I^{2}CRF: Incremental Interconnect Customization for Embedded Reconfigurable Fabrics

Jonghee W. Yoon*, Jongeun Lee§†, Jaewan Jung*, Sanghyun Park*, Yongjoo Kim*, Yunheung Paek* and Doosan Cho‡

*Seoul National University, Korea
{jhyoon,jwjung,shpark,yjkim}@optimizer.snu.ac.kr, ypaek@snu.ac.kr

§Corresponding author.
†UNIST, Korea
jlee@unist.ac.kr

‡Sunchon National University
Korea
dscho@sunchon.ac.kr

***Abstract*—Integrating coarse-grained reconfigurable architectures (CGRAs) into a System-on-a-Chip (SoC) presents many benefits as well as important challenges. One of the challenges is how to customize the architecture for the target applications efficiently and effectively without explicit design space exploration. In this paper we present a novel methodology for incremental interconnect customization of CGRAs that can suggest a new interconnection architecture that can maximize the performance for a given set of application kernels while minimizing the hardware cost. Applying the inexact graph matching analogy, we translate our problem into graph matching taking into account the cost of various graph edit operations, which we solve using the A^* search algorithm with a heuristic tailored to our problem. Our experimental results demonstrate that our customization method can quickly find application-optimized interconnections that exhibit 70% higher performance on average compared to the base architecture, with relatively little hardware increase in interconnections and muxes.**

I. Introduction

The ever increasing demand for faster and more efficient computation all require constant innovation in computer design such as heterogeneous multi-core processors, general-purpose GPU computing, and reconfigurable architectures (e.g., FPGAs). While reconfigurable architectures are mainly used as prototyping platforms, those used as *flexible hardware accelerators* can be easily be integrated into SoCs (Systems-on-Chips) along with other modules such as soft cores and on-chip memories [1], [2], motivated by the need to reduce the communication time between the reconfigurable accelerator and other processors. Unlike FPGAs, such reconfigurable accelerators tend to be coarse-grained, or at the granularity of words (hence called Coarse-Grained Reconfigurable Architectures, or CGRAs), and are often targeted for such application domains as media processing, cryptography, and communication [3].

CGRAs embedded in SoCs present new opportunities for further customization for target applications. In today's design processes, virtually every soft module that is integrated into an SoC goes through a configuration step before the SoC is optimized for various design objectives. Configuration may include, in the case of a CGRA, such obvious ones as adjusting its size or the number of processing elements, but one can easily envision more elaborate ones such as choosing the right set of operations for the processing elements and even tuning the interconnection network and the memory subsystem architectures. Such application-specific customization of CGRAs is analogous to Application-Specific Instruction set Processor (ASIP) design (e.g., [4]), but is much more complicated due to the exceedingly larger design space of CGRAs. The goal of such architecture customization is to increase performance and/or energy-efficiency, while not hampering compilation generality[1] or worsening VLSI implementation.[2] As sometimes architectural specialization may be the only option to boost the performance without increasing power dissipation or resorting to ASIC for high-performance or real-time applications, it is important to address the issue of how to make the best design/customization decisions for CGRAs embedded in SoCs.

In this paper we consider the problem of incrementally changing the interconnect architecture of embedded CGRAs to maximize their performance for given applications, which is more challenging than application mapping [6], [7] or architecture synthesis [8], [9]. We focus on the interconnect architecture, as interconnection topology is found to have a significant impact on both performance and energy efficiency of CGRAs [10], [11]. Our methodology changes the architecture incrementally, by adding extensions to the base architecture (e.g., a mesh network), so that regularity is maintained through the base architecture, but anything beyond the mesh provides some level of specialization for the target applications. By starting from a base architecture, which is already a reduced version compared to typical CGRAs (e.g.,

This work was supported in part by the Korea Science and Engineering Foundation(KOSEF) NRL Program grant funded by the Korea government(MEST) (No. 2009-0083190), the Engineering Research Center of Excellence Program of Korea Ministry of Education, Science and Technology(MEST) / Korea Science and Engineering Foundation(KOSEF) (Grant 2010-0001724), and the IDEC, and in part by the 2009 Research Fund of the UNIST, under Grant 090035.

[1]Recent compilation algorithms for CGRAs are very general so that an arbitrary interconnection topology poses no greater challenge for compilation than a regular interconnection [5].

[2]Architecture customization may help VLSI implementation by making the design simpler (for the same performance level), or by restricting the search space to the ones that preserve the structural regularity to exploit the ease of VLSI implementation of regular structures.

978-3-9810801-7-9/DATE11/©2011 EDAA

1-hop and diagonal on top of mesh), our technique can reduce the cost of the hardware while delivering the same level of performance.[3]

One alternative to architecture customization is design space exploration. However the extensive nature of CGRA design space has permitted only very limited explorations of the space so far [13], [14], [15]. Moreover, application mapping for CGRA is itself still a long-running process [7], [6] despite recent advances, mainly because it must involve placement and routing on a 2D array. The main challenge of our approach is thus to find the best architecture customization without explicit design space exploration. We do so by matching the application graph to the architecture graph with minimal extension of the latter, which is similar to *inexact graph matching* [16]. Employing the common strategy in inexact graph matching, we translate our problem into that of finding the *graph edit sequence* to best embed an application graph into an architecture graph, taking into account the cost of graph edit operations. The inexact graph matching problem is arguably at least as complex as subgraph isomorphism [17], which is NP-complete, and often solved with the A^* search algorithm [18], for which we develop a heuristic function tailored to our problem domain. Our experimental results using benchmark kernels from multimedia and digital signal processing demonstrate that our customization method can quickly find application-optimized interconnections that exhibit 70% higher performance on average compared to the base architecture, or 41% higher performance on average with substantially less hardware cost compared to a typical interconnect architecture.

II. Definitions and Problem Formulation

Kernel Graph An application loop i can be represented by a directed graph with labels, called kernel graph, $K_i = (V_i, E_i, \alpha_i)$, where the vertices in V_i are the operations in the loop, edges in E_i represent the dependence between the operations. Labeling α_i is a function assigning an operation type to each vertex.

CGRA Graph An $M \times N$ CGRA can be represented as another directed graph with labels $C = (P, L, \beta)$, where $p_{ij} \in P$ with $1 \leq i \leq M$ and $1 \leq j \leq N$, is the (i, j)-th Processing Element (PE) of the CGRA, and L is the set of all interconnection links. C_0 and L_0 denote the base CGRA and its interconnection links, respectively. For two PEs $p, q \in P$, there is an interconnection link $l = (p, q) \in L$ iff q can use the result of p that was generated in the previous cycle. The set of operation types supported by a PE is captured by the labeling function β.

Augmentation & Mapping Architecture extension is represented by an augmentation to C_0, which is the set of additional interconnection links $\Delta_L = \bigcup_i \Delta_L^i$, where Δ_L^i is the augmentation induced by K_i. A (kernel graph) mapping is a function $\phi : K_i \rightarrow C$, which consists of two sub-functions, $\phi_{V_i} : V_i \rightarrow P$, and $\phi_{E_i} : E_i \rightarrow 2^L$, where $L = L_0 \cup \Delta_L$.

- ϕ_{V_i} is an injective function that maps operations to compatible PEs. In other words, each vertex $v \in V_i$ is mapped to a distinct PE $p \in P$ such that $\alpha_i(v) \in \beta(p)$.
- ϕ_{E_i} is a multi-valued function that maps data dependence $e \in E_i$ of a kernel to a set of interconnection links $I \in 2^L$.

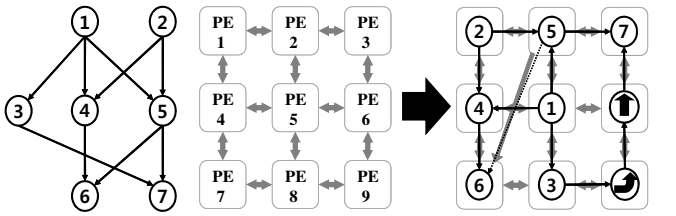
Constraints

- Simple Path Existence: For an edge $e = (u, v) \in E_i$, let $I = \phi_{E_i}(e) \in 2^L$. Then I must constitute a valid simple path.
- Routing PE: For a data dependence edge $e = (u, v) \in E_i$, RPE^e is the set of routing PEs for e, $RPE^e = \{q | \forall l = (p, q) \in \phi_{E_i}(e), q \neq \phi_{V_i}(v)\}$. We define RPE to be the set of all routing PEs, or $RPE = \bigcup_{e \in E_i} RPE^e$. A routing PE may be used to route only one value, and no computations may be performed on it in that particular cycle.
- Shared Resource Constraint: Each row of the CGRA may share resources such as floating-point unit and load-store unit [19]. To specify such a constraint we define the number of shared resources per row for each shared resource type.

Interconnection Insertion Cost Inserting interconnections to the base CGRA incurs additional hardware cost $c(\Delta_L^i) = \Sigma_{l \in \Delta_L^i} c_{li}(l)$, where $c_{li}(l)$ is the cost of inserting an interconnection link.

Now the problem is to find the best mapping for the given set of kernels, minimizing the cost of new interconnection links added to the base CGRA, C_0. A typical measure of application mapping for a loop is Initiation Interval (II). However since we consider multiple loops, and II has dependence on the size of a loop, we take IPC (Instructions Per Cycle).

Given a set of n kernels $K = \{K_i \mid 1 \leq i \leq n\}$, and a base CGRA $C = (P, L_0, \beta)$, find mappings $\phi(K_i)$ with the objective of $\min c(\Delta_L)$ *for maximal IPC, under i) simple path existence, ii) uniqueness of, and no computation on, routing PE, and iii) shared resource constraints.*



(a) Kernel K_i (b) Base CGRA C_0 (c) Augmentation and mapping (K'_i, C)

Fig. 1. CGRA augmentation and kernel mapping.

III. Our Approach: I^{2}CRF

A. Our Strategy

While application mapping for CGRA is finding a graph $X \subset C_0$ that is isomorphic to K, ours is finding a graph Y that is isomorphic to K and subset of C which is most similar to C_0. The dissimilarity between two graphs, or the

[3]The idea of superimposing a few long-range links on a mesh network to take advantage of both regularity and partial topology customization is not entirely new (e.g., [12] in the network-on-chip domain), but CGRA customization has a completely different set of challenges.

graph (edit) distance, is defined as the cost of the minimum-cost graph edit path from one graph to the other [16]. Suppose that we want to compute the graph distance between K_i and C_0 (see Figure 1). C_0 is not isomorphic to kernel graph K_i, nor is any subgraph of C_0. Thus we add or remove nodes and edges from K_i to match C_0, and the problem is to find the sequence of such graph edit operations with the minimum cost. This can be considered as a search problem in a graph-edit state tree where the root is K_i and each branch defines a graph edit operation with its cost. Then we need search for the minimum-cost path from the root to a leaf that is C_0. By searching for the minimum-cost edit path we can ensure that the required CGRA augmentation is minimal.

B. Graph Edit Operations

While our problem seems more complicated than pure inexact graph matching due to its application-to-architecture mapping aspect, an important observation is that once a CGRA augmentation and mapping solution is given, one can list all the necessary operations to convert the structure of a kernel graph to that of a CGRA graph, which can be considered as graph edit operations. Thus we say a kernel graph mapping $\phi = (\phi_{V_i}, \phi_{E_i})$ *induces* graph edit operations. Recall that a kernel graph mapping[4] defines which operation node in V_i is mapped to which PE in P (by ϕ_{V_i}) and which data dependence edge in E_i is mapped to which subset of L (by ϕ_{E_i}).[5] Therefore by comparing K_i and C_0, one can find all the graph edit operations necessary to convert K_i to C_0. We summarize in Table I the set of the graph edit operations that can arise from CGRA mappings and how they can be identified from a given mapping. In our scheme, node deletion does not occur, and we omit node substitution as it is trivial. It can be seen that the operations listed in the table are enough to transform Figure 1(a) to (b) through (c).

TABLE I
GRAPH EDIT OPERATIONS INDUCED BY A KERNEL GRAPH MAPPING

Graph Edit Op.	Induced by
Node insertion	A routing PE (e.g., p_6, p_9) or an unused PE
Edge insertion	An unused interconnection link; e.g., $(p_5, p_6), (p_6, p_5), (p_7, p_8), (p_8, p_7), (p_2, p_1), ...$
Edge substitution (identical)	A data dependence edge mapped to a single interconnection link included in L_0
Edge substitution (non-identical)	An edge mapped to multiple interconnection links via routing PE(s); e.g., $(v_3, v_7) \rightarrow \{(p_8, p_9), (p_9, p_6), (p_6, p_3)\}$
Edge deletion	A data dependence edge mapped to a single interconnection link not included in L_0 (included in Δ_L); e.g., (v_5, v_6)

C. Graph Edit Cost Model

Among the graph edit operations listed in Table I, only edge deletion directly implies architecture extension, or Δ_L. The cost of an edge deletion operation is what is termed *interconnection insertion cost* in our problem formulation, which we will describe in more detail. Another edit operation that can affect the quality of mapping and thereby indirectly impact the amount of architecture extension, is the node insertion operation due to routing PEs. In kernel graph mapping, often II has to be increased due to the lack of interconnection resources even though there are enough PEs. Using PEs as interconnection resources ("routing PE") can help, but only if routing PEs are judiciously used. Therefore, routing PEs can be considered as a cheaper alternative to adding interconnection links, and we associate a cost with it. Thus there are two kinds of cost: i) *edge cost* is the cost of adding a new, custom interconnection link, and ii) *vertex cost* is the cost of adding a new vertex for use as a routing PE.

Specifically, a kernel graph mapping ϕ induces a set of graph edit operations, the cost of which is defined as:

$$c(\phi) = \sum_{v \in RPE} c_v(v) + \sum_{e \in E_i^d} c_e(e) \tag{1}$$

where $c_v(v)$ is the cost of inserting vertex v for use as a routing PE, $c_e(e)$ is the cost of deleting edge e, and E_i^d is the set of edges that are deleted. Here, $c_e(e)$ is identical to $c_{li}(l)$ in II. The vertex and edge costs are defined as follows.

$$c_v(v) = c_1 \tag{2}$$

$$c_e(e) = c_2 \cdot (w_m(s) + w_w(e)) \tag{3}$$

where c_1 and c_2 are constants representing unit vertex cost and unit edge cost, respectively. The absolute value of c_1 is not important, since c_2 is defined as a multiple of c_1. We set c_2 large enough, so that routing through PEs is first sought out before creating a new, custom link to implement a data dependence edge. For our experiments we set $c_2 = c_1 \cdot (|P| - |V_i| + 1)$, which makes it much cheaper to use routing PEs than to create a new link. The edge cost, $c_e(e)$, consists of two parts: $w_m(s)$ is the cost due to the increased multiplexor complexity in the affected PEs, and $w_w(e)$ is the cost due to the newly added wires. These terms are defined as relative weights to the unit edge cost. The multiplexor cost is assumed to be constant in our experiment, but ideally a step function-like cost model can be employed to more accurately model the cost, for which the current state information s becomes necessary. The wire cost is assumed to be linearly proportional to the wire length (i.e., Manhattan distance divided by unit distance between PEs), but it is easy to extend it to a more complex model. We assume that long wires are pipelined so that they may not be on the critical path.

D. Heuristic Function for A^* Search

Since in our case graph edit operations are induced by a mapping, we search the mapping space for the minimum-cost path. As visiting all mapping instances may require a prohibitive amount of resources even for moderate-sized problems, we employ the A^* search algorithm [18], which requires a heuristic function to guide the search. As we search

[4]Our notion of *mapping* includes CGRA augmentation, as its target architecture already includes necessary architecture extension (Δ_L).

[5]Note that though the two sub-functions ϕ_{V_i}, ϕ_{E_i} overlap mostly, we need ϕ_{E_i} (or equivalently routing PE information) to specify which routing PEs, if any, are used to map data dependence edges.

the mapping space, different versions of *partial* mappings are generated, inducing a set of graph edit operations, the cost of which determines $g(s)$, or the cost of the optimal path from the root state to the current state s found by A^* so far. To compute $g(s)$, we first extract the subgraph of the kernel graph containing all the *mapped nodes* for a given partial mapping (which is a full mapping as far as the subgraph is concerned), and then generate the set of graph edit operations using the method listed in Table I. The sum of the costs of the graph edit operations is $g(s)$.

The heuristic function $h(s)$, which estimates the cost from s to a leaf state, comes from the portion of the kernel graph that is yet to be mapped, or the subgraph containing all the *unmapped edges*. (Combined, $g(s) + h(s)$ gives an assessment of the partial mapping s.) The key idea of $h(s)$ is to find the number of difficult-to-map edges. We assume *vertex scattering*-based mapping, which is essential in some mesh-based mapping algorithms to support certain architectural constraints such as shared resource constraints. Vertex scattering [20] partitions, or *splits*, a kernel graph, and each partition is mapped to a different row. Then a *fork*, which is a set of adjacent edges cut by a split, cannot be mapped to a mesh without additional routing resources such as a routing PE or a custom interconnection link ("difficult-to-map edge"). Also, an *overlength* edge, which is defined as an edge between different partitions that are not neighboring with each other, is a difficult-to-map edge. For those difficult edges there is some positive cost. The cost can be minimized if routing PEs are maximally used, because routing PEs are exceedingly cheaper than adding an interconnection link ($c_1 \ll c_2$). Therefore the minimum number of required routing PEs is $N_f + |E_o|$, or $N_f + \sum_{e \in E_o} l(e)$, where N_f is the number of *forks*, E_o is the set of *overlength* edges, and $l(e)$ is a conservative estimation of wire length (in terms of routing PEs necessary) for edge e. for edge e. For $l(e)$ we use the number of partitions passed by e.

Now our heuristic function $h(s)$ for a subgraph from s can be defined as:

$$h(s) = \begin{cases} c_1 N_r & \text{if } N_r \leq N_i \\ c_1 N_i + c_2(N_f + |E_o| - N_i) & \text{otherwise} \end{cases} \quad (4)$$

where $N_i = |P| - |V_i|$ is the number of available routing PEs, and updated for each state s. If there are enough routing PEs ($N_r \leq N_i$), the minimum cost is achieved by implementing all the difficult edges with PE routing. Otherwise after exhausting all the available PEs first, the remaining difficult-to-map edges must be implemented using custom interconnection links.

E. Example

Figure 2 illustrates with an example how to find the best CGRA augmentation and mapping. The input base CGRA graph, C, and the graph edit costs are shown in Figure 2(a). Figure 2(b) illustrates input kernel graph, K_i, and its vertex scattering result, which assigns v_1 and v_4 to the first row of C, and v_2 and v_3 to the second row. A fork (black thick arrows inFigure 2(b)) is identified at the vertex scattering stage. The A^* search tree is drawn in Figure 2(c), where each node has its state ID ("s"), incremental mapping information ($\{\cdot\}$), and its cost (in dark square). The symbol \$ in the mapping information means routing vertex insertion.

The root node has $g(0) = 0$ since nothing has been mapped yet, and $h(0) = 1$ from the fork node in the kernel graph. The search space is explored as mapping is incrementally made. At the second level of the tree all three states have the same cost, thus we pick any one at random ($s = 1$). At state $s = 5$, substituting p_3 for v_4 makes it necessary to insert a routing vertex, thereby increasing the $g(s)$ cost to 1, while at the same time those edit operations do not help remove the existing fork, which means that $h(5)$ is unchanged. Figures 2(d) through (h) depict extended kernel graphs (black solid line graphs) generated by complete mappings. At state $s = 14$ in Figure 2(h), the $g(s)$ cost becomes 1 as there is one PE routing only while $h(s)$ becomes 0, which gives the minimum overall cost, and thus an optimal edit path.

IV. Experiments

A. Experimental Setup

To evaluate the effectiveness of our approach, we compare our custom-interconnect CGRAs against CGRAs with predefined-interconnects of different complexity. The predefined interconnects are i) mesh, ii) 1-hop (connecting PEs of distance 2 or less, vertically and horizontally), iii) diagonal, and iv) mixed (combining all of the above). The rest of the CGRA architecture is based on the MorphoSys architecture [21]. The size of the PE array is 4x4, which is often used in other CGRA literature [6] for evaluation.

We use kernels from multimedia and digital signal processing benchmarks. Each kernel is fed to our I^{2}CRF interconnect customization tool along with a base CGRA architecture that has only mesh interconnections. Our tool generates a set of new uni-directional interconnections that makes the CGRA achieve the best performance (in terms of II) for the given set of kernels, while trying to minimize the cost (priority is in maximizing performance). If multiple kernels are presented as input, the set of new interconnection links derived from a kernel is added to the CGRA architecture, and the extended CGRA is used to find the architecture augmentation and mapping for the next kernel. Therefore the interconnection links added for the previous kernel can be used for free in the next kernel, which can lower the cost of the overall architecture extension for multiple kernels, compared to simply merging the architecture extensions separately derived from each kernel. In contrast, in a typical CGRA mapping approach, kernels are simply mapped to the CGRA architecture without any architecture customization, which often results in much higher II than is possible through customization.

The selection of a mapping algorithm seems important for fair comparison. We try two approaches. The first approach is to use our own A^* search engine with modifications that disable edge deletion operations. The second approach is to use a well-known mapping algorithm such as the EMS algorithm [6]. Both have merits and problems, and we combine

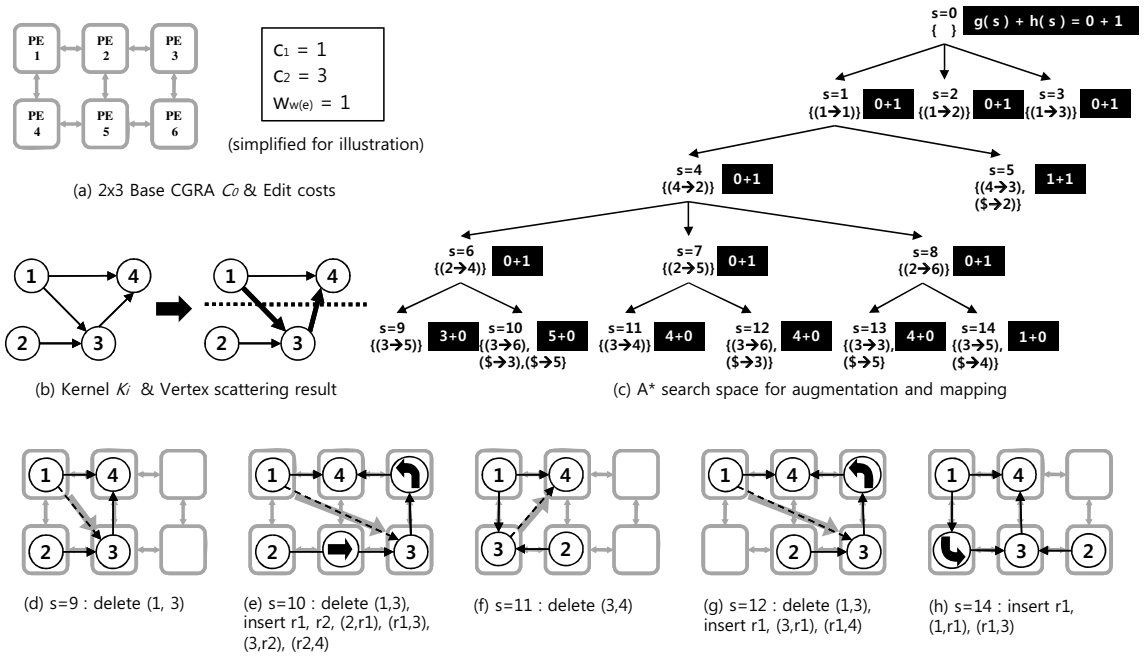


Fig. 2. Vertex scattering and searching for the best mapping using the A^* search.

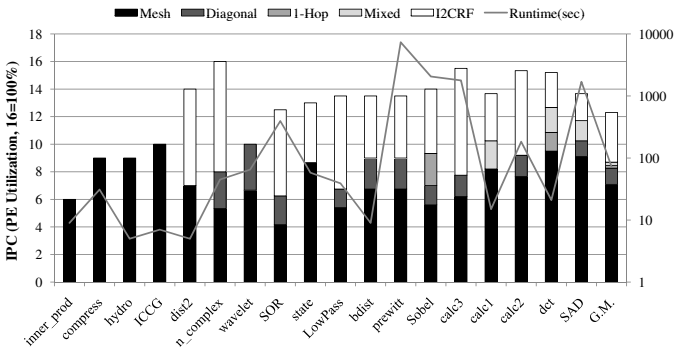


Fig. 3. Performance comparison between a mapping approach and our customization approach, for benchmark kernels (G.M. = Geometric Mean). IPC increases in the order of mesh $\leq$ diagonal $\leq$ 1-hop $\leq$ mixed $\leq$ I^{2}CRF. Runtime is shown on the right axis on a log-scale.

both approaches and report the best results. We limit the mapping time for the first approach to two hours per II (if no mapping is found within 2 hours, the next II is tried).

B. Benchmark Kernel Results

Figure 3 shows the performance of different CGRA architectures for benchmark kernels, in terms of IPC, or PE utilization (IPC of 16 is equivalent to 100% utilization as there are 16 PEs in the PE array). Routing PEs are not counted toward PE utilization. Overall, the PE utilization of CGRAs are not very high, especially for mapping-only approaches, regardless of the interconnection topology. This is in part due to the small size of the data flow graph. For instance, if a kernel has 17 operations, the smallest II is 2, and utilization will be limited to 53%. But in large part, it is limited by the lack of interconnection resources. Hence, as we see in the graph, the more versatile the interconnect is, the greater number of PEs are utilized for actual computation. On average, the PE utilization is increased by 23% through a more complicated interconnect (Mixed) compared to the Mesh architecture. In comparison, through our interconnection customization methodology the PE utilization and the IPC are increased by more than 70% on average compared to Mesh, or by 41% on average compared to Mixed.

TABLE II
CUSTOMIZATION OVERHEAD FOR BENCHMARK KERNELS

	Interconnection			Multiplexor		
Benchmark	#Links	AvgLen	MaxLen	AvgInc	MaxInc	#>Max
compress	0	0	0	0	0	0
LowPass	5	2.40	4	0.31	1	1
wavelet	4	2.75	3	0.25	1	1
calc1	5	3.20	4	0.31	1	1
calc2	13	2.54	5	0.81	1	4
calc3	9	3.11	4	0.56	1	2
Sobel	11	3.18	5	0.69	2	3
SOR	7	2.86	4	0.44	1	2
bdist	6	2.33	3	0.38	2	2
dct	21	2.71	6	1.31	3	6
dist2	5	2.40	3	0.31	1	1
hydro	0	0	0	0	0	0
ICCG	0	0	0	0	0	0
inner_prod	0	0	0	0	0	0
n_complex	3	2.33	3	0.19	0	0
prewitt	9	3.11	4	0.56	2	2
SAD	7	2.71	4	0.44	2	3
state	3	3.33	4	0.19	0	0
1-Hop	32	2.00	2	2.00	2	12
Diagonal	36	2.00	2	2.25	4	12
Mixed	68	2.00	2	4.25	6	16

Table II summarizes the overhead of our customization approach. The overhead mainly consists of the added intercon-

nection links and the increased mux size. The table lists, on the left half, the number of new interconnection links and their average/maximum lengths. The length of a link is calculated as Manhattan distance between the end points, with 1 being the distance between PEs (PEs are assumed to be square-shaped). On the right half, the table lists the average/maximum increase in mux sizes (in terms of the number of inputs), and the number of muxes that grow bigger than the largest mux before customization.[6] For comparison the table also lists the same statistics for three interconnection architectures (1-hop, diagonal, and mixed), against which our custom-generated architectures are compared.

Table II first reveals that the number of new interconnection links added by our customization methodology is very small compared to other typical interconnection architectures. The average lengths of the new interconnection links are also very modest (around 3) though there are a few long links, of up to 6 in the case of dct. When it comes to mux, our technique boasts a very marginal increase in the overall mux complexity compared to the base architecture (i.e., rarely more than 1 extra input is needed on average) whereas other architectures routinely have at least 2 extra inputs. While this is a direct consequence of having a small number of new interconnection links overall, it does suggest that our customization methodology adds interconnection links very judiciously.

C. Optimization Time

An alternative to algorithmic architecture customization such as ours is Design Space Exploration (DSE). While DSE can find the best architectures at least in theory, it is extremely slow. And it is not clear at all how one can perform DSE efficiently and effectively, especially since CGRAs have a number of architectural parameters and therefore vast design space. Further, unlike compilation for uniprocessors, mapping a kernel to a CGRA can take quarter to half an hour even by a heuristic algorithm [6]. In contrast, our approach can find competitive custom interconnection architectures taking reasonable time as shown in Figure 3, which is very much in the same range as the compilation times (not shown here), sometimes even faster. One reason why ours can be faster than a mapping algorithm is that a mapping algorithm takes longer if the final II is higher, and ours finishes very often at a lower II. Still our experimental results clearly demonstrate that our customization method can find competitive custom interconnection architectures very quickly.

V. Conclusion

We presented an interconnection customization method for coarse-grained reconfigurable architectures (CGRAs). The interconnection architecture of a CGRA ranges from a simple mesh to much more complicated ones, and has a potential to impact the utilization of processing elements and the performance. Our method exploits the similarity between the interconnection customization problem and inexact graph matching.

The experiments in our study have many limitations especially regarding the back-end implementation of CGRAs. For instance we assume that non-homogeneous extensions to a base interconnection architecture can be made without much penalty in VLSI design. We are currently working to find out the extent of the difficulty due to the non-homogeneity as well as find novel ways to mitigate any impact if necessary. We also plan to extend the scope of architecture customization for CGRAs.

[6]Even before customization mux sizes can vary depending on the location of a PE. E.g., fewer-input muxes may be found at the corners.

References

[1] A. Lodi *et al.*, "An embedded reconfigurable datapath for SoC," in *FCCM '05*, 2005.

[2] S. Khawam *et al.*, "Embedded reconfigurable array targeting motion estimation applications," in *IEEE ISCAS*, 2003.

[3] R. Hartenstein, "A decade of reconfigurable computing: a visionary retrospective," in *DATE '01*, 2001.

[4] P. Yiannacouras *et al.*, "Application-specific customization of soft processor microarchitecture," in *FPGA '06*, 2006.

[5] H. Park *et al.*, "Modulo graph embedding: mapping applications onto coarse-grained reconfigurable architectures," in *CASES '06*, 2006.

[6] H. Park *et al.*, "Edge-centric modulo scheduling for coarse-grained reconfigurable architectures," in *PACT '08*, 2008.

[7] B. Mei *et al.*, "DRESC: A retargetable compiler for coarse-grained reconfigurable architectures," in *FPT '02*, 2002.

[8] Z. Huang *et al.*, "The design of dynamically reconfigurable datapath coprocessors," *ACM Trans. Embed. Comput. Syst.*, vol. 3, no. 2, 2004.

[9] P. Brisk *et al.*, "Area-efficient instruction set synthesis for reconfigurable system-on-chip designs," in *DAC '04*, 2004.

[10] A. Lambrechts *et al.*, "Energy-aware interconnect optimization for a coarse grained reconfigurable processor," in *VLSI Design '08*, 2008.

[11] N. Bansal *et al.*, "Network topology exploration of mesh-based coarse-grain reconfigurable architectures," in *DATE '04*, 2003.

[12] U. Y. Ogras *et al.*, "Its a small world after all: Noc performance optimization via long-range link insertion," *IEEE Trans. VLSI*, vol. 14, 2006.

[13] B. Mei *et al.*, "Architecture exploration for a reconfigurable architecture template," *IEEE Design and Test of Computers*, vol. 22, 2005.

[14] R. W. Hartenstein *et al.*, "Design-space exploration of low power coarse grained reconfigurable datapath array architectures," in *PATMOS '00*, 2000.

[15] K. Karuri *et al.*, "A design flow for architecture exploration and implementation of partially reconfigurable processors," *IEEE Trans. VLSI*, vol. 16, 2008.

[16] H. Bunke, "Inexact graph matching for structural pattern recognition," *Pattern Recognition Letters*, vol. 1, no. 4, 1983.

[17] S. Thoresen, "An efficient solution to inexact graph matching with application to computer vision," Ph.D. dissertation, NTNU, Norway, 2007.

[18] P. E. Hart *et al.*, "A formal basis for the heuristic determination of minimum cost paths," *Autonomous Mobile Robots: Perception, Mapping, and Navigation (Vol. 1)*, 1991.

[19] Y. Kim *et al.*, "Resource sharing and pipelining in coarse-grained reconfigurable architecture for domain-specific optimization," in *DATE '05*, 2005.

[20] G. D. Battista *et al.*, "A split & push approach to 3D orthogonal drawing," in *Graph Drawing*, 1998.

[21] H. Singh *et al.*, "Morphosys: An integrated reconfigurable system for data-parallel and computation-intensive applications," *IEEE Trans. Comput.*, vol. 49, no. 5, 2000.