Resource Sharing in Custom Instruction Set Extensions

Marcela Zuluaga
g.m.zuluaga@sms.ed.ac.uk
Institute for Computing Systems Architecture
School of Informatics
University of Edinburgh

Nigel Topham
npt@inf.ed.ac.uk
Institute for Computing Systems Architecture
School of Informatics
University of Edinburgh

***Abstract*—Customised processor performance generally increases as additional custom instructions are added. However, performance is not the only metric that modern systems must take into account; die area and energy efficiency are equally important. Resource sharing during synthesis of instruction set extensions (ISEs) can reduce significantly the die area and energy consumption of a customized processor. This may increase the number of custom instructions that can be synthesized with a given area budget. Resource sharing involves combining the graph representations of two or more ISEs which contain a similar sub-graph. This coupling of multiple sub-graphs, if performed naively, can increase the latency of the extension instructions considerably. And yet, as we show in this paper, an appropriate level of resource sharing provides a significantly simpler design with only modest increases in average latency for extension instructions.**

Based on existing resource-sharing techniques, this study presents a new heuristic that controls the degree of resource sharing between a given set of custom instructions.

Our main contributions are the introduction of a parametric method for exploring the trade-offs that can be achieved between instruction latency and implementation complexity, and the coupling of design-space exploration with fast area-delay models for the operators comprising each ISE. We present experimental evidence that our heuristic exposes a broad range of design points, allowing advantageous trade-offs between die area and latency to be found and exploited.

I. Introduction

The customization of a processor through instruction set extensions is now a widely adopted technique in high performance embedded systems. One of the key challenges in the field of processor customization is how to increase processor speed across an application domain, without replicating logic which could otherwise be shared.

Most research to this point has assumed that each unique application will demand a uniquely customised processor. However, the non-recurrent engineering cost of producing an ASIC design increases with the introduction of each new technology generation. The approximate cost of creating a new SoC design has grown from $1 million in 1994 to current estimates of $20–50 million by 2010 [14]. A significant factor in the cost of each new chip design is the cost of mask production, which has grown from $100 thousand at 0.35μ to almost $9 million for a 45nm design [14].

This has an impact on the number of ASIC design starts, not least designs involving application-specific processors with instruction set extensions. Application specific instruction set processors (ASIPs) can be deployed easily on FPGA technologies, but FPGA cannot compete with ASIC implementation in die area (and therefore unit cost), energy efficiency, and maximum clock rate. Kuon and Rose [8] show that ASICs have an area advantage of at least 5x, a delay advantage of 3x or 4x, and a dynamic energy advantage of 14x. In mobile or low-power devices, where high performance and low cost are essential attributes, the standard cell ASIC approach remains highly competitive in all three key axes of performance.

Our work addresses these issues by focussing on the problem of how to explore the design space of customized processors which may support a wider collection of extensions, perhaps from an entire application domain, or indeed a large number of extensions from a single complex application.

Section II summarizes the prior work related to this topic, after which section III outlines our motivation for this research and presents the technical problem we address. Section IV then describes our proposal for a parameterised resource-sharing heuristic. Experimental methods and results are presented in sections V and VI, followed by concluding remarks in section VII.

II. Related Work

There is a significant body of previous work on automatic identification and selection of ISEs to create application specific processors [2], [6], [16]–[18], [20]. However, minimizing the area required to implement a set of ISEs is equivalent to the problem of constructing a minimal-cost weighted supergraph of a set of graphs, which is NP-Complete [4]. A heuristic approach to this problem is presented by Brisk *et al*., which transforms a set of ISEs into a single hardware datapath based on the classical problem of finding maximal subsequences and substrings thereof in the graph representation of ISEs [3]. The aim of their work is to maximize die area reduction through the construction of a *consolidation graph* representing merged ISEs. Similarly, Moreano *et al*. in [13] introduce a heuristic that uses the construction of a compatibility graph to reduce the problem of two data-path merging to a maximum weight

clique problem which is NP complete, they propose non-exact methods to solve this problem in polynomial time.

Our work differentiates itself from [3] and [13] in the introduction of latency constraints in the merging process, while they focus only in maximising the area savings.

Hardware resource sharing is also a design goal in the field of high-level synthesis. Zaretsky *et al*. present an algorithm for dynamically generating templates of re-occuring patterns for resource sharing in CDFGs [21], and Martinez and Kuchcinski present a constraint-solving approach based on graph-matching to implement resource sharing in CDFGs [12].

For the sake of simplicity and tractability these heuristics all assume a fixed cost for the area and delay of each type of operator. This is known to be idealistic [19]. This is corroborated by results in our paper, which show a wide range of die area and timing possibilities for typical arithmetic operators synthesised by commercial tools.

In [7], Ghiasi *et al*. present a polynomial time algorithm to maximise the delay assigned to operations in a DFG. The problem is solved by injecting delay to units until all the paths in the graph became critical.

Our paper builds on the work of Brisk *et al*. by introducing a *parametric* heuristic which uses the theory of timing budget management developed by Ghiasi *et al*. to allocate timing slack within a consolidation graph to non-critical nodes, thereby reducing their area-complexity.

The work of Lee *et al*. [10], Lorenz *et al*. [11], and Cheng and Tyson [5], has demonstrated that custom instruction-set extensions have a significant potential to reduce dynamic energy consumption. By helping to reducing die area, our work aims to also reduce static energy consumption.

III. Motivation

An ASIP design will be most cost-effective if it can address not a single application, but rather a whole *class* of applications. For example, an ASIP capable of efficient processing across the complete range of video standards has greater economy of scale compared with one which handles only MPEG2. When extending an instruction set to cover a complete class of applications we can expect larger numbers of extension instructions to be identified, effectively representing the union of the extension instructions required by each application in the class. Even with a single complex application it is possible to find large numbers of potential extension instructions, each of which adds to the die area of the system.

Clark *et al*. highlight the difficulty of finding exact subgraph matches in order to reuse application specific instructions across multiple applications in a single domain [6]. However, to avoid bloating the die area with large numbers of extension instructions, it is important to identify and exploit such commonality between instructions and, where possible, to share hardware resources when this represents a good trade-off between die area and execution time.

A. Problem definition

In this work we assume that instruction set extensions have been identified by a previous compiler phase, and they are represented as a collection of directed acyclic graphs (DAGs) annotated with execution frequency. The problem we address is how to merge such a collection of graphs to reduce the overall die area, whilst minimising the increase in execution latency.

Depending on the alignment of shareable paths in ISE graphs, we may find that the resulting latency is almost unchanged after merging, or we may find that latency increases significantly for some or all merged operations. Naturally we want to avoid merging a frequent operation with an infrequent operation if such a merge would add to the latency of the frequent one. Thus the optimisation process becomes highly complex when instruction latencies may be modified by merging. Figure 1 illustrates how the sharing of one node within graphs (a) and (b) creates a significantly longer path with multiplexers to isolate the graphs according to the operation they implement. As a minimum, the latency will increase due to muxing, but this form of sharing also creates the potential for structural hazards between extension instructions. To avoid creating functional units from merged extensions that produce outputs at different times, a pipelined implementation of the graph in figure 1.(c) may pipeline all paths to be the same length. In that case, the latency of both operations could be as large as the sum of the latencies of the graphs 1.(a) and 1.(b).

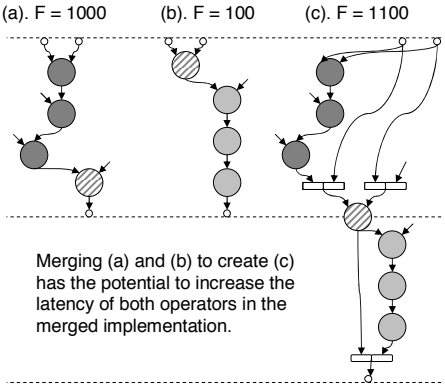

Fig. 1. Example of path merging

To evaluate the impact of graph-merging on die area and delay requires models for the operators of an ISE which reflect the wide variations in complexity that can be achieved under different timing constraints and logical context. For example, the marginal cost of an adder following a multiplier is less than the cost of an adder in isolation. Yehia *et al*. present examples of how arithmetic optimizations can reduce the combined latency of sequential operations [19]. Modern logic synthesis tools have the ability to perform similar arithmetic optimizations, such as folding ADD or SUBTRACT operations into the carry-save tree of a combinational multiplier. Even a single isolated arithmetic operator can be synthesised to a wide range of speed-area design points by specifying timing or area constraints during logic synthesis. Figure 2 shows how the gate count of a synthesised 32-bit floating-point adder can vary by

more than a factor of 2 as timing constraints are varied. For a 32-bit fixed-point adder the relative variation is even greater.

Design objectives will not always stipulate the extremes of minimum execution time, nor minimum die area: there are many possible intermediate points in the area-delay relationship, any one of which may be ideal for a given system. Our goal has been to develop a *parametric* resource-sharing algorithm that will enumerate extensive regions of the design space through the settings of a small number of real-valued parameters. Such an algorithm could be used in iterative design-space exploration methods, or may provide a means through which machine-learning based approaches can be trained to understand the characteristics of the design space. Those are topics for future work and are not addressed in this paper.

IV. Parametric Resource - Sharing Heuristic

The proposed heuristic is derived from a path-based resource-sharing algorithm, introduced by Brisk *et al*. in [3].

A DFG is a DAG represented by a set of vertices V and a set of edges E, where vertices are operators, inputs or outputs, and edges indicate the data dependencies between them. A path within a DFG is a sequence of vertices that traverses the graph, through the edges, from an input to an output.

Resource sharing is induced by the search for maximum common substrings between two paths. A maximum common substring is a subsequence of vertices that maximizes area reduction. The area of a substring is given by the sum of the areas of each operation within the substring.

A description of the proposed heuristic is illustrated in algorithm 1. The algorithm receives as inputs a set of n DFGs G_{in}, where each $G_i \in G_{in}$ represents an ISE to be synthesised. The algorithm is parameterised by three threshold values α_T, β_T and θ_T. Each of these is given a real value between 0 and 1. The output of the process is another set of graphs G_{out} containing the result of resource sharing.

The algorithm is divided into a *global* and a *local* phase. During each phase there is an exhaustive search for a maximum substring, comparing all pairs of paths belonging to different graphs. During global merging the maximum substring is referred as $MaxStrGlobal$ and in local merging as $MaxStrLocal$.

The global phase operates on G_{out}, which is initially copied directly from G_{in}. Consequently, before any resource sharing is applied, the number of input graphs is the same as the number of output graphs, *i.e.* $m = n$, where $m = |G_{out}|$. For each $G_i \in G_{out}$, a set of paths P_i is created with all the possible paths found in G_i. P aggregates all the sets of paths from P_i to P_m. Every path in P_i is compared with all other paths that belong to $P_{j \neq i}$ in order to find the $MaxStrGlobal$ between two DFGs. Graphs containing $MaxStrGlobal$, *i.e.* G_x and G_y will be merged into one graph G'. The process then switches to the local phase where a $MaxStrLocal$ is searched for, taking all pairs of paths of the merged graph G'; one path is found from G_x and the other from G_y. Once $MaxStrLocal$ is found, the paths are merged. The iterative search for further merging finishes when no further $MaxStrLocal$ instances can be found. The process goes back to the global phase where the number of graphs is decreased by one. This loop will be finished when no $MaxStrGlobal$ is found or when there is only one graph left in G_{out}.

A. Alpha and Alpha Threshold

Every graph G_i has an associated value α_i.

$$\alpha_i = F_i \times \frac{L'_i - L_i}{L'_i} \times (1 - M_i) \tag{1}$$

where F_i is the normalised execution frequency of G_i, defined by the execution frequency of G_i divided by the maximum execution frequency in the set G_{in}. L_i is the original latency of G_i, *i.e.* before the merging process. L'_i is the latency of G_i after being merged with other graphs. M_i is the percentage of area corresponding to operations in G_i that can be merged with other graphs, divided by the total area that could be merged in the whole process.

α_T is therefore a parameter that serves to omit graphs from the merging process if their corresponding ISE is executed very frequently, and if their latency, due to resource-sharing, is large. Additionally, this effect can be slightly cancelled out when the level of sharing found in G_i is high. The value of α associated with each graph, is compared with α_T to decide if the graph will be included in the merging process. As the value of α decreases, the probability to implement the graph separately increases.

B. Beta and Beta Threshold

Every G_i has an associated value β_i.

$$\beta_i = \frac{|\hat{L} - L_i|}{\max_{i=1}^{m} L_i} \times (1 - M_i) \tag{2}$$

where n is the number of input graphs and:

$$\hat{L} = \frac{\sum_i^n L_i}{n} \tag{3}$$

The β_T parameter tends to leave graphs unmerged if their latency is much larger than the rest of the ISEs. This is indicated by the difference between the average latency of all input graphs and the latency of the graph in question. If β_i is greater than β_T, G_i will not be considered during the merging process, thus preventing G_i from affecting the latency of the other graphs. The value of β associated with every graph is compared with the value of β_T in order to decide if the graph will be included in the merging process.

When global merging is finished, the values of β and α are calculated for every $G_i \in G_{in}$. These values will indicate if any input graph is to be left separate. The set of graphs G^* keeps track of the graphs that will not be included in merging. If $G^* \neq \emptyset$, the merging process will start again from $G_{out} = G_{in}$ - G^*.

1: $G_{out} \leftarrow G_{in}$
2: $G^* \leftarrow \emptyset$
3: $P^* \leftarrow \emptyset$
4: **for** j = 0 to 1 **do**
5: **if** j = 1 **then**
6: $G_{out} \leftarrow G_{in}$ - G^*
7: $P^* \leftarrow \emptyset$
8: **end if**
9: **repeat**
10: find paths in each graph of G_{out}: P=$\{P_1...P_m\}$
11: find maximum area common substring between
12: 2 graphs: $MaxStrGlobal$ such that $MaxStrGlobal \notin P^*$
13: merge graphs G_x and G_y that contain $MaxStrGlobal$ in G'
14: **repeat**
15: find maximum area common substring in G': $MaxStrLocal$
16: merge $MaxStrLocal$
17: **until** no $MaxStrLocal$ is found in G'
18: find critical path of G'
19: find area of G'
20: find θ_x and θ_y
21: **if** $\theta_x < \theta_T$ and $\theta_y < \theta_T$ **then**
22: replace G_x by merged graph G'
23: remove G_y from G_{out}
24: $m \leftarrow m - 1$
25: **else**
26: $P^* \leftarrow P^* + MaxStrGlobal$ {exclude $MaxStrGlobal$}
27: **end if**
28: **until** no $MaxStrGlobal$ is found
29: **for all** $G_i \in G_{out}$ **do**
30: add multiplexors needed in G_i
31: find critical path of G_i
32: find area of G_i
33: **end for**
34: **if** j = 0 **then**
35: **for all** $G_i \in G_{in}$ **do**
36: find β_i and α_i
37: **if** $\alpha_i > \alpha_T$ or $\beta_i > \beta_T$ **then**
38: $G^* \leftarrow G^* + G_i$ {exclude G_i}
39: **end if**
40: **end for**
41: **if** $G^* = \emptyset$ **then**
42: **exit loop**
43: **end if**
44: **end if**
45: **end for**
46: **return** G_{out}

Algorithm 1: Parametric Resource Sharing

C. Theta and Theta Threshold

If graphs G_x and G_y are selected to be merged, a graph G' is obtained as a result. We define,

$$\theta_x = \frac{L_{G'} - L_x}{L_{G'}} \times \left(1 - \frac{A_x + A_y - A_{G'}}{A_x + A_y}\right) \quad (4)$$

$$\theta_y = \frac{L_{G'} - L_y}{L_{G'}} \times \left(1 - \frac{A_x + A_y - A_{G'}}{A_x + A_y}\right) \quad (5)$$

where $L_{G'}$, L_x and L_y are respectively the critical paths of G', G_x and G_y,and $A_{G'}$, A_x and A_y are respectively the areas of G', G_x and G_y.

In contrast with α and β, θ is a value that will be calculated every time merging two graphs is considered. The first term in each θ-equation represents the increase in latency, whereas the second term represents the area savings that result from merging G_x and G_y.

Every time a $MaxStrGlobal$ is found, a G' is formed and local merging is applied. When no further merging is possible in G', θ_x and θ_y are calculated. If either θ_x or θ_y is greater than θ_T, G' will not be considered and other opportunities for sharing will be searched for between the available graphs. Thus, the global phase starts again from G_{out} with m unchanged, forcing the search for another $MaxStrGlobal$ such that when G' is found, it satisfies θ_T. The set of paths P^* is used to record the paths that are not eligible for merging.

D. Area and Delay of Graphs

Several times during the execution of the algorithm, information regarding the latency and area of DFGs is required. It is not feasible to perform full logic synthesis for each graph in order to get accurate values. We therefore developed a piecewise-linear model of area and delay for each type of operation that can appear in a DFG. Each operation type is modelled by four discrete points from its curve of area versus delay. Each point represents a possible trade-off between hardware cost (area) and speed (delay). Figure 2 shows the curves that were obtained for a selection of the operators used.

The curves of area versus delay for each operator were obtained using Synopsys' DC Ultra synthesis tool and a 0.13μ standard cell library. Two points correspond to the extreme cases of minimum delay and minimum area. The other two points are intermediate points chosen in order to give a good approximation to the non-linear area-delay relationship using a small number of piecewise linear segments. Specific area and delay values for each operator are found by linear interpolation from the four known points on the curve.

The latency of an ISE graph is estimated as the sum of the minimum delay that can be obtained for each operator in the critical path of the graph.

The area estimation is found by adapting the budget management technique given by Ghiasi *et al*. in [7]. Each operator in the graph is initially assigned a latency and area given by its minimum delay point. Then a zero slack algorithm is applied in order to relax the area of the operators that are off the critical path. An iterative slack distribution process takes place until

no further slack is found in the graph. At this point, each operator has an area value that corresponds to the maximum permissible delay of the operator such that the critical path of the graph is not increased. The sum of the areas of all the operators in the graph determines the estimated ISE area.

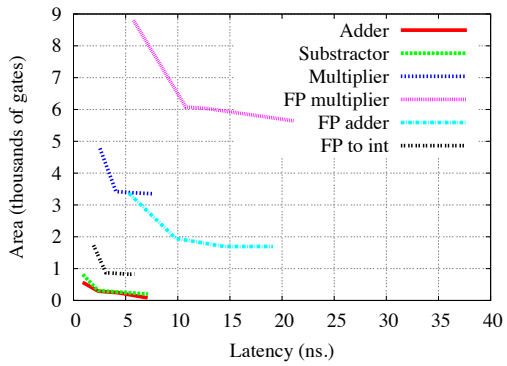

Fig. 2. Area-delay curves

V. Experimental Evaluation

We extracted a set of basic blocks from the Linear Predictive Coding (LPC) program, from the UTDSP benchmark suite, using an existing ISE identification technique. The DAGs from these blocks were presented as the initial set of n ISEs $G_{in} = G_1 \ldots G_n$ to our resource-sharing heuristic. Additionally, we include summarised results of performing the same experiment with other two benchmarks: ADPCM encoder from the UTDSP benchmark suite [9] and ADPCM encoder/decoder from SNU-RT benchmark suite [1]. This is with the intention of demonstrating that the results shown are consistent independently of the input set.

The characterists of the input sets can be seen in table I.

Benchmark	UTDSP LPC	UTDSP ADPCM	SNU-RT ADPCM
Number of ISEs	14	9	25
Largest area (gates[a])	66,296	44,370	43,550
Smallest area (gates)	12,324	8,834	292
Largest latency (ns)	29.97	16.67	39.92
Shortest latency (ns)	11.66	5.75	2.96
Max. number of operators	19	15	21
Min. number of operators	4	4	4

TABLE I
Input sets characterisation

[a]Gate counts are calculated as the standard cell area divided by the area of a strength-1 2-input NAND gate.

A. Algorithm Implementation

The parameterised resource-sharing algorithm was implemented as a system that takes input graphs expressed in XML. It performs resource sharing and outputs a description of the resulting merged logic in Verilog to enable subsequent logic synthesis and integration with an existing processor core.

For each experiment, the algorithm was executed several times with different values of α_T, β_T and θ_T, varying from 0 to 1 in steps of 0.05. This resulted in 9261 points to explore the design space of potential solutions. The execution time of the resource-sharing algorithm varied slightly with the value of θ_T. For the chosen set of inputs the average execution time was 40 seconds.

B. Design Space and Metrics

For our study, we have defined the latency of a particular ISE, L'_{out}, as the critical path of the, possibly merged, graph in which it appears. In order to evaluate a solution, every point which represents an implementation alternative has two associated metrics: *Application-specific Functional Unit (AFU) area* $\mathcal{A}$, and *average ISE latency* $\mathcal{L}$. $\mathcal{A}$ is the total area that the AFU will use, and can be computed by summing the area of each output graph in G_{out}. $\mathcal{L}$ is the average of the product $F \times L'_{out}$, over all ISEs considered as inputs.

Thus the design space is bounded by two opposing and extreme solutions: maximum resource sharing, with highly compromised delay, and no resource sharing, with minimal delay. The objective of our experiments is to expose a wide range of potentially interesting points in the design space, each showing a different trade-off between area and delay.

C. Multiplexer Insertion

Multiplexers have to be added at the inputs of a vertex when it has more than one predecessor per input as a result of sharing the resource associated with that vertex. In the case of vertices with just one input the solution is straightforward: an N-input multiplexer is added when there are N unique predecessors, and the number of selection bits is given by $\lceil log_2 N \rceil$. Vertices with two inputs can potentially have multiplexers in each input. Since multiplexors are inserted as a result of merging two different graphs, the predecessors are always from different operators. For this reason the input balancing problem exposed in [15] is not an issue in this process. Commutability of operations is exploited where any input of the ISE is part of the inputs of a two-input vertex in order to balance the assignment of inputs to the multiplexors.

The final area of the merged graphs in our experiments includes the contribution of all multiplexors inserted.

VI. Results

The resource-sharing solutions found as a result of executing the algorithm several times with varying parameters using the input set extracted from the LPC program, are plotted in figure 3. Five solutions that correspond to Pareto points of the design-space have been highlighted. Table II details the solutions represented by each of these points. The solution given by the No RS point has no resource sharing applied, and therefore has the smallest $\mathcal{L}$ but the largest $\mathcal{A}$. In contrast, for Max RS, where all parameters are set to 1, the algorithm merges all ISEs without any restrictions, as in non-parameterised approaches. This yields the smallest $\mathcal{A}$, but also the largest $\mathcal{L}$. Points Pareto 1, Pareto 2 and

Pareto 3, represent possible implementation alternatives with a better trade-off between $\mathcal{L}$ and $\mathcal{A}$. As shown in table II, a small reduction in area savings can yield a major reduction in latency, and a small increase in latency can yield a major reduction in area when the space is searched in the proposed manner.

The values of the three parameters that produce the five solutions are also specified in table II. For Pareto 1, one graph was left separate as it did not meet the constraint β_T=0.2. Similarly, the solution that represents Pareto 2 separates two graphs as they did not meet the contraint α_T=0.55. In a different way, Pareto 3 results in two output graphs that were consequence of constraining the merging process repetitively with θ_T=0.6. Thus, we show that the combination and variation of the 3 parameters is useful in finding good solutions.

We expect the solutions found by the resource-sharing techniques described in [3] and [13] to be comparable with the solution Max RS, as the focus of these previous methods is area reduction.

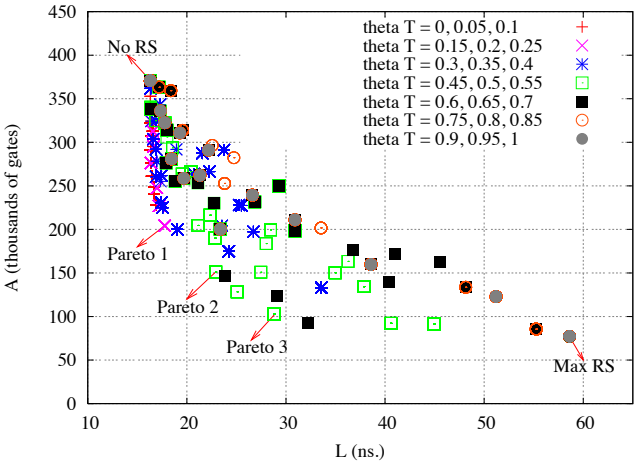

Fig. 3. Effect of varying α_T, β_T and θ_T

Point	$\mathcal{L}$	Area Saved	Output Graphs	β_T	α_T	θ_T
No RS	16.32 ns	0%	14	0	0	0
Pareto 1	17.77 ns	44%	4	0.2	1	0.2
Pareto 2	22.91 ns	59%	4	1	0.55	0.5
Pareto 3	28.90 ns	72%	2	1	1	0.55
Max RS	58.60 ns	79%	1	1	1	1

TABLE II
INTERESTING POINTS IN THE DESIGN SPACE

To illustrate the specific effect of each parameter, the graphs in figures 4 and 5 are included. In figure 4, three values of θ_T were selected and, for each value, β_T and α_T were varied. It can be noticed that as θ_T is reduced, the resulting solutions are pushed to the left. This illustrates how θ_T can be used to favor merged graphs, resulting in shorter latency ISEs. On the other hand, in figure 5, three values for β_T and α_T were chosen and for each of them θ_T was varied. In this case, β_T and α_T partition the points along the Pareto optimal curve. Thus, θ_T generates a *horizontal* exploration of the space while the other parameters are fixed. Conversely, β_T and α_T generate mainly a *vertical* search while θ_T is fixed.

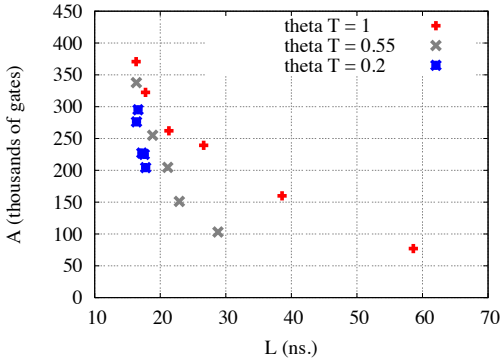

Fig. 4. β_T and α_T variation: $\theta_T \in \{1, 0.55, 0.2\}$

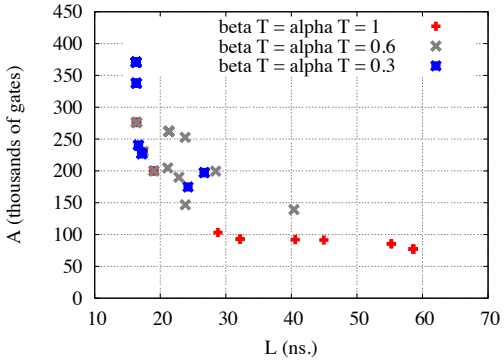

Fig. 5. θ_T variation: $\alpha_T, \beta_T \in \{1, 0.6, 0.3\}$

Figure 6 shows the sumarised results of the experiments performed with the three input sets described in table I. Latency and area values of three points in the design space were extracted in each of the three experiments. Max RS, No RS and an additional solution that has been taken from the resulting Pareto curve refered as solution P. P was chosen with the criteria of minimising the distance to the origin in the area-latency plane. From figure 6, it can be noticed that for all of the experiments, Max RS, as expected, has long latency and small area results. The point No RS shows always short latency and large area. In contrast, Max RS represents the solutions found as a result of the exploration of the parameterised space where there is an equilibrium between average ISE latency and area savings.

VII. Conclusions

This paper presents a new parametric algorithm for sharing hardware resources between multiple instruction set extensions that have been selected *a priori* for the performance improvements they can make to a given application. The algorithm combines a path-based resource-sharing algorithm

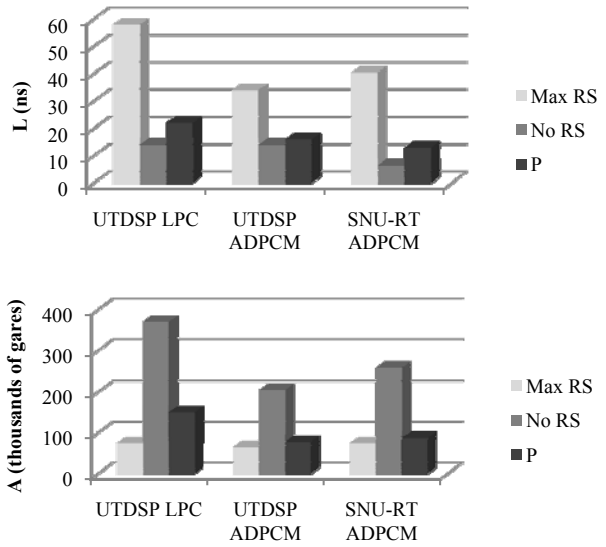

Fig. 6. Sumarised results from different input sets

with a timing budget management scheme to merge ISE graphs and allocate slack time in the resulting graphs so as to minimize implementation cost. The primary goal of our work has been to develop a method by which the design space of resource sharing can be explored. Our results show that die area is excessive when resource sharing is disabled, whereas very aggressive resource sharing leads to high latency. Our new heuristic is able to explore the space between these two extremes to find solutions that achieve the right compromise between latency and area. This work contributes to the goal of generating customised processors that support a whole class of applications, while still meeting timing constraints within a given silicon area.

ACKNOWLEDGMENT

The authors would like to thank the anonymous reviewers for their insightful comments and EPSRC for their financial support under grant EP/D50399X/1. Thanks go to Richard Bennett for providing the input ISEs used in the experiments reported in this paper.

REFERENCES

[1] SNU-RT real time benchmarks.

[2] ATASU, K., POZZI, L., AND IENNE, P. Automatic application-specific instruction-set extensions under microarchitectural constraints. In *DAC '03: Proceedings of the 40th conference on Design automation* (New York, NY, USA, 2003), ACM Press, pp. 256–261.

[3] BRISK, P., KAPLAN, A., AND SARRAFZADEH, M. Area-efficient instruction set synthesis for reconfigurable system-on-chip designs. In *DAC '04: Proceedings of the 41st annual conference on Design automation* (New York, NY, USA, 2004), ACM Press, pp. 395–400.

[4] BUNKE, H., GUIDOBALDI, G., AND VENTO, M. Weighted minimum common supergraph for cluster representation. *Proc. 2003 Int. Conf. on Image Processing, ICIP 2003 2* (14-17 Sept. 2003), II–25–8 vol.3.

[5] CHENG, A. C., AND TYSON, G. S. An energy efficient instruction set synthesis framework for low power embedded system designs. *IEEE Trans. Comput. 54*, 6 (2005), 698–713.

[6] CLARK, N., ZHONG, H., AND MAHLKE, S. Processor acceleration through automated instruction set customization. In *MICRO 36: Proceedings of the 36th annual IEEE/ACM International Symposium on Microarchitecture* (Washington, DC, USA, 2003), IEEE Computer Society, p. 129.

[7] GHIASI, S., BOZORGZADEH, E., CHOUDHURI, S., AND SARRAFZADEH, M. A unified theory of timing budget management. In *ICCAD '04: Proceedings of the 2004 IEEE/ACM International conference on Computer-aided design* (Washington, DC, USA, 2004), IEEE Computer Society, pp. 653–659.

[8] KUON, I., AND ROSE, J. Measuring the gap between FPGAs and ASICs. *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems 26*, 2 (Feb. 2007), 203–215.

[9] LEE, C., AND STOODLEY, M. UTDSP benchmark suite, 1992.

[10] LEE, J., CHOI, K., AND DUTT, N. D. Energy-efficient instruction set synthesis for application-specific processors. In *ISLPED '03: Proceedings of the 2003 international symposium on Low power electronics and design* (New York, NY, USA, 2003), ACM Press, pp. 330–333.

[11] LORENZ, M., MARWEDEL, P., DRÄGER, T., FETTWEIS, G., AND LEUPERS, R. Compiler based exploration of DSP energy savings by SIMD operations. In *ASP-DAC '04: Proceedings of the 2004 conference on Asia South Pacific design automation* (Piscataway, NJ, USA, 2004), IEEE Press, pp. 838–841.

[12] MARTINEZ, A. F., AND KUCHCINSKI, K. Graph matching constraints for synthesis with complex components. *10th Euromicro Conf. on Digital System Design Architectures, Methods and Tools, 2007. DSD 2007* (29-31 Aug. 2007), 288–295.

[13] MOREANO, N., BORIN, E. C. D. S., AND ARAUJO, G. Efficient datapath merging for partially reconfigurable architectures. *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems 24* (Jul. 2005), 969 – 980.

[14] PANDINI, D., DESOLI, G., AND CREMONESI, A. Computing and design for software and silicon manufacturing. *IFIP Int. Conf. on Very Large Scale Integration, 2007. VLSI - SoC 2007* (15-17 Oct. 2007), 122–127.

[15] PANGRLE, B. On the complexity of connectivity binding. *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems 10*, 11 (Nov. 1991), 1460–1465.

[16] POZZI, L., ATASU, K., AND IENNE, P. Exact and approximate algorithms for the extension of embedded processor instruction sets. *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems 25*, 7 (July 2006), 1209–1229.

[17] POZZI, L., AND IENNE, P. Exploiting pipelining to relax register-file port constraints of instruction-set extensions. In *CASES '05: Proceedings of the 2005 international conference on Compilers, Architectures and Synthesis for Embedded Systems* (New York, NY, USA, 2005), ACM, pp. 2–10.

[18] VERMA, A. K., BRISK, P., AND IENNE, P. Rethinking custom ISE identification: a new processor-agnostic method. In *CASES '07: Proceedings of the 2007 international conference on Compilers, Architectures and Synthesis for Embedded Systems* (New York, NY, USA, 2007), ACM, pp. 125–134.

[19] YEHIA, S., CLARK, N., MAHLKE, S., AND FLAUTNER, K. Exploring the design space of LUT-based transparent accelerators. In *CASES '05: Proceedings of the 2005 international conference on Compilers, Architectures and Synthesis for Embedded Systems* (New York, NY, USA, 2005), ACM, pp. 11–21.

[20] YU, P., AND MITRA, T. Scalable custom instructions identification for instruction-set extensible processors. In *CASES '04: Proceedings of the 2004 international conference on Compilers, Architectures and Synthesis for Embedded Systems* (New York, NY, USA, 2004), ACM, pp. 69–78.

[21] ZARETSKY, D., MITTAL, G., DICK, R., AND BANERJEE, P. Dynamic template generation for resource sharing in control and data flow graphs. *19th Int. Conf. on VLSI Design, 2006. Held jointly with 5th Int. Conf. on Embedded Systems and Design* (3-7 Jan. 2006).