An Operation and Interconnection Sharing Algorithm for Reconfiguration Overhead Reduction Using Static Partial Reconfiguration

Sungjoon Jung, *Member, IEEE*, and Tag Gon Kim, *Senior Member, IEEE*

***Abstract*—In most reconfigurable architectures, reconfiguration overhead is a great bottleneck for repetitive reconfigurations. In this paper, we introduce a resource sharing algorithm exploiting static partial reconfiguration to reduce reconfiguration overhead between consecutive configurations. The proposed algorithm allows parameterizing reconfiguration overhead for individual resources. We also present both conservative and aggressive pruning techniques to reduce computation time. Experiments have been conducted with generated graphs and real benchmarks using a synthesizable intermediate representation. The results report that the algorithm could share up to 6.82% more resources than a previous interconnection sharing technique, and that the algorithm could reuse 80.9% resources in the selected benchmarks.**

***Index Terms*—Reconfiguration overhead management, sharing algorithm, static partial reconfiguration.**

I. Introduction

In RECENT decades, reconfigurable architectures (RAs) have proven themselves as one of the most promising solutions in computing domain. However, reconfiguration overhead in most RAs is a great bottleneck for repetitive reconfigurations [1]–[3]. Frequent changes in application requirements may make the reconfiguration overhead to overwhelm the computational speedups, and eventually to degrade the overall system performance. Therefore, to amortize the overhead, RAs are supposed to execute a kernel thousands of times after a single configuration. In order to relieve the overhead problem, Virtex [4], PACT XPP [5], and other state-of-the-art RAs support *partial reconfiguration* that only configures small differences between consecutive configurations.

To effectively employ the partial reconfiguration and to reduce the reconfiguration overhead, this paper adopts resource sharing to reuse the already configured resources between consecutive configurations or temporal partitions. The execution model that reconfigures a device every time entering temporal partitions is basically different from a regular time-multiplexing model where all configurations are kept valid on a device throughout the execution. Since it only maintains a single partition at a time, the given model results relatively small area, and gets especially useful when loop bodies are unmappable due to area constraint and there is no other option but dividing loops into several parts [6]. Based on the execution model, this paper introduces a graph model that combines interconnection and operation sharing problems, and proposes a resource sharing algorithm derived from a traditional graph theory. Additionally, we provide pruning techniques designed for the proposed model.

The rest of this paper is organized as follows. Section II gives us an overview of previous sharing schemes for RAs. In Section III, we detail and model the operation and interconnection sharing problem. Section IV explains our algorithm for the sharing problem and the pruning criteria to reduce computation time. Section V reports the implementation and results of the proposed algorithm. Finally, the paper concludes with Section VI.

II. Related Work

Resource sharing is a traditional problem in high level synthesis area that is usually divided into data-level, operation-level, and interconnection-level. Singh [2] proposed a data sharing scheme, which reuses data in registers and minimizes data transfer between data memory and configuration memory. Even though data sharing seems effective to save reconfiguration time, it is also restrictive and can be applied to a few limited applications having specific data patterns. Operation sharing is a kind of traditional graph theory, and is well stabilized as a maximum bipartite matching. Huang and Malik [7] applied and extended the bipartite matching to interconnection sharing in RAs with weights of edges. In their model, highly weighted edges have high probability to share operations and interconnections.

Although the maximum bipartite based algorithm guarantees fast computation, it produces poor resource sharing due to an improper model of interconnection sharing that does not exploit the entire search spaces. Moreano *et al.* [8] proposed a maximum clique based algorithm that only models interconnections and covers all possible mappings of interconnections. Since the maximum clique problem is in NP class, they employed a heuristic, and showed 24% fewer configuration for interconnections than Huang and Malik's solution. Operation sharing in their algorithm is a result of interconnection sharing or a result of simple post-pass mapping between remaining unshared operations after interconnection sharing. This is mainly due to their architecture model that is based on interconnection networks, and such preference to interconnection sharing makes their algorithm weaker when the overhead of operation reconfiguration is higher than that of interconnection reconfiguration.

Manuscript received May 17, 2006; revised April 11, 2007. Current version published November 19, 2008.

The authors are with the Electrical Engineering Department, Korea Advanced Institute of Science and Technology (KAIST), Daejeon 305-701, Republic of Korea (e-mail: sjjung@smslab.kaist.ac.kr).

Digital Object Identifier 10.1109/TVLSI.2008.2000973

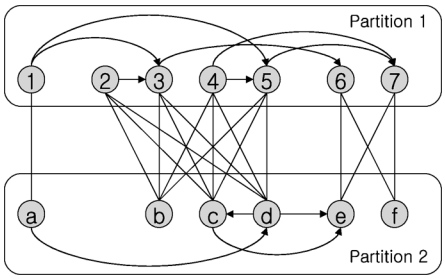


Fig. 1. Two partitions and compatibility edges.

III. Graph Model For Operation and Interconnection Sharing Problem

In this paper, an application is a list of temporal partitions. A temporal partition, a unit to be configured and executed on a device at one time, is a set of data flow graphs (DFGs), where a DFG is a set of *operation nodes* and *dependency edges*. A dependency edge is a directed edge to represent a dataflow between two operation nodes. Given two DFGs, there are relations, called *compatibility edges*, to describe which operation is sharable with another in the adjacent partition. For example, two partitions and compatibility edges between them are illustrated in Fig. 1.

From now on, we shall use notations n_{number} for nodes in Partition 1 or $G_1 = (N_1, E_1)$, n_{alphabet} for nodes in Partition 2 or $G_2 = (N_2, E_2)$, e_{ij} for dependency edges between n_i and n_j, and $c_{\alpha\beta}$ for compatibility edges between n_α and n_β, where n_i and n_j are in the same graph, and n_α and n_β are in different graphs. Finally, CE is the set of all compatibility edges.

Since compatibility edges represent sharable nodes, n_i becomes sharable only with n_j when all compatibility edges except one are removed from both nodes, or when a compatibility edge c_{ij} is *selected*. For example, n_3 in Fig. 1 can be reused by n_d if and only if c_{3d} is selected and all the other compatibility edges are removed from n_3 and n_d. When both c_{1a} and c_{3d} are selected, n_1 and n_3 are reused by n_a and n_d, therefore, e_{13} can also be reused by e_{ad}. In other words, an interconnection sharing could be said to select two compatibility edges according to dependency edges, as well as an operation sharing is represented as selecting just one compatibility edge.

Definition 1: For a subset SCE of CE, an operation sharing OS is $\{(n_a, n_b) | n_a \in N_1, n_b \in N_2, (n_a, n_b) \in \text{SCE}\}$ and an interconnection sharing IS is

$$\{(e_{ij}, e_{kl}) | e_{ij} = (n_i, n_j) \in E_1, e_{kl} = (n_k, n_l) \in E_2, \{(n_i, n_k), (n_j, n_l)\} \subset \text{SCE}\}.$$

The only requirement in selecting compatibility edges is that one node cannot be shared more than once, which will be called the *single reuse constraint*.

Definition 2: For any $\{(n_i, n_j), (n_i, n_k)\} \subset \text{SCE}$ or $\{(n_j, n_i), (n_k, n_i)\} \subset \text{SCE}$, $n_j = n_k$.

When the pairwise cost functions for operation and interconnection sharing are $\text{RO}_o(n_a, n_b)$ and $\text{RO}_i(e_{ij}, e_{kl})$, the cost function for the given resource sharing is defined as follows.

Definition 3: The total sharing advantage TS by selecting SCE is $\sum_{(n_a,n_b)\in \text{OS}} \text{RO}_o(n_a, n_b) + \sum_{(e_{ij},e_{kl})\in \text{IS}} \text{RO}_i(e_{ij}, e_{kl})$.

The cost function is intended to give capabilities to separatively model resource overheads. Although it is sometimes hard to determine precise reconfiguration overheads without synthesis, the cost function may represent a user's sharing tendency to specific resources over the others, as will be described in Section IV-B.

Now, we can formulate the operation and interconnection sharing problem as follows.

Definition 4: Given two DFGs, find a subset SCE of CE to maximize TS.

The traditional operation sharing problem is a subproblem of the operation and interconnection sharing problem. We can simply derive it by eliminating all dependency edges in DFGs.

IV. Algorithm For Operation and Interconnection Sharing Problem

In this section, we propose a solution for the operation sharing problem which is a subset of our problem, then extend it for operation and interconnection sharing.

A. Algorithm For Operation Sharing Problem

In Section III, we suggested that the traditional operation sharing problem can be obtained from the operation and interconnection sharing problem. The operation sharing problem between two graphs can be stated as a procedure to compare every node pair in two graphs, which is similar to finding subgraph isomorphism. In this paper, we adopt a part of the backtracking based enumeration algorithm [9]: the mapping matrix and the enumeration procedure.

Let us introduce the mapping matrix M to be a $|N_1|(\text{rows}) \times (|N_2| + 1)(\text{columns})$ matrix whose elements are 1's and 0's. If $m_{ij} \in M$ is 1, there exists a compatibility edge between $n_i \in N_1$ and $n_j \in N_2$. The single reuse constraint is modeled as a condition that any row and any column in the resulting matrix cannot contain more than one 1. If $m_{ij} = 1$ and the other elements in the same row and the same column are all 0's, then $n_i \in N_1$ is said to be *shared* with $n_j \in N_2$, which is equivalent with selecting a compatibility edge c_{ij}. To a matrix, we added a column to denote *not shared*. As an exceptional case, the added column can have more than one 1 because any node may remain unshared. The enumeration procedure now systematically changes each and every 1's in M to 0's and tests if the single reuse constraint is satisfied. In the operation sharing problem, the cost function is the number of shared nodes, which is identical to the number of selected columns.

Fig. 2(a) shows a corresponding mapping matrix to Fig. 1. Fig. 2(b) is a resulting mapping matrix that satisfies the single reuse constraint after the enumeration procedure. Remaining 1's mean shared nodes, while unshared nodes are greyed out. As shown, selecting six compatibility edges $\{c_{1a}, c_{2b}, c_{3c}, c_{4d}, c_{6e}, c_{7f}\}$ among 17 compatibility edges [the number of 1's in Fig. 2(a)] results maximum operation sharing.

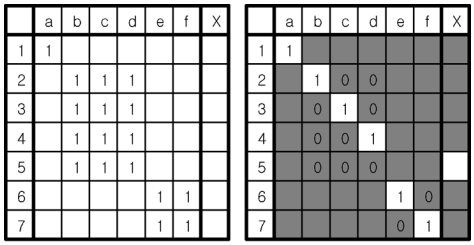


Fig. 2. Mapping matrix and best operation sharing for Fig. 1.

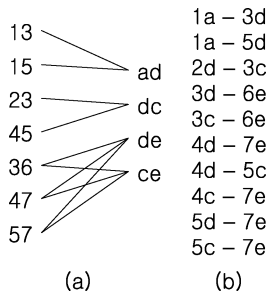


Fig. 3. Possible mapping of dependency edges and relation list of compatibility edges.

B. Extension For Interconnection Sharing

In Section III, we explained that operation sharing and interconnection sharing can be represented as selecting one and two compatibility edges, respectively. For example, c_{3d} in Fig. 1 may result an interconnection sharing of e_{13} and e_{ad} if and only if c_{1a} is selected at the same time. Without c_{1a} being selected, c_{3d} cannot initiate the interconnection sharing. To clearly identify such relations between compatibility edges, we generate all possible mapping of dependency edges between G_1 and G_2 as Moreano *et al.* did in their work [8].

Fig. 3(a) shows all possible mapping of dependency edges for Fig. 1. In Fig. 3, the possible mapping $(13,\ ad)$ means that e_{13} and e_{ad} can be shared if and only if n_1 and n_3 are reused by n_a and n_d.

Once all possible mapping of dependency edges are generated, we can identify the effect of selecting compatibility edges.

Definition 5: Given two compatibility edges, they are said to have a *relation* to each other if selecting both compatibility edges results an interconnection sharing.

For example, compatibility edges c_{1a} and c_{3d} in Fig. 1 are related, since both e_{13} and e_{ad} exist. The relation explains that *once n_1 and n_a are shared, sharing n_3 with n_d becomes more beneficial than sharing n_3 with another node*. All relations between compatibility edges can be easily found from possible mappings by the formula,

$$(n_i n_j,\ n_k n_l) \Rightarrow (n_i n_k,\ n_j n_l)$$

where $n_i, n_j \in N_1$ and $n_k, n_l \in N_2$. Fig. 3(b) shows the entire relation list for Fig. 3(a).

To deal with such relations between compatibility edges in the enumeration procedure, here we define the varying matrix VM that is enhanced from the mapping matrix M. Each element m_{ij} in VM has a relation list or a list of elements to which m_{ij} has relations. When an element m_{ij} in VM is selected, the values of elements related to m_{ij} are increased. Therefore, elements in VM may have any non-negative values as well as 0's and 1's. Positive values denote the profits of selecting elements, where larger values mean more beneficial mapping than less ones.

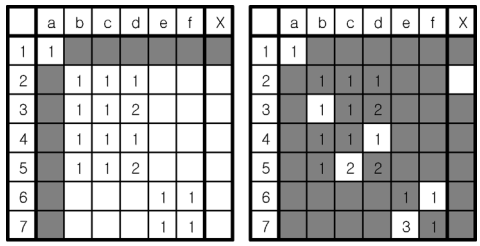


Fig. 4. Varying matrix and best mapping for Fig. 1.

Fig. 4 gives an example of VM for Fig. 1, when the configuration overheads for operations and interconnections are the same to 1. Fig. 4(a) denotes the case that $m_{1a} = 1$ is selected. Because additional selection of m_{3d} or m_{5d} will initiate interconnection sharing of e_{13} or e_{15}, the values of m_{3d} and m_{5d} are increased by 1, which is the advantage we get from interconnection sharing. Now, $m_{3d} = 2$ means that selecting c_{3d} may result not only operation sharing but also additional interconnection sharing. Fig. 4(b) shows the best result. To show the final status of VM, we do not change the unselected elements to 0's, but just grayed them out. As shown, m_{5c} is 2, m_{7e} is 3, and all the other selected elements are 1's, while row 2 remains unselected. The result is interpreted as all operations except n_2 are shared and e_{45}, e_{57}, and e_{47} are reused by e_{dc}, e_{ce}, and e_{de}.

For multi-terminal nets, it is sufficient to decompose them into two-terminal nets and to identify sharing possibilities, while we need to handle branching points of the nets in routing phase. If two nets are partially sharable, wiring points or switching matrices in any cells having branching points have to be controlled to deal with routings only for the successive temporal partitions, where such reconfigurations will result additional overheads. The overhead introduces another backend issue of *configuration sharing*. We will briefly present the issue later in Section V-C.

The proposed algorithm simultaneously performs operation and interconnection sharing, and also gracefully handles various cases of reconfiguration overhead; whether reconfiguration overhead for operations is higher or lower than that for interconnections, or how many different operation and interconnection types there are. For example, there are three different interconnection types for Pegasus intermediate representation (IR) including data, token, and predicate [10]. By differentiating the increasing amounts, we can emphasize some interconnections over the others. Moreover, with initial values in the varying matrix, reconfiguration overheads for operations can be individually parameterized. For example, logical operations for events are occasionally implemented as wires, where the initial values for the corresponding operation sharing can be set lower than the others.

About the time complexity, the worst case happens in the case that all pairs of operations are compatible, and the asymptotic

	a	b	c	d	e	f	M	T
1	1						1	12
2		1	1	1			1	11
3		1	2	2			2	10
4		1	1	1			1	8
5		1	2	2			2	7
6					2	1	2	5
7					3	1	3	3

Fig. 5. Maximum advantage matrix $A_{\max}$.

upper bound of the algorithm becomes $|N_2|(|N_1|)^{|N_2|+1}$, where $|N_2| \leq |N_1|$.

C. Conservative and Iterative Pruning Techniques

In this section, we propose conservative and aggressive pruning techniques to reduce computation time.

First of all, let us introduce a maximum advantage matrix $A_{\max}$ for the branch-and-bound scheme. $a_{ij} \in A_{\max}$ is a *maximum expected advantage* that we can expect by sharing $n_i \in N_1$ and $n_j \in N_2$. For example, $(4c, 7e)$, $(4d, 7e)$, $(5d, 7e)$, and $(5c, 7e)$ in Fig. 3(b) mean that selecting c_{7e} may result four interconnection sharing as long as the corresponding compatibility edges are selected. However, c_{4c}, c_{4d}, c_{5d}, and c_{5c} cannot be selected simultaneously since it violates the single reuse constraint. The maximum number of the expected interconnection sharing with the selection of c_{ij} can be conservatively formulated as $\min(|S_1|, |S_2|)$, where S_1 and S_2 are subsets of N_1 and N_2 that appear in the paired compatibility edges of c_{ij} in the relation list. For example, c_{7e} may induce $\min(\{n_4, n_5\}, \{n_c, n_d\}) = 2$ interconnection sharing at most. Therefore, $A_{\max}$ is constructed with the equation

$$a_{ij} = C_o + C_i * \min(|S_1|, |S_2|)$$

where C_o is an operation reconfiguration cost and C_i is an interconnection reconfiguration cost.

The maximum advantage matrix $A_{\max}$ for Fig. 1 is shown in Fig. 5. The column named M is the maximum advantage of each row. The last column named T is the sum of maximum advantages in remaining rows. The search space is pruned if the following equation is satisfied:

$$\mathrm{Adv}_{\max} > \mathrm{Adv}_{\mathrm{current}} + T_{\mathrm{remaining}}$$

where $\mathrm{Adv}_{\max}$ is a maximum advantage found at the moment, $\mathrm{Adv}_{\mathrm{current}}$ is a sum of advantages up to currently processing row, and $T_{\mathrm{remaining}}$ is the sum of maximum expected advantages in remaining rows.

In adopting the branch-and-bound scheme, if the earlier stage of the algorithm finds higher advantage $\mathrm{Adv}_{\max}$, it is possible to prune more search spaces. Since interconnection sharing is highly dependent on operation sharing, operation sharing to boost interconnection sharing, or *visiting elements in descending order of the size of relation list* may increase the probability to share more interconnections, which eventually increase the total sharing.

As an aggressive heuristic to downscale the total search space, we also designed an iterative framework that initially performs a resource sharing of important elements in the varying matrix and later matches less-important elements. An element is regarded as important if its selection is more probable to increase interconnection sharing. The enumeration algorithm processes elements having lots of relations, and resolves resource conflicts between them under the single reuse constraint. A threshold to separate elements into important and less-important ones can be empirically tuned with total number of important elements, total size of relation lists of important elements, or computation time of the enumeration procedure. Ignoring elements whose relation is less than a threshold may dramatically reduce the total search space. Once resolving conflicts between most-important elements, the algorithm processes the remaining elements, or applies another threshold.

V. Experimental Results

To demonstrate the efficiency of the proposed graph model and the algorithm in reconfiguration overhead management, we have conducted two experiments with our SW/HW co-design framework, where the frontend is based on SUIF [11] and the backend IR is similar to Pegasus [10]. The first experiment shows the effectiveness of the algorithm over a previous work in a set of randomly generated graphs, and the other employs real benchmarks to show how many resources can be shared using the algorithm. Furthermore, we also provide a preliminary result on configuration overhead reduction with our backend enhanced to deal with resource sharing.

A. Experiments on Randomly Generated Graphs

To show the effectiveness of the proposed graph model and the algorithm, we have conducted experiments using randomly generated graphs. The random graph generator was parameterized with number of nodes and probability to create compatibility edges and dependency edges. It generated 100 graphs for each experiment. As a counterpart, we also implemented Moreano's algorithm. To find a solution for maximum clique, we did not employ the same heuristic as Moreano did, but traversed all the search space. Since Moreano's algorithm only deals with interconnections, we performed additional post-pass mapping for remaining unshared operations. However, the exponential time complexity of maximum clique algorithm limited the input graph size, while our algorithm has several pruning schemes, even aggressive ones to reduce computation time. In each experiment, the sharing improvement of our algorithm against Moreano's and the reduction in search space were obtained and presented in Table I.

Table I shows the statistics of parameters and total sharing using the equation in Definition 3 $\%\mathrm{Imp}$, which denotes the relative improvement of total sharing of the proposed algorithm against Moreano's. We slightly increased C_o by 1 compared to C_i to emphasize operation sharing. #Same means the number of graphs out of 100 graphs whose total sharing is same to Moreano's. In more than half graphs, the results of two algorithm were reported the same. We reused experiment 8 for experiment 12 since parameters used in two experiments are identical.

TABLE I
IMPROVEMENT IN SHARING AND REDUCTION IN SEARCH SPACE

Exp. ID	$\|G_1\|$	$\|G_2\|$	%DE	%CE	$\|DE \in G_1\|$	$\|DE \in G_2\|$	$\|CE\|$	%Imp.	#Same	TSS	%SS
1	5	10	30	30	6.3	27.23	15.24	1.45	91	889.35	20.52
2	6	10	30	30	9.17	26.11	17.57	1.69	84	2209.52	14.64
3	7	10	30	30	12.85	26.98	20.56	2.10	80	6425.2	12.86
4	8	10	30	30	16.09	26.49	24.32	3.00	67	21240.01	9.38
5	9	10	30	30	21.31	27.08	27.06	3.22	57	53435.46	11.29
6	8	10	30	10	16.89	26.32	7.83	2.52	88	310.44	35.53
7	8	10	30	20	16.77	26.66	15.82	3.78	67	3278.94	17.51
8(12)	8	10	30	30	16.37	26.41	24.27	2.41	72	21474.19	10.83
9	8	10	30	40	16.52	26.77	32.51	1.02	86	92542.91	6.80
10	8	10	10	30	6.16	9.03	24.46	6.82	45	20649.2	6.04
11	8	10	20	30	10.92	17.64	23.75	4.20	62	18344.76	8.98
12(8)	8	10	30	30	16.37	26.41	24.27	2.41	72	21474.19	10.83
13	8	10	40	30	22.68	36.01	24.27	0.97	85	19934.13	9.05

$|G|$: average number of nodes in graph G
%DE : % probability to generate dependency edges
%CE : % probability to generate compatibility edges
$|DE \in G|$: average number of dependency edges in graph G
$|CE|$: average number of compatibility edges
%Imp. : % improvement against Moreano's
#Same : number of results among 100 graphs same to Moreano's
TSS : total search space
%SS : % of reduced search space against TSS, with conservative pruning techniques

In Table I, we can figure out some interesting relations of results to parameters. First of all, as the number of nodes increases, the number of same results decreases. This is because increasing the number of nodes generates more cases in which the preference to interconnection sharing prevents operation sharing without interconnections. However, increasing the number of compatibility edges or the number of dependency edges multiplies the number of interconnections and results that Moreano's algorithm also exploits the same sharing. Not only with high $\%\mathrm{CE}$, but sharing efficiency also decreases with low $\%\mathrm{CE}$. This is simply because no algorithm could generate any interconnection sharing or operation sharing without compatibility edges.

About the reduced search space $\%\mathrm{SS}$, it seems directly related to $\%\mathrm{CE}$. Actually, according to Table I, the total search space for low $\%\mathrm{CE}$ is very small, which makes the pruning technique not so useful and $\%\mathrm{SS}$ becomes high. Rather, we have to put more emphasis on the reason why the total space increases rapidly as $\%\mathrm{CE}$ increases. As explained in Section IV-B , the proposed algorithm consists of exploiting search space and managing relation lists. Both are proportional to density of the varying matrix, which is a function of $\%\mathrm{CE}$. Therefore, as $\%\mathrm{CE}$ increases, total search space seems to explode. This limitation is also applicable to Moreano's algorithm where the problem size of maximum clique is also exponentially proportional to the number of compatibility edges. However, even in those cases, our conservative pruning technique was efficient and successfully reduced the search space down to 6.80%.

B. Experiments on Real Benchmarks

In addition to the previous experiment on the graph model and the algorithm, we also conducted another experiment to show sharing effectiveness in real benchmarks. We used MediaBench [11] that consists of applications having mostly parallel computations and is the best match for hardware implementation or RAs.

Out of the benchmark suite, we selected several functions having interesting patterns to leverage resource sharing. Those functions are mainly composed of regular and repetitive computations of data streams, and such regularity guarantees many of operations and interconnections to be shared. Table II shows how many operations and interconnections are shared between two partitions.

TABLE II
OPERATION AND INTERCONNECTION SHARING

Bench.	Part.	Total	Op.	Int.
mpeg.a	P1	216	84	132
	P2	240	93	147
	Shared	198 (91.7%)	76 (90.4%)	122 (92.4%)
mpeg.b	P1	403	144	259
	P2	538	193	345
	Shared	282 (70.0%)	136 (94.4%)	146 (56.4%)
mpeg.c	P1	148	61	87
	P2	148	61	87
	Shared	136 (91.9%)	55 (90.2%)	81 (93.1%)
jpeg.a	P1	635	224	411
	P2	341	121	220
	Shared	201 (58.9%)	106 (87.6%)	95 (43.1%)
jpeg.b	P1	349	133	216
	P2	353	135	218
	Shared	283 (81.1%)	128 (94.8%)	155 (71.8%)
gsm	P1	215	76	139
	P2	276	99	177
	Shared	197 (91.6%)	73 (73.3%)	124 (89.2%)

In Table II, first two rows in each benchmark represent resource characteristics of two partitions, and the next row means how many resources are shared. The number of shared resources is also normalized against the smaller resources among two partitions, where 100% of sharing represents that one partition is a subset of the other. On average, the algorithm could share 80.9% of resources. For three benchmarks, mpeg.a, mpeg.c, and gsm, conservative pruning techniques using maximum advantage matrix and sorting are sufficient to traverse all the search spaces, while the others need the important-first, aggressive pruning heuristics. According to Table II, resource sharing in jpeg.a is relatively low, which is mainly due to its huge search space. The next two tables, Tables III and IV, detail benchmark characteristics and resource sharing.

Table III shows operation sharing in detail. In Table III, a column named *Data* means arithmetic and logical operations.

TABLE III
OPERATION SHARING IN DETAIL

Bench.	Part.	Total	Data	Control	Memory
mpeg.a	P1	84	23	58	3
	P2	93	30	60	3
	Shared	76	23	50	3
mpeg.b	P1	144	91	11	42
	P2	193	118	15	70
	Shared	136	86	8	42
mpeg.c	P1	61	18	23	20
	P2	61	18	23	20
	Shared	55	18	17	20
jpeg.a	P1	224	125	22	77
	P2	120	45	22	55
	Shared	106	39	12	55
jpeg.b	P1	133	82	11	40
	P2	135	84	11	40
	Shared	128	80	8	40
gsm	P1	76	31	3	42
	P2	99	37	15	47
	Shared	73	31	0	42

TABLE IV
INTERCONNECTION SHARING IN DETAIL

Bench.	Part.	Total	Data	Pred.	Token
mpeg.a	P1	132	66	30	36
	P2	147	71	40	36
	Shared	122	59	30	33
mpeg.b	P1	259	174	22	63
	P2	345	223	32	90
	Shared	146	94	22	30
mpeg.c	P1	87	46	6	35
	P2	87	46	6	35
	Shared	81	41	6	34
jpeg.a	P1	411	250	46	115
	P2	220	106	27	87
	Shared	95	25	24	46
jpeg.b	P1	216	153	3	60
	P2	218	155	3	60
	Shared	155	120	3	32
gsm	P1	139	80	0	59
	P2	177	103	4	70
	Shared	124	70	0	54

Control stands for control dependencies, speculation, and predication, and *Memory* for token operations resolving memory dependencies as well as load and store. For the benchmarks except gsm, many of control operations are reported sharable. Since control operations usually consists of operations for loop carried dependencies and loop controls, it means that those operations are reusable in successive temporal partitions which have similar loop patterns. For example, in mpeg.a whose sharing of control operations is notable, both partitions have triple-nested loops, and only the second partition has two more conditional branches that are translated into multiplexors. According to the table, memory operations seem highly reusable. It is mainly due to simple memory-related instructions. However, those sharing in memory operations does not necessarily conclude that memory patterns in both partitions are identical, as we can see in the next results about interconnection sharing.

Table IV details interconnection types in the benchmarks and sharing results. The column *Pred.* stands for predicate interconnections for control dependencies, and the column *Token* for explicit memory dependencies. In Table IV, the sharing of predicate interconnections are usually high due to similar control paths. The results for token sharing especially in mpeg.b, jpeg.a, and jpeg.b, show that their memory access patterns are different in spite of highly sharable memory operations. However, in the other three benchmarks, there exist many sharable tokens due to their similar array accesses. In jpeg.a, the data sharing looks relatively low partly because of rather different data manipulation patterns and partly because of the huge search space.

Although resource sharing highly depends on IR, partitioning, and architecture model, here are brief comparisons of the results to other studies. According to the maximum bipartite based work by Huang and Malik [7], ADPCM was described by two partitions having 178 and 111 resources, respectively. Their algorithm could share 78 resources among them, 31 operations and 47 interconnections. The result corresponds to 70.1% of resource sharing in our metric. Also, Moreano *et al.* conducted an experiment with 4 benchmarks having 104–276 resources, and reported 30%–50% of interconnection sharing [8]. In their metric, our experiment corresponds to 174–631 resources and 17.7%(jpeg.a)-87.0%(mpeg.c) interconnection sharing. In spite of differences in IR, our algorithm deals with heavier graphs, where search space grows exponentially with an input graph size, and produces results comparable to or better than the previous works.

C. Preliminary Results on Configuration Sharing

Although we have proposed the algorithm to share resources between temporal partitions, there still remain other practical issues to apply the algorithm to real architectures. Since most RAs have their own smallest addressable units, it is sometimes hard to selectively change bits of current configuration for upcoming temporal partitions. Therefore, in order to guarantee resource sharing in configuration level, it is strongly required to separate shared resources from non-shared ones. This means a synthesis process including placement and routing has to be enhanced to deal with resource sharing. Our recent research introduces *configuration sharing* that is the resource sharing problem in a configuration level [12]. In the work, we extend a traditional min-cut placer with a method to move and swap nodes between spatial partitions by which nets cut is reduced and spatial resource constraints throughout temporal partitions are satisfied. Additionally, a negotiation-based router adopts a newly designed cost function to avoid using shared resources as long as congestion and delay allow.

Fig. 6 shows the preliminary results on configuration sharing. The figure explains that the amount of configuration sharing is related to that of resource sharing. mpeg.b shows relatively low configuration sharing compared to resource sharing because of congestions. On the contrary, jpeg.a accomplishes many of resources to be reused due to free routable resources. Although we sacrificed critical path lengths to reduce configuration overhead, the results show that the resource sharing between temporal partitions is a tradeoff problem between configuration overhead and computation time, and that it may appeal to applications where loops have relatively small iteration counts compared to reconfiguration overhead or loops are unmappable onto RAs. We believe that there remain additional issues of adjusting and improving placement and routing algorithms to deal with resource sharing between consecutive partitions.

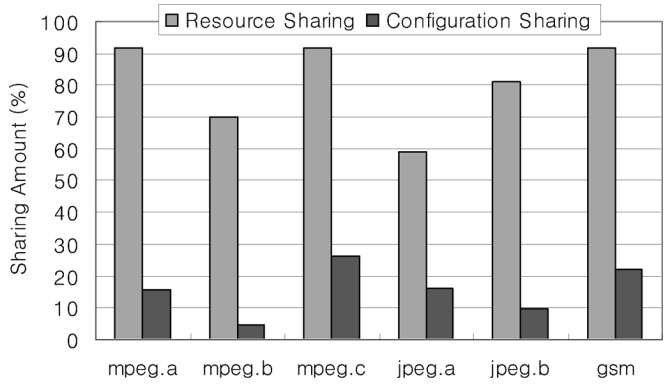


Fig. 6. Preliminary results on configuration sharing.

VI. Conclusion

In this paper, we addressed the operation and interconnection sharing problem, and proposed a graph model. The graph model is solved by the varying matrix that is enhanced from the mapping matrix in graph isomorphism. Furthermore, the varying matrix enables us to individually parameterize reconfiguration overhead for operations and interconnections. To reduce computation time, we also presented the conservative techniques and the iterative framework. The results obtained from generated graphs report that the proposed model and the algorithm outperform the previous work by up to 6.82% when overhead of reconfiguring operations is higher than that of reconfiguring interconnections. The experiment with benchmarks shows the algorithm could share 80.9% resources on average. Finally, our on-going research including back-end tools reveals the resource sharing scheme could eventually decrease the configuration overhead.

References

[1] T. Callahan, J. Hauser, and J. Wawrzynek, "The garp architecture and C compiler," *IEEE Computer*, vol. 33, no. 4, pp. 62–69, Apr. 2000.

[2] H. Singh, M.-H. Lee, G. Lu, N. Bagherzadeh, F. J. Kurdahi, and E. M. C. Filho, "MorphoSys: An integrated reconfigurable system for data-parallel and computation-intensive applications," *IEEE Trans. Comput.*, vol. 49, no. 5, pp. 465–481, May 2000.

[3] Z. Li, "Configuration management techiniques for reconfigurable computing," Ph.D. dissertation, Northwestern Univ., Evanston, IL, 2002.

[4] Xilinx, San Jose, CA, "Virtex-II Pro Platform FPGAs: Complete data sheet," 2004.

[5] V. Baumgrarte, F. May, A. Nückel, M. Vorbach, and M. Weinhardt, "PACT XPP – A self-reconfigurable data processing architecture," in *Proc. Int. Conf. Eng. Reconfigurable Syst. Algorithms*, 2001, pp. 64–70.

[6] J. M. P. Cardoso, "Loop dissevering: A technique for temporally partitioning loops in dynamically reconfigurable computing platforms," in *Proc. 17th Int. Symp. Parallel Distrib. Process. (IPDPS)*, 2003, p. 181.2.

[7] Z. Huang and S. Malik, "Managing dynamic reconfiguration overhead in systems-on-a-chip design using reconfigurable datapaths and optimized interconnection networks," in *Proc. Conf. Des., Autom. Test Eur. (DATE)*, 2001, p. 735.

[8] N. Moreano, G. Araujo, Z. Huang, and S. Malik, "Datapath merging and interconnection sharing for reconfigurable architectures," in *Proc. 15th Int. Symp. Syst. Synth. (ISSS)*, 2002, pp. 38–43.

[9] J. R. Ullmann, "An algorithm for subgraph isomorphism," *J. ACM*, vol. 23, no. 1, pp. 31–42, 1976.

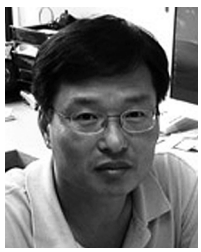
[10] M. Budiu, "Spartial computation," Ph.D. dissertation, Comput. Sci. Dept., Carnegie Mellon Univ., Pittsburgh, PA, 2003.

[11] C. Lee, M. Potkonjak, and W. H. Mangione-Smith, "Mediabench: A tool for evaluating and synthesizing multimedia and communicatons systems," in *Proc. MICRO*, 1997, pp. 330–335.

[12] S. Jung and T. G. Kim, "Configuration sharing to reduce reconfiguration overhead using static partial reconfiguration," *IEICE Trans. Inf. Syst.*, to be published.

Sungjoon Jung (S'05–M'08) received the B.S., M.S., and Ph.D. degrees in electrical engineering and computer science from Korea Advanced Institute of Science and Technology (KAIST), in 2000, 2002, and 2008, respectively.

He is currently working at KAIST as a Post-Doctoral Researcher. His research interests include a conventional software compiler and a hardware compiler as well as synthesis algorithms for reconfigurable architectures.



Tag Gon Kim (SM'95) received the Ph.D. degree in computer engineering with a specialization in systems modeling/simulation from the University of Arizona, Tucson, in 1988.

In Fall, 1991, he joined the Electrical Engineering Department, KAIST, Daejeon, Korea, as an Assistant Professor and has been a Full Professor with the Department of Electrical Engineering and Computer Science since Fall, 1998. Between 1980 and 1983, he was a full-time Instructor with the Communication Engineering Department, Bookyung National University, Pusan, Korea, and from 1989 to 1991, he was an Assistant Professor with the Electrical and Computer Engineering Department, University of Kansas, Lawrence. His research interests include methodological aspects of systems modeling simulation, analysis of computer/communication networks, and development of simulation environments. He has published more than 150 papers on systems modeling, simulation, and analysis in international journals/conference proceedings. He is a coauthor (with B. P. Zeigler and H. Praehofer) of *Theory of Modeling and Simulation (2nd ed.)* (Academic Press, 2000).

Dr. Kim was the Editor-in-Chief of *SIMULATION: Transactions of SCS* published by the Society for Computer Simulation International (SCS). He is a member of Eta Kappa Nu.