Mapping Visual Signal Processing onto Multi-Core Platform via Algorithm/Architecture Co-Exploration

Chun-Fu Chen, Gwo Giun (Chris) Lee, Zheng-Han Yu, Chun-Hsi Huang

Department of Electrical Engineering,
National Cheng Kung University,
Tainan City, Taiwan

n28991146@mail.ncku.edu.tw, clee@mail.ncku.edu.tw, n26020062@mail.ncku.edu.tw, n26024993@mail.ncku.edu.tw

Abstract—Degree of parallelism and data communication should be investigated to achieve high performance for mapping algorithm onto multi-core platform since multi-core platform would concurrently process multiple tasks and lots of data would be transferred between storages and processors. This paper proposes a method to resolve the burden of increase on data transfer rate in parallel processing via the analysis on the dependency matrix of data flow graph. The proposed method does not bias any multi-core platform since it just considers the intrinsic characteristics of algorithm, i.e., data flow graph. This paper utilizes dependency matrix, which conveys the causality of data transfer, for quantifying data transfer rate and corresponding storage requirement; as a consequence, a feasible mapping result, which has smaller data transfer rate and acceptable storage requirement, was exploited. Furthermore, in conjunction with degree of parallelism quantification, this paper presents a comprehensive exploration on design space for mapping algorithm onto multi-core platform through dependency matrix. IBM Cell Broadband Engine is selected to be the targeted multi-core platform in this paper. Experimental results show that when six cores are used, our result can speedup 5.75x on average as compared to single-core case; in addition, by integrating the proposed method on data transfer analysis, about 46% cycles of data transfer could be saved and overall performance could be further increased to 7.51x on average in comparison with the scenario of single-core without data reuse.

Keywords—algorithm/architecture co-exploration, degree of parallelism, multi-core platform, data transfer rate, IBM Cell Broadband Engine

I. INTRODUCTION

Multi-core or many-core platforms have been utilized to enhance the performance via using more processors than single-thread processor; however, the system performance did not increase as growing ratio of the number of processors due to the fact that the algorithm did not be concurrently revised to fit multi-core platform. Currently, there is no principal to appropriately indicate the design paradigm for multi-core platform. In addition, data transfer rate on multi-core platform usually becomes larger because more data are required to be available for multiple cores; and hence, storage, which is used to store the to-be-processed data, gets greater, too. Consequently, to mapping on algorithm onto multi-core platform, not only algorithm should be re-examined but also

the characteristics of platform should be investigated to assist in developing algorithm. Moreover, emerging applications, such as gene analysis, social network analysis, data mining, or other applications related to Internet of Things (IoTs), big data, etc., are encountered the problem that volume of to-be-processed data and number of tasks are dramatically scaled up to an extremely high level.

To achieve better performance in both algorithm and architecture, G. G. Lee *et al.* [1] introduce a new design paradigm, Algorithm/Architecture Co-exploration (AAC), to resolve the design difficulties from both perspectives of algorithm and architecture. To maximize the performance of multi-core platform, the issue of efficient resource management is discussed in this paper; and the resources discussed here include computation, storage, and data transfer. Several researchers had studied the mechanisms that are used to balance the workload of each processor, minimize bandwidth, and reduce storage requirement.

To reduce bandwidth requirements, S. Wuytack *et al.* [2] present the method of flow graph balancing to minimize the required bandwidth prior to architecture design. In their work, conflict graph is used to present the occurrence of memory accessing for each processing. C.-Y. Chen *et al.* [3] present data reuse scheme for motion estimation. In their work, various data flows are utilized for exploring design space and corresponding ratio of data reusing and memory increasing are reported. However, this work only targets at a specific application and the proposed scheme on data reuse cannot be dynamic on-the-fly.

E. Kijispongse *et al.* [4] present the method to balance the workload on heterogeneous platform via a task pool model. This model distributes tasks to each processor according to processors' capability and workload; then, upon any one of processors completes its job, a new task would be assigned to this processor to avoid the idle processor. This strategy could be considered as a first-fit manner to manage the available resources; however, it is also a greedy approach and hence, it might be trapped in a sub-optimal solution. I. Anagnostopoulos *et al.* [5] apply a divide and conquer approach to dynamically distribute resources. They use local authorization concept instead of centralized control since they thought there are some defects in centralized control. However, their method might focus on the optimization at local region since the authority of

resource management was distributed to local processors. In this case, a global optimization might be ignored since their method just considers the optimal solution determined by local processors. J. Zhong *et al.* [6] split GPU kernel into fine-grained kernelets to fulfill the utilization of each kernelet and consequently, the performance could be enhanced; furthermore, the dynamic schedule method is developed according to the availability data of GPU; if all required data of current task has been loaded on GPU, priority of this task would be high. However, their method might not be capable of maximizing the utilization of resource since they consider memory issue prior to other factors.

In this paper, we propose the dynamical management on data transfer rate and storage requirement; in addition, in cooperation with our previous work on degree of parallelism quantification, we present a comprehensive exploration of design space to intelligently handle resources in multi-core platforms. Furthermore, since our approach is based on data flow model and corresponding matrix representation, the proposed method would be general for different multi-core platforms, including homogeneous and heterogeneous multi-core platforms.

This paper is organized as follows. Section II briefly describes design methodology of algorithm/architecture co-exploration and introduces the presented method to quantify complexity metrics that are used to explore design space systematically through data flow graph. Subsequently, section III presents a case study of mapping algorithm onto multi-core platform via the presented method and the experimental results are demonstrated at section IV. Consequently, this paper is concluded at section V.

II. PROPOSED METHOD OF MAPPING VISUAL SIGNAL PROCESSING ONTO MULTI-CORE PLATFORM

To seamlessly map algorithms to architectures, AAC successfully connects algorithmic design and architectural design via data flow model. Intrinsic complexity that is transparent to algorithm and architecture would be quantified through data flow model and used to estimate resource cost in the future if the desired algorithm was adopted. Therefore, G. G. Lee *et al.* apply the complexity, including number of operations, degree of parallelism, data storage requirement, and data transfer rate, to characterize the algorithmic complexity and AAC has been proven to grant great benefit for both algorithm and architecture design [7]. The proposed design method is developed based on the combination of data flow model and linear algebra. In the following subsections, we present the proposed method in connection with data flow model to map algorithm to architecture based on the quantification on data transfer rate and data storage requirement.

A. Data Flow Model

Data flow is a kind of computation models that concurrently reveals algorithmic behavior and architectural characteristics and hence data flow could be considered as a bridge which connects algorithm and architecture. Therefore, one representation of data flow models, Data Flow Graph

(DFG) which is derived from signal flow graph [8], is applied to systematically explore design space. In a DFG, which is composed of vertices and edges, a vertex denotes the computation tasks or data feeders/receivers and an edge present the data or information communication between tasks, i.e., vertices. Furthermore, DFG could model an algorithm in different abstraction levels, e.g. when considering a vertex as one module for video decoding and edges in this DFG could be video bitstream; in this case, the abstraction level of DFG is high; in contrast, if a vertex presents one addition operation and an edge denotes transfer of one data sample, then DFG represents this algorithm in low level of abstraction. In addition, when assigning the weights on edges in DFG, the amount of data transfer could be also modeled in DFG.

To systematically extract the information embedded in graph, matrix representation is commonly used to represent a DFG. For instance, adjacent matrix introduces the connections among vertices and Laplacian matrix also displays the connectivity embedded in graph. These matrix representations are usually in behalf of undirected graph; however, in the study of data transfer of visual signal processing, data causality is also a significant information that should be retained in matrix representation. Hence, a dependency matrix conveying data causality of a directed or undirected graph is required, and its mathematical expression is illustrated as (1). We will use the dependency matrix to develop our method.

$$\mathbf{M}(i, j) = \begin{cases} -1, & \text{if vertex } v_j \text{ is the tail of edge } e_i \\ 1, & \text{if vertex } v_j \text{ is the head of edge } e_i \\ 0, & \text{otherwise} \end{cases} \quad (1)$$

B. Complexity Metrics Quantification via Data Flow Graph

G. G. Lee *et al.* [9] introduce a linear algebra approach to systematically quantify the degree of parallelism embedded in DFGs; furthermore, a multi-grain concept has been introduced, too. Lee's method uses Laplacian matrix to quantify the number of operation sets could be computed in parallel; moreover, the null space spanned by the dependency matrix is also capable of quantifying the degree of parallelism. In addition, the components of each independent operation set also could be indicated by the elements in null vectors.

To further explore design space from different perspectives, we propose the method to quantify data transfer rate and corresponding data storage requirement via dependency matrix. To assess data transfer rate in DFG, edge cut is applied since edge cut is a cut that results in a connected DFG into several

Fig. 1. A simple DFG and an edge cut separate vertices into two sides

$$\mathbf{M} = \begin{matrix} & v_1 & v_2 & v_3 & v_4 \\ \begin{matrix} e_1 \\ e_2 \\ e_3 \end{matrix} & \begin{bmatrix} 1 & 0 & -1 & 0 \\ 0 & 1 & -1 & 0 \\ 0 & 0 & 1 & -1 \end{bmatrix} \end{matrix}, \mathbf{x} = \begin{bmatrix} 1 \\ -1 \\ -1 \\ 1 \end{bmatrix} \Rightarrow \mathbf{M}\mathbf{x} = \begin{bmatrix} 2 \\ 0 \\ -2 \end{bmatrix} \quad (2)$$

$$\text{Amount of data transfer} = \frac{1}{2} \sum_{i=1}^N |\mathbf{M}\mathbf{x}(i)| = 2$$

disconnected sub-DFGs by removing the edges in this cut. Therefore, the size of edge cut (or number of edges in this cut) could be used to estimate the amount of data would be transferred among sub-DFGs due to the fact that data should be sent or received (via edges) by tasks (vertices). On the other hand, the behavior of edge cut in DFG is equivalent to applying an indicator vector $\mathbf{x}$ that separates vertices in DFG into two sides for dependency matrix, $\mathbf{M}$. For example, a simple DFG of an average filter is shown in Fig. 1, the indicator vector $\mathbf{x}$ of corresponding edge cut is $[1, -1, -1, 1]^T$, then this edge cut separates v_1 and v_4 into one group and v_2 and v_3 belong to the other group. (The vertices at the side with more input data would be set as 1.) Furthermore, by computing $\mathbf{M}\mathbf{x}$, the characteristics of edges in DFG would be revealed. In this example, $\mathbf{M}\mathbf{x}$ is $[2, 0, -2]^T$ and there are three type of edges that are introduced by $\mathbf{M}\mathbf{x}$, including in-edge-cut (value in $\mathbf{M}\mathbf{x}$ is positive, e_1), out-edge-cut (value in $\mathbf{M}\mathbf{x}$ is negative, e_3), non-edge-cut (value in $\mathbf{M}\mathbf{x}$ is zero, e_2). According to $\mathbf{M}\mathbf{x}$, the amount of data transfer was equal to the half of the summation of all absolute values in $\mathbf{M}\mathbf{x}$. Corresponding dependency matrix ($\mathbf{M}$), indicator vector ($\mathbf{x}$), characteristics of edges ($\mathbf{M}\mathbf{x}$), and amount of data transfer are depicted in (2). Therefore, $\mathbf{M}\mathbf{x}$ clearly presents the number of edges crossed by this edge cut and hence corresponding data transfer rate could be systematically quantified due to the fact that data transaction are occurred on the edges in DFG. Consequently, the amount of data transfer of this edge cut is 2.

In general, DFG presents a process for computing one Data Granularity (DG) of an algorithm and then this DFG is applied iteratively until all to-be-computed DGs are accomplished; for example, we might build up a DFG for block-based Motion Estimation (ME) and hence one DG is one block; as a consequence, to achieve ME for one frame, this DFG is used for all DGs in one frame. Therefore, when we combine consecutive processes for different DGs into one DFG, there are some data could be concurrently used for two DGs, i.e., the data could be reused and the amount of data transfer would be reduced. For example, two consecutive processes of Fig. 1 is

Fig. 2. A DFG composed of two consecutive DGs

$$\mathbf{M} = \begin{matrix} & v_1 & v_2 & v_3 & v_4 & v_5 & v_6 & v_7 \\ \begin{matrix} e_1 \\ e_2 \\ e_3 \\ e_4 \\ e_5 \\ e_6 \end{matrix} & \begin{bmatrix} 1 & 0 & 0 & -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & -1 \end{bmatrix} \end{matrix}, \mathbf{x} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ -1 \\ -1 \\ -1 \\ -1 \end{bmatrix} \Rightarrow \mathbf{M}\mathbf{x} = \begin{bmatrix} 2 \\ 2 \\ 2 \\ 2 \\ 0 \\ 0 \end{bmatrix} \quad (3)$$

$$\text{Amount of data transfer} = \frac{1}{2} \sum_{i=1}^N |\mathbf{M}\mathbf{x}(i)| = 4$$

presented in Fig. 2, and another edge cut crosses all input data. Its corresponding dependency matrix $\mathbf{M}$, indicator vector $\mathbf{x}$, and characteristics of edges $\mathbf{M}\mathbf{x}$ are illustrated in (3). We could find out the corresponding amount of data transfer would be four when directly computing absolute summation over $\mathbf{M}\mathbf{x}$. However, it is clear that if v_2 could be reused for both DG_{N-1} and DG_N , when DG_N denotes the DFG computes the n -th DG, the amount of data transfer would be reduced from four to three but one extra storage size is required. Here we present a systematic approach to indicate how many data could be reused and which data would be reused through the dependency matrix.

Dependency matrix concurrently conveys the direction of data transaction and the dependency of tasks, so it can indicate that the location where data are transacted through the determined edge cut. To clearly explain the proposed method, here we define the symbols used hereafter; an edge characteristic vector $\mathbf{y}$, equals to $(1/2)\mathbf{M}\mathbf{x}$ and one operator, $\cap$, an element-wise operation which reserves the elements with the same sign and set others as zero. Therefore, through vector $\mathbf{y}$ and operator $\cap$, the reusable data could be indicated. $\cap$ operator remains the elements that exchange data in this edge cut and hence we could create a new matrix $\mathbf{M}'$ whose i -th column is $\text{col}_i(\mathbf{M}) \cap \mathbf{y}$, where $\text{col}_i(\mathbf{M})$ is i -th column of matrix $\mathbf{M}$. After that, as shown in (4), for each column in $\mathbf{M}'$, a maximum operator would be performed on the elements with positive values to calculate maximum numbers of data should be sent from this vertex; on the other hand, for each column in $\mathbf{M}'$, the remaining elements with negative values would be summed up to be the amount of output data. Hence, the amount of data transfer with data reuse could be quantified systematically. Furthermore, when we merge more DGs into

$$\begin{aligned} & \text{Data could be reused} \\ & \mathbf{y} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 0 \\ 0 \end{bmatrix} \quad \mathbf{M}' = \begin{matrix} & v_1 & v_2 & v_3 & v_4 & v_5 & v_6 & v_7 \\ \begin{matrix} e_1 \\ e_2 \\ e_3 \\ e_4 \\ e_5 \\ e_6 \end{matrix} & \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix} \end{matrix} \quad (4) \\ & |1| + |1| + |1| + 0 + 0 + 0 + 0 = 3 \end{aligned}$$

$$\text{Amount of data transfer} = 3 \text{ (} v_2 \text{ is reused)}$$

one DFG, we have the potential to reduce more data transfer; however, storage requirement would also be increased if more DGs are considered at the same time. As a result, we have a systematic manner to explore design space in terms of amount of data transfer and storage requirement.

In the example, unweighted graph is applied; however, our method could be extended to weighted graph and the weight on edge denotes the proportion of data within DFG; thus, we could also indicate where is the bottleneck of data transfer and then resolve the critical part. Moreover, benefiting from the advantages of DFG, we could encapsulate the vertices into super-vertex to illustrate DFG in different levels of abstract; thus, amount of data transfer and storage requirement could be assessed at multi-grain sizes. Consequently, three major factors, including degree of parallelism, data storage requirement, and data transfer rate, in designing a system could be quantified in conjunction with the quantification on degree of parallelism through dependency matrix.

III. CASE STUDY

To evaluate the proposed design methodology, we map our previous work, Motion-Compensated Frame Rate Up-Converter (MC-FRUC) [10] onto the targeted multi-core platform, IBM Cell Broadband Engine (Cell BE) in PlayStation 3 (PS 3) [11].

A. Introduction to Selected Visual Signal Processing Algorithm and Targeted Multi-core Platform

MC-FRUC was an emerging technology that is used to enhance visual quality in temporal domain by interpolating virtual frames between the original frames. Visual signal processing algorithm which uses motion information is usually bandwidth-intensive and computation-intensive; and hence it will be suitable to be case study to evaluate our proposed method. MC-FRUC, whose block diagram is displayed in Fig. 3, hierarchically performs block-size ME, including Coarse ME (CME) and Refined ME (RME), to accurately extract Motion Vectors (MVs). The CME uses a spatial-temporal recursive ME to accurately track the motion trajectory of object; however, there is highly dependency between the processing of each coarse-grain block due to the fact that MVs are recursively updated by spatial neighboring blocks and temporal blocks. On the other hand, in algorithmic consideration, fixed block-size ME would suffer from the inaccurate MVs at objects boundaries; hence, the RME uses fine-grain block to refine coarse-grain MVs by re-examining neighboring coarse-grain MVs; the procedure of each fine-grain block is independent since one fine-grain block would use four coarse-grain blocks to refine or smooth MVs. Subsequently, upon having fine-grained MVs, Multiple Block

Fig. 3. Block diagram of MC-FRUC system

Fig. 4. Architecture of IBM Cell Broadband Engine

Candidates (MBC) derivation would indicate several blocks located at two consecutive frames be the candidates of current to-be-interpolated block based on the motion trajectory of fine-grained MVs in both forward and backward directions. MBC resolves the problems in unilateral MVs, such as motion holes and motion block overlapped, by referencing neighboring block candidates. Subsequently, Motion Compensated Interpolation (MCI) performs pixel-wise filter among block candidates to fill out the to-be-interpolated frame.

IBM Cell BE in PS 3 is a multi-core platform that features high-bandwidth and high-performance processor and its architecture is depicted in Fig. 4. There are six Synergistic Processing Elements (SPEs) and one Power Processor Element (PPE) in the model of PS 3. PPE plays the role as central controller which of assigning tasks to SPEs as a central brain to communicate all elements on IBM Cell BE. SPE is the major computational processor which supports Single-Instruction-Multiple-Data (SIMD) architecture that could computed multiple data within one instruction to speed up the computation as compared to scalar computation, and there are 256 Kbytes as local storage in each SPE. The element interconnect bus can handle the data transfer up to theoretically 96 bytes per cycle among all elements and bandwidth of each individual element is up to 16 bytes per cycle. Therefore, our design methodology will apply all advantages in IBM Cell BE to increase the performance, including thread-level parallelization (SPE) and data-level parallelization (SIMD). In addition, we also resolve the issue on data transfer by the proposed method to reduce the cycles in transferring data for multiple SPEs. As a result, all complexity metrics would be considered simultaneously to determine the optimal solution.

B. Mapping MC-FRUC onto IBM Cell BE via Complexity Metrics Quantification

We span design space from three perspectives, including degree of parallelism at thread-level, amount of data transfer, and storage size by varying data granularity. To explore design space, we establish the DFG to model MC-FURC. Take CME as an example, its DFG is illustrated in Fig. 5 (a). Every vertex is one task that computes ME of one coarse-grain block and edges denote the referenced spatial neighboring MVs.

Fig. 5. DFG of CME and level of dependency of each vertex

We could apply the methodology developed by our previous work [9] to quantify degree of parallelism at multi-grain granularity according to various level of data dependency. When data granularity is larger than one task, it is hard to exploit the degree of parallelism due to the fact all tasks are connected sequentially; in contrast, when we narrow down data granularity into one task, the parallelization possibility of CME is increased. In Fig. 5 (b), level of data dependency are listed at each vertex and the dash lines split DFG according to level of data dependency; then, vertices in identical level of data dependency are independent. Hence, in accordance with the quantification method developed by our previous work, the degree of parallelism can be systematically quantified via dependency matrix of DFG. Consequently, the maximum degree of parallelism of CME is dynamic according to level of data dependency. In the beginning, degree of parallelism is 1 and then incremented to the bound of available processors, i.e., six in this case study. On the other hand, RME, MBC derivation, and MCI are also applied the same approach to exploit the degree of parallelism to maximize the performance on thread-level. We also use SIMD to enhance performance at data-level; however, we only apply SIMD for partial operations, such as similarity measurement in CME and RME or coarse-grain MVs refinement in RME, trajectory tracking for multiple blocks in MBC derivation, and multiple interpolations in MCI, due to the fact that we focus on thread-level parallelization in this paper.

To reduce the data transaction between storages, we utilize the proposed method of data transfer analysis on the transfer between local storage and external storage. We expand DFG over time to explore the data reusability; and then indicate that the data would be reused for previous DG and current DG. As a result, we could systematically determine the suitable DG with highest data reusability; then, we select this data granularity for our architecture. Although the number of data reuse is deterministic in this example due to the regular DFG of MC-FRUC; however, the presented method could also dynamically determine the ratio of data reuse when data flow of targeted algorithm is irregular or dynamic since the proposed method just depends on DFG. Take MBC derivation and MCI as an example, MBC derivation uses fine-grained MVs to derive MBC according to motion trajectory for MCI. Hence, we investigate the DFG of MBC derivation for computing consecutive DGs, DG_{N-1} and DG_N , in Fig. 6 and the weights in DFG denotes the ratio of data size with respect to the maximum one. From the figure, a part of fine-grained MVs and reference pixels would be used for both DG_{N-1} and DG_N ;

Fig. 6. DFG of MBC derivation for computing DG_{N-1} and DG_N

that is, by using the proposed method, we could indicate how many data could be reused under the size of current DG and which data would be reused, then these data would be kept to avoid unnecessary data transaction from external storage. Then, dependency matrix of the DFG composing of DG_{N-1} and DG_N is built to systematically achieve smaller data transfer rate; in the implementation, we encapsulate 16×16 pixels as one vertex and 16 fine-grained MVs as one vertex in DFG to avoid huge dependency matrix. We utilize the proposed method for CME, RME, MBC derivation, and MCI, respectively, to significantly reduce data transfer rate with acceptable storage requirement.

IV. EXPERIMENTAL RESULTS

We achieve the realization of mapping MC-FRUC onto IBM Cell BE based on our previous work, degree of parallelism quantification, and then on top of that, we integrate the proposed method in this paper to reduce the cycles of data transfer between local storage and main memory. Three scenarios at various DGs are provided in this case study for performance evaluation, including single-core and multi-core implementation, and multi-core realization with data reuse scheme; in addition, data-level parallelization (SIMD) are applied for these three scenarios with the identical mechanism. The parameters of MC-FRUC are illustrated in Table I. DG size is defined in horizontal and vertical direction, respectively, and the unit of one DG is 16×16 block, e.g., if $DG_H = 40$ and $DG_V = 2$, then 80 16×16 blocks would be one DG. Due to the issue of memory address alignment, DG size in horizontal direction should be the multiples of eight. Table II displays the cycle requirement per frame, including computation and data transfer, and the ratio of speed up for different mapping methods at various DGs. X SPE denotes the number of SPEs is used and w/ DR or w/o DR stand for the results with or without data reuse scheme, respectively. In addition, the algorithmic performance did not change when we map the MC-FURC on the IBM Cell BE since we explore the embedded algorithmic characteristics via DFG rather than modifying the algorithm.

From Table II, larger DG is fast than smaller DG since larger DG shares more data, including intermediate and input data, to reduce the cycles of computation and data transfer. Therefore, when six SPEs are used, total cycle requirement is reduced dramatically and the speed of six-SPE case increases about 5.75x as compared to single-SPE configuration at each

TABLE I. PARAMETERS SETTING OF MC-FRUC

Resolution of test video	1920×1088
Coarse-grain block-size	16×16
Fine-grain block-size	4×4
Search range	H: [-64, +64]; V: [-32, +32]
{DG_H, DG_V}	{8, 1}, {8, 2}, {24, 1}, {24, 2}, {40, 1}, {40, 2}

TABLE II. PERFORMANCE OF DIFFERENT MAPPING METHODS

Data granularity		Cycles per frame ($\times 10^7$ cycles) (Computation + Data transfer)			Speedup ratio	
DG_H	DG_V	(a) 1 SPE w/o DR	(b) 6 SPEs w/o DR	(c) 6 SPEs w/ DR	(b) vs. (a)	(c) vs. (a)
8	1	6.61	1.15	0.92	5.75	7.18
8	2	5.97	1.02	0.81	5.85	7.38
24	1	5.60	0.97	0.74	5.75	7.52
24	2	5.32	0.92	0.70	5.81	7.62
40	1	5.46	0.96	0.71	5.68	7.65
40	2	5.21	0.92	0.68	5.68	7.70
Ave.					5.75x	7.51x

DG on average. Fig. 7 displays the detail of cycles of computation and data transfer when DG_H is 40 and DG_V is 2 at different numbers of SPEs. Before applying the proposed method, the cycles of data transfer are occupied about 55% of total cycles; thus, data transaction is the bottleneck for performance enhancement. Consequently, the cycles of data transfer were decreased about 46% at this DG on average after applying the proposed method. Hence, from the last column of Table II, we could achieve 7.51x fast in comparison with single-core scenario without data reuse on average. In addition, our method achieves similar reduction ratio at different DGs since our method depends on DFG which is invariant from platform configurations. Furthermore, when comparing the case of single SPE without data reuse at DG_H = 8 and DG_V = 1 to the case of six SPEs with data reuse at DG_H = 40 and DG_V = 2, the speedup ratio could be up to 9.76x. As a consequence, our mapping method could efficiently explore design space to find a better solution.

V. CONCLUSION

The proposed analysis method on data transfer rate and data storage requirement could efficiently assist designers in exploring design space; hence, we resolve the important issue, data communication, when mapping algorithm onto multi-core platform. The presented method could search the feasible solutions from the global viewpoint and would not be biased by platforms; in addition, by integrating the multi-grain concept, this method can measure the data transactions at different levels of abstract. The experimental results show that our method can successfully enhance the system performance. Without data reuse scheme, the performance of six SPEs could achieve 5.75x on average as compared to the performance of single SPE. On the other hand, after applying data reuse scheme of proposed method, the speed can be increased to 7.51x on average in comparison with the single-SPE scenario without data reuse. Experimental results justify our design methodology is efficiently to manage resource in reducing unnecessary data communication; hence, the performance of multi-core platform could be enhanced. Furthermore, power consumption could be reduced since the data transaction

Fig. 7. Performance evaluation on IBM Cell BE with different number of SPEs and data reuse scheme.

between local storage and external storage is usually power-intensive. Consequently, by cooperating the previous work on degree of parallelism quantification, a comprehensive analysis method for mapping algorithm onto multi-core platform in terms of three important aspects is presented in this paper.

REFERENCE

- [1] G. G. Lee, Y.-K. Chen, M. Mattavelli, and E. S. Jang, "Algorithm/Architecture Co-Exploration of Visual Computing on Emergent Platforms: Overview and Future Prospects," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 19, Iss. 11 pp. 1576-1587, 2009.
- [2] S. Wuytack, F. Catthoor, G. de Jong, B. Lin, and H. De Man, "Flow graph balancing for minimizing the required memory bandwidth," in *9th International Symposium on System Synthesis, 1996. Proceedings.*, 1996, pp. 127-132.
- [3] C.-Y. Chen, C.-T. Huang, Y.-H. Chen, and L.-G. Chen, "Level C+ data reuse scheme for motion estimation with corresponding coding orders," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 16, Iss. 4 pp. 553-558, 2006.
- [4] E. Kijsipongseand S. U-ruekolan, "On load balancing of hybrid OpenCL/Global Arrays applications on heterogeneous platforms," in *2012 9th International Conference on Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology (ECTI-CON)*, 2012, pp. 1-4.
- [5] I. Anagnostopoulos, A. Bartzas, G. Kathareios, and D. Soudris, "A divide and conquer based distributed run-time mapping methodology for many-core platforms," in *2012 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2012, pp. 111-116.
- [6] J. Zhong and B. He, "Kernelet: High-Throughput GPU Kernel Executions with Dynamic Slicing and Scheduling," *IEEE Transactions on Parallel and Distributed Systems*, vol. 25, Iss. 6 pp. 1522-1532, 2014.
- [7] G. G. Lee, M.-J. Wang, H.-Y. Lin, W.-C. Su, and B.-Y. Lin, "Algorithm/Architecture Co-Design of 3-D Spatio-Temporal Motion Estimation for Video Coding," *IEEE Transactions on Multimedia*, vol. 9, Iss. 3 pp. 455-465, 2007.
- [8] A.V. Oppenheim and R.W. Schaefer, *Discrete-Time Signal Processing*. Prentice-Hall, 1989.
- [9] G. G. Lee, H.-Y. Lin, C.-F. Chen, and T.-Y. Huang, "Quantifying Intrinsic Parallelism Using Linear Algebra for Algorithm/Architecture Coexploration," *IEEE Transactions on Parallel and Distributed Systems*, vol. 23, Iss. 5, pp. 944-957, 2012.
- [10] G. G. Lee, C. F. Chen, C. J. Hsiao, and J. C. Wu, "Bi-Directional Trajectory Tracking With Variable Block-Size Motion Estimation for Frame Rate Up-Converter," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 4, pp. 29-42, 2014.
- [11] J. A. Kahle, M. N. Day, H. P. Hofstee, C. R. Johns, T. R. Maeurer, and D. Shippy, "Introduction to the Cell multiprocessor," *IBM Journal of Research and Development*, vol. 49, pp. 589-604, 2005.