Fast, Nearly Optimal ISE Identification With I/O Serialization Through Maximal Clique Enumeration

Ajay K. Verma, Philip Brisk, *Member, IEEE,* and Paolo Ienne, *Member, IEEE*

***Abstract*—The last decade has witnessed the emergence of the application-specific instruction-set processor (ASIP) as a viable platform for embedded systems. Extensible ASIPs allow the user to augment a base processor with instruction set extensions (ISEs) that execute on dedicated hardware application-specific functional units (AFUs). Due to the limited number of read and write ports in the register file of the base processor, the size and complexity of AFUs are generally limited. Recent papers have focused on overcoming these constraints by serializing access to the register file. Exhaustive ISE enumeration methods are not scalable and generally fail for larger applications and register files with a large number of read and write ports. To address this concern, a new approach to ISE identification is proposed. The approach presented in this paper significantly prunes the list of the best possible ISE candidates compared to previous approaches. Experimentally, we observe that the new approach produces optimal results on larger applications where prior approaches either fail or produce inferior results.**

***Index Terms*—Custom ISE identification, I/O access serialization, maximal clique.**

I. Introduction and Goal

APPLICATION-specific instruction set processors (ASIPs) are commonly used in embedded systems. An ASIP is a lightweight general-purpose embedded processor, typically a reduced instruction set computer (RISC) that has been augmented with application-specific hardware functionality in the form of instruction set extensions (ISEs). The design and verification processes for ASIPs are lower than for application-specific integrated circuits (ASICs), because the same base processor can be used across a variety of product lines. To further reduce the design effort for ASIPs, algorithms to automatically identify ISEs for a set of applications have been proposed; after the ISE is identified, it is synthesized as a hardware application-specific functional unit (AFU), which is then interfaced with the hardware description of the processor.

This paper advances the state-of-the-art in ISE identification and AFU synthesis in several respects. Firstly, we prove that the optimal ISE for a processor can be identified using a speedup model that is independent of the details of the execution pipeline of the processor. Our method relies on two assumptions: that the base processor is a RISC, meaning that it is single-issue and does not perform out-of-order execution or register renaming, and that an AFU is never slower than a software implementation of an ISE. The latter assumption generally does not hold for systems where the processor is implemented in fixed logic and the ISEs are synthesized on a slower reconfigurable datapath; however it does hold for ASIC technologies.

Previous ISE identification methods [2]–[8] use the input/output (I/O) bandwidth of the register file to limit the number of inputs and outputs of a permissible ISE; this yielded effective pruning criteria that reduced the size of the search space for ISE identification. Pozzi and Ienne [9] developed a technique for multicycle I/O serialized ISE execution: during each cycle, data is read from and written to the base processor's register file. Under this execution model, there is no need to limit the number of inputs and outputs of each ISE. Unfortunately, this eliminates the most effective pruning criteria that allowed previous ISE enumeration methods to converge.

As a second contribution we show, assuming I/O serialized ISE execution that we only need to consider only maximal ISEs. The size of ISEs are generally limited by "forbidden nodes," i.e., operations that are not permitted to be moved into customized hardware. Typically, forbidden nodes include memory accesses, instructions such as branches that modify the program counter, and variable latency operations such as floating-point arithmetic or hardware division.

A third contribution is a faster heuristic for I/O serialization compared to the one proposed by Pozzi and Ienne [9]. Due to the high runtime cost of I/O serialization, Pozzi and Ienne used a single-cycle speedup model to identify the best ISE, and then serialize its I/O access; however, after serialization, the ISE may no longer be optimal.

We have discussed earlier that at least one maximal ISE will maximize the overall speedup; however, a smaller ISE may also have the same speedup as the best ISE. We propose an approach that finds the smallest ISE contained within a larger ISE that has the same speedup as the larger ISE. Empirically, we show that maximal ISEs result only in a marginal area penalty.

A. Limitations of Previous Approach

The previous approach to ISE generation, shown in Fig. 1, is problematic for several reasons: First, I/O serialization changes the optimality criteria for ISEs, as shown in Fig. 2. The

Manuscript received July 29, 2009. Current version published February 24, 2010. This paper was recommended by Associate Editor P. Eles.

A. K. Verma and P. Ienne are with the Processor Architecture Laboratory, Ecole Polytechnique Fédérale de Lausanne, Lausanne CH-1015, Switzerland (e-mail: ajaykumar.verma@epfl.ch; paolo.ienne@epfl.ch).

P. Brisk is with the Department of Computer Science and Engineering, University of California, Riverside, CA 92507 USA (e-mail: philip.brisk@epfl.ch).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TCAD.2010.2041849

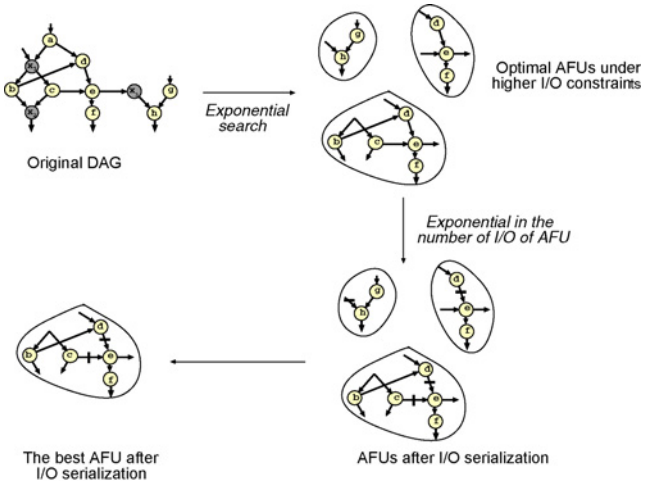


Fig. 1. Overall description of the previous approach for ISE identification.

software latency of each node is 1.0 cycles, and the timing constraint requires that each execution stage have a period of no more than 1.0 cycles. The optimal ISE, using the original method, is a path containing six nodes, where nodes have alternating delays of 0.5 and 0.6 cycles. The suboptimal ISE has two nodes, of delays 0.2 and 0.3 cycles. Due to the timing constraint, the only way to implement the optimal ISE is to create an AFU with six execution stages and a latency of 6 cycles: the same as software execution. In the suboptimal ISE, both nodes can form a single-cycle AFU, compared to a 2-cycle latency in software.

Second, the I/O serialization algorithm of Pozzi and Ienne [9] repeats the enumeration process for all I/O constraints from $(2, 1)$ to (N, N), where N is the number of nodes in the directed acyclic graph (DAG). Due to runtime considerations, the search is stopped at $(10, 5)$. Both the ISE search and I/O serialization algorithm are exponential in the number of input and output nodes of the DAG.

The third problem is that if the software/hardware latency of some instruction changes, the result of the ISE enumeration and AFU generation algorithm is no longer guaranteed to be optimal. For example, a custom instruction that is optimal for a speedup model based on 0.18 μm complementary metal oxide semiconductor (CMOS) technology may become sub optimal for 0.13 μm CMOS technology. One must either repeat the entire process or accept a potentially sub optimal solution. The approach presented here, in contrast, finds a sparse set of potential ISEs. Irrespective of the specific technology used, the optimal ISE is always included in this set.

B. Outline of the Presented Approach

Fig. 3 illustrates the new approach for ISE identification. The algorithm has seven main steps.

1) *Step 1 (Clustering):* Nodes are grouped into equivalence classes based on assumptions regarding the speedup model, which will be discussed in Section IV. If nodes x and y belong to the same equivalence class C, then any ISE that includes x but not y, or vice versa, is sub-optimal. Note that, this step depends only on the topology of the input data flow graph.

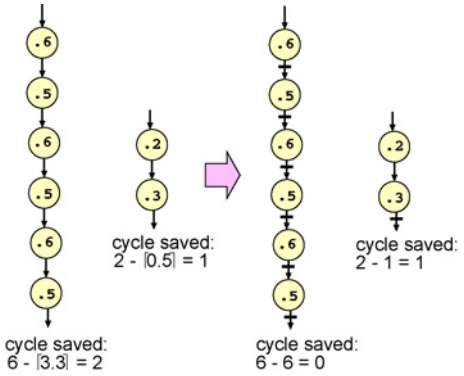


Fig. 2. Example showing that the relative merit of AFUs might change after I/O serialization. The software latency of each node is 1.0.

2) *Step 2 (Cluster Graph Construction):* Each equivalence class is compressed into a single vertex. An edge is placed between every pair of equivalence classes that could be merged into the same ISE without violating convexity constraints. The resulting undirected graph, which is called a *Cluster Graph*, has fewer vertices and edges than the original DAG. This reduces the ISE identification search space. Similar to the previous step, this step also depends only on the topology of the input DAG.
3) *Step 3 (Pruning):* This step is optional. Nodes and edges from the cluster graph that can be removed without affecting the optimal ISE are removed to further reduce the search space. This step depends on the specific details of the processor.
4) *Step 4 (Clique Enumeration):* Each clique in the cluster graphs corresponds to a potentially optimal ISE. We exhaustively enumerate all maximal cliques in the cluster graph in search of the optimal ISE. In practice, clique enumeration is faster than subgraph enumeration of the original DAG [4], [5]. This step depends only on the topology of cluster graph, and is independent of the specific details of processor/technology used.
5) *Step 5 (Clique Pruning):* This step is optional. Properties of the base processor and speedup model are used to bound the speedup of each ISE. Based on these bounds, some ISEs can be discarded without explicitly computing their speedups.
6) *Step 6 (I/O Serialization):* This step serializes the I/O accesses of the ISEs corresponding to each remaining clique. The ISE whose speedup is maximal is then selected.
7) *Step 7 (Area Optimization):* The final step is to reduce the size of the chosen ISE, i.e., to find a smaller ISE that offers the same speedup.

This process repeats to select multiple ISEs from the same DAG. Each ISE that is identified is replaced with a single node which is marked as forbidden; this prevents overlapping ISEs.

II. Related Work

Most prior techniques for ISE generation use the number of I/O ports of the register file to constrain the set of subgraphs

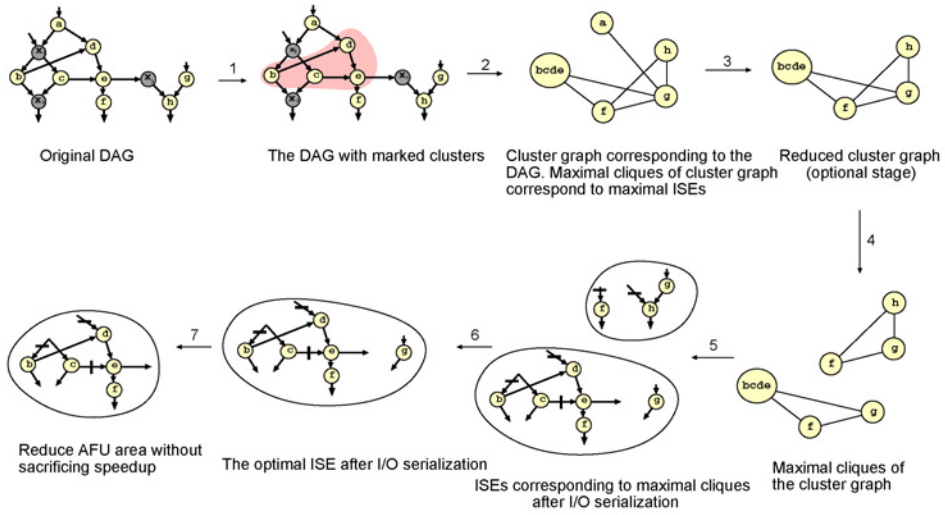


Fig. 3. Summary of our approach to optimal ISE identification and serialization.

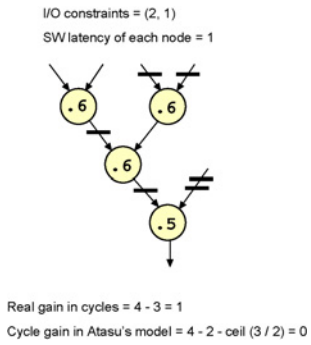


Fig. 4. Example showing that the penalty for I/O accesses is not constant, and can be zero in some cases.

that can be enumerated [2]–[8]; however, Pozzi and Ienne's [9] I/O serialization approach eliminates the need for these constraints.

Pothineni *et al.* [10] make similar assumptions regarding the monotonicity of convex subgraphs as we do; however, their algorithm considers only connected subgraphs as potential AFUs. They use an unconstrained MaxMIMO algorithm to enumerate ISEs. Afterward, they serialize the I/O accesses of the MaxMIMOs using the algorithm of Pozzi and Ienne. Rather than searching for the best ISE, they find an overlapping set of ISE candidates and then select a nonoverlapping subset using a covering algorithm similar in principle to Guo *et al.* [11]. Pothineni *et al.* do not describe any techniques that are comparable to our methods for clustering and clique enumeration.

Atasu *et al.* [12] have presented an algorithm based on *Integer Linear Programming (ILP)* to solve the same problem with relaxed I/O constraints. However, for each extra I/O access they impose a constant penalty on hardware latency. This leads to an inaccurate calculation of the speedup, because some I/O accesses can be done in parallel to execution stages resulting in no penalty. An example is shown in Fig. 4, where the ISE has five inputs and one output. Under register file I/O constraints of (2, 1), there are three extra inputs, hence Atasu's algorithm will impose a penalty of $\lceil \frac{3}{2} \rceil = 2$ cycles on hardware latency. This will result in a gain of zero cycles; however, the extra inputs can be accessed in parallel to execution stages, resulting in a hardware latency of 3 cycles, i.e., a gain of 1 cycle. We have presented an ILP [13] based on ILP similar to Atasu's, but with an accurate measure of cycle gain; it provides optimal solutions, but runs considerably slower than the method presented here.

The cluster graph described in this paper has some similarities to the all pairs common slack graph (APCSG) described by Brisk *et al.* [14]. The difference is that APCSG edges are placed between operations that can be scheduled in parallel. Their approach is not optimal and focuses primarily on finding very long instruction word (VLIW)-style parallel instructions.

Other ISE generation techniques have discarded I/O constraints for reasons unrelated to I/O serialization. Kastner *et al.* [15], for example, find ideal IP blocks to integrate into a reconfigurable fabric. Likewise, Cadambi and Goldstein [16] build a macro-generator library for FPGAs. These techniques do not extend a base processor with constraints on the register file. Kingshuk *et al.* [17] advocate the use of VLIW-style clustered register files to increase the bandwidth to an AFU; however, this creates a coalescing problem, as inter-cluster register transfers must be minimized.

III. Problem Statement

Each basic block can be represented as a DAG $G = (V, E)$ where nodes correspond to primitive operations (e.g., ADD, MUL, LOAD) and edges correspond to data dependencies between operations. We can extend G into a larger DAG, $G^+ = (V \cup V^+, E \cup E^+)$, where V^+ is the set of inputs and outputs of the basic block, and E^+ is the set of edges connecting vertices in V and V^+. To simplify notation, we will henceforth use G in place of G^+.

Along with G, we are given a subset $F \subseteq V$ of forbidden nodes that cannot be included in any ISE. Initially, forbidden

nodes correspond to operations such as LOAD, STORE, and JUMP, which require access to main memory or modify the program counter. F also includes variable-latency operators such as hardware division and floating-point arithmetic. If the ISE generation algorithm is run multiple times on the same DAG, then already identified ISEs are also marked as forbidden.

For each node $u \in V$, there are two positive real values SW_u and HW_u, which are the latencies of u implemented in software (on the base processor) and hardware. A convex subgraph $S \subseteq V$ is a subset of nodes, such that for every pair of nodes $u, v \in S$, every path from u to v in G consists solely of nodes in S. For a subgraph S, $SW(S)$ and $HW(S)$ are the latencies of S when implemented in hardware and software, respectively.

The total number of cycles saved by implementing S as an ISE is $M(S) = N_S(SW(S) - HW(S))$, where N_S is the number of times the basic block is executed. Since N_S is the same for all ISEs in the basic block, it will not affect the relative merit of the ISEs in a basic block. The ISE generation problem is to find a convex subgraph S of G that contains no forbidden nodes and maximizes $M(S)$. In general, both $SW()$ and $HW()$ are processor-specific functions. For example, in a RISC processor, $SW(S)$ can be approximated by adding the software latencies of all nodes in S, as follows:

$$SW(S) = \sum_{u \in S} SW_u.$$

$HW(S)$, in contrast, is dependent on specific issues of AFU synthesis (algorithms, ASIC versus. FPGA, etc.). Clearly, $HW(S)$ depends also on the available I/O ports between the AFU that implements S as an ISE and the register file in the base processor. If m and n are the number of input and output ports of the register file, then $HW(S)$ can be computed by serializing the I/O accesses of the AFU S under I/O constraints (m, n) [9].

I/O serialization is a complicated problem. Let $G_S = (S, E_S)$ be the subgraph of G induced by S. We extend G_S with two additional nodes, v_{src} and v_{sink}, which are connected to all of the inputs and outputs of S, respectively, by edge sets E_{src} and E_{sink}. Let $S^+ = S \cup v_{\text{src}} \cup v_{\text{sink}}$ and $E_S^+ = E_S \cup E_{\text{src}} \cup E_{\text{sink}}$. I/O serialization is then applied to the resulting DAG $G_S^+ = (S^+, E_S^+)$. Let R denote the total latency of G_S^+ after I/O serialization, which is achieved by inserting registers on the edge set E_S^+. Let $\rho(u, v)$ denote the number of registers inserted onto edge (u, v).

Given these definitions, I/O serialization can be formulated as an optimization problem [9]:

Problem 1: Minimize R under the following constraints.

1) *Latency Constraints:* The maximum register-to-register latency of the circuit cannot exceed λ, a user-specified clock period. For any path p through G_S^+ whose edges contain no registers, the sum of the hardware latencies of the operations in p cannot exceed λ.
2) *Legality:* All operands of a node must arrive at the same time. In other words, for any node $v \in S^+$, all paths from v_{src} to v must contain the same number of registers. The number of registers on any path between v_{src} and v_{sink} will be $R - 1$ (yielding R execution stages).
3) *I/O Serialization:* At any cycle, at most m inputs can be read from the register file and at most n outputs can be written back. Formally, let $S_{\text{in}}(i)$ be the set of input nodes whose incoming edges have exactly i registers, meaning that these nodes read their values from the register file at the ith clock cycle. Likewise, let $S_{\text{out}}(j)$ be the set of output nodes whose outgoing edges have exactly j registers. Then $| S_{\text{in}}(i) | \leq m$ and $| S_{\text{out}}(j) | \leq n$.

The optimal value of R corresponds to $HW(S)$. Consequently, one must solve the I/O serialization problem optimally in order to solve the ISE generation problem optimally as well.

IV. Algorithms for ISE Generation and I/O Serialization

If there are no inherent assumptions regarding the performance model of the base processor and AFUs, then it is impossible to compare the merits of two different ISEs, S_1 and S_2, without comparing their speedups, i.e., computing $M(S_1) - M(S_2)$. Without assumptions, the only feasible approach for ISE generation is to enumerate all convex subgraphs, compute their speedups, and choose the best one; however, the speedup model is not a random function, and it is certainly reasonable to make assumptions about it, as long as the properties used to justify the assumptions are relatively safe. The algorithms for ISE generation presented here exploit these properties to reduce the size of the search space significantly.

A. Monotonicity of Speedup Model

The most important property of the speedup model is *Monotonicity*, and is defined as follows.

Definition 1: A speedup model is monotonic, if for any two convex subgraphs S_1 and S_2

$$(S_1 \subseteq S_2) \Rightarrow M(S_1) \leq M(S_2).$$

Theorem 1, which follows, shows that the speedup model for RISC processors is monotonic; and that its monotonicity is independent of the hardware and software latencies of operations.

Theorem 1: The speedup model for ISE generation for RISC processor is monotonic, under the assumption that for any convex subgraph S, $SW(S) \geq HW(S)$.

Proof: Let S_1 and S_2 be convex subgraphs of G such that $S_1 \subset S_2$. S_2 can be obtained from S_1 by some sequence of the following three operations.

1) Choose a convex subgraph T, such that $T \cap S_1$ is empty and there are no paths from any node in S_1 to a node in T, and vice versa. Let $S_1' = S_1 \cup T$.
2) Choose a node v from outside S_1 such that v is the predecessor of at least one of the nodes in S_1. Let $P(v, S_1)$ be the subgraph of G induced by the set of nodes on all paths from v to nodes in S_1 and let $S_1' = S_1 \cup P(v, S_1)$.
3) Choose a node v from outside S_1 such that v is the successor of at least one of the nodes in S_1. Let $P(S_1, v)$ be the subgraph of G induced by the set of nodes on all paths from nodes in S_1 to v, and let $S_1' = S_1 \cup P(S_1, v)$.

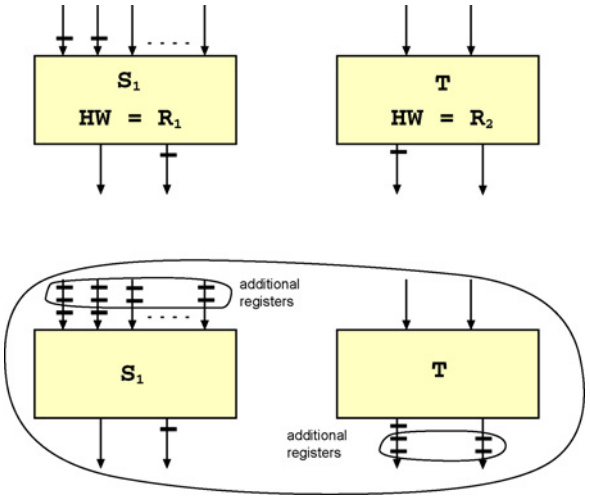


Fig. 5. Disjoint union of two convex subgraphs increases the speedup.

Since S_1 is convex, S_1' must also be convex if it is constructed using these three operations. Given S_1 and S_2, as described above, we can construct S_2 from S_1 by repeatedly applying these three operations, yielding the following sequence of subgraphs:

$$S_1 = S^{(0)} \subset S^{(1)} \subset \cdots \subset S^{(k)} = S_2.$$

We prove that none of these transformations reduces the speedup of resulting subgraph.

Consider the first transformation where $SW(S_1)$ and $SW(T)$ are the software latencies of AFUs corresponding to subgraphs S_1 and T. After I/O serialization, let R_1 and R_T be the number of registers inserted for S_1 and T respectively. Now we serialize the I/O accesses of S_1'.

One possibility is to add R_T additional registers to each incoming edge of inputs to S_1 and R_1 additional registers on each outgoing edge of outputs of T, as shown in Fig. 5. We argue that no more than m input nodes of S_1' have the same number of registers on their incoming edges. This constraint is already satisfied among the input nodes of S_1 and T individually. Since $HW(T) \leq R_T$, each input of T has fewer than R_T registers on its incoming edges. Each input of S_1 has at least R_T registers on its incoming edges. This means that no input of S_1 can have the same number of registers on its incoming edge as any input of T. This ensures that the input constraint is satisfied for S_1'. Likewise, the analogous argument holds for the outputs of S_1' as well.

For S_1', the number of registers on every path from v_{src} to v_{sink} is $R_1 + R_T$. This ensures that $M(S_1') \geq SW(S_1) + SW(T) - (R_1 + R_T) \geq M(S_1) = SW(S_1) - R_1$.

Now, consider the second transformation and assume that there are t paths from v to S_1. Consider the set $X = \{v_1, v_2, \ldots, v_r\}$ of nodes on these paths that are directly connected to the nodes of S_1. The smallest convex graph H containing all of the nodes in X is the union of the t paths. Let $HW(H)$ be the number of registers inserted after H. Under the assumption of monotonicity, $HW(H) \leq SW(H)$.

Once again, consider supergraph S_1'. Place $HW(H)$ additional registers on each incoming edge of the inputs of S_1 and $HW(S_1)$ addition registers on the outgoing edges of outputs of H that are not connected to inputs of S_1. By applying the same argument as above, this ensures that no more than m input nodes of S_1' will have the same number of registers on their incoming edges, and no more than n output nodes of S_1' will have the same number of registers on their outgoing edges.

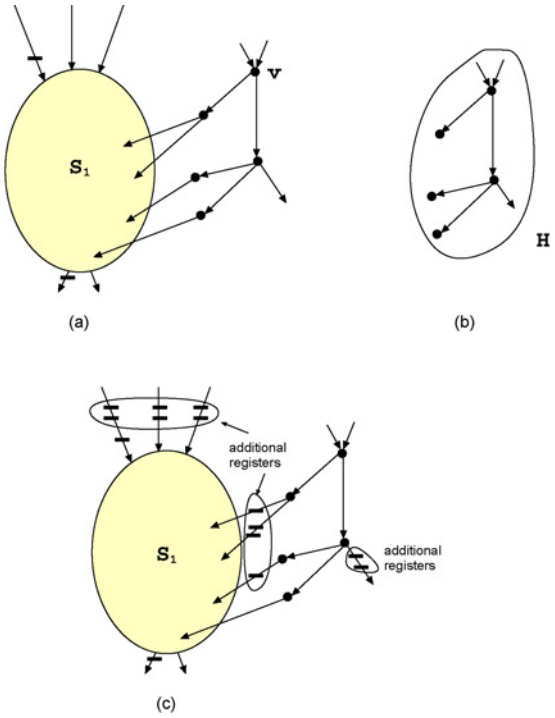


Fig. 6. Adding a single node to a convex subgraph and all paths from this node to the subgraph increases the speedup. (a) Addition of all paths from a node to original subgraph. (b) Graph corresponding to the added node. (c) Resulting ISE after I/O serialization.

Now, let $N(S_1)$ be a set of nodes in S_1 that are directly connected to nodes in H. This adds extra paths from v_{src} to nodes in $N(S_1)$, which come via H. To satisfy the legality constraint for these nodes, we must insert extra registers on some edges. Note that all paths from v_{src} to nodes in $N(S_1)$ that pass through S_1 will have at least $HW(H)$ registers; however, all paths from v_{src} to nodes in $N(S_1)$ that pass through H will have fewer than $HW(H)$ registers. It suffices to insert additional registers on the edges between nodes in H and nodes in S_1 to satisfy the legality constraint. This does not affect the quality of the original implementation because these edges do not exist in S_1 or H alone. It is important to note that the addition of these registers do not increase the maximum number of registers on all paths from v_{src} to v_{sink}.

In the entire procedure, we have added only $HW(H)$ extra registers to each path through S_1 and $HW(S_1)$ extra registers on each path through H. Thus, the number of registers from v_{src} to v_{sink}, e.g., $HW(S_1')$ cannot exceed $HW(S_1) + HW(H)$. As discussed above, the registers added on the paths containing nodes from both H and S_1 do not increase the number of registers between v_{src} and v_{sink}. Thus, $M(S_1') \geq SW(S_1) + SW(H) - (HW(S_1) + HW(H)) \geq M(S_1)$. Fig. 6 illustrates the key points of the proof in this case.

The proof for the third transformation is analogous to the proof of the second and has been omitted to conserve space. ■

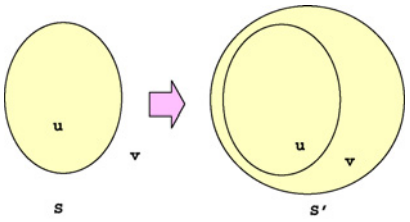


Fig. 7. Two nodes are related if any convex subgraph containing one of them can be extended to a convex subgraph containing both of them.

The above discussion has not considered interconnect delays because they are complicated to model, and they depend on several parameters other than the topology of the circuit. Theoretically, it is possible that the monotonicity property may not hold after considering interconnect. However, in practice this phenomenon occurs only for ISEs which are close to each other in terms of size and software latency, because $M(S)$ is dominated by the software latency, which is usually $10\times$ greater than the hardware latency. After considering the interconnect the hardware latency of an ISE may increase by a few cycles, but is unlikely to reduce $M(S)$ significantly. Also, the hardware execution of an ISE, the number of stages are mainly governed by the serialization of the I/O access rather than the critical path delay of the ISE. We have verified empirically that interconnect delays do not affect monotonicity for the benchmarks we have studied.

B. Reducing the DAG Size via Clustering

Here, we show how to generate maximal subgraphs that are *Valid*, meaning they are convex and contain no forbidden nodes. We want to reduce the size of the original DAG in such a way that the set of maximal valid subgraphs is preserved. We find pairs (or sets) of certain nodes that will always occur together in maximal valid subgraphs. Such nodes are said to be *Clusterable*. We cluster these nodes together into a single node, reducing the number of nodes and edges in the DAG.

1) *Equivalence Relationship Between Clusterable Nodes:* We define a relation "$\sim$" between two nodes u and v, such that $u \sim v$ is true if u and v are clusterable, and false otherwise. This relation is formalized as follows:

Definition 2: For two nodes u and v in G, $u \sim v$ if and only if any valid subgraph containing u or v can always be extended to a valid subgraph containing both u and v. Fig. 7 illustrates the preceding concept.

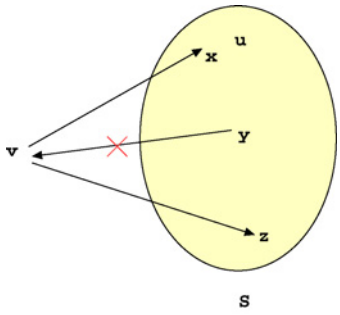
It is trivial to see that the relationship is reflexive ($u \sim u$) and symmetric ($u \sim v \Rightarrow v \sim u$). To prove that $\sim$ is an equivalence relation, we must also prove that it is transitive.

Theorem 2: The relation $\sim$ is a transitive relation.

Proof: Let u, v, and w be nodes in G, and assume that $u \sim v$ and $v \sim w$. To establish transitivity, we must prove that $u \sim w$. Assume that there is a valid subgraph S that contains u, but not w. If $v \in S$, then S can be extended to a valid subgraph S' such that $w \in S'$, since $v \sim w$. If $v \notin S$, then S can be extended to a new subgraph S' such that $v \in S'$ since $u \sim v$. Since $v \in S'$, then S' can be extended to a new subgraph S'' such that $w \in S''$ since $v \sim w$. The same argument can be used to show that any subgraph S containing w can be extended to contain both u and w. ■

```
checkMembership (node u, node v, set F) {
// The function takes two nodes u,v and the set of
// forbidden nodes F and decides whether v ∈ P(u).
  (Pred_u, Pred_v) = (predecessors(u), predecessors(v));
  (Succ_u, Succ_v) = (successors(u), successors(v));
  if (({u, v} ∩ F ≠ ∅) or (Pred_u ∩ Succ_v ∩ F ≠ ∅) or
  (Succ_u ∩ Pred_v ∩ F ≠ ∅))
    return false;
  return true; }
```

Fig. 8. Algorithm to test the membership of v in $P(u)$.



Since S is convex, all paths between x and y must be inside S

Fig. 9. All paths from a node to a convex subgraph must be monodirectional.

Theorem 2 shows that we can partition the nodes of a DAG into equivalence classes, defined by the relation $\sim$. If C is an equivalence class, any maximal valid subgraph must contain either all nodes in C, or none. This implies that the nodes of an equivalence class can be clustered into a single node, thereby reducing the number of nodes and edges in the DAG. The clustering transformation is analogous to the edge contraction method described by Kastner *et al.* [15].

2) *Determining the Equivalence Relation:* We desire an efficient algorithm to evaluate $u \sim v$.

Definition 3: The *Consistent Set* $P(u)$ of a node u is the set of all nodes x, such that there exists at least one valid subgraph containing both u and x. If $v \notin P(u)$, then no valid subgraph can contain both u and v.

If f is a forbidden node, then $P(f)$ is empty. If u and v are not forbidden, and there is no path from u to v as well as from v to u, then $S = \{u, v\}$ is a valid subgraph, i.e., $v \in P(u)$ and vice versa.

Now, let there be at least one path from u to v. If there is a forbidden node along at least one path, then $v \notin P(u)$; if there are no forbidden nodes, then the subgraph corresponding to the union of all of these paths will be a valid subgraph containing both u and v, hence $v \in P(u)$ and vice versa. Based on these observations, an algorithm to determine membership of v in $P(u)$ is given in Fig. 8. Theorem 3, which follows, provides the foundation to determine the relationship $\sim$ efficiently.

Theorem 3: For nodes u and v, $u \sim v \Leftrightarrow P(u) = P(v)$.

Proof: Assume that $P(u) \neq P(v)$. Then there is a node x such that $x \in P(u)$ and $x \notin P(v)$, or vice versa. If $x \in P(u)$, then there is a valid subgraph S containing u and x. Since $x \notin P(v)$, v cannot be included in S, so $u \not\sim v$.

Now, assume that $P(u) = P(v)$. Consider a valid subgraph S, such that $u \in S$ and $v \notin S$. For each node $s \in S$, $s \in P(u)$ by definition. Since $P(u) = P(v)$, $S \subseteq P(v)$. Now, consider all

paths between v and the nodes in S. Since S is convex, these paths are monodirectional, i.e., all of them will be from v to S, or from S to v; this is illustrated in Fig. 9. Moreover, none of these paths can contain a forbidden node, since $S \subseteq P(v)$. If we include all of the nodes of these paths in S, we have a convex graph with no forbidden nodes that contains both u and v. In other words, any valid subgraph containing u or v, but not both, can be extended to a valid subgraph containing u and v. ■

Theorem 3 proves that one can easily find the clusters in an original DAG. First, the consistent set $P(u)$ is found for every node u. Then all of the nodes having the same consistent set are put into the same cluster. Since all forbidden nodes have a null consistent set, they will be placed in a single cluster, which henceforth will be called a *Forbidden Cluster*. A maximal subgraph is found by taking the union of a subset of clusters.

C. Cluster Graph and Its Maximal Cliques

Definition 4: The cluster graph $\mathcal{C}(G)$ of a DAG G is an undirected graph whose nodes correspond to the nonforbidden clusters of G, and an edge is placed between nodes corresponding to clusters C_1 and C_2 if and only if there exist two nodes u and v in G such that $u \in C_1$, $v \in C_2$ and $v \in P(u)$.

Since all nodes in a cluster have the same consistent sets, all nodes in that cluster will be consistent with all of the nodes in a neighboring clusters. Likewise, if two clusters are not neighbors in the cluster graph, then there cannot be any valid subgraph containing nodes from both clusters.

Theorem 4: Each maximal valid subgraph of G corresponds to a maximal clique in the cluster graph and vice versa.

Proof: Applying Theorem 3, suppose that a valid subgraph S corresponds to the union of clusters $C_1, \ldots, C_m$. Since all nodes in a valid subgraph are consistent with one another, for any two nodes u and v in the subgraph: $v \in P(u)$ and $u \in P(v)$. If $u \in C_i$ and $v \in C_j$ then there is an edge between C_i and C_j in the cluster graph, so $C_1, \ldots, C_m$ is a clique.

To establish the converse, assume that $C_1, \ldots, C_m$ is a maximal clique in the cluster graph, and that the corresponding subgraph in G is nonconvex. Then there must be two nodes $u \in C_1$ and $v \in C_2$ in the subgraph such that there is a path from u to v that includes nodes lying outside of the subgraph. Let $w \in C_{m+1}$ be a node on this path that lies in a cluster outside of the maximal clique.

Since C_1 and C_2 are connected by an edge in the cluster graph, u and v must be consistent, i.e., all paths between u and v contain no forbidden nodes; thus, all paths from u to w and from w to v contain no forbidden nodes as well. In other words, $u \in P(w)$ and $v \in P(w)$, indicating the presence of edges (C_1, C_{m+1}) and (C_2, C_{m+1}) in the cluster graph. Now, suppose that there is a cluster C_i in the clique that is not adjacent to C_{m+1}. This can only be possible if $w \notin P(u')$ for any node $u' \in C_i$. In other words, there must be a path from w to u (or u to w) that contains a forbidden node. This means that a forbidden node exists on one of the two paths $u \rightarrow w \rightarrow u'$ or $u' \rightarrow w \rightarrow v$. This leads to a contradiction, because $u \in P(u')$ and $v \in P(u')$. Therefore, C_{m+1} is connected to all clusters in the clique $C_1, \ldots, C_m$, and, thus, $C_1, \ldots, C_{m+1}$ is a clique, contradicting the assumption that the former clique is maximal. This proves that any maximal clique in the cluster graph corresponds to a valid subgraph of G. ■

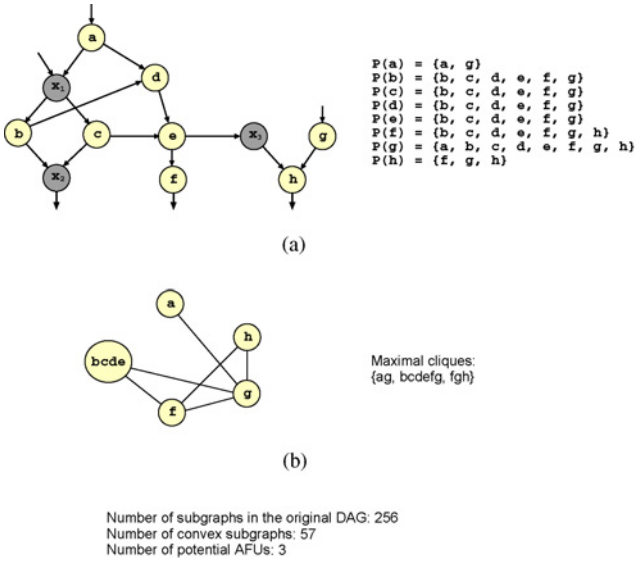


Fig. 10. Example of forming a cluster graph from the original graph and enumerating its maximal cliques. The nodes labeled as x_i's in the graph indicate forbidden nodes. (a) Original DAG and the consistent set of its nodes. (b) Cluster graph and its maximal cliques.

Fig. 10 shows an example illustrating the execution of this method. This approach generates only the maximal valid subgraphs of the input DAG using the maximal clique enumeration algorithm of Johnson *et al.* [18]. In a recent paper, Atasu *et al.* [19] has proved that the number of such maximal cliques is bounded by $2^{|F|}$, where F is the set of forbidden nodes, which can still be significantly large. Without any additional information about the speedup model, we cannot prune this initial set of potentially optimal ISEs any further; however, with more information, such as the hardware and software latencies of individual nodes, we can reduce the size of the cluster graph, and hence that of the search space, even further.

Suppose that we know some lower bound α on the number of cycles saved by the AFU that implements the optimal ISE. A lower bound on the maximal number of cycles saved by the optimal AFU can be found by computing the number of cycles saved by any valid subgraph corresponding to any maximal clique in the cluster graph.

For the first example, let $N[C_1]$ be a set containing cluster C_1 and all its neighboring clusters. Let T_1 be the subgraph of the original DAG induced by all of the nodes in subgraphs corresponding to clusters in $N[C_1]$. Then, if $SW(T_1) \leq \alpha$, we can trivially remove C_1 from the cluster graph. The reason is that any clique that contains C_1 corresponds to an AFU whose cycle savings is provably less than α, and the speedup attributable to this AFU is clearly suboptimal.

For the second example, suppose that there exist adjacent cluster nodes C_1 and C_2. Let $N[C_1 \cup C_2] = N[C_1] \cap N[C_2]$, and let T_{12} be the subgraph of the original DAG induced by all of the nodes in subgraphs corresponding to clusters in $N[C_1 \cup C_2]$. If $SW(T_{12}) \leq \alpha$, then edge (C_1, C_2) can be removed from the cluster graph using a similar reasoning as above.

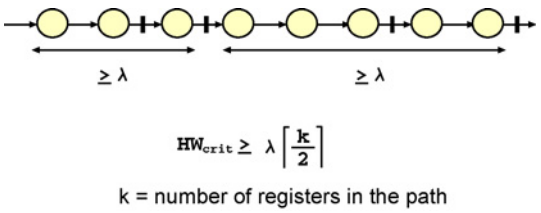


Fig. 11. Upper bound on the number of registers on a path.

D. Pruning the Set of ISE Candidates Without I/O Serialization

Here, we determine the optimal ISE in terms of speedup achievable by its AFU. In order to compute this speedup, we must serialize its I/O accesses, which involves solving another optimization problem, which is quite expensive for a large set of ISEs. To reduce the number of ISEs for I/O serialization, we first calculate upper and lower bounds on the speedup of each ISE without actually serializing its I/O accesses. We use these results to eliminate some candidate ISEs from consideration prior to I/O serialization. Specifically, if the upper bound on the speedup of some ISE is less than the lower bound of another, then there is no need to consider the first and it can be discarded from the set of potentially optimal ISEs.

Theorem 5: The following upper bound on the speedup of an ISE S holds

$$M(S) \leq \sum_{u \in S} SW_u - \max\left(\frac{\mathsf{IN}(S)}{m}, \frac{\mathsf{OUT}(S)}{n}\right).$$

Proof: According to the third constraint in Section III, when we serialize the I/O accesses of an ISE S, for any i, at most m input nodes can have i registers on their incoming edges. Since there are $\mathsf{IN}(S)$ input nodes, at least one input node will have $\frac{\mathsf{IN}(S)}{m}$ or more registers on its incoming edge. Similarly there will exist at least one output node which will have $\frac{\mathsf{OUT}(S)}{n}$ or more registers on its outgoing edge. This means there will be at least $\max\left(\frac{\mathsf{IN}(S)}{m}, \frac{\mathsf{OUT}(S)}{n}\right)$ registers on the path from v_{src} to v_{sink}. In other words, $HW(S) \geq \max\left(\frac{\mathsf{IN}(S)}{m}, \frac{\mathsf{OUT}(S)}{n}\right)$. Since $M(S) = SW(S) - HW(S)$, $M(S)$ cannot exceed $\sum_{u \in S} SW_u - \max\left(\frac{\mathsf{IN}(S)}{m}, \frac{\mathsf{OUT}(S)}{n}\right)$. ■

Theorem 6: The following lower bound on the speedup of an ISE S holds:

$$M(S) \geq \sum_{u \in S} SW_u - \left\lceil \frac{\mathsf{IN}(S)}{m} \right\rceil - \left\lceil \frac{\mathsf{OUT}(S)}{n} \right\rceil - 2\left\lceil \frac{HW_{\text{crit}}(P)}{\lambda} \right\rceil.$$

Proof: From Theorem 7, there exists a path from v_{src} to v_{sink}, such that the number of registers between v_{src} to v_{sink} remains the same, even if we remove all vertices but the ones on this path between v_{src} and v_{sink}. In worst case, this path can have exactly $\left\lceil \frac{\mathsf{IN}(S)}{m} \right\rceil$ registers on its first edge (i.e., the incoming edge of its input), as well as $\left\lceil \frac{\mathsf{OUT}(S)}{n} \right\rceil$ on its last edge.

To understand the additional factor of $2\lceil HW_{\text{crit}}(S) \rceil$ consider any three consecutive registers in this path. The sum of the hardware latencies of all nodes between the first and third register must exceed λ; otherwise we could remove the middle register. If there are k registers in the path, then the sum of hardware latencies of the nodes in the path will be at least $2k\lambda$, i.e., the number of registers in the path cannot exceed $2\left\lceil \frac{HW_{\text{crit}}(P)}{\lambda} \right\rceil$ as shown in Fig. 11. It follows that the number of registers between v_{src} and v_{sink} cannot exceed $2\left\lceil \frac{HW_{\text{crit}}(P)}{\lambda} \right\rceil + \left\lceil \frac{\mathsf{IN}(S)}{m} \right\rceil + \left\lceil \frac{\mathsf{OUT}(S)}{n} \right\rceil$. ■

E. Formulation of I/O Serialization as a Matrix Problem

We are left with a set of potentially optimal ISEs for which we must compute the actual cycle gain by serializing their I/O accesses. Due to concerns about the exponential runtime of Pozzi and Ienne's I/O serialization algorithm [9], we have implemented an efficient heuristic that finds high quality solutions in practice.

Definition 5: The *Integral Path Delay* is the minimum number of registers that need to be inserted in a path such that the sum of hardware latencies of all nodes between two consecutive registers does not exceed λ.

Definition 6: The *Integral Critical-Path Delay (ICD)* from node u to v in a DAG, $ICD(u, v)$, is the maximal integral path delay along all paths from u to v. If there is no path from u to v, then $ICD(u, v) = -\infty$.

Definition 7: The *Residual Hardware Latency* from u to v, $RHL(u, v)$, is the maximal sum of hardware latencies of all nodes after the $ICD(u, v)$th register among all paths between u and v.

$ICD(u, v)$ and $RHL(u, v)$ can be computed by traversing the nodes of the DAG in topological order.

Once we determine the number of registers on incoming edges of the inputs of the AFU, there is no need to schedule the DAG in order to compute the optimal number of registers between v_{src} and v_{sink}. Theorem 7, which follows from Lemma 1, shows that $HW(S)$ can be computed using ICD values alone, without explicitly computing a schedule.

Lemma 1: For any node v, there exists a path p (defined as the *Dominant Path* for v) from v_{src} to v, such that if we remove all nodes from the DAG except those in p, then the number of registers from v_{src} to v, as well as the RHL, will remain the same.

Proof: We use induction on the height of node v, $h(v)$, i.e., the maximum path length (in terms of nodes) from v_{src} to v. If v is an input node, there is one dominant path p of length 0.

For the induction step, assume that the lemma holds for all nodes v, such that $h(v) < r$. Now, let $h(v) = r$. Since all predecessors u of v have $h(u) < r$, there exist dominant paths from v_{src} to u. Now, remove all nodes from the DAG except for v and these paths. This will not affect v, since there will be no difference in the number of registers placed on the dominant paths from v_{src} to each predecessor u; likewise the RHL remains the same. Let u_j be the input node that has the maximal number of registers from v_{src}; ties can be broken using the RHL. Thus, u_j determines the number of registers prior to v and $RHL(u_j, v)$. Hence, the dominant path for the corresponding predecessor of v will also be the dominant path for v. This completes the induction step. ■

Theorem 7: Let S be an AFU with input nodes $u_1, \ldots, u_k$ and $x_1, \ldots, x_k$ registers on each incoming edge. Let $R(v)$

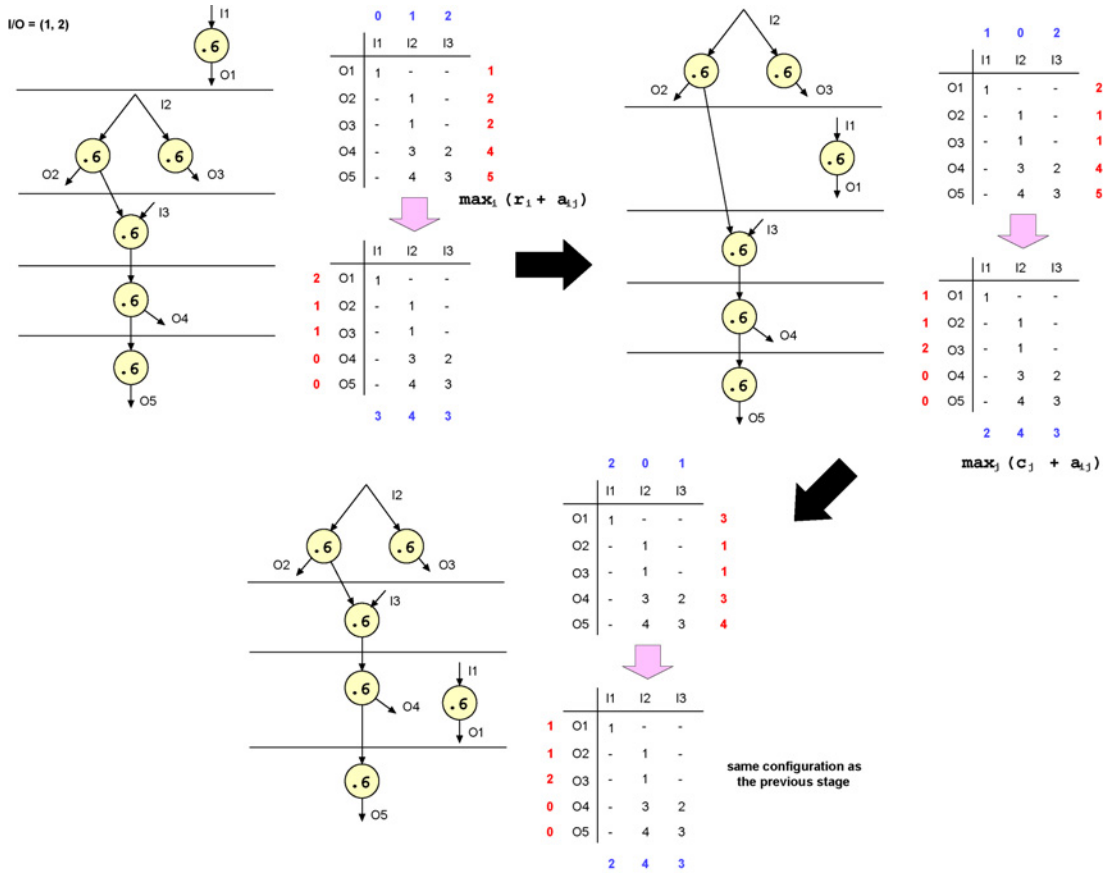


Fig. 12. Execution of the Ping-Pong Algorithm on a simple example; it finds the optimal solution only in two iterations.

denote the maximum number of registers placed on a path from v_{src} to v. Then

$$R(v) = \max_{1 \le i \le k}(x_i + ICD(u_i, v)).$$

Proof: By Lemma 1, if the dominant path for v passes through input u_1, then $R(v) = x_1 + ICD(u_1, v)$. ■

Note that for each input node u_i, $ICD(u_i, v)$ can be computed without knowing the specific value of x_i; moreover, the ICDs can be used to determine x_i. If the AFU has l outputs, $v_1, \ldots, v_l$ with $y_1, \ldots, y_l$ registers on their outgoing edges, then the total number of registers from v_{src} to v_{sink} is

$$R(v_{\text{sink}}) = \max_{(1,1) \le (i,j) \le (k,l)}(x_i + ICD(u_i, v_j) + y_j).$$

Exactly m input nodes will have zero registers on their incoming edges, m inputs will have one register on their incoming edges, etc. The same holds for the number of registers on the outgoing edges of output nodes. Thus, we must find the best mapping between the number of registers and AFU inputs and outputs. The I/O serialization problem translates into the following matrix problem.

Problem 2: Given an $m \times n$ integer matrix A, an m dimensional integer array R, and an n dimensional integer array C, find permutations $\pi : \{1, 2, \ldots, m\} \rightarrow \{1, 2, \ldots, m\}$ and $\sigma : \{1, 2, \ldots, n\} \rightarrow \{1, 2, \ldots, n\}$, such that the following expression is minimized:

$$F(A, R, C, \pi, \sigma) = \max_{i,j}(R[\pi(i)] + a_{ij} + C[\sigma(j)]).$$

R and C correspond to the number of registers placed on each input and output node respectively, and A is the set of ICD values for every pair of nodes in the graph.

F. Heuristic to Solve the Matrix Problem

If all entries of C were zero, then the permutation σ would be meaningless. The resulting problem would be to find a permutation $\pi : \{1, 2, \ldots, m\} \rightarrow \{1, 2, \ldots, m\}$ such that the following simpler equation was minimized:

$$F(A, R, \pi) = \max_{1 \le i \le m}(R[\pi(i)] + a_{ij}).$$

This problem can be solved very easily by assigning the smallest value in $R[.]$ to the largest value of $\max_j(a_{ij})$, the second smallest value of $R[.]$ to the second largest value of $\max_j(a_{ij})$, etc. In other words, the permutation π corresponds to a reverse sorting order of $\max_j(a_{ij})$ values.

Similarly, if permutation σ is fixed, we can find the optimal permutation π by sorting $\max_j(a_{ij} + C[\sigma(j)])$; an analogous situation occurs when π is fixed and we want to find an optimal permutation σ. The heuristic that we use to compute π and σ is based on these observations. We start with randomly generated permutations for π and σ. First, we find the optimal π, given σ. Next, we find the optimal σ, given π. Then, once again, the optimal π, given σ. The process continues as long as at least one of the permutations changes. This heuristic will henceforth be referred to as the *Ping-Pong Algorithm*. Fig. 12 shows

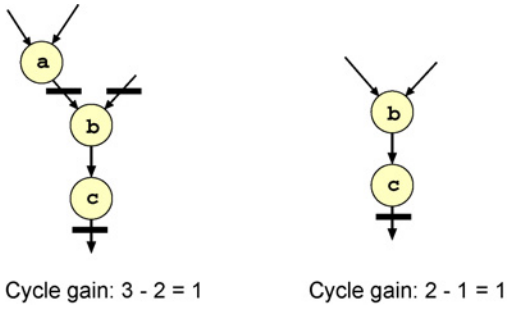


Fig. 13. Two ISEs have the same speedup, however the second one has smaller area. The I/O constraint is (2, 1).

an example illustrating its execution. Theorem 8 establishes convergence of Ping-Pong Algorithm.

Theorem 8: If all entries in the input matrix A and R, C are bounded, then the Ping-Pong Algorithm will converge.

Proof: For any permutation π and σ, consider the mn-ary vector W which contains the values $R[\pi(i)] + a_{ij} + C[\sigma(j)]$ in decreasing order. The claim is that in every iteration of ping-pong this vector decreases lexicographically. It is easy to see that this suffices to prove convergence.

Given two sequences $a_1, a_2, \ldots, a_r$ and $b_1, b_2, \ldots, b_r$ of integers; sort the a_i's in decreasing order. Now if we sort the b_j's in increasing order using bubble sort, at each step we will swap two entries b_i and b_{i+1} (i.e., originally $b_i > b_{i+1}$). Now observe the vector $U = (a_1 + b_1, a_2 + b_2, \ldots, a_r + b_r)$ in this process. The only entries that change are $a_i + b_i$ and $a_{i+1} + b_{i+1}$, which are replaced by $a_i + b_{i+1}$ and $a_{i+1} + b_i$.

Since $a_i > a_{i+1}$ and $b_i > b_{i+1}$

$$a_i + b_i > \mathsf{max}(a_i + b_{i+1}, a_{i+1} + b_i).$$

In other words, the maximum of two entries in the vector U decreases, i.e., if we have a vector V, which has the same entries as U, but in decreasing sorted order, then in the swapping operation V decreases lexicographically. Sorting $b_1, b_2, \ldots, b_r$, decreases V lexicographically.

Suppose for a fixed σ we want to find π by sorting $\mathsf{max}_j(C[\sigma(j)] + a_{ij})$. If π was not already in the reverse sorted order of this sequence, then by the same logic as above we reduce W lexicographically. ■

G. Area Minimal ISE With the Same Speedup

Suppose that we find an ISE R which has the maximal speedup after I/O serialization. Although R is optimal with respect to speedup, it may be large. Hence, one would like to find an ISE that has the maximal speedup among all ISEs, while having the smallest area among all such ISEs.

Let T be the ISE that has the maximal speedup and whose area is minimal. Clearly, $T \subseteq R$, where R is an ISE corresponding to a maximal clique in the cluster graph. According to the monotonicity property the speedup of T cannot exceed that of R.

As an example consider Fig. 13. Here we have an I/O constraint of (2, 1), i.e., in each cycle at most two inputs can be read, and at most one output can be written. Based on these constraints, if we serialize the I/O accesses of the two ISEs, we find they have the same speedup; however, the second ISE is smaller, and is therefore preferable over the first one.

We only need to find the smallest ISE contained in a maximal ISE, which has the same speedup. We define the problem as follows:

Problem 3: Given a convex subgraph G, containing only valid nodes, find the maximal subset of nodes that can be removed from G without violating the convexity constraint, or deteriorating the speedup achievable by the corresponding ISE.

We can prune the search space effectively by exploiting following property.

Property 1: We are given two convex graphs G and H, such that $H \subset G$. If there are three nodes x, y, and z in G, such that y is a successor of x, and z is a predecessor of x, and $x \in G/H$, then both y and z cannot be in H.

Since H is convex and does not contain x, it cannot contain both y and z, otherwise we will have a path from z to y which is not completely contained in H. However, this cannot be the case as H is convex.

If we remove a node from G, then we have to remove either all its predecessors, or all its successors; otherwise the resulting graph will be nonconvex. Hence, there must exist two sets P and S such that in order to obtain H from G, one needs to remove all the nodes in P and their predecessors nodes, and all the nodes in S and their successors.

For a node $p \in P$, consider the subgraph G_p which is obtained by removing p and all its predecessors from G. It is easy to see that G_p is convex. G_p also satisfies the property that $H \subseteq G_p \subset G$. According to the monotonicity property $M(H) \leq M(G_p) \leq M(G)$; but since $M(H) = M(G)$, it follows that $M(G_p) = M(G)$. This means that P can contain a node p only if $M(G_p) = M(G)$. This can be used to find the potential members of P. Similarly, we can find the potential members of S. Lets call the two set of potential members as $\mathcal{P}$ and $\mathcal{S}$.

By the following theorem one should consider only those subsets of $\mathcal{P}$ and $\mathcal{S}$ in which no two nodes have predecessor–successor relationships.

Theorem 9: For any given P and S there exist $P' \subseteq P$ and $S' \subseteq S$, such that no two nodes in P' and S' have predecessor–successor relationship, and the subgraph resulting by removing nodes of P (S) and its predecessors (successors) is the same as the one resulting from removing nodes of P' (S') and its predecessors (successors).

Proof: Let us assume that there are two nodes u and v are in P such that v is a predecessor of u. The set of predecessors of v is a subset of the set of predecessors of u. Hence, the process of removing the predecessors of u will also remove v and its predecessors. Thus, removing v from P will not affect the resulting subgraph. We can keep removing nodes from P until no two nodes of P are related via a predecessor–successor relationship. The same can be done for S. ■

Based on Theorem 9 we can further restrict the subsets of $\mathcal{P}$ and $\mathcal{S}$. We can construct an undirected graph X consisting of nodes of $\mathcal{P}$ and $\mathcal{S}$. If a node occurs in both $\mathcal{P}$ and $\mathcal{S}$, there should be two nodes in X corresponding to this node. We draw an edge between two nodes in X, if either both of them

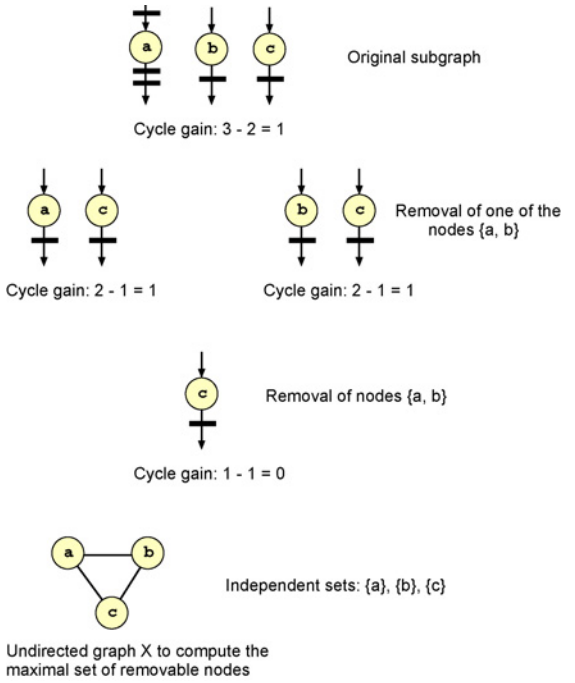


Fig. 14. Example showing that it may be possible to remove two nodes individually without affecting the gain, but not both simultaneously. The I/O constraint is (2, 1).

correspond to nodes in $\mathcal{P}$ or both of them correspond to nodes in $\mathcal{S}$, and they also have a predecessor–successor relationship. Only those subsets of $\mathcal{P}$ and $\mathcal{S}$ should be considered whose corresponding nodes in X form an independent set; by enumerating the independent sets of X we can enumerate the potential sets of nodes which should be removed, along with their predecessors/successors, from the subgraph without affecting its speedup. The subgraph corresponding to each independent set is generated and its speedup is evaluated. The subgraph which has the smallest size, and has the same speedup as the original ISE is then chosen.

There may exist two nodes u and v in $\mathcal{P}$, that are not related by a predecessor–successor relationship, such that removing either of the two nodes and its predecessor does not affect the speedup, but removing both nodes and their predecessors decreases the speedup. An example is shown in Fig. 14. For the I/O constraint $(2, 2)$ the original subgraph has a gain of $(3-2) = 1$ cycles. Removal of any of the three nodes a, b, or c does not reduce the gain in cycles. However, if we remove two of these nodes, the cycle gain reduces to $(1-1) = 0$ cycles. Using this criteria we can further prune the search space.

In X, we can choose any two nodes which are not connected by an edge, and then remove their predecessors (successors) to see if this reduces speedup. If so, then the predecessors (successors) of the two nodes cannot be removed simultaneously, hence, we can add an edge between the two nodes in X. The overall algorithm is shown in Fig. 15. In order to enumerate all independent sets of X, we use the method presented in [18] to generate all maximal independent sets, and then all subsets of these maximal independent sets are generated.

```
areaMinimal (graph H) {
// The function takes graph H and outputs all potential
// subgraphs of H which have the same speedup as H.
  P = removableNodesWithPredecessors (H);
  S = removableNodesWithSuccessors (H);

  X = constructUndirectedGraphWithNodes (P ∪ S);
  foreach pair of nodes (u, v) in X do
    if (u, v cannot be removed simultaneously)
      addEdge (X, (u, v));

    if (({u, v} ⊆ P or {u, v} ⊆ S) and (u, v) have a
    predecessor-successor relationship)
      addEdge (X, (u, v));
  od;

  I_X = independentSets (X);
  foreach I in I_X return the subgraph obtained by removing
  the nodes in I and its predecessors/successors.
}
```

Fig. 15. Algorithm to generate potential area-minimal subgraphs that have the same speedup as a given subgraph.

V. Experiments

We use *Clarity*, a commercial compiler for extensible processor customization to generate the dataflow graphs used in our experiments [20]. Each node in the graph is affiliated with the name of the corresponding operation, bitwidth of its inputs and outputs, and the software and hardware latency of the operation. The software latency of an instruction is estimated to be the latency of the execution stage of the RISC pipeline. The hardware latency of an instruction is estimated by synthesizing the corresponding operator on a UMC $0.18\,\mu$m CMOS technology standard cell library; this latency is then normalized to the delay of a 32-bit multiply accumulate operation.

We run our algorithm on each DFG to find a set of nonoverlapping ISEs in it. This is done by using an iterative approach. First an ISE is chosen with the maximal speedup. To find the next ISE, all the nodes of first ISE are clustered into a single node, and this node is marked as forbidden. Then we use our algorithm on the modified DAG to find the second and all subsequent ISEs. Each ISE is then serialized and pruned as described in the preceding section.

The k best ISEs from the whole program are chosen based on the merit function $M()$. In our experiments, we choose k to be 16 similar to Pozzi and Ienne's paper [9]. Note that, we do not consider any hardware area constraints; however, if such constraints are present, then we can use the approach used by Atasu *et al.* [12] to choose a set of ISEs with highest cumulative merit, satisfying the area constraints.

The speedup for the whole program is computed as follows:

$$\text{speedup} = \frac{\text{numCycles in software}}{\text{numCycles in software} - \sum_S \text{merit}(S)},$$

where the summation is taken over all chosen ISEs.

A. Results

We evaluated our algorithm on eight benchmarks. The first two benchmarks *ADPCM CODER* and *ADPCM DECODER* are from Mediabench applications [21]; *AES*, *DES*, and *MD5* are cryptography applications; and *FFT*, *VITERBI*, and *UNEPIC* are EEMBC benchmarks [22].

On each benchmark we ran our algorithm, Atasu *et al.*'s algorithm [4] with no I/O serialization, and Pozzi and Ienne's

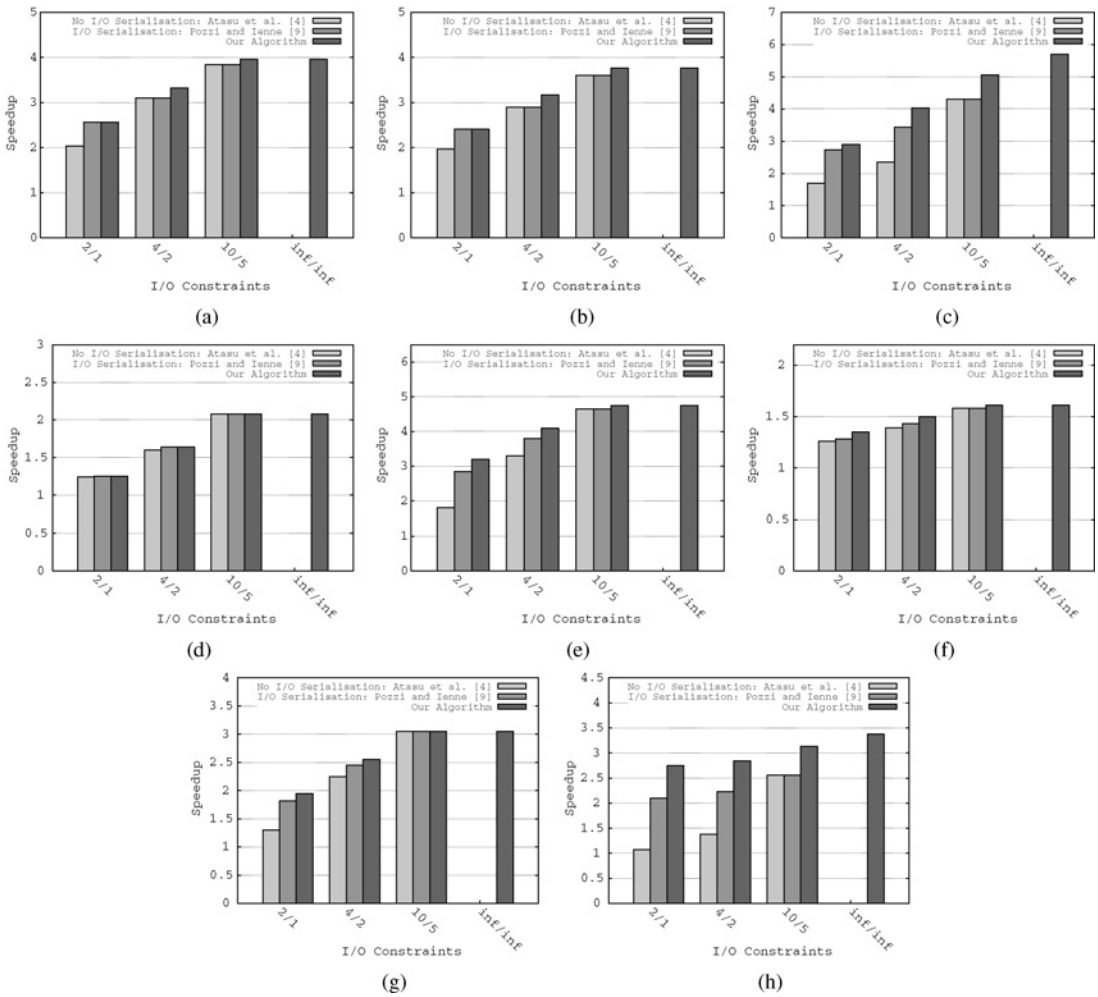


Fig. 16. Comparison of the speedup from ISEs/AFUs generated by our algorithm with the state of art techniques. (a) *ADPCM CODER*. (b) *ADPCM DECODER*. (c) *AES*. (d) *VITERBI*. (e) *DES*. (f) *UNEPIC*. (g) *FFT*. (h) *MD5*.

algorithm [9] with I/O constraints $(2, 1), (4, 2), (10, 5)$, and (∞, ∞), the last of which indicates no constraints on the I/O. We also compare with an *Integer Linear Programming (ILP)* presented by Verma *et al.* [13] to solve the same problem, which is similar to the one presented by Atasu *et al.* [12], but does not assume the constant penalty per extra I/O access. When the ILP solver was able to terminate, it produced the same results as our algorithm.

The results are shown in Fig. 16. I/O serialization increases the speedup for all benchmarks because it can identify larger subgraphs as ISEs. In all eight benchmarks, our algorithm provides speedups equal to or exceeding that of Pozzi and Ienne's algorithm [9]. In some cases, the previous method, although a heuristic, also found optimal solutions.

For *AES*, which has a 696 node DFG, for an I/O constraint of $(10, 5)$, the new algorithm selects an ISE that results in a speedup of 5.05, compared to a speedup of 4.3 generated by the algorithm of Pozzi and Ienne. Our algorithm found an ISE having 22 inputs and 22 outputs. Pozzi and Ienne's algorithm, in contrast, only considers subgraphs having I/O constraints up to a fixed value, which was limited to $(10, 5)$. If the fixed value was relaxed to $(22, 22)$, their approach could have found the same solution; however, they advise against relaxing the I/O constraints beyond $(10, 5)$ due to the exponential runtime of their approach. A similar phenomenon occurs in the case of *MD5*, which has a 1149 node DFG: our algorithm found an ISE with 52 inputs and 4 outputs resulting in a much higher speedup than found by Pozzi and Ienne's algorithm. For other

TABLE I
COMPARISON OF RUNTIME OF OUR ALGORITHM WITH POZZI AND IENNE'S ALGORITHM AND THE ILP SOLVER

Benchmark	Pozzi and Ienne's Algorithm [9] (sec)	Our Algorithm (sec)	ILP Solver [13] (sec)
AES	>4 hours	50.4	>4 hours
MD5	>4 hours	41.7	>4 hours
ADPCM CODER	67.5	3.8	80.1
ADPCM DECODER	66.4	3.8	82.3
VITERBI	13.1	1.3	8.0
DES	1661.7	37.4	230.7
UNEPIC	15.4	0.8	17.8
FFT	122.4	8.1	30.1

We show the runtime only for I/O constraints of $(4, 2)$. The runtime for other I/O constraints is similar.

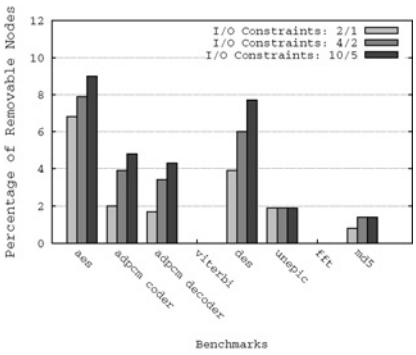


Fig. 17. Percentage reduction in the size of the chosen ISEs without affecting the speedup.

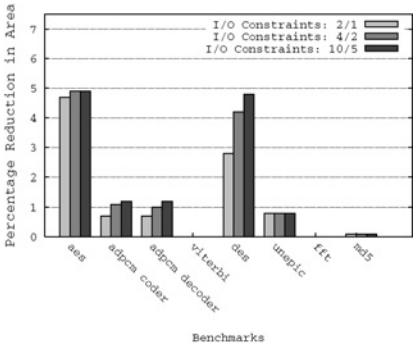


Fig. 18. Percentage reduction in the hardware area of the chosen ISEs without affecting the speedup.

benchmarks, our algorithm yields marginal improvements in ISE quality, because the optimal ISEs have relatively low I/O constraints and can be found by existing algorithms.

Table I compares the runtime of our algorithm, including I/O serialization, with the runtime of Pozzi and Ienne's algorithm [9] and Verma *et al.*'s ILP formulation [13] for a base processor with I/O constraints of $(4, 2)$; we observed similar results for other I/O constraints because all steps dependent on I/O constraints do not contribute much to the overall runtime. This is the case in both our algorithm and in Pozzi and Ienne's. In our algorithm, the majority of time is due to clique enumeration, which is independent of I/O constraints. However, in Pozzi and Ienne's algorithm, for any I/O constraints, all ISEs with number of inputs under ten and number of outputs under five are enumerated, which contributes significantly to the overall runtime of their algorithm.

Table I shows that our algorithm is 15–20$\times$ faster on smaller benchmarks compared to both Pozzi and Ienne's algorithm, and the ILP solver. However, for larger benchmarks such as *AES* and *MD5* Pozzi and Ienne's algorithm and the ILP solver did not finish in the allotted time (4 h). Fig. 16 reported the best solution found by their algorithm in the allotted time. In contrast, our algorithm took less than a minute for both of these benchmarks.

Figs. 17 and 18 show the effectiveness of our area reduction algorithm presented in Section IV-G. Fig. 17 shows the percentage of nodes than can be removed from the chosen ISEs without reducing the speedup. The values of Fig. 18 have been obtained after synthesizing the ISEs before and after removing the nodes, using a common standard cell library for UMC $0.09\,\mu$m CMOS technology. Both figures indicate that only a marginal reduction is possible without sacrificing the speedup. For some benchmarks such as *VITERBI* and *FFT* the original ISEs were also area-optimal; however, for *AES* and *DES*, the size of the original ISE was reduced by 5–9% without reducing the speedup obtained.

The runtime of the area reduction algorithm is small, never exceeding 5 s. The first step is the recognition of removable nodes, and the construction of graph X. The second step enumerates the independent sets of X. The first step calls the Ping-Pong Algorithm at most $O(n)$ times, where n is the number of nodes in the ISE. The second step has exponential time complexity on the number of nodes in X; however, it never ran on large input instance in this paper.

VI. Conclusion

A new approach for ISE generation for extensible processors has been presented. It exploits the fact that the use of I/O serialization eliminates the I/O constraints imposed by previous formulations of the problem. Without I/O constraints, prior techniques for ISE generation suffer from runtimes that are exponential in the number of nodes in their graphs. The approach presented in this paper, in contrast, exploits the monotonicity of the speedup function to reduce the number of nodes in the graph via clustering. A cluster graph has been introduced to model the compatibility between clusters and permit the construction of large ISEs by combining clusters. Clique enumeration in the cluster graph has been empirically observed to be much faster than subgraph enumeration in the original DFG. We have also introduced a faster heuristic for I/O serialization than the current algorithm developed by Pozzi and Ienne [9]. Since we enumerate only maximal ISEs, this might result in the inclusion of nodes that we can remove without reducing the speedup. Our experiments show that only a small fraction of nodes can be removed.

We have also shown that for a RISC base processor the speedup model used to evaluate the benefit of an ISE is independent of the specific details of the microarchitecture. This ensures that engineers do not need to study the architectural details of the pipeline in order to identify good ISEs.

Altogether, our approach to ISE generation runs significantly faster than the prior approach of Atasu *et al.* [5]. Pozzi and Ienne [9] use an I/O constraint of $(10, 5)$ to limit the size of the ISEs identified; we have found that for AES, the optimal ISE has an I/O constraint of $(22, 22)$; standard subgraph enumeration methods require several hours to identify a suboptimal $(10, 5)$ ISE, while the new approach found the optimal ISE in approximately 16 s, testifying to both the efficacy and efficiency of the proposed technique.

References

[1] A. K. Verma, P. Brisk, and P. Ienne, "Rethinking custom ISE identification: A new processor-agnostic method," in *Proc. Int. Conf. Compilers Architectures Synthesis Embedded Syst.*, Sep. 2007, pp. 125–134.

[2] N. Clark, H. Zhong, and S. Mahlke, "Processor acceleration through automated instruction set customization," in *Proc. 36th Annu. Int. Symp. Microarchitecture*, Dec. 2003, pp. 129–140.
[3] P. Yu and T. Mitra, "Scalable custom instructions identification for instruction set extensible processors," in *Proc. Int. Conf. Compilers Architectures Synthesis Embedded Syst.*, Sep. 2004, pp. 69–78.
[4] K. Atasu, L. Pozzi, and P. Ienne, "Automatic application-specific instruction-set extensions under microarchitectural constraints," in *Proc. 40th Des. Autom. Conf.*, Jun. 2003, pp. 256–261.
[5] L. Pozzi, K. Atasu, and P. Ienne, "Exact and approximate algorithms for the extension of embedded processor instruction sets," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 25, no. 4, pp. 1209–1229, Jul. 2006.
[6] P. Yu and T. Mitra, "Disjoint pattern enumeration for custom instruction identification," in *Proc. 17th Int. Conf. Field-Programmable Logic Appl.*, Aug. 2007, pp. 273–278.
[7] P. Bonzini and L. Pozzi, "Polynomial-time subgraph enumeration for automated instruction set extension," in *Proc. Des. Autom. Test Eur. Conf. Exhibition*, Apr. 2007, pp. 1331–1336.
[8] X. Chen, D. L. Maskell, and Y. Sun, "Fast identification of custom instructions for extensible processors," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 26, no. 2, pp. 359–368, Feb. 2007.
[9] L. Pozzi and P. Ienne, "Exploiting pipelining to relax register-file port constraints of instruction-set extensions," in *Proc. Int. Conf. Compilers Architectures Synthesis Embedded Syst.*, Sep. 2005, pp. 2–10.
[10] N. Pothineni, A. Kumar, and K. Paul, "Application-specific datapath extension with distributed I/O functional units," in *Proc. 20th Int. Conf. Very-Large-Scale Integration Des.*, Jan. 2007, pp. 551–558.
[11] Y. Guo, G. J. Smit, H. Broersma, and P. M. Heysters, "A graph covering algorithm for a coarse-grain reconfigurable system," in *Proc. Assoc. Comput. Machinery Conf. Languages Compilers Tools Embedded Syst.*, Jun. 2003, pp. 199–208.
[12] K. Atasu, R. Dimond, O. Mencer, W. Luk, C. Özturan, and G. Dünda, "Optimizing instruction-set extensible processors under data bandwidth constraints," in *Proc. Des. Autom. Test Eur. Conf. Exhibition*, Mar. 2007, pp. 588–593.
[13] A. K. Verma, P. Brisk, and P. Ienne, "Fast, quasi-optimal, and pipelined instruction-set extensions," in *Proc. Asia South Pacific Des. Autom. Conf.*, Jan. 2008, pp. 334–339.
[14] P. Brisk, A. Kaplan, R. Kastner, and M. Sarrafzadeh, "Instruction generation and regularity extraction for reconfigurable processors," in *Proc. Int. Conf. Compilers Architectures Synthesis Embedded Syst.*, Oct. 2002, pp. 262–269.
[15] R. Kastner, A. Kaplan, S. Ogrenci Memik, and E. Bozorgzadeh, "Instruction generation for hybrid reconfigurable systems," *Assoc. Comput. Machinery Trans. Des. Autom. Electron. Syst.*, vol. 7, no. 4, pp. 605–627, Oct. 2002.
[16] S. Cadambi and S. C. Goldstein, "CPR: A configuration profiling tool," in *Proc. 7th IEEE Symp. Field-Programmable Custom Comput. Mach.*, Apr. 1999, pp. 104–113.
[17] K. Karuri, A. Chattopadhyay, M. Hohenauer, R. Leupers, G. Ascheid, and H. Meyr, "Increasing data-bandwidth to instruction-set extensions through register clustering," in *Proc. Int. Conf. Comput. Aided Des.*, Nov. 2007, pp. 166–171.
[18] D. S. Johnson and C. H. Papadimitriou, "On generating all maximal independent sets," *Inf. Process. Lett.*, vol. 27, no. 3, pp. 119–123, 1988.
[19] K. Atasu, O. Mencer, W. Luk, C. Özturan, and G. Dünda, "Fast custom instruction identification by convex subgraph enumeration," in *Proc. 19th Int. Conf. Appl.-Specific Syst. Architectures Processors*, Jul. 2008, pp. 1–6.
[20] *Clarity*, Mimosys SA, 2008.
[21] C. Lee, M. Potkonjak, and W. H. Mangione-Smith, "MediaBench: A tool for evaluating and synthesizing multimedia and communicatons systems," in *Proc. 30th Annu. Int. Symp. Microarchitecture*, Dec. 1997, pp. 330–335.
[22] T. R. Halfhill, "EEMBC releases first benchmarks," Microprocessor Rep., May 1, 2000.

Ajay K. Verma received the B.S. degree in computer science from the Indian Institute of Technology Kanpur, Kanpur, India, in 2003. Since 2004, he has been pursuing the Ph.D. degree from the Processor Architecture Laboratory, Ecole Polytechnique Fédérale de Lausanne, Lausanne, Switzerland.

His research interests include logic synthesis, optimization of arithmetic circuits, and design automation for application-specific processors.

Mr. Verma was the recipient of the Best Paper Award at the International Conference on Compilers, Architecture, and Synthesis, in 2007.

Philip Brisk (M'09) received the B.S., M.S., and Ph.D. degrees, all in computer science, from the University of California, Los Angeles, in 2002, 2003, and 2006, respectively.

From 2006 to 2009, he was a Postdoctoral Scholar with the Processor Architecture Laboratory, Ecole Polytechnique Fédérale de Lausanne, Lausanne, Switzerland. Since 2009, he has been an Assistant Professor with the Department of Computer Science and Engineering, University of California, Riverside. His current research interests include FPGA architecture and mapping algorithms, compiler and design automation, and architecture for application-specific processors.

Dr. Brisk was the recipient of the Best Paper Award at the International Conference on Compilers, Architecture, and Synthesis, in 2007, and the International Conference on Field Programmable Logic and Applications, in 2009. He was the General Co-Chair of the IEEE Symposium on Industrial Embedded Systems, in 2009. He has been a Member of the technical program committees of several international conferences and workshops, including the Design Automation and Test in Europe, the IEEE Symposium on Application-Specific Processors, the International Conference on Architecture of Computing Systems, the Reconfigurable Architecture Workshop, and the International Workshop on Software and Compilers for Embedded Systems. He is a Member of the Association for Computing Machinery.

Paolo Ienne (S'94–M'96) received the Dottore degree in electronic engineering from the Politecnico di Milano, Milan, Italy, in 1991, and the Ph.D. in computer science degree from the Ecole Polytechnique Fédérale de Lausanne (EPFL), Lausanne, Switzerland, in 1996.

In December 1996, he was with the Semiconductors Group of Siemens AG, Munich, Germany (which later became Infineon Technologies AG). After working on datapath generation tools, he became the Head of the Embedded Memory Unit in the Design Libraries Division. Since 2000, he has been with EPFL, where he is currently a Professor and heads the Processor Architecture Laboratory. His research interests include various aspects of computer and processor architecture, computer arithmetic, reconfigurable computing, and multiprocessor systems-on-chip.

Prof. Ienne was a recipient of a Best Paper Award of the Design Automation Conference in 2003, the International Conference on Compilers, Architecture, and Synthesis (CASES), in 2007, and the International Conference on Field Programmable Logic and Applications (FPL), in 2009. He was the General Co-Chair of the Sixth IEEE Symposium on Application-Specific Processors in 2008 and a Guest Editor for a special section of the IEEE TRANSACTIONS ON VERY-LARGE-SCALE INTEGRATION SYSTEMS ON APPLICATION SPECIFIC PROCESSORS. He has been a Member of the program committees of several international conferences and workshops, including the Design Automation and Test in Europe, the International Conference on Computer Aided Design, the CASES, the International Symposium on Low Power Electronics and Design, the International Symposium on High-Performance Computer Architecture, the FPL, and the IEEE International Symposium on Asynchronous Circuits and Systems.