Datapath Merging and Interconnection Sharing for Reconfigurable Architectures

Nahri Moreano[†*] and Guido Araujo[*]

[*] IC-UNICAMP
Campinas, SP 13084-971
[†] DCT-UFMS
Campo Grande, MS 79070-900
Brazil

{nahri,guido}@ic.unicamp.br

Zhining Huang and Sharad Malik

Department of Electrical Engineering
Princeton University
Princeton, NJ 08544
USA

{znhuang,malik}@ee.princeton.edu

ABSTRACT

Recent work in reconfigurable computing research has shown that a substantial performance speedup can be achieved through architectures that map the most relevant application inner-loops to a reconfigurable datapath. Any solution to this problem must be able to synthesize a datapath for each loop and to merge them together into a single reconfigurable datapath. The main contribution of this paper is a novel graph-based technique for the datapath merge problem. This approach is based on the solution of a maximum clique problem that merges datapaths one at a time. A set of experiments, using the MediaBench benchmark, shows that the proposed technique produces 24% fewer datapath interconnections than a previous solution to this problem.

Categories and Subject Descriptors

C.1.3 [**Processor Architectures**]: Other Architecture Styles—*Adaptable architectures*

General Terms

Performance

Keywords

Reconfigurable computing, high level and architectural synthesis

1. INTRODUCTION

The availability of large/cheap arrays of programmable (reconfigurable) logic has created a new set of architectural alternatives for the design of complex digital systems. Reconfigurable logic brings together the flexibility of software to the performance of hardware. In most reconfigurable architecture designs, an array of programmable logic is coupled to a microprocessor, enabling the designer to partition the application between (slow) software running on the processor and (fast) hardware running on the reconfigurable array.

There are a number of solutions to the design of reconfigurable systems. In general, the resulting architectures can be classified according to: the level of coupling between the reconfigurable array and the processor; and the size of the logic blocks in the array [4]. In highly coupled systems, reconfigurable hardware is used solely to provide functional units within a host processor [18, 13]. The reconfigurable array can be used as a co-processor [7, 14] or as another processing unit of a multiprocessor architecture [6] in medium coupled systems. In loosely coupled systems, reconfigurable units communicate with the host processor through a network. When the size of the logic block is considered, reconfigurable hardware can be divided into: (a) fine-grained units, where logic blocks are boolean function cells (e.g., typical FPGAs); (b) medium-grained units, formed by bit-slices of functional units (e.g., 4-bit adder) that can be used to implement wider operation units (e.g.,32-bit adder) [7, 11]; and (c) coarse-grained units containing entire functional units [5] (or tiny processors [12]) interconnected so as to implement word-width datapaths.

Recent work in reconfigurable computing research has shown that a substantial performance speedup can be achieved through architectures that map the most relevant application inner-loops to a reconfigurable datapath [17, 2]. At runtime, as each loop of the application starts to execute, the system reconfigures the datapath so as to perform the loop computation. In the case of coarse-grained architectures, in which a set of functional units communicate through a reconfigurable network, any solution to this problem must be able to perform two tasks: (a) synthesize a datapath for each such loop; and (b) to merge them together into a single reconfigurable datapath.

The reconfigurable datapath should have as few hardware blocks and interconnections as possible, in order to reduce its cost, area, power consumption and reconfiguration overhead. Hence we want to reuse the hardware blocks and interconnections across the loop datapaths as much as possible. The datapath merging process enables this reuse by identifying similarities among the loop datapaths and producing a resulting datapath that can be dynamically reconfigured to work for each loop datapath, with the minimum number of hardware blocks and interconnections.

ISSS'02, October 2–4, 2002, Kyoto, Japan.

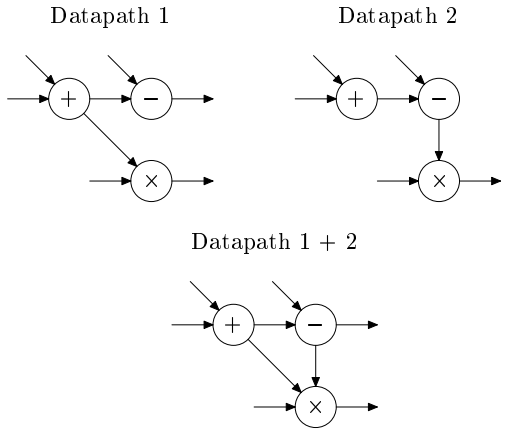

Figure 1: Datapaths merging.

Figure 1 illustrates the concept of datapath merging. The goal is to design a reconfigurable datapath which incorporates all the loop datapaths and has as least functional units and interconnections as possible. When the Datapaths 1 and 2 from Figure 1 are merged, we get the resulting Datapath 1 + 2, also shown in the figure. Notice that in the resulting datapath there are interconnections originated from only one datapath (e.g., the (+,×) interconnection from Datapath 1) and interconnections shared by both datapaths (e.g., the (+,−) interconnection).

Huang and Malik [8] proposed a technique to merge the individual loop datapaths into a single reconfigurable datapath. Their heuristic adds one datapath to the final datapath at a time. At each step, it solves a maximum weight bipartite matching problem that maps hardware blocks (functional units and registers) vertices while trying to maximize the sharing of interconnections. A similar approach was presented in [15], which describes a method for combining two designs into a reconfigurable one, based on the identification of common components of these designs. This work is also based on an algorithm for vertex matching on weighted bipartite graphs, but at a lower granularity.

The main contribution of this paper is a novel graph-based technique for the datapath merge problem. Our approach is based on the solution of a maximum clique problem that merges datapaths one at a time. Contrary to [8], which is based on vertex mapping, our approach maps datapath interconnections (arcs) to compute the reconfigurable datapath. Experimental results, using the MediaBench benchmark [10], reveal that this technique produces on average 24% fewer datapath interconnections than the solution in [8], while using the same number of hardware blocks.

As described in Section 4, a preliminary *Integer Linear Programming* (ILP) lower bound analysis [16] shows that this approach produces, in the worst case, 8.6% more interconnections than the optimum solution, for the MPEG application, and finds optimal datapaths for two other applications in MediaBench.

This paper is divided as follows. Section 2 describes our reconfigurable architecture model. Section 3 details the datapath merge problem and describes our maximum clique approach used to solve this problem. In Section 4 a set of experiments are described to support the proposed approach. Finally, Section 5 concludes the work.

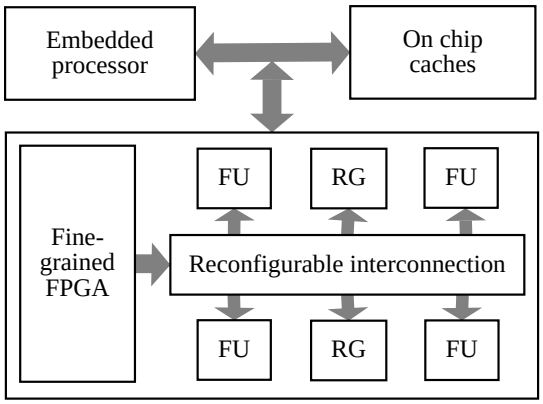

Figure 2: Architectural model.

2. ARCHITECTURAL MODEL

The reconfigurable architecture studied in this paper is a medium-coupled coarse-grained system (Figure 2) similar to the one proposed in [8]. In this model, a set of functional units is organized around an interconnection network resulting in a programmable datapath. As shown in Figure 2, it consists of an embedded processor coupled to an on-chip SRAM and a reconfigurable array through a bus. The reconfigurable array is composed of a set of *Functional Units* (FUs) and *Registers* (RGs) wired together to an interconnection network. A fine-grained FPGA is used for the control logic required to re-shape the network so it implements the desired datapath.

As the computation progresses, the system reconfigures the datapath through the interconnection network, such that computational intensive pieces of the application are mapped to it. Given the (coarse) granularity of the logic blocks (FUs and RGs), the number of bits required to encode them is much smaller than in the case of fine-grained architectures. As a result, fewer bits are needed to reconfigure the datapath, thus diminishing the size of the memory required to store the reconfiguration bits (the so called reconfiguration *context*). This is a central issue in SOC designs where on-chip area is a premium asset. Moreover, the smaller the size of the context the smaller the time overhead required for reconfiguration. Reconfiguration time is a critical feature in such systems, given that the final performance is determined by the sum of the computation time and the reconfiguration latency (if latency hidden techniques are not used).

3. THE DATAPATH MERGING PROBLEM

The approach to reconfigurable computing used in this work follows closely the one proposed in [8]. The IMPACT compiler [3] is used to extract profiling information from programs. Enough experimental evidence exist to support the fact that inner loops account for the largest share of program execution time. Therefore, these loops are the best candidates for mapping onto the reconfigurable logic.

In order to do that, the execution time (cycle count) of each loop is measured using the scheduled/allocated *lcode*[1] representation of the program. Inner loops are then ranked according to their contribution to program execution time,

[1]lcode is the intermediate representation format of IMPACT.

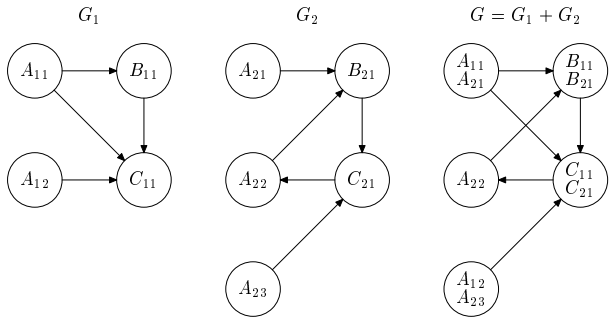

Figure 3: Two graphs G_1 and G_2 and the resulting graph $G = G_1 + G_2$.

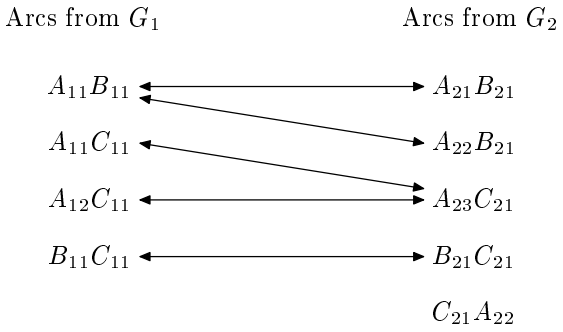

Figure 4: All possible mappings of arcs from G_1 and G_2.

and the set of the most 7-10 relevant loops are selected. After loop profiling, a direct mapping technique [9] is used to generate the datapath for each selected loop, from IMPACT lcode. The output of each loop synthesis is the design of an optimized datapath that implements the loop computation. Notice that not all loops are amenable for synthesis. Loops that violate the hardware resource limits of the target reconfigurable array should be discarded.

The next step in this approach is to merge together all loop datapaths into a single datapath. The resulting datapath should be reconfigured as program execution reaches each mapped loop. To achieve that, a flexible interconnection network is used, combined with a distributed cache mechanism to store the loop configuration contexts. The work in [8] describes this reconfiguration mechanism in details, and thus it will not be described further.

3.1 Graph Modeling

In this section we formulate datapath merge as a graph theoretical problem. More formally, we want to merge several datapaths (corresponding to application loops), in order to build a reconfigurable datapath that has as least hardware blocks (functional units and registers) and interconnections as possible.

Each loop datapath i is modeled as a directed graph $G_i = (V_i, E_i)$, where the vertices in V_i represent the hardware blocks in the datapath, and the arcs in E_i are associated to the interconnections between the hardware blocks. The types of hardware blocks (e.g. adders, multipliers, registers, etc) are modeled by a labeling function L_i of V_i, such that, for each vertex $u \in V_i$, $L_i(u) = T_{ij}$ is a label that represents the type of the hardware block associated to u. More specifically, we say that vertex u in graph G_i is associated to the j^{th} hardware block of type T. Consider, for example, Figure 3. It shows two directed graphs G_1 and G_2, corresponding to two loop datapaths. Each vertex in G_1 and G_2 is identified by its label. For instance, vertex A_{23} is associated to the third unit of type A in graph G_2.

The resulting reconfigurable datapath is the merge of all N loop datapaths represented by $G_i, i = 1 \ldots N$. As before, it can also be modeled as a directed graph $G = (V, E)$ and a labeling function L of V, such that $G_i \subseteq G$, for all G_i. Each vertex $u \in V$ maps to one vertex v in at least one V_i, such that $L(u) = L_i(v)$ [2]. Moreover, each arc of E maps to one arc from at least one E_i. The final graph $G = \sum_{i=1}^{i=N} G_i$ is the result of overlapping all G_i, such that only vertices with the same label can be overlapped. A good solution for G should overlap vertices and arcs from all G_i as much as possible. Ideally, the optimum solution for G is the one in which: (a) $|E|$ is minimum (i.e. the resulting graph G has the smallest possible number of interconnections); and (b) the number of hardware blocks of type T in G is equal to the maximum number of blocks of that type encountered across all datapaths G_i. To compute that we have to find out which mapping of vertices from all G_i, among several possibilities, gives the best mapping of arcs, i.e. the one that overlaps the interconnections the most. The resulting graph $G = G_1 + G_2$ after merging G_1 and G_2 is shown in Figure 3. Each vertex (arc) in G is identified with a(n) vertex (arc) from G_1 and/or G_2.

[2]We say that two labels are the same if they are associated to the same type of hardware block.

3.2 The Compatibility Graph

We solve the problem of finding the resulting graph $G = \sum_{i=1}^{i=N} G_i$ using an arc mapping approach. Initially all possible arc mappings between two graphs G_i and G_j are generated. Two arcs (interconnections), (t, u) and (v, w), from G_i and G_j respectively, can be mapped (overlapped) if and only if, $L_i(t) = L_j(v)$ and $L_i(u) = L_j(w)$. In other words, the source vertex of the arcs must have the same label, as well as their destination vertices. Figure 4 lists those arcs from graphs G_1 and G_2 in Figure 3 that can be overlapped to each other. In Figure 4 each mapping is represented by a double-arrow line uniting the arcs that can be overlapped. We represent a possible mapping using a bar, e.g. in $(A_{11}, B_{11})/(A_{21}, B_{21})$ arcs (A_{11}, B_{11}) and (A_{21}, B_{21}) can be overlapped.

A compatibility graph H is constructed, where each vertex of H corresponds to a possible mapping of two arcs, one from G_i and another from G_j. There exist an edge between two vertices of H if the arc mappings represented by the vertices are compatible. In order to build the compatibility graph we need to define the notion of *mapping compatibility*. Two arc mappings are not compatible if and only if they map the same vertex of G_i to two different vertices of G_j, or vice-versa. This problem is illustrated in Figure 5. In that figure two loop datapath graphs G_i and G_j are shown. There are two possible arc mappings between G_i and G_j, which are $(A_{i1}, B_{i1})/(A_{j1}, B_{j1})$ and $(A_{i1}, C_{i1})/(A_{j2}, C_{j1})$. These two mappings are incompatible since they map the same vertex A_{i1} from G_i to two different vertices, A_{j1} and A_{j2}, in G_j.

By using the compatibility criterion discussed above the compatibility graph H can be easily constructed. Figure 6 shows the compatibility graph H resulting from the mappings of arcs from G_1 and G_2 in Figure 4. Consider, for

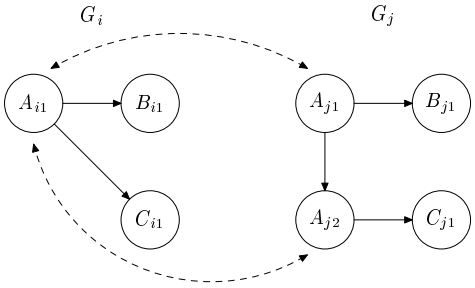

Figure 5: Incompatible mappings: A_{i1} maps to A_{j1} and A_{j2}.

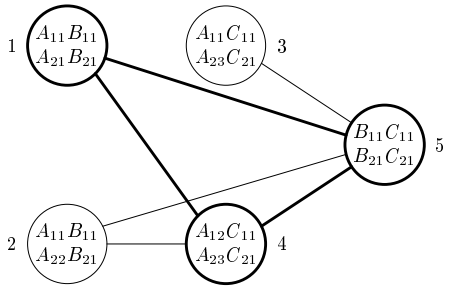

Figure 6: Maximum clique on the compatibility graph H.

example, mappings $(A_{11}, B_{11})/(A_{21}, B_{21})$ (vertex 1 in H) and $(B_{11}, C_{11})/(B_{21}, C_{21})$ (vertex 5 in H). For those mappings, no vertex from G_1 maps to two distinct vertices in G_2 and vice-versa. As a result, these two mapping are compatible, and an edge (1,5) is required in H. On the other hand, no edge exist in H between vertices 2 and 3. The reason is that the mappings represented by 2 and 3 are incompatible, since A_{11} in G_1 maps to both A_{22} and A_{23} in G_2.

3.3 Maximum Clique Solution

In order to determine the resulting graph $G = \sum_{i=1}^{i=N} G_i$ such that $|E|$ is minimum, it is necessary to find the maximum number of arc mappings that are compatible to each other. This can be achieved by computing the maximum clique of the compatibility graph H. The maximum clique problem is known to be NP-complete, and thus a heuristic polinomial-time algorithm is used to solve it [1]. For example, in the compatibility graph H of Figure 6, a possible maximum clique has vertices 1, 4, and 5.

Finally, the mappings represented by the vertices from the maximum clique of H are used to construct the resulting graph G. Each vertex from the clique gives an arc mapping between G_i and G_j (and their corresponding vertices). After that, the vertices from G_i that were not mapped can be mapped to any vertex of G_j, provided it has the same label and has not been mapped yet. If no such vertex is available the vertex of G_i is introduced into G. The same is also valid for the remaining vertices from G_j.

The solution presented above merges two datapath graphs. In order to merge several graphs, this method is used as a heuristic and applied iteratively. First, two input graphs are merged, then the resulting graph is merged with another input graph, and so on, as described in Algorithm 1. This algorithm runs in polynomial time, since we use a polynomial time heuristic for the maximum clique problem.

Algorithm 1 Datapath Merging

```
procedure Datapath_Merging(G_1, ..., G_N, G)
input: N directed graphs G_i = (V_i, E_i) and
  labeling functions L_i : V_i → T, 1 ≤ i ≤ N
output: directed graph G = (V, E) and
  labeling function L : V → T,
  such that |V| and |E| are minimum, and ∀u ∈ V,
  u maps possibly one vertex with label L(u) from
  each V_i

  /* Initially, G is the first input graph G_1 */
  G ← G_1;
  L ← L_1;
  /* Iteratively merge G with input graph G_i */
  for i ← 2 to N do
    H ← Construct_Compatibility_Graph(G, G_i, L, L_i);
    C ← Find_Maximum_Clique(H);
    (G, L) ← Reconstruct_Resulting_Graph(C, G, G_i, L, L_i);
```

4. EXPERIMENTAL RESULTS

The solution to the merging datapath problem presented above was applied to a number of programs from the MediaBench benchmark (MPEG, GSM, G.721 and ADPCM) [10]. Each program was compiled using the IMPACT compiler, and profiled at the *lcode* level, so as to determine which loops contributed the most to the program execution time. To maximize the speedup through datapath execution, predication and software pipelining between different loop iterations (if permitted by data dependence analysis) were applied. The code of each loop was synthesized as a loop datapath, from which a datapath graph was generated. The graphs were then merged iteratively using Algorithm 1. Three experiments were designed to evaluate the efficiency of the proposed algorithm.

In the first experiment, the number of interconnections of the final reconfigurable datapath was measured after merging all loop datapaths. This number was then compared to the number of interconnections produced by a previous technique [8]. As shown in Table 1, Algorithm 1, based on arc mapping, produces on average 24% fewer interconnections in its resulting datapath G, than the number of interconnections from datapath G', obtained using the vertex mapping approach described in [8]. For instance, 28.9% fewer interconnections result when Algorithm 1 is used to compute the GSM datapath.

The goal of the second experiment was to compare the number of interconnections $|E|$ of the resulting reconfigurable datapath G to its lower and upper bound. We define

Table 1: Comparison of arc mapping and vertex mapping approaches

Program	Datapath G' *		Datapath G †		Arcs reduction
	Vertices	Arcs	Vertices	Arcs	
MPEG	51	114	51	88	22.8%
GSM	67	197	67	140	28.9%
ADPCM	108	219	108	168	23.3%
G.721	42	77	42	62	19.5%

* Obtained with the vertex mapping approach [8]
† Obtained with the arc mapping approach

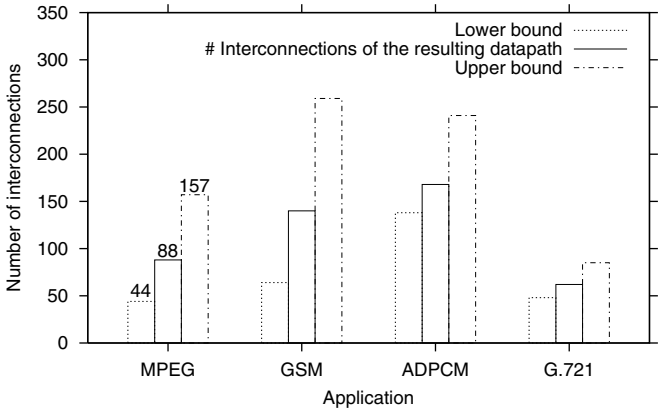

Figure 7: Number of interconnections, lower and upper bounds.

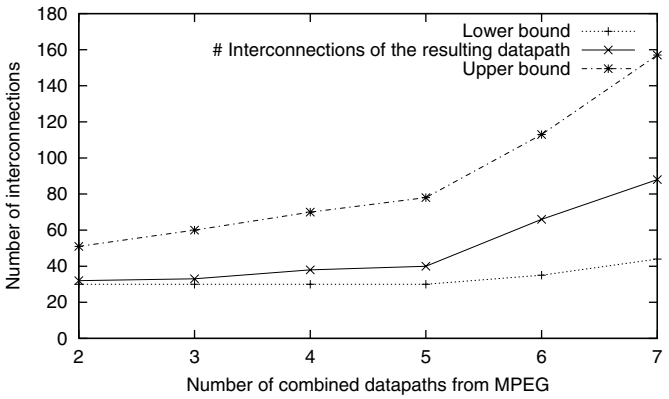

Figure 8: Number of interconnections, lower and upper bounds, for each iteration of MPEG.

the lower bound of $|E|$ as the maximum number of interconnections from one loop datapath, across all loop datapaths. The upper bound of $|E|$ is defined as the sum of the interconnections from all loop datapaths. So, for an application in which the N loop datapaths are represented by the directed graphs $G_i, i = 1 \ldots N$, we have:

$$\textit{Lower bound of } |E| = \max_{i=1}^{N} |E_i|$$

$$\textit{Upper bound of } |E| = \sum_{i=1}^{N} |E_i|$$

Notice that this lower bound is highly optimistic, while the upper bound is highly pessimistic. The lower bound represents the datapath merging in which all loop datapaths are entirely embedded into one of them, i.e. no extra interconnections are required. The upper bound represents the situation in which the loop datapaths are merged without any interconnections sharing.

Figure 7 shows the number of interconnections of the resulting datapaths, as well as its lower and upper bounds, for four applications from MediaBench. In Figure 8 we show the number of interconnections (and its lower and upper bounds) of the partial reconfigurable datapaths obtained at each iteration during the construction of the MPEG reconfigurable datapath. Since seven inner-loops were extracted, six iterations were needed to merge each loop datapath to the final datapath.

A preliminary ILP lower bound analysis [16] permitted the evaluation of the maximum error of the solution obtained with the proposed technique, when compared to an optimal solution (with respect to the sharing of interconnections), as shown in Table 2. This analysis showed that there is no resulting datapath with less than 81 interconnections, for the MPEG application. So, Algorithm 1 produced, in the worst case, only 8.6% more interconnections than the optimum solution, for this application. This result stress our observation that the lower bounds in Figures 7 and 8 are extremely optimistic. The lower bound from Figure 7 for MPEG is 44, but from ILP analysis is 81. For the ADPCM and G.721 benchmarks, our approach found optimal datapaths. The maximum error of the GSM solution is 44.9%. This is probably because more loop datapaths are merged, so more iterations of Algorithm 1 are required, accumulating the error produced by our greedy approach. Notice that this ILP analysis runs in exponential time, while our datapath merging algorithm is polynomial.

In the third experiment, we evaluated the ability of the proposed technique to enable the sharing of the interconnections from the datapaths of a given application. To do that, we measured the percentage of the interconnections in the final datapath which result from the overlap of n interconnections, $n = 1 \ldots N$, N the number of loop datapaths in the application. Notice that this measure reflects not only the ability of Algorithm 1 to maximize the interconnection sharing, but also how similar the structure of the datapaths are. Observe from Figure 9, that on average 50%-60% of the interconnections in the final datapath are the result of no overlap, i.e. around half of the resulting interconnections are not shared. Moreover, only nearly 30% of the interconnections are shared by two datapaths. As the *sharing number* (horizontal axis in Figure 9) increases, the percentage of interconnections in the final datapath having this sharing number decreases almost exponentially. For applications having more than three loops, we have not noticed any case where an interconnection of the resulting datapath is shared by all loop datapaths.

In Figure 9 the results for the G.721 application show that nearly 70% of the interconnections of the reconfigurable datapath are not shared. Interesting enough, from Table 2 we observe that the resulting datapath, constructed for this program using the proposed algorithm, is optimum, i.e., it has the minimum number of interconnections. We conclude from this that the sharing efficiency of this application is low because its loop datapaths are structurally very different.

5. CONCLUSIONS

This paper presented a novel graph-based technique for the datapath merge problem. Performance speedup can be achieved through architectures that map the most relevant application inner-loops to a reconfigurable datapath. We synthesized datapaths for each such loops and merged them together into a single reconfigurable datapath. Our approach merges the individual loop datapaths into a single reconfigurable datapath one at a time. At each step, it solves a maximum clique problem that matches FUs while maximizing the sharing of interconnections.

Experiments were performed to evaluate the efficiency of the proposed algorithm. First, the number of interconnec-

Table 2: Maximum error analysis (wrt. optimal)

Program	Datapath G † (# arcs)	Lower bound* (# arcs)	Maximum error
MPEG	88	81	8.6%
GSM	140	98	42.9%
ADPCM	168	168	0%
G.721	62	62	0%

† Obtained with the arc mapping approach
* Obtained with ILP analysis [16]

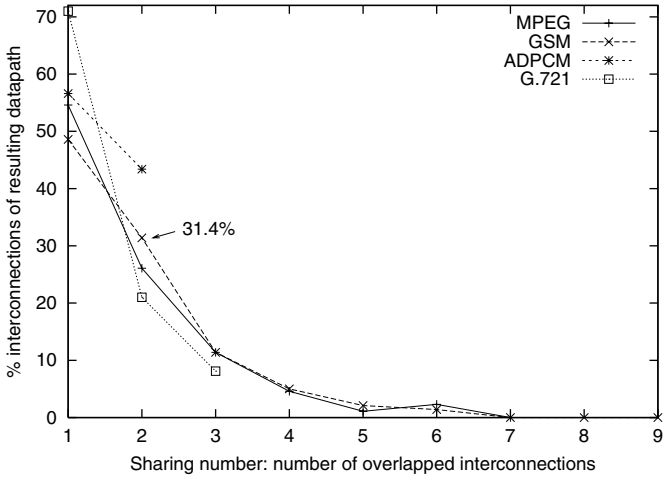

Figure 9: Interconnection sharing efficiency.

tions in the final reconfigurable datapath was compared to the number of interconnections produced by a previous approach and our technique produced, on average, 24% fewer interconnections. Moreover, an Integer Linear Programming lower bound analysis showed that, for two applications, our technique found optimum solutions. For the MPEG application, we produced, in the worst case, only 8.6% more interconnections than the optimum solution.

We also evaluated the ability of the proposed approach to maximize the interconnection sharing and how similar the structure of the datapaths are. We observed that an average 50%-60% of the interconnections in the final datapath are the result of no overlap, i.e. around half of the resulting interconnections are not shared. As the sharing degree increases, the percentage of interconnections in the final datapath having this degree decreases almost exponentially.

6. ACKNOWLEDGMENTS

This work was partially supported by ProTem-CC CNPq/NSF project 68.0059/99, CNPq research grant 300156/97, FAPESP research grant 2000/15083-9, and a CAPES fellowship award.

7. REFERENCES

[1] R. Battiti and M. Protasi. Reactive local search for the maximum clique problem. *Algorithmica*, 29(4), 2000.

[2] T. J. Callahan, J. R. Hauser, and J. Wawrzynek. The Garp architecture and C compiler. *IEEE Computer*, April 2000.

[3] P. P. Chang, S. A. Mahlke, W. Y. Chen, N. J. Warter, and W. W. Hwu. IMPACT: An architectural framework for multiple-instruction-issue processors. In *18th International Symposium on Computer Architecture*, 1991.

[4] K. Compton and S. Hauck. Reconfigurable computing: A survey of systems and software. *ACM Computing Surveys*, 34(2), 2002.

[5] C. Ebeling, D. C. Cronquist, and D. Franklin. RaPid – Reconfigurable pipelined datapath. *Lecture Notes in Computer Science 1142 — Field Programmable Logic: Smart Applications, New Paradigms and Compilers*, 1996.

[6] S. C. Goldstein et al. PipeRench: A reconfigurable architecture and compiler. *IEEE Computer*, 33, 2000.

[7] J. R. Hauser and J. Wawrzynek. Garp: A MIPS processor with a reconfigurable coprocessor. In *IEEE Symposium on Field-Programmable Custom Computing Machines*, 1997.

[8] Z. Huang and S. Malik. Managing dynamic reconfiguration overhead in systems-on-a-chip design using reconfigurable datapaths and optimized interconnection networks. In *Design, Automation and Test in Europe Conference*, 2001.

[9] Z. Huang and S. Malik. Exploiting operation level parallelism through dynamically reconfigurable datapaths. In *39th Design Automation Conference*, 2002.

[10] C. Lee, M. Potkonjak, and W. H. Mangione-Smith. MediaBench: A tool for evaluating and synthesizing multimedia and communication systems. In *30th Annual International Symposium on Microarchitecture*, 1997.

[11] A. Marshall et al. A reconfigurable arithmetic array for multimedia applications. In *ACM/SIGDA International Symposium on FPGAs*, 1999.

[12] T. Miyamori and K. Olukotun. A quantitative analysis of reconfigurable coprocessors for multimedia applications. In *IEEE Symposium on Field-Programmable Custom Computing Machines*, 1998.

[13] R. Razdan and M. D. Smith. A high-performance microarchitecture with hardware-programmable functional units. In *27th Annual International Symposium on Microarchitecture*, 1994.

[14] C. R. Rupp et al. The NAPA adaptive processing architecture. In *IEEE Symposium on Field-Programmable Custom Computing Machines*, 1998.

[15] N. Shirazi, W. Luk, and P. Cheung. Automating production of run-time reconfigurable designs. In *IEEE Symposium on Field-Programmable Custom Computing Machines*, 1998.

[16] C. C. Souza and A. M. Morais. *Private communication*. Applied Combinatorics Laboratory, IC-UNICAMP, March 2002.

[17] M. Weinhardt and W. Luk. Pipeline vectorization for reconfigurable systems. In *IEEE Symposium on Field-Programmable Custom Computing Machines*, 1999.

[18] A. Ye et al. Chimaera: A high-performance architecture with a tightly-coupled reconfigurable functional unit. In *27th Annual International Symposium on Computer Architecture*, 2000.