[24] Jun Zhang, Tan Deng, Qiuyan Gao, Qingfeng Zhuge, and Edwin H.-M. Sha, "Optimizing data allocation for loops on embedded systems with scratch-pad memory," in *IEEE 18th International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, 2012, pp. 184 – 191.