A Scalable Communication-Aware Compilation Flow for Programmable Accelerators

Jason Cong, Hui Huang and Mohammad Ali Ghodrat
Computer Science Department
University of California, Los Angeles
Los Angeles, CA 90095, USA
{cong, huihuang, ghodrat}@cs.ucla.edu

***Abstract*—Programmable accelerators (*PA*) are receiving increased attention in domain-specific architecture designs to provide more general support for customization. In a PA-rich system, computational kernels are compiled into predefined PA templates and dynamically mapped to real PAs at runtime. This imposes a demanding challenge on the compiler side – that is, how to generate high-quality PA mapping code. Another important concern is the communication cost among PAs: if not handled properly at compile time, data transfers among tens or hundreds of accelerators in a PA-rich system will limit the overall performance gain. In this paper we present an efficient PA compilation flow, which is scalable for mapping large computation kernels into PA-rich architectures. Communication overhead is modeled and optimized in the proposed flow to reduce runtime data transfers among accelerators. Experimental results show that for 12 computation-intensive standard benchmarks, the proposed approach significantly improves compilation scalability, mapping quality and overall communication cost compared to state-of-art PA compilation approaches. We also evaluate the proposed flow on a recently developed PA-rich platform [1]; the final performance gain is improved by** 49.5% **on average.**

I. Introduction

Customization is an appealing approach to increasing performance-power efficiency, which is one of the primary design concerns in the age of dark silicon. A recent industry trend to address this is the design and integration of fixed-function computation accelerators on the die, targeting application domains demanding high performance and power-efficient execution. Graphics, media, audio, and imaging processing are example domains for this [2], [3]. Although fixed-function accelerators can be designed to provide the best performance/energy efficiency for a specific domain, they suffer from poor flexibility, and hence are not suitable for the domains with constantly changing use protocols.

To address this problem, the programmable accelerator (*PA*) has been proposed to enable varying degrees of customization in accelerator-rich systems [1], [4], [5], [6]. In a standard PA architecture, a programmable accelerator template is implemented in each PA unit to support a selected set of computation tasks with reasonable hardware design costs. The entire predefined PA template may support a relatively complex computation task, while it can also be reconfigured dynamically to perform a set of simpler but more general subtasks. Therefore, each accelerator unit in a PA-rich system can be customized for computation tasks with different granularity, which enables efficient switching among varying degrees of customization at runtime. For example, the PA template used in [4] can be configured by hardware control signals at runtime to support all the 4-input 2-output computation patterns with dependency depth less than 5.

On the other hand, the emergence of PA-based designs imposes a demanding challenge on the compiler side to generate high-quality PA mapping code and efficiently utilize the programmable execution units in a PA-rich architecture. Considering that the number of PA candidates grows exponentially with the size of the input data flow graph and PA template, the PA mapping, which itself is NP-complete [7], may become intractable even for medium-size DFG blocks and cannot generate desirable mapping solutions. This leads to the PA compilation scalability problem, which has become a major challenge that existing PA compilation work struggles to resolve.

Another important problem which is crucial to PA execution efficiency is the communication overhead among accelerators. A computing platform equipped with multiple (or even a sea of) accelerators are usually connected together via bus or NoC for better scalability [1], [8], [9], [10], [11], [12], [13]. The computation capacity of each accelerator has its upper-bound, and a large task has to be composed by multiple PAs. There will be communication among these PAs through the system interconnects, and this incurs extra performance and energy overhead. Note that a PA can compute >100x faster than a general-purpose processor [14] and usually needs to consume a large amount of data every clock cycle. This indicates that the communication among PAs will incur much higher traffic on the system interconnects than the communication among multiprocessors. The high traffic brought by accelerators can easily exceed the bandwidth that the interconnects can afford, and it correspondingly limits the performance gain of accelerators. To the best of the authors' knowledge, existing PA compilation work mainly focuses on accelerating computations and has ignored the possible effect of communication patterns on the overall performance.

The contributions of this work include:

(i) A scalable PA compilation flow to efficiently utilize the available on-chip accelerator resources and support a higher level of parallel execution. The PA mapping quality has been improved by 23.8% and 32.5% on average, compared to

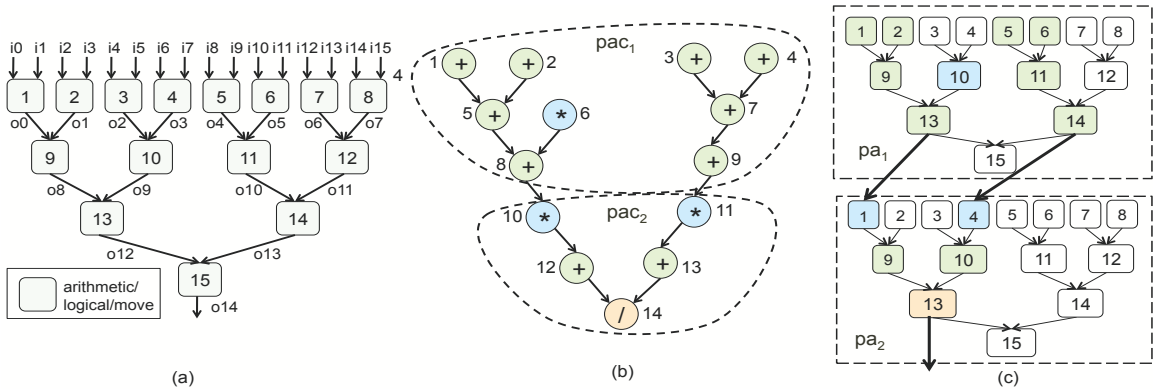

Fig. 1. (a) Sample PA template. (b) One PA mapping solution for *rician-denoise*. (c) Runtime PA configuration of (b).

representative previous work [7] and [5].

(ii) Efficient mapping size reduction techniques to improve the mapping scalability. Compared to the PA compilation approaches proposed in [7] and [5], our approach achieves at least 2X improvement on the overall compilation time.

(iii) Communication cost modeling and optimization to achieve better runtime communication behavior. The results show that the runtime data transfer is reduced by up to 20% compared to the compilation solution without considering the communication pattern.

The paper is organized as follows: Section II uses a real-life application to illustrate the PA compilation flow; the effect of communication patterns is discussed in Section III. Section IV introduces the proposed PA compilation approach, followed by Section V which reports experimental results. The representative previous PA compilation work is discussed in Section VI.

II. PA Compilation Example

In this section we use a real-life medical imaging application *Rician-denoise* [15] to illustrate the PA compilation results on a sample PA template.

As shown in Figure 1(a), the sample PA template is organized as a 4-level binary-tree structure, which is similar to the *POLY*16 PA template proposed in [1]. Each template node can either perform arithmetic operations or forward the input value to its output. The interconnect between two levels is designed in such a way that each data can be transferred to the next level or be directly accessed as a PA output.

Figure 1(b) shows the simplified data flow graph of the kernel loop in *Rician-denoise*, which contains 14 arithmetic operation nodes (*add*, *multiply* and *divide*). As shown in Figure 1(b), the entire DFG is covered by two PA candidates pac_1 and pac_2. Let's first look at the connected candidate pac_2: it is isomorphic to subgraph {1, 4, 9, 10, 13} of the sample PA template and therefore can be identified as a PA candidate. The corresponding runtime PA configuration is shown in Figure 1(c). A 15-node PA unit pa_2 is dynamically reconfigured to perform the 5 computations in pac_2. The remaining nodes in pa_2 will be bypassed and will not perform real computations in this mapping.

Compared to the connected-only case, PA compilation with disjoint PA candidates takes better advantage of the instruction-level parallelism inside a PA template. For example, pac_1 in Figure 1(b) contains two connected subgraphs. Since the two subgraphs are both PA-executable and can be mapped to a single template at the same time, pac_1 itself is also a PA candidate. As shown in Figure 1(c), nodes 1, 2, 5, 6, 8 in pac_1 are mapped to template nodes 1, 2, 9, 10, 13; nodes 3, 4, 7, 9 are mapped to template nodes 5, 6, 11, 14. At runtime, only one PA unit is needed to execute this disjoint PA candidate.

III. Impact on Communication

In this section we use a recent PA-rich platform CHARM [1] to analyze the impact of different PA mapping solutions on the runtime communication overhead. The overall architecture of CHARM consists of cores, L2 cache banks, memory controllers, PA islands and an *accelerator block composer* (ABC). PAs are building blocks of the application-specific accelerators. PA islands consist of a series of PAs which share a scratchpad memory and a DMA-controller. The ABC is responsible for dynamically composing PAs to create coarser-grained accelerators (loosely coupled accelerators or LCAs).

There are two types of communication in CHARM:

Output-oriented communication. As shown in Figure 1(c), pa_1 produces two output data streams from nodes 13 and 14, which are consumed by pa_2 as inputs. If pa_1 and pa_2 are allocated to the same PA island at runtime, the communication between the two PAs includes two SPM write accesses by pa_1 and two corresponding SPM read accesses by pa_2. If they are allocated to different PA islands, additional NoC traffic will be incurred in order to transfer data between PA islands. The amount of data transfer for intermediate results can be modeled as the total number of communication edges across PA units. (Note that when an output of one PA is fed into two different nodes in another PA as inputs, it will only be counted once even though there are two communication edges). The communication between PAs is through the scratchpad memories. Depending on which island a PA is assigned to, there are two types of communications:

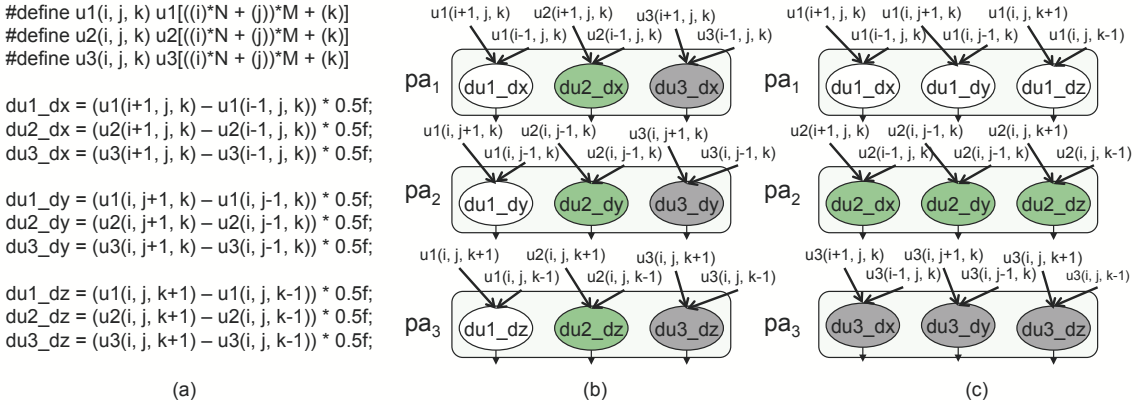

Fig. 2. (a) Kernel codes of *registration*. (b) PA mapping solution I. (c) PA mapping solution II.

intra-island communication when the two PAs are on the same island, and inter-island communication when the two PAs are assigned to two different islands. Each case has its own overhead source: in the intra-island case, the overhead comes from accessing the local scratchpad memory (contention on memory ports); in the inter-island case the overhead is due to communication through NoC.

Input-oriented communication. Another source of communication is the input data transfer into the PA system. Figure 2(a) shows a kernel code piece in another medical imaging application called *registration* [15]. Figure 2(b) and (c) represent two mapping solutions containing the same number of PA units. As we can see from Figure 2(b), all of the three PA units require data arrays $u1$, $u2$ and $u3$ as inputs. In this case, one data element may be requested by multiple PA units since data reuse exists among their inputs. If pa_1, pa_2 and pa_3 are assigned to different PA islands at runtime, the DMA controller needs to send the same data set to multiple destinations, which will introduce additional data transfer overhead as well as energy consumption. Another possible mapping solution is shown in Figure 2(c), in which the operation nodes sharing the same input data set are packed into one PA unit; thus, the duplicate data transfer overhead in solution I can be reduced accordingly. Note that in this example, both of the mapping solutions consume the same amount of PA units, and thus will be considered as equivalent solutions in the existing PA mapping work [7] and [5]. However, the communication overhead in solution II is much less than solution I, as explained above.

IV. Scalable Communication-Aware PA Compilation Flow

A. Problem Formulation

To formally convey the proposed maximal PA compilation problem, in this section we first introduce the necessary definitions and problem formulation.

Definition 1: Given a PA template $T < V_T, E_T >$ and an input data flow graph $G < V,\ E >$, a subgraph $G^* \subseteq G$ is called a **PA candidate**, if there exists a data path $T^* \subseteq T$, which is isomorphic to G^*.

Definition 2: A PA candidate $G^* < V^*,\ E^* >$ is called a **maximal PA candidate**, if $\forall v_i \in V - V^*$, the expanded subgraph $G^+ < V^* \cup \{v_i\},\ E^* >$ is not a PA candidate.

For example, PA candidate pac_2 in Figure 1(b) is not a maximal PA candidate since it can be expanded by adding nodes 8 or 9, and the expanded graph is still a PA candidate. On the other hand, pac_1 is a maximal PA candidate, since none of its expanded subgraph can be mapped to the PA template in 1(a) without violating the data dependency constraint.

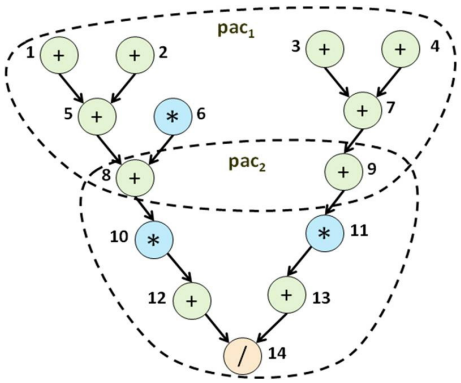

Fig. 3. Two compatible maximal PA candidates.

Note that PA candidates may overlap each other at a certain set of nodes. If it is possible to distribute each overlapping node to exactly one maximal PA candidate, and the transformed subgraphs are still PA candidates, the overlapping candidates are called *compatible PA candidates*. Figure 3 shows two maximal PA candidates pac_1 and pac_2, which overlap each other at nodes 8 and 9. They are compatible with each other since after removing nodes 8 and 9 from pac_2, the remaining subgraph of pac_2 is still a PA candidate.

Based on the concept of a maximal PA candidate, we propose a maximal PA compilation flow, which can be decomposed into the two sub-problems:

Problem 1: **Maximal PA candidate identification**. Given an input data flow graph G and PA template T, identify all the maximal PA candidates in G, which can be executed on PAs.

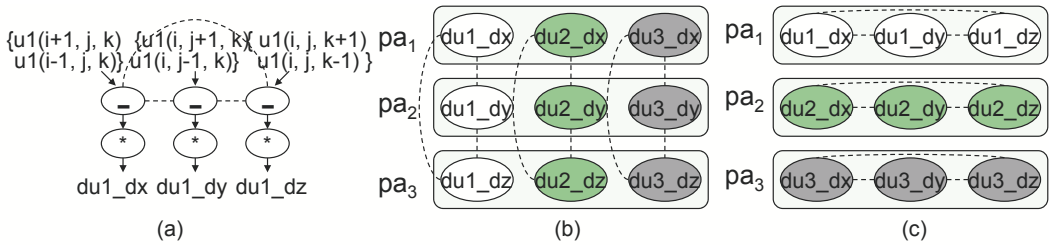

Fig. 4. (a) Expanded DFG of Figure 2(a). (b) Communication pattern of PA mapping solution I in Figure 2(b). (c) Communication pattern of PA mapping solution II in Figure 2(c).

Problem 2: **Maximal PA mapping**. Given an input data flow graph G and a set of maximal PA candidates, select n compatible maximal PA candidates which covers the entire G with co-optimized mapping size n and communication cost c, denoted by $f(n,\ c)$.

Theorem 1: The optimal solution for Problem 2 using only maximal PA candidates equals the optimal solution for the original PA mapping problem using all possible PA candidates.

Proof: Assume that the optimal solution for the maximal PA mapping problem contains N maximal PA candidates with communication cost C_N, and the optimal solution for the original PA mapping problem contains M PA candidates with communication cost C_M.

(i) $f(N, C_N) \geq f(M, C_M)$ (defined by optimality).

(ii) Given an optimal PA mapping solution containing M non-overlapping PA candidates $\{P_1,\ P_2,\ ...,\ P_M\}$, for each P_i in the optimal solution, if P_i is a maximal PA candidate, let $P_i^* = P_i$; if P_i is not a maximal PA candidate, expand it by adding neighboring nodes until a corresponding maximal PA candidate P_i^m is generated – let $P_i^* = P_i^m$. The derived set of PA candidates $\{P_1^*,\ P_2^*,\ ...,\ P_M^*\}$ contains M maximal PA candidates, which cover the entire data flow graph and are compatible with each other (the corresponding overlapping-free subgraphs are $\{P_1,\ P_2,\ ...,\ P_M\}$). Therefore $\{P_1^*,\ P_2^*,\ ...,\ P_M^*\}$ is one feasible solution for the maximal PA mapping problem, and we have $f(M, C_M) \geq f(N, C_N)$.

From (i) and (ii), we can get $f(N, C_N) = f(M, C_M)$. ■

Theorem 1 demonstrates the optimality of the proposed maximal compilation flow, in which the original PA mapping problem can be transformed to the maximal PA mapping problem with a much smaller problem size.

B. Maximal PA Candidate Identification

Efficient pattern identification techniques have been investigated in a wide range of work [16] [17] [18]. In our flow, the subgraph identification and isomorphism checking techniques proposed in [16] are employed to generate connected PA candidates efficiently. If subgraph G with $k + 1$ nodes is a PA candidate, all the subgraphs of G with k nodes will be marked as non-maximal. In this case, when k increases to the maximal PA size, all the maximal connected PA candidates can be generated. Note that instead of generating all the disjoint PA candidates in an input data flow graph, we only target those which can be mapped to the pre-given PA template. Therefore, the microarchitectural constraints in the PA template, such as depth, size, number of inputs/outputs, can be applied to prune the identification space.

C. Communication Modeling

As discussed in Section III, the amount of intermediate result transfers between PA units can be estimated as the total number of communication edges between the corresponding PA candidates in the mapping solution. However, the impact of input-oriented communication is not captured by the original data flow graph. To model the communication overhead imposed by the primary input data streams, we introduce pseudo-communication edges and expand the original kernel data flow graph to represent the input data reuse. As shown in Figure 4(a), inputs of the three "–" nodes access the same array with possibly overlapped data ranges; therefore, three edges are inserted between each two nodes. Different from the existing edges in the kernel DFG, which indicate data flow dependencies, the newly added edges represent the existence of input data reuse between two operations.

Figure 4(b) and (c) show the communication edges in PA mapping solution I and II. By comparing Figure 4(b) and (c), we can conclude that solution II is better than I, in which all the communication edges are encapsulated inside each PA unit. In mapping solution I, there are nine communication edges across PA units, which will be directly reflected in the runtime data transfer overhead.

D. Communication-Aware Maximal PA Mapping

Now that we have a set of maximal PA candidates, a subset of those candidates needs to be selected and mapped to PA units. The entire mapping flow can be divided into two major phases: *mandatory-selection* and *max-cover*.The following metric is used as the optimality measurement for a given mapping solution $\mathbb{MP}^*$:

$$\frac{|\mathbb{MP}^*|}{BSF\_count} + \alpha \cdot \frac{comm(\mathbb{MP}^*)}{BSF\_comm} \tag{1}$$

The first part of Equation 1 measures the mapping size optimality, where BSF_count corresponds to the current optimal mapping size (*Best-So-Far*). Similarly, the second part measures the optimality of $\mathbb{MP}^*$'s communication cost, which can be viewed as the degree of closeness to the smallest communication cost obtained so far (BSF_comm). The function *comm* returns the estimated communication cost incurred

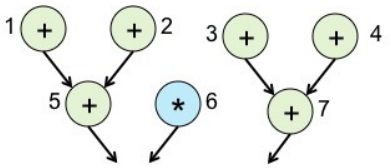

Fig. 5. Kernel data flow graph of *rician-denoise* after mandatory-selection.

by a given mapping solution. The variable α is a parameter which users can adjust based on the system requirements to trade off between area minimization and communication reduction. Note that when α is set to zero, this problem is reduced to the area-optimal mapping problem discussed in [7].

Phase 1: mandatory-selection In exact PA compilation algorithms, the complete set of PA candidates is enumerated and considered as inputs of the mapping phase. Therefore, in most cases a node in G will be covered only by more than one PA candidate, unless it is disconnected from other nodes in G. For example, node 14 in Figure 3 is covered by possible PA candidates such as {14}, {12, 14}, {13, 14}, etc. On the other hand, when we only include maximal PA candidates in the mapping phase, node 14 is only contained in one maximal PA candidate pac_2 in Figure 3. In this case, we can directly conclude that PA candidate pac_2 will be selected in the optimal covering solution, and remove all the nodes covered by pac_2 from G. Then the mapping process only needs to be applied to the remaining data flow graph with less PA candidates. Figure 5 shows the effect of mandatory-selection on Figure 3. After pac_2 is selected, the remaining graph contains 7 nodes, which is only half of the original size.

Phase 2: max-cover A branch-and-bound based covering algorithm is applied to the reduced data flow graph after mandatory selection. For each maximal PA candidate, it can be either included or excluded in a feasible solution. When the entire graph is covered after adding a new PA candidate, the corresponding covering solution will be compared to the current optimal solution. If the newly generated solution turns out to be better, compatibility checking is performed on the selected PA candidates. The current optimal solution will be updated if the selected PA candidates are compatible with each other. Note that BSF_count and BSF_comm are unknown values; the current optimal mapping size and communication cost are used to approach the optimal one.

Assume each overlapping node v_i is covered in n_i PA candidates. In the worst case $\prod n_i$, non-overlapping node assignment schemes need to be evaluated to decide whether a set of overlapping PA candidates are compatible or not. To perform fast compatibility checking, *tight* nodes are first removed from the overlapping node set. Here an overlapping node v is called a *tight* node of PA candidate P if P is no longer a PA candidate whenever v is removed. For example, the overlapping nodes located in a path between two nodes in P are tight nodes if the corresponding two nodes do not belong to the overlapping set. Therefore, it should be directly assigned to P; otherwise the convexity of P cannot be maintained.

Note that the communication overhead for each edge can be modeled based on the amount of data reuse between the two PA units connected by this edge; therefore, the corresponding edge weights may be non-uniform values. For example, the communication edge between $du1\_dy$ and $du1\_dz$ in Figure 4 can be assigned a larger weight compared to the one between $du1\_dx$ and $du1\_dy$ since the reuse distance between nodes $du1\_dy$ and $du1\_dz$ is smaller and more likely to happen .

V. Experimental Results

A. Experimental Setup

We evaluate the proposed maximal PA compilation flow on 12 computation-intensive applications from widely known benchmark suites and computing domains. The testcases include five benchmarks from the $SPEC2006$ suite [19] (*calculix*, *leslie3d*, *povray*, *bwaves* and *lbm*), four applications from the medical imaging domain [15] (*compressive sensing*, *registration*, *rician-denoise* and *segmentation*), and three applications from the *Rodinia* benchmark suite [20] (*heartwall*, *leukocyte* and cfd), which is designed for heterogeneous computer systems with accelerators.

Our PA compilation flow is implemented with the LLVM compiler infrastructure [21]. Omega library [22] is used for for memory reuse analysis. In the experiments, the tested benchmarks are compiled with all the standard optimization in O3 turned on. The compilation time is obtained on a 4-core Intel Xeon CPU (E5404) running at 2 GHZ. To further evaluate our compilation flow, we have extended Simics [23] and GEMS [24] and conduct cycle-accurate simulations on a CHARM-like architecture [1].

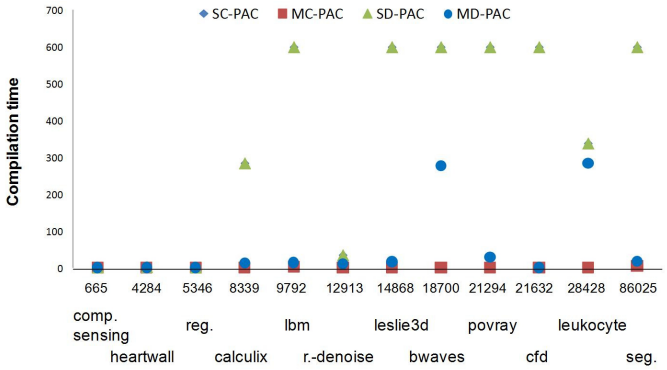

Fig. 6. Compilation time (sec) vs. input problem size.

B. Comparison Results

In this section we show the comparison results of four PA compilation flows – scalable connected PA compilation (SC-PAC) [7], the proposed maximal connected PA compilation (MC-PAC), scalable disjoint subgraph mapping (SD-PAC) [5] and the proposed maximal disjoint PA compilation (MD-PAC). Among the four approaches, the first two only target connected PA candidates, and the last two consider both connected and disjoint candidates.

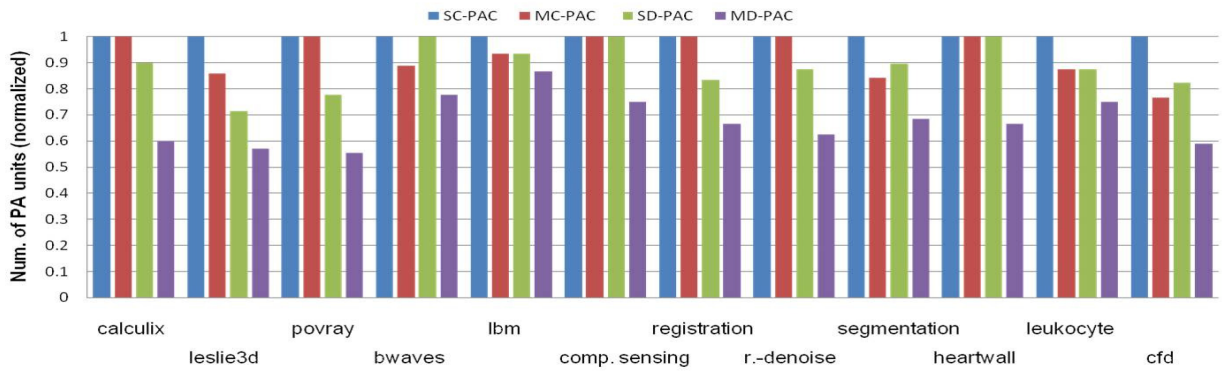

Fig. 7. Comparisons on PA compilation results of *SC-PAC* [7], *MC-PAC*, *SD-PAC* [5] and *MD-PAC*.

Compilation time. Table I shows the comparison results on the PA compilation time. To perform a fair comparison, we use the same maximal time limit in [7] and [5], upon which the PA compilation will be terminated and the best solution generated by this time point will be reported. Note that 1 *sec*. in Table I means that the compilation can complete in one second. The tradeoff parameter α is set to be zero in this comparison, which makes our mapping problem equivalent to [7].

From Table I, we can make the following observations:

(1) The compilation times of SC-PAC and SD-PAC are very close to each other. The reason is that SC-PAC is a subroutine of SD-PAC. In SD-PAC, the optimal connected PA mapping solution is first generated with SC-PAC. After that, a greedy grouping operation is performed on the selected PA candidates with negligible time overhead, as shown in Table I.

(2) In the connected compilation case, the maximal PA compilation algorithm can complete in less than 10 seconds for all the benchmarks, while the SC-PAC flow fails to complete for six test cases, and its compilation time increases quickly when the compilation problem size grows.

(3) The disjoint compilation results are similar – MD-PAC completes in no more than 300 seconds for all testcases.

The large gap in algorithm runtime between SC(D)-PAC and MC(D)-PAC can be explained with Table II and Table III. The problem size of PA mapping is related to two factors – the target DFG size and the total number of PA candidates which can be selected into a mapping solution. As we discussed, with the proposed concept of maximal PA candidates, both factors can be efficiently reduced. From Table II we can see that by only including the maximal ones, the total number of PA candidates in the mapping phase can be reduced significantly. Table III shows the reduction on the number of nodes to be covered in the kernel DFG, after *mandatory-selection*. SC-PAC and SD-PAC normally need to cover the size of the entire DFG, since most DFG nodes belong to more than one PA candidate and cannot be selected directly. In MC(D)-PAC, the number of nodes to cover can be reduced by 20% on average, as shown in Table III.

Algorithm Scalability. To illustrate the scalability of the proposed maximal PA compilation flow, we plot compilation time with the corresponding problem size for the 12 benchmarks. Here the compilation problem size is estimated as the product of target DFG size and the number of PA candidates.

As shown in Figure 6, SC-PAC and SD-PAC run fairly fast for moderate-size applications, while exhibiting limited scalability when the problem size grows. Note that *leukocyle* is one application which can be compiled within 600 seconds even with a large problem size. This is because the real runtime will also be influenced by other factors, such as subgraph overlapping and the efficiency of the initial greedy solution. The estimated problem size is used here to provide an insight into the overall trend.

Considering the maximal PA compilation flow, the increased problem size has a small effect on the MC-PAC runtime, and it can finish quickly for all 12 benchmarks. When disjoint PC candidates are included, the corresponding compilation flow MD-PAC gradually slows down as the problem size increases, but it still can finish in less than 300 seconds for all the benchmarks tested. As we discussed, when the estimated problem size exceeds a given threshold, a greedy MD-PAC process will be invoked, and the corresponding compilation time falls drastically while still generating reasonable mapping quality – which will be shown later in this section.

Mapping optimality. Figure 7 shows the comparison results on the final mapping size, which equals the number of selected PA candidates to cover the target DFG.

From the results we can see that compared to the optimal approach SC-PAC, MC-PAC generates better mapping solutions at six applications with relatively large kernel size. This because with those large testcases, SC-PAC cannot finish within 600 seconds and thus cannot obtain the actual optimal result even though the approach itself is optimal. On average, MC-PAC can achieve a 14% improvement over SC-MAC in terms of the mapping quality, and MD-PAC can achieve a 23.8% improvement over the heuristic SD-PAC approach and a 32.5% improvement compared to the results of SC-PAC with connected PA candidates.

Figure 8 shows the comparison results on the amount of

TABLE I
COMPARISONS ON PA COMPILATION TIME (SEC)

	calculix	leslie3d	povray	bwaves	lbm	comp. sensing	reg.	rician-denoise	seg.	heart-wall	leuko-cytes	cfd
SC-PAC[8]	284.3	> 600	> 600	> 600	> 600	1	1	34.2	> 600	2.1	338	> 600
MC-PAC	1	1	1	1	4.1	1	1	1	5.2	1	1	1
SD-PAC[6]	284.9	> 600	> 600	> 600	> 600	1	1	34.6	> 600	2.2	339	> 600
MD-PAC	14	17	29	277	14.6*	1	1	11	18.9*	1	284.1	1*

TABLE II
COMPARISONS ON THE NUMBER OF PA CANDIDATES

	calculix	leslie3d	povray	bwaves	lbm	comp. sensing	reg.	rician-denoise	seg.	heart-wall	leuko-cytes	cfd
SC(D)-PAC[6,8]	269	413	507	425	204	35	198	349	1147	252	618	416
MC-PAC	12	16	29	21	36	11	14	19	44	11	33	27
MD-PAC	98	31	378	215	54	28	103	162	57	11	270	46

TABLE III
KERNEL SIZE REDUCTION WITH *pre-selection*

	calculix	leslie3d	povray	bwaves	lbm	comp. sensing	reg.	rician-denoise	seg.	heart-wall	leuko-cytes	cfd
Original	31	36	42	44	48	19	27	37	75	17	46	52
MC(D)-PAC	22	26	32	39	45	13	19	30	55	13	35	47

data transfer reduction by applying the communication optimization, which is collected based on the CHARM platform. In our experiments, each communication edge is counted with unit communication cost, which can be further refined by using weighted communication edges to capture the difference in the communication overhead incurred by each edge.

On average, the proposed flow achieves a 13.7% reduction on the total amount of data transfers, with the maximal reduction up to 20%. We also observe that no obvious improvements have been achieved for *lbm*, *heartwall* and *leslie3d*. The reason is that there is less data reuse inside the kernels, and the original PA mapping solution is already the one with the optimal communication cost.

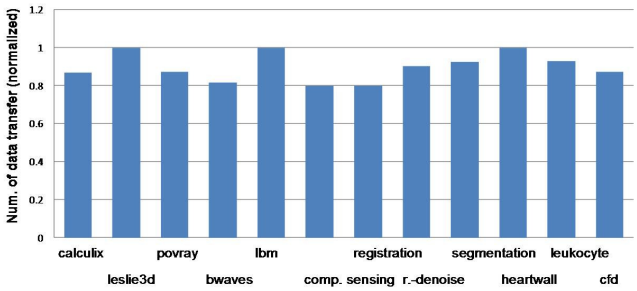

Fig. 8. Comparisons on communication overhead.

Performance. Figure 9 shows the comparison of execution time on the CHARM platform. The performance gain comes from two factors – improved data-level parallelism and reduced communication overhead. Given a limited number of accelerator units, a smaller mapping solution implies a higher capability of accelerator duplication to support parallel execution, so that multiple copies of accelerators can execute at independent loop iteration space [1]. Note that there is no obvious performance gain for *lbm*. This is because the accelerator resource saving in the mapping solution is not enough to create another parallel execution copy. On average, the overall performance gain of MD-PAC is improved by 49.5% over SC-PAC and 31.1% over SD-PAC.

VI. RELATED WORK

As discussed in Section I, both the PA candidates' identification and PA mapping problems are difficult to solve. Heuristic approaches have been employed in previous work to reduce the mapping complexity. A widely employed heuristic method is to perform greedy enumeration and immediate selection. One example is the DySER compilation flow [6]. Another set of work employs an exact PA compilation flow targeting optimal mapping solutions, which can be described as full enumeration followed by optimal mapping [7]. Here, full enumeration means enumerating all the possible PA candidates in the target kernels. Then an optimal mapping algorithm, such as ILP-based or branch-and-bound approach, will be applied on the full set of PA candidates. As discussed in Section I, scalability becomes the major compilation challenge in such approaches as the kernel size increases.

There already exists some relevant work investigating the development of scalable PA mapping methods to obtain optimal solutions for moderate-size application kernels. For example, in [7], a scalable subgraph mapping algorithm is proposed to generate optimal PA mapping solutions with connected PA candidates. The limitation of this work is the lack of support for disjoint PA candidates due to the scalability problem; thus it cannot fully utilize the existing parallelism in

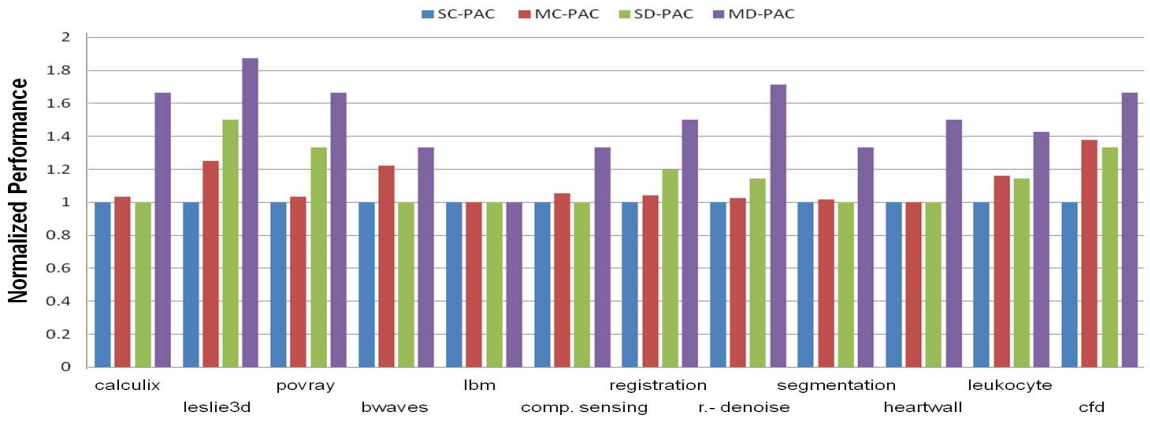

Fig. 9. Performance comparison of *SC-PAC* [7], *MC-PAC*, *SD-PAC* [5] and *MD-PAC*.

a PA template. An extension of this work is discussed in [5], in which the optimally selected connected accelerator patterns are greedily grouped into disjoint ones. However, there is no guarantee on the optimality of the resulting PA mapping solution. Another limitation of the previous PA compilation work [7], [5], [6] is the lack of communication optimization. The proposed mapping algorithms merely focus on the computation resources, e.g., the work in [7] targeting an area-optimal mapping solution without considering the underlying communication overhead in the PA-rich platform.

VII. Conclusion and Future work

The performance and energy efficiency of accelerator-rich platforms comes at the price of a number of compiler challenges. In this work we introduced a scalable communication-aware PA compilation flow based on maximal PA candidates. The proposed flow shows significant improvements in terms of mapping quality, scalability and communication overhead. One thing to note here is we only consider PA candidates identified by subgraph isomorphism techniques in the current flow, instead of full equivalence checking to determine whether two subgraphs are functionally equivalent or not. This will be investigated in the future work.

VIII. ACKNOWLEDGMENTS

This work is partially supported by MARCO Gigascale Systems Research Center (GSRC), the Center for Domain-Specific Computing (CDSC) funded by the NSF Expedition in Computing Award CCF-0926127 and Intel Corporation with matching support from the NSF InTrans Award CCF-1436827, and the NSF grant CCF-0903541.

References

[1] J. Cong, M. A. Ghodrat, M. Gill, B. Grigorian, and G. Reinman, "Charm: A composable heterogeneous accelerator-rich microprocessor," in *ISLPED*, 2012, pp. 379–384.

[2] *Intel Moorestown*, http://www.intel.com/pressroom/archive/reference/Moorestown _backgrounder.pdf.

[3] *The OMAP5430 Platform*, http://www.ti.com/ww/en/omap/omap5/omap5-OMAP5430.html.

[4] N. Clark, J. Blome, M. Chu, S. Mahlke, S. Biles, and K. Flautner, "An Architecture Framework for Transparent Instruction Set Customization in Embedded Processors," in *Proc. ISCA*, 2005.

[5] A. Hormati, N. Clark, and S. Mahlke, "Exploiting Narrow Accelerators with Data-Centric Subgraph Mapping," in *Proc. CGO*, 2007, pp. 341–353.

[6] V. Govindaraju, C. Ho, and K. Sankaralingam, "Dynamically Specialized Datapaths for Energy Efficient Computing," in *Proc. HPCA*, 2011, pp. 503–514.

[7] N. Clark, A. Hormati, S. Mahlke, and S. Yehia, "Scalable Subgraph Mapping for Acyclic Computation Accelerators," in *Proc. CASES*, 2006, pp. 147–157.

[8] H. Franke, J. Xenidis, C. Basso, B. M. Bass, S. S. Woodward, J. D. Brown, and C. L. Johnson, "Introduction to the wire-speed processor and architecture," *IBM Journal of Research and Development*, vol. 54, no. 1, pp. 3:1–3:11, Jan. 2010.

[9] *Convey computer*, http://conveycomputer.com.

[10] L. Seiler *et al.*, "Larrabee: A Many-Core x86 Architecture for Visual Computing," in *IEEE.Micro*, vol. 29, no. 1, 2009, pp. 10–21.

[11] *Nallatech development systems*, http://www.nallatech.com.

[12] R. Hou, L. Zhang, M. C. Huang, K. Wang, H. Franke, Y. Ge, and X. Chang, "Efficient data streaming with on-chip accelerators: Opportunities and challenges," in *International Symposium on High Performance Computer Architecture*, 2011, pp. 312–320.

[13] R. Razdan and M. D. Smith, "A high-performance microarchitecture with hardware-programmable functional units," in *Iternational symposium on Microarchitecture*, 1994, pp. 172–180.

[14] J. Cong, M. Ghodrat, M. Gill, B. Grigorian, and G. Reinman, "Architecture Support for Accelerator-Rich CMPs," in *DAC*, 2012.

[15] J. Cong, V. Sarkar, G. Reinman, and A. Bui, "Customizable domain-specific computing," in *IEEE Design and Test of Computers*, 2010, pp. 5–15.

[16] J. Cong and W. Jiang, "Pattern-based behavior synthesis for fpga resource reduction," in *FPGA*. New York, NY, USA: ACM, 2008, pp. 107–116.

[17] P. Bonzini and L. Pozzi, "Polynomial-time subgraph enumeration for automated instruction set extension," in *DATE*. New York, NY, USA: ACM Press, 2007, pp. 1331–1336.

[18] P. Yu and T. Mitra, "Disjoint Pattern Enumeration for Custom Intruction Identification," in *Proc. FPL*, 2007, pp. 273–278.

[19] *SPEC CPU2006*, http://pec.it.miami.edu/cpu2006/.

[20] S. Che, M. Boyer, J. Meng, D. Tarjan, J. Sheaffer, S. Lee, and K. Skadron, "Rodinia: A Benchmark Suite for Heterogeneous Computing," in *Proc. IISWC*, 2009, pp. 44–54.

[21] *The LLVM Compiler Infrastructure*, http://llvm.cs.uiuc.edu.

[22] *Omega Library*, http://www.cs.umd.edu/projects/omega.

[23] P. Magnusson, M. Christensson, J. Eskilson, D. Forsgren, G. Hallberg, J. Hogberg, F. Larsson, A. Moestedt, and B. Werner, "Simics: A full system simulation platform," in *IEEE Computer*, 2002, pp. 50–58.

[24] M. Martin, D. Sorin, B. Beckmann, M. Marty, M. Xu, A. Alameldeen, K. Moore, M. Hill, and D. Wood, "Multifacet's general execution-driven multiprocessor simulator(GEMS) toolset," in *Computer Architecture News*, 2005, pp. 92–99.