Application-Specific Clustered VLIW Datapaths: Early Exploration on a Parameterized Design Space

Viktor S. Lapinskii, *Member, IEEE*, Margarida F. Jacome, *Senior Member, IEEE*, and Gustavo A. de Veciana, *Senior Member, IEEE*

***Abstract*—Specialized clustered very large instruction word (VLIW) processors combined with effective compilation techniques enable aggressive exploitation of the high instruction-level parallelism inherent in many embedded media applications, while unlocking a variety of possible performance/cost tradeoffs. In this work, the authors propose a methodology to support early design space exploration of clustered VLIW datapaths, in the context of a specific target application. They argue that, due to the large size and complexity of the design space, the early design space exploration phase should consider only design space parameters that have a first-order impact on two key physical figures of merit: clock rate and power dissipation. These parameters were found to be: maximum cluster capacity, number of clusters, and bus (interconnect) capacity. Experimental validation of their design space exploration algorithm shows that a thorough exploration of the complex design space can be performed very efficiently in this abstract parameterized design space. Moreover, an empirical study carried out on a representative set of computation-intensive benchmarks suggests that "penalties" of clustered versus centralized datapaths are often minimal and that clustering indeed unlocks a variety of valuable design tradeoffs.**

***Index Terms*—Application-specific processors, compilers, custom VLIW datapaths, design space exploration, embedded systems, power optimization.**

I. Introduction

A SIGNIFICANT segment of embedded multimedia applications exhibits high instruction-level parallelism (ILP) in the most time-critical inner loop bodies. Very large instruction word (VLIW) application specific instruction set processors (ASIPs) provide a means to efficiently exploit such ILP. In this work, we focus on early design space exploration of datapaths for these processors.

A "simple" VLIW datapath may include a centralized register file (RF) with several functional units connected to it through read and write ports. With a sufficient number of functional units and an adequate memory bandwidth, a compiler can potentially utilize all of the available static ILP present in the time-critical segments of the application. However, as the number of functional units (and thus register file ports) increases, "centralized" architectural solutions tend to become exceedingly costly [1]–[3] in terms of clock rate, power, area, and overall design complexity.

Manuscript received June 1, 2001; revised November 6, 2001. This work was supported in part by the National Science Foundation under ITR Grant ACI-0081791 and Grant CCR-9901255 and by the Texas Higher Education Coordinating Board under Grant ATP-003658-0649. This paper was recommended by Associate Editor R. Camposano.

The authors are with the Department of Electrical and Computer Engineering, University of Texas at Austin, Austin, TX 78712 USA (e-mail: lapinski@ece.utexas.edu; jacome@ece.utexas.edu; gustavo@ece.utexas.edu).

Publisher Item Identifier 10.1109/TCAD.2002.800451.

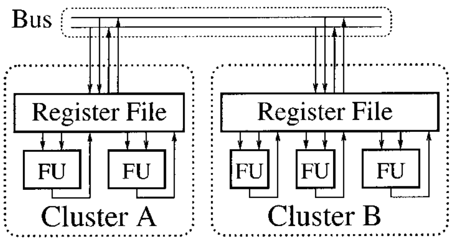

Fig. 1. Clustered datapath model.

In order to control the penalties associated with an excessive number of RF ports, while still providing all functional units necessary to exploit the available ILP, one can *restrict the connectivity* between functional units and registers, in particular, by structuring a VLIW datapath into a set of *clusters*. Each cluster in the datapath contains a set of functional units connected to a local register file (see Fig. 1). The idea of restricting connectivity is not new and in fact has been extensively used in ASIP synthesis [4]–[7] and in high-performance computer architectures [8]. Recent industry and research projects specifically targeting clustered VLIW architectures include [9]–[14]. In such architectures explicit *data transfer* operations are typically required to move data from one cluster to another, which may lead to increased schedule latency and energy consumption. The number and type of functional units instantiated in each cluster also directly affects the performance of the system.

The focus of this paper is on early design space exploration aimed at identifying clustered datapath configurations that are likely to be suitable for a given target application, i.e., effectively execute a (typically) small set of time/energy-critical code segments. For the vast majority of the applications of interest these code segments comprise a few loop kernels. Design space exploration of clustered VLIW ASIP datapaths involves complex tradeoffs among *clock rate*, schedule *latency*, and *power/energy* consumption. (Note that, as mentioned above, we focus on the application's time-critical loop bodies. Therefore, the latency of a kernel's schedule is, in fact, the *initiation interval* of the corresponding loop.) For example, as the number of functional units per cluster increases, one may expect the penalty associated with data transfers between clusters to decrease, leading to shorter schedule latencies. However, the corresponding increase in the number of register ports leads to a nonlinear (up to N^3) growth in combinatorial delay, power dissipation, and area [1]. Thus,

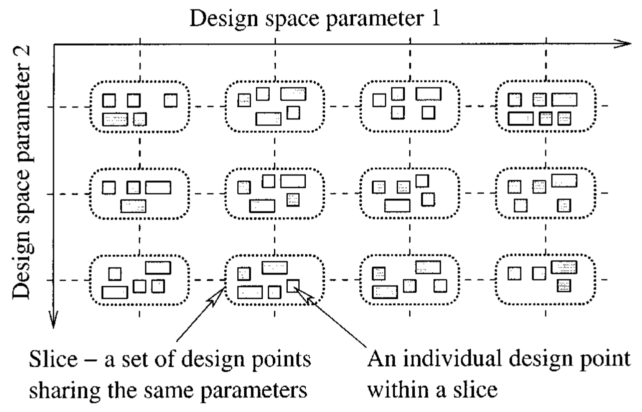
Design space parameter 1
Design space parameter 2
Slice – a set of design points sharing the same parameters
An individual design point within a slice

Fig. 2. Parameterized design space.

although the schedule latency (i.e., loop initiation interval) decreases, the clock rate may also have to be decreased, possibly leading to decreased throughput.

As is typical in dealing with complex CAD and compiler problems, to successfully explore the huge design space of clustered VLIW ASIP datapaths, we resort to *abstraction* and *problem decomposition*. Along with finding the appropriate abstractions, the challenge is to identify the critical dimensions of the design space, i.e., those capturing the salient performance/cost tradeoffs, and to develop complementary algorithms for design space exploration. As will be discussed shortly, this goal is achieved by using a suitable *parameterization of the design space* of clustered VLIW datapaths [15].

A. Figures of Merit

In our exploration, we consider two groups of figures of merit: physical and application-specific.

1) *Physical figures of merit* characterize the performance of a datapath itself, regardless of the specific code running on it. Examples of such figures include achievable clock rate f, power dissipation P, and silicon area A.
2) *Application-specific figures of merit* reflect characteristics which depend on the application. They are further divided in two subcategories.

— *Basic* application-specific figures of merit do not depend on physical ones. Examples include the number of clock cycles L in the schedule for a given kernel (i.e., loop body) and the number of data transfers between clusters N_{MV} during the execution of the kernel.

— *Derived* application-specific figures of merit do depend on physical figures of merit—for example, the kernel's energy consumption E and execution delay L_τ. The latter depends on the clock rate: $L_\tau = L/f$. We shall refer to the reciprocal of execution delay, $1/L_\tau = f/L$, as the throughput.

B. Design Space Parameterization

Let us introduce the concept of design space parameterization, initially in a domain-independent (more abstract) form. In our methodology, we partition the space into *design space slices*. By definition, elements of each slice share the same values for a set of *design space parameters* (see Fig. 2). The key idea of our approach is that the design space parameters must be carefully selected so that they have a *first-order impact* on the most important physical figures of merit. Intuitively, configurations within a given slice are likely to have a certain degree of commonality in terms of their physical figures of merit. This, in turn, enables decomposing the design space exploration process itself into exploration on the level of slices and exploration within each visited slice. The former focuses on considering datapaths with distinct physical figures of merit, the latter explores datapath configurations with respect to the corresponding basic application-specific figures of merit.

The goal of the exploration is to "sample" the design space, for various ranges of physical figures of merit, i.e., properly select a subset of design slices to be explored and, for each visited slice, identify a design point that is likely to be one of the best in the slice, in terms of its basic application-specific figures of merit.

C. Exploration on a Parameterized Design Space of Clustered VLIW Datapaths

The central idea driving the proposed early design space exploration approach is to make sure that the majority of relevant regions of the design space are adequately visited/sampled during the exploration, that is, to ensure that promising combinations of physical figures of merit (such as clock rate and power) are exposed during this early exploration. Accordingly, for each potentially interesting design space region (slice), our algorithm derives the corresponding basic application-specific figures of merit (e.g., the kernel's initiation interval) for a good datapath, representative of the slice.

In summary, the primary objective of the proposed early exploration phase is to expose good datapath configurations among the entire set of slices' representatives. These, in turn, are analyzed in subsequent exploration/design phases in terms of derived application-specific figures of merit.[1]

According to the discussion in Section I-B, we state that at our early exploration steps, one should identify and consider only the datapath configuration parameters that have a first-order impact on two key physical figures of merit: clock rate f and power dissipation P (silicon area, *per se*, is not necessarily of major importance, since today's fabrication technology allows one to cost-effectively place a large number of devices on a single chip [16]). Let us now consider the rationale behind identifying and selecting these design space parameters.

As mentioned earlier, clustered VLIW architectures address the problem of superlinear growth of delay, power, and area with the number of functional units (and thus—RF ports) in a centralized datapath. Thus, clustering can be viewed as enforcing hierarchical/distributed organization of the datapath resources, so as to enable scaling of the architecture to accommodate high levels of ILP. An additional advantage of designing systems that adhere to some form of hierarchical/structured organization is that such organization increases fidelity and accuracy of early estimation of some key system figures of merit (such as delay, power, and area) [1].

[1]At that later phase (not addressed here), detailed optimizations can be performed on this reduced set of clustered datapaths.

The parameters that describe the hierarchical structure[2] of a clustered datapath should naturally expose the critical elements of such VLIW datapaths. These parameters are likely to form a basis of design space parameterization (introduced in Section I-B), as discussed in the sequel.

- First, observe that the intercluster communication structure (bus) is a global resource, in that it must connect all datapath clusters instantiated in the datapath. Therefore, as empirically verified also by Rixner *et al.* [1], the number of global buses instantiated in the datapath (which we call bus capacity) is likely to have a first-order impact on the datapath performance, power, and area.
- Second, the number of clusters instantiated in the datapath has a first-order impact on the total area of the datapath, and thus on the delay and power of the global communication structure, and therefore, again, is a parameter of major importance that should be accounted for during early exploration.[3]
- Finally, we consider maximum cluster capacity—a parameter of a more "local" nature defining the maximum number of issue slots used in any given cluster in the datapath. Capacity of a cluster directly correlates to the number of ports in the corresponding local register file. This parameter (unlike the total number of functional units in the datapath) gives us explicit control over the super linear growth of delay, power, and area with respect to the number of ports in a register file. Clusters with fewer functional units will have fewer register ports in our model and the register file delay in those units will be smaller. Thus, at this level of abstraction, the largest cluster dominates. Accordingly, maximum cluster capacity was chosen as a design space parameter.

We denote the above-mentioned design space parameters (first introduced in an earlier publication [15]) as follows:

- *cluster capacity*—number of functional units in the largest cluster—N_F;
- *number of clusters*—N_C;
- *bus (interconnect) capacity*—N_B.

As described in Section I-B, datapath configurations sharing the above parameters are grouped into a *design space slice* denoted by $S(N_F, N_C, N_B)$. The datapath in Fig. 1, for example, is a member of slice $S(3, 2, 2)$.

In summary, then, we explore the space by varying the ILP supported by individual clusters, the number of clusters, that is, the actual ILP supported by the VLIW machine, and the capacity of the intercluster communication network. Recall that the parameters exposed during our early first-cut exploration are aimed at aggressively identifying potentially good tradeoff regions for a given application, rather than synthesizing the final datapath configuration (i.e., fully implementing a specific point in the design space).

[2]Such hierarchical structure is sometimes called design style, see, e.g., Geurts *et al.* [4].

[3]Note that different datapath design styles, such as the ASU-based style of Cathedral [4] also expose the maximum number of hierarchical elements (i.e., ASUs) allowed on the datapath. In other words, they consider such a number as a primary exploration/search parameter.

For an early design space exploration to be informative, one must leave out all nonessential (secondary) parameters, as well as parameters that show a strong correlation/dependence to those which are already exposed. Register file size is such a parameter since it should be strongly correlated to the number of ports/issue width of a cluster. Indeed, it is known that schedules with high ILP increase register pressure [17]–[21]. The "cost" of hardware support for such ILP (cluster capacity N_F) is already exposed in the proposed parameterization. In a well-designed (balanced) datapath, the sizes of register files should be selected to adequately support the computation demands (ILP) enabled by the number of issue slots available on each cluster, and thus should strongly correlate to cluster capacity. Therefore, we dropped this parameter from explicit consideration, acknowledging that there is always a price to pay when automation and time considerations are introduced in a design flow.

Similar factors influenced our decision to remove from this early exploration phase the detailed characterization of the types of operations supported by functional units associated with each issue slot of a cluster (see Section II-E). It must be noted that the definition of special-purpose functional units is of primary importance in certain design methodologies, especially ones associated with designing high performance hardware accelerators, such as [4], [22], and [23]. In contrast, our target architecture style can be classified as "custom VLIW datapath" (see also Faraboschi *et al.* [9]) which targets processors that are supposed to execute not only the critical kernels, but the entire application or a significant part of it. Naturally, if this style alone fails to deliver the performance/energy targets, one or more hardware accelerators may need to be used for some of the time-critical segments of the application. Design of hardware accelerators is beyond the scope of this paper.

For simplicity (related to datapath implementation and compiler), in our custom VLIW architecture style we consider only functional units implementing primary operation types. Note that this decision is consistent with some top of the line commercial clustered VLIW DSPs, such as TMS320C6000 [13] which does not even contain multiply-accumulate units. In the research arena, a similar strategy was adopted by HP's Lx architecture [9]. Thus, our main concern during this exploration was to ensure a good utilization of costly RF ports, by using quasiuniversal functional units, and to focus on identifying tradeoffs, related to the amount of ILP supported by the machine (see Section II-E). As in the previous case, we acknowledge that some interesting design points may be missed during our early exploration, yet this is the cost of limiting the complexity of the design space.

The phases following the proposed early design space exploration (see Section II) include detailed synthesis of an actual datapath for the target machine. Detailed optimizations should be performed on candidate clustered datapaths members of the identified promising design space regions. These later steps are beyond the scope of this paper.

D. Overview of the Algorithm

Given a set of time-critical kernels of interest, the design space exploration algorithm presented in this paper assists the

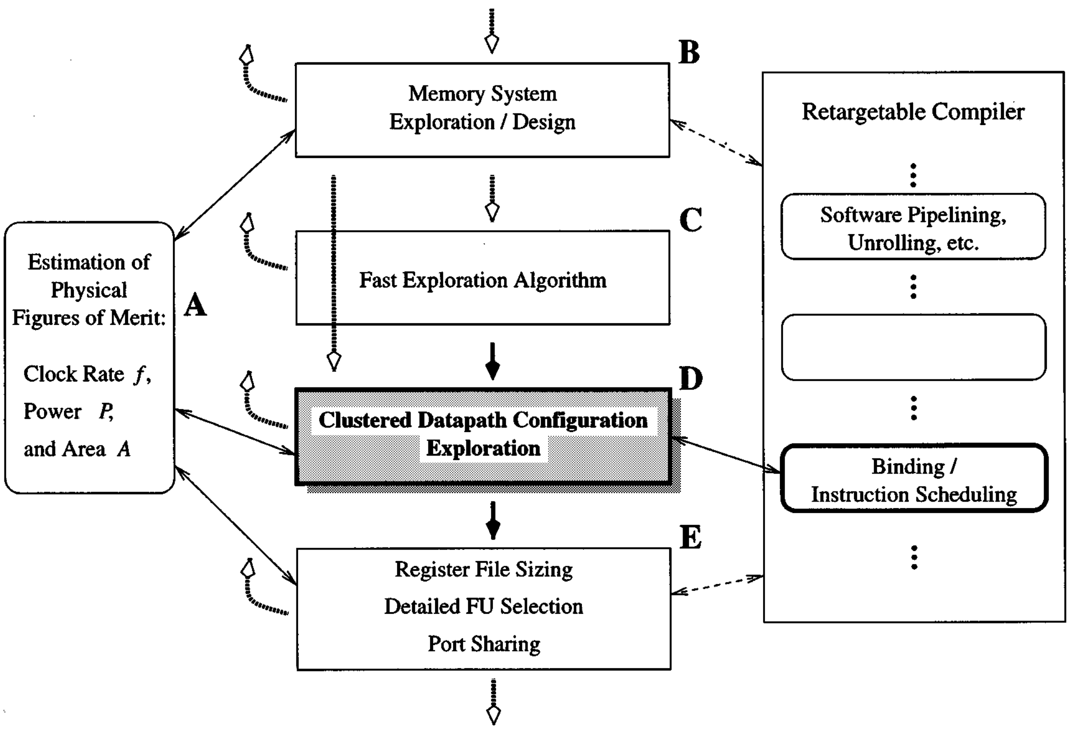
B
Memory System Exploration / Design
C
Fast Exploration Algorithm
D
Clustered Datapath Configuration Exploration
E
Register File Sizing
Detailed FU Selection
Port Sharing
A
Estimation of Physical Figures of Merit:
Clock Rate f,
Power P,
and Area A
F
Retargetable Compiler
Software Pipelining, Unrolling, etc.
Binding / Instruction Scheduling

Fig. 3. The context for our design space exploration phase.

designer in aggressively pruning the huge design space down to a limited number of promising datapath configurations. The inputs to the algorithm are the time-critical application kernels in a form of dataflow graphs and, optionally, ranges for the key parameters N_F, N_C, and N_B to be considered during the exploration. The search/exploration is systematically performed on a highly structured ("sliced") design space—as illustrated in Fig. 5. For each design space slice $S(N_F, N_C, N_B)$, the algorithm finds the most *promising datapath configurations* focusing on the *basic* application-specific figures of merit discussed above.

A promising configuration is one likely to optimize such basic application-specific figures of merit—in particular, the schedule latency (initiation interval) L and the number of data transfers between clusters N_{MV} for that optimal/minimum L. As will be shown later, the search/exploration within each slice $S(N_F, N_C, N_B)$ of the design space is performed by considering the execution of the target set of kernels on a carefully selected small subset of datapath configurations in S.

E. The Underlying Compilation Techniques

Code generation for processors with "irregular" architectural features [17], such as distributed RF organization found in clustered machines, require special, sophisticated techniques. At the core of our exploration methodology lies an efficient algorithm for binding operations in a dataflow graph (usually a time-critical loop body) to the clusters of a given datapath configuration. The binding algorithm is designed to generate solutions that first minimize latency and, second, minimize the number of data transfers. The core binding algorithm, designed to deliver high-quality results, comprises two phases: 1) generation of an initial binding solution and 2) iterative improvement of the initial solution. Details of our algorithm can be found elsewhere [24], [25].

The rest of the paper is organized as follows. Section II introduces the general framework that includes our design space exploration phase. Based on empirical observations on the structure of the "promising" datapaths over design space slices in Section III, we propose an efficient algorithm for identifying the set of datapath configurations within each slice that are likely to be effective for the kernels of interest. In Section IV, we present experimental results generated by the proposed design space exploration algorithm for a set of representative benchmarks. Our case studies show that the algorithm thoroughly explores the large and complex design space and does so in an efficient way by carefully selecting for consideration only a small subset of datapaths within each slice. We further present the results of an empirical study of performance/cost tradeoffs associated with clustered versus nonclustered machines. We contrast our work with previous research in Section V and conclude in Section VI.

II. Context for the Design Space Exploration Phase

Fig. 3 shows the exploration framework that includes our proposed design space exploration phase. In what follows, we briefly discuss its components in order to provide the context for the proposed exploration phase.

A. On Estimation of Physical Figures of Merit

As mentioned above, our algorithm (see block D in Fig. 3) finds datapath configurations that are "promising" with respect to the *basic application-specific* figures of merit L and N_{MV}. Indeed, one of the distinctive features of our design space exploration methodology is its independence of implementation technology. A key advantage of such an approach is that, by decoupling physical from application-specific figures of merit, collections of "promising" datapath configurations can be generated up-front for various fundamental algorithm kernels. The resulting design space characterization (see Section IV, Fig. 6 for an example) may be stored in an *implementation-independent library*, together with the kernels, for future use.

When an application-specific processor for these kernels needs to be implemented in a new target technology, f and P estimates can be determined for the new technology [see Fig. 3(a)] and used to compute the *derived* figures of merit. Estimation of physical figures of merit is beyond the scope of this paper; see, e.g., [4] and [26] for a good overview of such techniques.

B. Memory Subsystem

Memory subsystem design decisions typically have a major impact on the performance of data-intensive high-throughput applications; see e.g., [4], [27]–[30]. Thus, design space exploration pertaining to memory organization precedes our datapath exploration phase, as indicated in Fig. 3(b). The output of this phase should include a broad schedule of memory operations, possibly defining a lower bound on schedule latency (or initiation interval) L. Note that these load and store operations may be software pipelined, in order to hide their latencies.[4]

C. Narrowing the Design Space

The efficiency of the overall design space exploration process may be a concern when one needs to explore the design space for a *large* kernel. Under such conditions, the fast design space exploration algorithm proposed in our previous work [15] can be used first, as shown in Fig. 3(c) This will help to quickly identify the ranges of parameters N_F, N_C, and N_B of interest for such kernels, i.e., the area of the design space that ought to be explored in detail by the proposed design space exploration algorithm.

D. Register Files

As shown in Fig. 3(e), after our design space exploration phase identifies promising datapath configurations, a more detailed exploration is to follow, including register file sizing. Indeed, as mentioned in Section I-C, during our design space exploration we do not explicitly size the register file in each cluster. Yet, we assume that the sizes of these register files *will* be such that the number of expensive spills to memory (i.e., expelling of *intermediate* results from a register file) will be small and not significantly affect performance. We argue that this is a reasonable assumption, since in general it is undesirable to have a machine with a large number of functional units and yet have spills to memory degrading performance to a level achievable by a "smaller" machine, i.e., by a machine with less parallelism. If this is the case, a more compact machine/datapath (also to be found by our design space exploration algorithm) is a superior solution for that performance level.

[4]If high throughput is desirable, the full kernel may be software pipelined to decrease the initiation interval [31].

E. Functional Units

Functional unit selection and port sharing exploration are also performed after our design space exploration phase [see Fig. 3(e)]. Indeed, one of the key abstractions underlying our approach lies in the special treatment given to functional unit types. We argue that during early design space exploration, the precise definition of functional unit types should be postponed and abstract types coarsely grouping the operation types pertinent to the kernel of interest should be used instead. This abstraction allows us to assume that exclusive (nonshared) register ports are assigned to each such functional unit instantiated in a cluster, while still ensuring a good utilization of such costly resources as register file ports. In other words, given N_F (the maximum number of functional units per cluster defined for a slice S), by using abstract/coarse functional units we are likely to generate good performance estimates for a specified kernel, without engaging in a prohibitively costly simultaneous search for precise port sharing configurations among fine-grained functional units.

Note that these abstract functional unit types are not intended to be actually used in the final datapath implementation. Once the set of promising clustered datapath configurations is identified, more detailed specialization steps should be undertaken in order to refine these "coarse-grain" functional units , as indicated in Fig. 3(e). Such refinement steps include the implementation of each abstract type in terms of a number of actual functional units sharing an issue slot (i.e., register file ports), the possible elimination of support for selected (fine grain) operation types on individual issue slots, etc.

III. Design Space Exploration

We start this section by defining the quality metrics used for identifying promising datapath configurations. Then, we define our proposed approach for searching for these promising configurations in the parameterized design space and give some insights on the characteristics of this design space. Finally, we will present our design space exploration algorithm and show that it efficiently takes advantage of these characteristics.

A. Quality Metrics

We define the combined latency (initiation interval) L of an application's set of time/energy-critical loop kernels (executed on a given datapath) to be the weighted sum of the latencies (initiation intervals) of those individual kernels. The weights reflect the relative criticality of the kernels in the application. For simplicity of the notation, throughout the rest of this section, we assume this set contains a single kernel. For a given datapath $d \in S(N_F, N_C, N_B)$, we let $L(d)$ and $N_{MV}(d)$ denote the latency and the number of moves obtained using an efficient binding and scheduling of a target kernel on d, and let $T_F(d)$ denote the total number of functional units instantiated in the datapath d.

The datapath $d \in S(N_F, N_C, N_B)$ is said to be *lexicographically minimal* if its associated vector $(L(d), M_{MV}(d), T_F(d))$ is lexicographically the smallest among all datapaths in S. In other words, it achieves the minimum latency, and, then, among

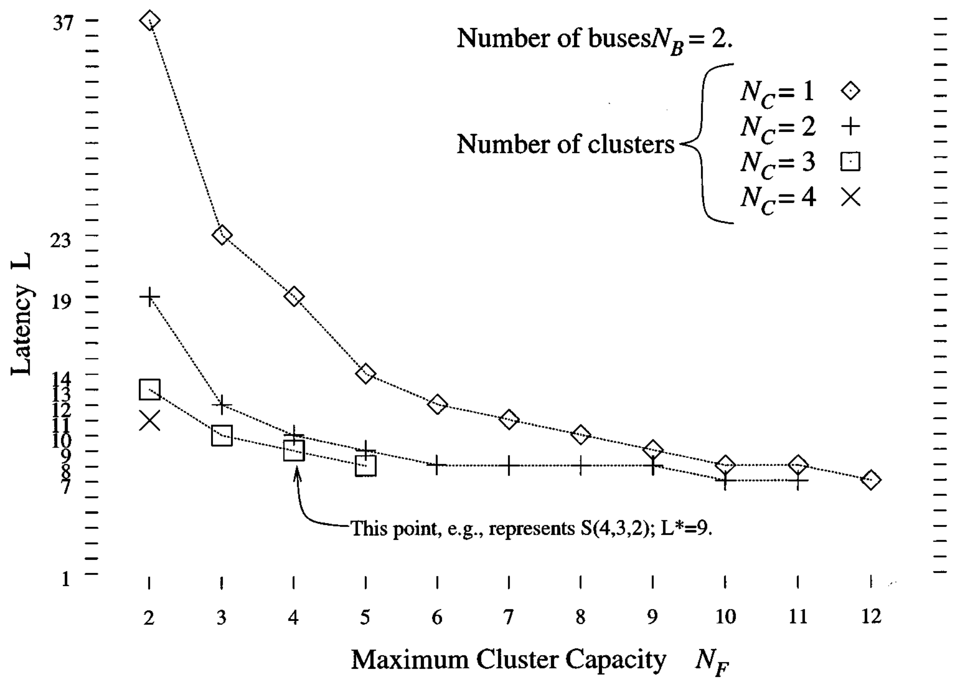

Fig. 4. Relevant slices of design space and their corresponding best latencies L^* for DCT-DIT.

all datapaths with the minimum latency, it achieves a minimum number of moves and, finally, among those it includes a minimum number of functional units. The rationale for this ordering is to give priority to minimizing latency, then reducing the energy consumption associated with moves, and finally the total number of required functional units. We noted, however, that in practice it could be useful to identify sets of possible configurations achieving the minimum latency but relax the requirement on the minimum number of moves, while attempting to minimize the total number of functional units allowed in the datapath. Thus, we decided not to limit ourselves to the lexicographically optimal datapath configurations mentioned above. We formalize the relaxations implemented in our approach as follows:

$$\begin{aligned}
L^* &= \min_{d \in S(N_F,N_C,N_B)} L(d)\\
D_{L^*} &= \operatorname*{argmin}_{d \in S(N_F,N_C,N_B)} L(d)\\
N^*_{MV} &= \min_{d \in D_{L^*}} N_{MV}(d)\\
D_{L^*,N^*_{MV}} &= \operatorname*{argmin}_{d \in D_{L^*}} N_{MV}(d)\\
T^*_F &= \min_{d \in D_{L^*,N^*_{MV}}} T_F(d)\\
D^* &= \{d | d \in D_{L^*} \text{ and } T_F(d) \le T^*_F\}.
\end{aligned}$$

In other words, in the context of a given basic block and a slice $S(N_F, N_C, N_B)$ of the design space:

1) L^* is the best achievable latency (initiation interval) among the datapaths within the slice;
2) D_{L^*} is a subset of $S(N_F, N_C, N_B)$ such that each datapath in D_{L^*} exhibits the optimal latency L^*;
3) N^*_{MV} denotes the minimum number of moves achievable for the best latency L^*;
4) $D_{L^*,N^*_{MV}}$ is a subset of D_{L^*}, where each datapath d produces the minimum number of moves N^*_{MV};
5) T^*_F denotes the minimum total number of functional units in a datapath achieving a minimum latency L^* and a minimum number of moves for that L^*;
6) D^* is the set of datapath configurations which achieve the minimum latency and have no more functional units than a lexicographically minimal solution, but may require more moves.[5]

Later on, we sometimes specify the parameters of the slice explicitly, e.g., $L^*(N_F, N_C, N_B)$ or $D^*(N_F, N_C, N_B)$.

B. Searching the Design Space

In order to investigate and gain insight into the challenging problem of exploring the design space of clustered datapath configurations, we started by developing a simple brute-force algorithm that recursively enumerates all datapaths within a specified slice and performs an exhaustive search over all slices within a given set of parameter ranges. Thus, for each datapath $d \in S(N_F, N_C, N_B)$, we computed $L(d)$ and $N_{MV}(d)$. Among all datapaths in $S(N_F, N_C, N_B)$, we then identified lexicographically minimal configurations, as well as the set D^* of "promising" configurations. Fig. 4 shows a representative set of results for one such exhaustive exploration, for the DCT-DIT benchmark (see benchmark descriptions in Section IV). Each point on the graph corresponds to the best (minimum) latency L^* achieved by executing the kernel on datapaths D^* within a given slice $S(N_F, N_C, N_B)$. In this

[5]Observe that the set of configurations of interest D^* includes all lexicographically minimal datapaths.

TABLE I
DCT-DIT KERNEL: DATAPATH CONFIGURATIONS D^* FOR SLICES WITH $N_B = 2$

$N_F \rightarrow$ $N_C \downarrow$	2	3	4	5	6
1 (◇)	$\boxed{\|1,1\|} - 0$ $L^* = 37$	$\boxed{\|2,1\|} - 0$ $L^* = 23$	$\|3,1\| - 0$ $\boxed{\|2,2\|} - 0$ $L^* = 19$	$\boxed{\|3,2\|} - 0$ $L^* = 14$	$\boxed{\|4,2\|} - 0$ $L^* = 12$
2 (+)	$\boxed{\|1,1\|1,1\|} - 7$ $L^* = 19$	$\boxed{\|2,1\|2,1\|} - 7$ $L^* = 12$	$\|2,2\|2,2\| - 8$ $\boxed{\|3,1\|2,2\|} - 5$ $L^* = 10$	$\|3,2\|3,1\| - 8$ $\|4,1\|2,2\| - 7$ $\|3,1\|2,3\| - 7$ $\|3,2\|2,2\| - 6$ $\boxed{\|4,1\|3,2\|} - 5$ $L^* = 9$	$\|2,2\|4,2\| - 5$ $\boxed{\|3,3\|4,2\|} - 4$ $L^* = 8$
3 (□)	$\boxed{\|1,1\|1,1\|1,1\|} - 13$ $L^* = 13$	$\|1,1\|2,1\|2,1\| - 13$ $\boxed{\|2,1\|2,1\|2,1\|} - 11$ $L^* = 10$	$\|2,1\|2,1\|2,2\| - 12$ $\boxed{\|1,1\|3,1\|2,2\|} - 7$ $L^* = 9$	$\boxed{\|1,2\|4,1\|3,2\|} - 7$ $L^* = 8$	
4 (×)	$\boxed{\|1,1\|1,1\|1,1\|1,1\|} - 13$ $L^* = 11$				

$N_F \rightarrow$ $N_C \downarrow$	7	8	9	10	11	12
1 (◇)	$\|5,2\| - 0$ $\boxed{\|4,3\|} - 0$ $L^* = 11$	$\|6,2\| - 0$ $\|5,3\| - 0$ $\boxed{\|4,4\|} - 0$ $L^* = 10$	$\|6,3\| - 0$ $\boxed{\|5,4\|} - 0$ $L^* = 9$	$\|7,3\| - 0$ $\boxed{\|6,4\|} - 0$ $L^* = 8$	$\|8,3\| - 0$ $\|7,4\| - 0$ $\boxed{\|6,5\|} - 0$ $L^* = 8$	$\boxed{\|7,5\|} - 0$ $L^* = 7$
2 (+)	inferior $L^* = 8$	inferior $L^* = 8$	inferior $L^* = 8$	$\|3,1\|5,5\| - 5$ $\|2,2\|5,5\| - 4$ $L^* = 7$	$\|1,1\|6,5\| - 4$ $\boxed{\|1,2\|6,5\|} - 3$ $L^* = 7$	

example, $N_B = 2$, N_C is represented with different glyphs (see the legend), and N_F changes along the horizontal axis. Slices $S(N_F, N_C + k, N_B)$ for $k > 0$, which achieve the same latency L^* as slice $S(N_F, N_C, N_B)$, are deemed inferior and are not shown in the graph.

For each slice associated with a point in Fig. 4, Table I shows the corresponding relaxed "promising" datapath configurations in D^*, as described in Section III-A. In the text, we use a vector-set notation: (Ai, Mj) represents a cluster with i ALUs and j multipliers. For presentation purposes, clusters in the tables are shown as $|i, j|$ pairs, where i denotes the number of ALUs and j is the number of multipliers in the corresponding cluster. The number on the right of each symbolic datapath configuration in Table I is the number of data transfers N_{MV} needed when executing the kernel on that datapath. Every configuration highlighted with a box is a member of the slice's D^*, identified by our design space exploration algorithm, to be discussed shortly.

Let us consider the set D^* for the slice $S(N_F, N_C, N_B) = S(6, 2, 2)$, shown in Table I. This D^* contains two datapath configurations: $D^* = \{\{(A2, M2), (A4, M2)\}, \{(A3, M3), (A4, M2)\}\}$. By definition, both configurations exhibit the same latency $L^* = 8$. Note however, that the first configuration $\{(A2, M2), (A4, M2)\}$ has fewer functional units at the expense of one additional data transfer. The three cells in Table I marked "inferior" represent slices whose large D^* sets only contain configurations inferior to the best ones in $S(2, 6, 2)$. Specifically, the best configurations in these slices have the same latency as $S(2, 6, 2)$ (i.e., $L^* = 8$) but higher cluster capacity N_F (they correspond to a horizontal segment with the "$+$" glyphs in Fig. 4).

C. Design Space Exploration Algorithm Description

Despite the high level of abstraction at which the search is performed, the design space remains large, i.e., any exhaustive algorithm becomes quickly impractical. For example, the exhaustive exploration for the DCT-DIT benchmark discussed above took over 20 h. Thus, we felt that there was a need to further prune the search space.

Based on the data we collected for reasonably sized benchmarks, we made an important *structural observation* concerning lexicographically minimal datapaths across slices of the design space. We observed that, typically, in $D^*(N_F, N_C + 1, N_B)$ there exists a datapath d_j, such that

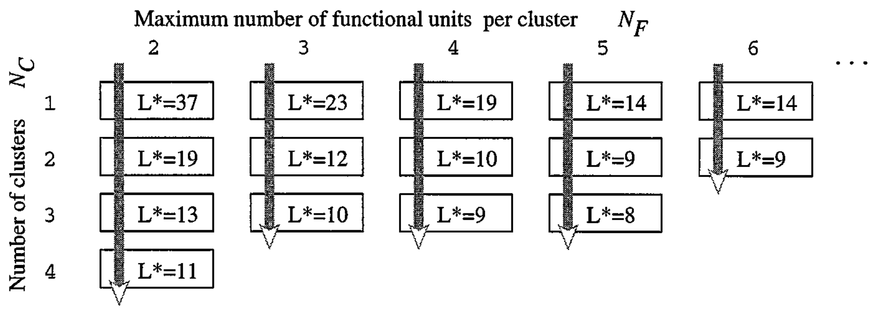

Fig. 5. Traversal of design space slices.

N out of its $N+1$ clusters match those in some datapath d_i contained in $D^*(N_F, N_C, N_B)$:

$$\exists d_i \in D^*(N_F, N_C, N_B),$$
$$d_j \in D^*(N_F, N_C+1, N_B) : d_j \subset d_i.$$

For example, in Table I, consider the set of configurations D^* in $S(5,\mathbf{3},2)$ (i.e., $N_F = 5$, $N_C = 3$, $N_B = 2$). It consists of one datapath: $d_i = \left\{(A1, M2), \underline{(A4, M1), (A3, M2)}\right\}$. In $S(5,\mathbf{2},2)$, there is a configuration $d_j = \left\{\underline{(A4, M1), (A3, M2)}\right\}$ such that $d_j \subset d_i$ (the common clusters are underlined). Similarly, for the single-cluster configuration, $d_k = \{(A3, M2)\} \subset d_j$. Note that the same kind of regularities can be observed in all but one column of Table I.

We conjecture that this characteristic is a natural property of the binding/clustering problems addressed in this paper and may be applied to exploring the design space. By doing exhaustive exploration on a representative set of benchmarks (see Section IV), we verified that this property could be robustly applied to improve efficiency of early design space exploration, since only a few (around 5%) potentially good design points were missed or incorrectly characterized as a result of such heuristic pruning. This simple observation permitted our design space exploration algorithm to dramatically prune the design space with no significant impact on the quality of its results.

Our proposed algorithm generates a promising datapath configuration for each design space slice, as presented in Table I (boxed configurations). In other words, for each slice S, our algorithm identifies one datapath configuration, a member of the corresponding set D^*.[6] The algorithm works in a context of a given bus capacity N_B and traverses the slices S of the design space as illustrated in Fig. 5. It begins with a current maximum cluster capacity (default initial value $N_F = 2$) and number of clusters $N_C = 1$. Then, it starts increasing the number of clusters, which corresponds to filling a column of cells in Table I. The "vertical" iteration over N_C stops when adding a new cluster ceases to improve the kernel performance (i.e., L^*).[7] The algorithm then increases N_F and starts building another column of the table. We consider the slices with smaller parameter values first, i.e., we start with datapaths that are likely to have better f and P characteristics. Below is a more detailed description of the exploration process.

[6]Recall that D^* includes configurations with minimum latency (initiation interval) and no more functional units than in a lexicographically minimal solution, but possibly requiring more moves.

[7]Recall that the performance estimation relies on our cluster assignment algorithm and a simple scheduler.

For a given N_B, the central procedure, genNF(), is invoked for $N_F = 2, 3 \ldots$ until the critical path latency for the original (unbound) dataflow graph or the desired initiation interval (optionally given as input) is reached. As shown below, genNF() first initializes an empty (i.e., $N_C = 0$) datapath configuration and then repeatedly increases the number of clusters N_C by calling addBestCluster(). The procedure genNF() terminates when the introduction of an extra cluster does not improve (L_best, Nmv_best). A simplified pseudocode description of genNF() follows.

```
genNF(nf,nb){
 L = INFNTY
 Nmv = INFNTY
 d = emptyDatapath(nb)
 nc = 0
 do{
   L_best = L
   Nmv_best = Nmv
   d = addBestCluster(d) // the key element here
   nc++
   L = getL(d)
   Nmv = getNmv(d)
   improved = (L < L_best) or
    ((L == L_best) and (Nmv < Nmv_best))
   if (improved) storeDatapath(nf, nc, nb, d)
 }while(improved)
}
```

Inside genNF(), the procedure call addBestCluster(d) finds the best configuration for the new cluster (leaving the existing clusters "untouched") by executing three procedures: initialCluster(), balanceCluster(), and minimizeCluster(). We found through exhaustive experimentation that the best results are consistently obtained when initialCluster() allocates the available N_F functional unit slots evenly among different functional unit types (ALUs and multipliers, for our examples). When N_F does not permit an even assignment, the remaining slot is assigned to the functional unit type that supports the dominating operation type in the dataflow. Such initial balance delivers the best results

TABLE II
BENCHMARK SUITE

Name	# nodes	# connected components	critical path	exploration time, sec
EWF	34	1	14	196
ARF	28	1	8	47
FFT	38	1	4	1705
DCT-LEE	49	2	9	393
DCT-DIF	41	2	7	412
DCT-DIT	48	1	7	1150
DCT-DIT-2	96	2	7	23135
SWIM1	26	3	4	760
AMMP	24	1	3	728
MEST-S	24	1	6	537
MEST-A	24	1	6	455
MEST-S-2	48	3	6	663

even for highly asymmetric dataflows like DCT-DIT with 36 ALU operations and only 12 multiplications. The rationale is to use the least biased cluster (if there is a choice) to provide more flexibility for binding when new clusters are added.

Then, `balanceCluster()` tries to improve the datapath performance by reassigning some of the functional unit slots in the new cluster to a different functional unit type. As mentioned above, L and N_{MV} are estimated for each generated datapath configuration using our binding algorithm [24] along with a list scheduler. Note that the procedure `balanceCluster()` does not change the number of functional units in the new cluster, which is set to the maximum (i.e., N_F). Thus, in the worst case the algorithm needs to check $N_F - 3$ configurations. In practice, however, we have observed that `balanceCluster()` rarely has to explore more than three configurations, even for a large N_F.

During the last stage, we optimize the new cluster. Specifically, `minimizeCluster()` removes functional units from the cluster one by one (interleaving the functional unit types) and terminates when attempts to further remove any functional units fail to preserve `L_best` and `Nmv_best`.

IV. Design Space Exploration Algorithm Validation and Results

Table II summarizes key characteristics of the representative benchmarks selected for experimental validation of our design space exploration algorithm. These include an elliptic wave filter (EWF), an auto regression filter (ARF), a version of a fast Fourier transform (FFT) algorithm which is the main kernel in the RASTA benchmark from MediaBench [32], various discrete cosine transform (DCT) and motion estimation (MEST) algorithms, as well as the DCT-DIT-2 and MEST-S-2–unrolled versions of the DCT-DIT and MEST-S algorithms.[8] In order to assess the performance of the algorithms in a different application domain we also selected some additional computationally intensive kernels from SPEC2000 (floating point) [33]—SWIM1 and AMMP.

A. Algorithm Validation

As stated in Section III, the objective of the design space exploration algorithm is to identify one datapath configuration in the set D^* associated with each slice S it considers. In doing so, it also determines the best achievable latency L^* for each such slice S. As mentioned above, for investigation and validation purposes, we performed an exhaustive exploration to determine the complete sets D^* for each of the slices to be considered, for most of our benchmarks.[9] A set of results of such an exhaustive exploration for the DCT-DIT was presented in Table I.

Depending on the kernel, the number of slices generated by the exhaustive exploration ranged from 7 to 25 per benchmark. A total of 112 slices over all benchmarks were built by the exhaustive exploration. The design space exploration algorithm was able to identify one element in D^* for 107 out of the 112 slices (i.e., succeeded in about 95% of the cases). In some cases where the design space exploration algorithm failed to identify a datapath in the corresponding D^* set (e.g., $S(10,2,2)$ in Table I), it still found a datapath configuration achieving the optimal latency L^*, but with a total number of functional units larger than T_F^*. In a few other cases (e.g., $S(6,2,1)$ in MEST-S benchmark for a single cycle bus) the algorithm failed to identify a datapath with the optimal latency L^*. As mentioned above, the FFT and DCT-DIT-2 benchmarks had to be validated manually. The design space exploration algorithm generated 46 design space slices for these two benchmarks and only failed to correctly identify two configurations, missing the best latency L^* by one clock cycle in both cases. This brings the overall experimental success rate of the design space exploration algorithm to just above 95%.

The CPU time (in seconds) required by the design space exploration algorithm when executed on an IBM RS6000 595 is presented in Table II. Most of the execution time is spent on the underlying binding algorithm. We must add that the design space exploration algorithm delivers a dramatic reduction in execution time as compared to the exhaustive exploration, e.g., for the DCT-DIT benchmark it executed 60 times faster.

B. Discussion of Results

Tables III–VI summarize the results obtained by the design space exploration algorithm. For these experiments we assume datapaths with two buses ($N_B = 2$), with the bus latency $lat(BUS) = 1$ and $lat(BUS) = 2$ clock cycles (left and right group of columns correspondingly). Each data cell in the table corresponds to a design space slice defined by the remaining two parameters N_F and N_C. The first set of results for each benchmark is the best schedule latency (initiation interval) L^* obtained for each slice identified by the design space exploration algorithm. The second and the third sets are latency and functional unit penalties discussed in the sequel. The L^* values for the DCT-DIT-2 example are presented graphically in Fig. 6.

A simple initial observation of the results shows that, for a given time-critical kernel and overall design space region (specified by the designer), there may be several slices exhibiting the same or close latencies. For example, in Table IV, for the DCT-DIT kernel and datapaths with $lat(BUS) = 1$, slices $S(8,1,0)$[10] and $S(3,3,2)$ have $L^* = 10$. The two

[8]MEST-S-2 was additionally software pipelined after unrolling.

[9]Unfortunately, exhaustive exploration was not feasible for two benchmarks, FFT and DCT-DIT-2, due to the very large space of possible designs.

[10]Note that no intercluster communication is required in a single-cluster datapath ($N_B = 0$).

TABLE III
DESIGN SPACE EXPLORATION RESULTS FOR EWF, ARF, FFT—($N_B = 2$)

	EWF $lat(BUS) = 1$											**EWF** $lat(BUS) = 2$										
N_F	2	3	4	5	6	7	8	9	10	11	12	2	3	4	5	6	7	8	9	10	11	12
N_C	Schedule latencies																					
1	27	16	16	14								27	16	16	14							
2	17	15	15									17	15	15								
3	15		14									16										
	Latency penalty																					
1	–	–	–	–								–	–	–	–							
2	1	1	1									1	1	1								
3	1		0									2										
	Functional unit penalty																					
1	–	–	–	–								–	–	–	–							
2	1	2	2									1	2	2								
3	2		4									2										
	ARF $lat(BUS) = 1$											**ARF** $lat(BUS) = 2$										
N_F	2	3	4	5	6	7	8	9	10	11	12	2	3	4	5	6	7	8	9	10	11	12
N_C	Schedule latencies																					
1	19	13	10	10	8							19	13	10	10	8						
2	11	10										13	12									
3																						
	Latency penalty																					
1	–	–	–	–	–							–	–	–	–	–						
2	1	2										3	4									
3																						
	Functional unit penalty																					
1	–	–	–	–	–							–	–	–	–	–						
2	0	2										1	3									
3																						
	FFT $lat(BUS) = 1$											**FFT** $lat(BUS) = 2$										
N_F	2	3	4	5	6	7	8	9	10	11	12	2	3	4	5	6	7	8	9	10	11	12
N_C	Schedule latencies																					
1	26	18	14	11	10	8	8	7	7	6	6	26	18	14	11	10	8	8	7	7	6	6
2	14	10	8	7	6	6	6	6	5	5	5	14	10	8	7							
3	10	8	7	6	6		5	5				10	8									
4	9	7	7																			
5	8	7	6																			
	Latency penalty																					
1	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0							
3	0	1	1	1	1		0	1				0	1									
4	1	1	2																			
5	1	1	1																			
	Functional unit penalty																					
1	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
2	0	0	0	1	1	1	1	2	6	1	2	0	0	0	1							
3	0	2	3	3	4		5	8				0	2									
4	1	2	8																			
5	3	6	6																			

configurations in these slices are interesting to compare because they show that the DCT-DIT kernel is actually very "cluster-friendly" (despite its dataflow graph having a single connected component)—the latency achieved by executing it on a datapath with three functional units per cluster is identical to that achieved on a (costly) centralized datapath with eight functional units.

This example emphasizes the importance and convenience of the technology-independent design space exploration phase addressed in this work. As mentioned in Section II-A, a key advantage of our approach is in decoupling physical from application-specific figures of merit. Thus, collections of promising datapaths along with L^* for each slice (such as those shown in Fig. 6 or Table I) can be generated and stored in a technology-independent library for various *fundamental algorithm kernels*, DCTs, FFTs, etc. When the kernel is needed for an application, delay and power estimates can be performed (and/or upgraded to a new implementation technology).

Another interesting observation is that the increase of the data transfer operation latency (see Tables III–VI) often does not affect significantly the L^* of slices with a small number of clusters. As one may expect, though, datapaths with a larger number of clusters and a larger N_F are more affected by a slower bus. Thus, bus latency increase effectively reduces the number of relevant slices. This is especially noticeable in the case of the FFT benchmark.

C. On Effectiveness of Clustered Datapaths

We conclude this discussion with an analysis of our experimental data aimed at roughly assessing cost/performance tradeoffs implemented by clustered versus nonclustered datapaths. For this purpose, we introduce two metrics: latency penalty and functional unit penalty. We define the *latency penalty* $\Delta L(d)$ of a given clustered datapath d as the increase in schedule latency, as compared to that achievable on a centralized datapath with the *same total number* of functional units T_F available in d and the best functional unit mix. Similarly, the *functional unit penalty* $\Delta T_F(d)$ corresponds to the increase in the number of functional units in d, with respect to the T_F of the smallest centralized datapath that achieves the *same schedule latency* L. It is important to keep in mind that these "penalties" are very

TABLE IV
DESIGN SPACE EXPLORATION RESULTS FOR DCT ALGORITHMS—($N_B = 2$)

	DCT-LEE $lat(BUS) = 1$											**DCT-LEE** $lat(BUS) = 2$										
N_F	2	3	4	5	6	7	8	9	10	11	12	2	3	4	5	6	7	8	9	10	11	12
N_C	Schedule latencies																					
1	31	21	16	13	11	10	9					31	21	16	13	11	10	9				
2	16	12	10									16	13	10								
3	12											13										
	Latency penalty																					
1	–	–	–	–	–	–	–					–	–	–	–	–	–	–				
2	0	1	1									0	2	1								
3	2											3										
	Functional unit penalty																					
1	–	–	–	–	–	–	–					–	–	–	–	–	–	–				
2	0	0	1									0	1	1								
3	0											1										
	DCT-DIF $lat(BUS) = 1$											**DCT-DIF** $lat(BUS) = 2$										
N_F	2	3	4	5	6	7	8	9	10	11	12	2	3	4	5	6	7	8	9	10	11	12
N_C	Schedule latencies																					
1	29	15	13	10	8	8	8	7				29	15	13	10	8	8	8	7			
2	15	10	9	8			7					15	10	10	8			7				
3	11											12										
	Latency penalty																					
1	–	–	–	–	–	–	–	–				–	–	–	–	–	–	–	–			
2	2	2	1	0			0					2	2	2	0			0				
3	3											4										
	Functional unit penalty																					
1	–	–	–	–	–	–	–	–				–	–	–	–	–	–	–	–			
2	1	1	1	0			2					1	1	2	0			2				
3	1											1										
	DCT-DIT $lat(BUS) = 1$											**DCT-DIT** $lat(BUS) = 2$										
N_F	2	3	4	5	6	7	8	9	10	11	12	2	3	4	5	6	7	8	9	10	11	12
N_C	Schedule latencies																					
1	37	23	19	14	12	11	10	9	8	8	7	37	23	19	14	12	11	10	9	8	8	7
2	19	12	10	9	8	8	8	8	7			19	12	11	9	9	8	8	8	7		
3	13	10	9	8								13		10								
4	11											12										
	Latency penalty																					
1	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
2	0	0	0	1	1	1	1	1	0			0	0	1	1	3	1	1	1	0		
3	1	1	1	1								1		2								
4	1											2										
	Function unit penalty																					
1	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
2	0	0	0	1	2				2			0	0	0	1	1				2		
3	0	1	1	3								0		1								
4	1											2										

TABLE V
DESIGN SPACE EXPLORATION RESULTS FOR SWIM1 AND AMMP KERNELS FROM SPEC2000 FP—($N_B = 2$)

	SWIM1 $lat(BUS) = 1$											**SWIM1** $lat(BUS) = 2$										
N_F	2	3	4	5	6	7	8	9	10	11	12	2	3	4	5	6	7	8	9	10	11	12
N_C	Schedule latencies																					
1	18	11	9	7	6	6	5					18	11	9	7	6	6	5				
2	9	7	6	5		5						9	7	6	5		5					
3	6	5				4						7	6				4					
	Latency penalty																					
1	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
2	0	1	0	0		0						0	1	0	0		0					
3	0	0				0						1	1				0					
	Functional unit penalty																					
1	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
2	0	1	1	1		1						0	1	1	1		1					
3	0	1				0						1	2				0					
	AMMP $lat(BUS) = 1$											**AMMP** $lat(BUS) = 2$										
N_F	2	3	4	5	6	7	8	9	10	11	12	2	3	4	5	6	7	8	9	10	11	12
N_C	Schedule latencies																					
1	15	12	8	8	6	5	5	5	4	4	4	15	12	8	8	6	5	5	5	4	4	4
2	8	6	5					4				8	7	6								
3	6											7										
	Latency penalty																					
1	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
2	0	0	0					0				0	1	1								
3	0											1										
	Functional unit penalty																					
1	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
2	0	0	1					1				0	0	2								
3	0											1										

TABLE VI
DESIGN SPACE EXPLORATION RESULTS FOR MOTION ESTIMATION KERNELS—($N_B = 2$)

	MEST-A $lat(BUS) = 1$											**MEST-A** $lat(BUS) = 2$										
N_F	2	3	4	5	6	7	8	9	10	11	12	2	3	4	5	6	7	8	9	10	11	12
N_C	Schedule latencies																					
1	13	10	8	8	7	7	6					13	10	8	8	7	7	6				
2	9	8	7									10	8									
3	8	7										8										
4	7																					
	Latency penalty																					
1	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
2	1	1	0									2	0									
3	0	1										1										
4	0																					
	Functional unit penalty																					
1	–	–	–	–	–	–	–	–	–	–	—	–	–	–	–	–	–	–	–	–	–	–
2	1	2	1									1	1									
3	1	2										2										
4	0																					
	MEST-S $lat(BUS) = 1$											**MEST-S** $lat(BUS) = 2$										
N_F	2	3	4	5	6	7	8	9	10	11	12	2	3	4	5	6	7	8	9	10	11	12
N_C	Schedule latencies																					
1	16	13	9	9	8	8	7	7	7	7	7	16	13	9	9	8	8	7	7	7	7	7
2	10	9	8		7							11	10	8								
3	9											10	9									
	Latency penalty																					
1	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
2	1	1	0		0							2	1	1								
3	1											2	1									
	Functional unit penalty																					
1	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
2	1	2	1		0							1	1	2								
3	2											2	3									
	MEST-S-2 $lat(BUS) = 1$											**MEST-S-2** $lat(BUS) = 2$										
N_F	2	3	4	5	6	7	8	9	10	11	12	2	3	4	5	6	7	8	9	10	11	12
N_C	Schedule latencies																					
1	32	18	16	11	10	8	8	7	7	7	7	32	18	16	11	10	8	8	7	7	7	7
2	16	10	8			7						16	11	9	8			7				
3	11	9	7									11	9	8								
4	9	8										10										
	Latency penalty																					
1	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
2	0	0	0			0						0	1	1	1							
3	1	1	0									1	2	1								
4	1	1										2										
	Functional unit penalty																					
1	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
2	0	0	1			2						0	1	2	2							
3	1	1	3									1	3	3								
4	2	3										2										

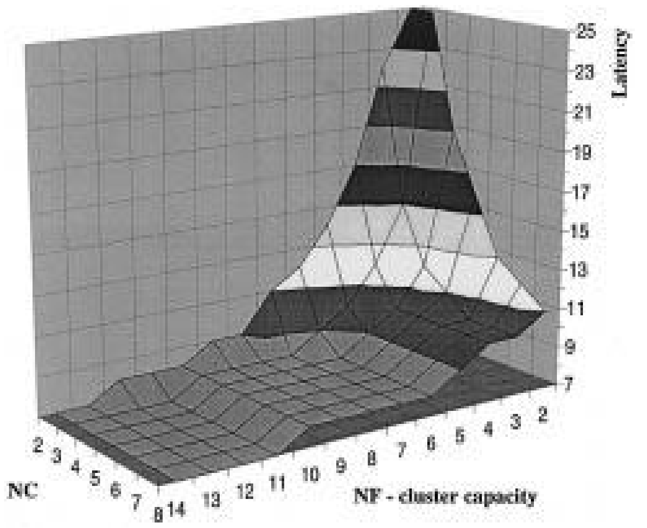

Fig. 6. Design space characterization for DCT-DIT-2 ($N_B = 2$, $lat(BUS) = 1$).

pessimistic, since, for example, typically a clustered datapath would support a faster clock rate than the centralized machine that was used as the baseline, and thus an increased latency does not necessarily translate to an increased execution delay.

The experimental results in Tables III–VI suggest that clustered datapaths often incur a very low (sometimes zero) latency penalty, and thus the cost/performance tradeoffs unlocked by considering such datapaths are likely to be systematically beneficial from the point of view of execution delay, energy/power consumption, and silicon area. This is particularly true when the latency of data transfers is not significantly larger than the latencies of regular operations.

V. PREVIOUS WORK

Hekstra *et al.* [34] present methods and tools to perform design space exploration concerning the precise functional unit configuration for a VLIW core to be used in future TriMedia processors. Prior to the design space exploration step discussed in the paper, it was decided that the CPU64 VLIW core would have a single register file and five issue slots, in which 30 different types of functional units could be placed. The objective was thus to determine how many functional units of each type were needed and where to place these in the issue slots. Note that the set of functional units placed in a single issue slot share two read ports and one write port to the register file. In summary, then, the parameters addressed in this paper had already been defined prior to the design space exploration step discussed by Hekstra *et al.* [34], namely, number of clusters $(= 1)$ and the cluster capacity $(= 5)$. This example confirms that, in practice, the first datapath parameters to decide upon should be those addressed in this paper, and that other important (more detailed) optimizations, including port sharing, should be undertaken after the vast design space is properly reduced to a few slices of interest.

Another important body of research focuses on design space exploration and synthesis using trace-driven simulation techniques. Wu and Wolf [35], for example, describe a method for performing datapath design space exploration for a clustered VLIW datapath aimed at implementing an MPEG-2 encoder. The main contributions of the paper are the use of a trace-driven scheduling algorithm and the development of a method for quickly generating a rough estimate of the impact on cycle count when varying certain datapath resources with respect to a baseline. Still, the authors remark that only a limited number of datapath configurations could be examined in reasonable time using their technique, and plan to devise search techniques that can explore the space more efficiently.

Capitanio *et al.* [36] performed design space exploration of clustered (“limited connectivity”) VLIW datapaths in the context of straight line code in loops. However, in their work they considered only predefined homogeneous clusters, i.e., clusters with identical number of “universal” functional units. In contrast, our design space exploration algorithm actually *generates* heterogeneous cluster configurations.

Fisher *et al.* [37] described a methodology to design clustered VLIW processors customized for a given application or a set of applications. A highly retargetable production-level compiler was used in the design loop. The general philosophy of this methodology is to “build simple hardware that does the basic, simple operations, but uses a lot of ILP to get a speedup” and try to match the “structures and sizes of the architecture to the application rather than specific opcodes”. Extensive experiments were performed with various datapaths configurations and benchmarks. In order to “avoid an exponential explosion of runtime and data,” however, only *homogeneous* clusters with only certain (predefined) combinations of ALU’s and multipliers were considered during the exploration. As mentioned earlier, our algorithm is capable of exploring design space of *heterogeneous* clusters, fine-tuning each cluster to match application’s kernel(s).

A large body of work is available in the high-level synthesis literature addressing design space exploration focusing on area, schedule length, and clock rate dimensions. Many such approaches assume that the delay of functional units dominates the system delay; see e.g., [38]. Unfortunately, such an assumption does not hold for our problem of interest; see the previous discussion on critical design space parameters.

Extensive research is being performed in the area of high performance datapath synthesis for digital signal processing applications, e.g., [4], [22], [39], [40]. Below we discuss a representative example, the Cathedral [4] high level synthesis system, developed at IMEC. Cathedral uses an architectural style based on ASU. ASUs are datapaths comprised of chained functional units “matching” carefully selected parts of the application flow graph. Thus, their adopted *structural hierarchy* defines a design space, and an exploration methodology, quite different from those addressed in this paper. (Similar contrasts can be made to other high-level synthesis approaches.)

Yet another contribution relevant to the work discussed in this paper is the program-in-chip-out (PICO) system synthesis and design space exploration tool [41]–[45] developed at HP Labs. In what follows, we discuss PICO-VLIW [43], [44], the component of the PICO tool that designs application-specific VLIW processors. In PICO-VLIW, the design of an application-specific VLIW processor is supported by three closely interrelated subsystems: 1) Spacewalker, the design space explorer; 2) VLIW Architecture Synthesis Subsystem, which takes the abstract architecture specification generated by Spacewalker and outputs an RTL-level VHDL description of the processor, as well as an area estimate for the processor; and (3) Elcor, a retargetable compiler for VLIW processors, which generates the application code (for the custom VLIW processor) and evaluates overall performance by counting the number of cycles taken to execute the program [41], [44]. The authors remark that, during the design space exploration, the decision on the next abstract architecture to consider is done by the Spacewalker, using the previously generated area and cycle estimates and relying on sophisticated search strategies and heuristics [41], [44], [45]. The ultimate goal is to identify all Pareto-optimal design points with respect to latency (number of execution steps) and area. The abstract architecture generated by the Spacewalker (at each search iteration) includes: 1) a specification of the target machine’s register files (including width in bits and number of registers, but not number of ports); 2) a specification of the machine’s operations, a subset of the HPL-PD repertoire [46]; and 3) a characterization of the machine’s “level of ILP,” specified in terms of a number of concurrent and exclusion operation groups/constraints—note that such constraints are to be later used by the Architecture Synthesis Subsystem, to explore opportunities for register port sharing, as well as sharing of instruction bits [44].

PICO-VLIW is a remarkable tool in that it successfully automated the design of custom VLIW processors. In contrast to our work, however, in publications on PICO-VLIW we have not found concrete information on strategies and/or heuristics directly or indirectly *exploring* clustering of datapaths. For example, the abstract architecture specification used to illustrate the discussion on PICO-VLIW [44] contains a single general-purpose register file, and no information is provided on how such a decision was made. Moreover, as alluded to above, in PICO-VLIW the number of ports on each of the machine’s register files is to be determined by the VLIW Architecture Synthesis Subsystem, using the operation constraints specified by the Spacewalker; thus, the Spacewalker can only indirectly exercise control over this important parameter. In contrast, our early design space exploration focuses *directly* on those aspects of the VLIW processor datapath configuration (i.e., of the machine’s hardware) that are likely to have a first-order impact on the most critical physical figures of merit, that is, on the machine's sustainable clock rate f and power dissipation P. In doing so, it explicitly considers/explores fundamental strategies for hierarchical organization of datapath resources that are known to be effective for high-performance VLIW processors, such as clustering. (In addition, see also discussion in Section II on postponing decisions on refinement of functional units/port sharing and register file sizes).

Dutta and and Wolf [47] describe a detailed design methodology for video signal processing datapath elements. The article shows how the insight on circuit implementation is used in architectural design of a programmable video signal processor. Critical tradeoffs between the number of register file ports, number of registers, the depth of datapath pipeline, and the clock frequency are considered by the methodology. The authors developed parameterized delay estimation models for register files of different size and number of ports as well as for adders and multipliers with different pipeline depths. The models were verified by designing actual CMOS layouts. The methodology was evaluated for DCT and motion estimation algorithms. For each algorithm, a number of design points with different degree of pipelining and different number of register ports were generated. The structure of processing elements in the explored architecture is fixed (two adders, one multiplier, and one accumulator). The possibility of heterogeneous processing elements considerably increases the complexity of the design space and requires an algorithm that can automatically assign operations to functional units in different processing elements.

Finally, as mentioned in Section II-C, Jacome *et al.* [15] proposed a clustered ASIP VLIW datapath design space exploration methodology. The algorithm [15] is to be used prior to the design space exploration phase described in this paper, as illustrated in Fig. 3(c), to quickly identify regions of interest (that is, ranges of parameters N_F, N_C, and N_B) in the design space.

VI. Conclusion and Future Work

We have presented a kernel-specific and technology-independent methodology for performing early design space exploration of clustered VLIW ASIP datapaths. A design space parameterization and key abstractions were introduced to support the proposed design space exploration. The design space exploration phase discussed in this paper is one of the components of the NOVA framework [48], [49], currently under development, illustrated in part in Fig. 3.

Acknowledgment

The authors would like to thank the reviewers for their suggestions and constructive criticism.

References

[1] S. Rixner, W. J. Dally, B. Khailany, P. Mattson, U. J. Kapasi, and J. D. Owens, "Register organization for media processing," in *Proc. 26th Int. Symp. High-Performance Computer Architecture*, May 1999.

[2] V. Agarwal, H. S. Murukkathampoondi, S. W. Keckler, and D. C. Burger, "Clock rate versus IPC: The end of the road for conventional microarchitectures," in *Proc. 27th Int. Symp. Computer Architecture*, June 2000, pp. 248–259.

[3] K. I. Farkas, P. Chow, N. P. Jouppi, and Z. Vranesic, "The multicluster architecture: Reducing cycle time through partitioning," in *Proc. 30th Ann. Int. Symp. Microarchitecture*, Dec. 1997.

[4] W. Geurts, F. Catthoor, S. Vernalde, and H. DeMan, *Accelerator Data-Path Synthesis for High-Throughput Signal Processing Applications*. Boston, MA: Kluwer, 1997.

[5] H. Wang, N. Dutt, A. Nicolau, and K.-Y. S. Siu, "High-level synthesis of scalable architectures for IIR filteres using multichip modules," in *Proc. 30th ACM IEEE Design Automation Conf.*, 1993.

[6] J. Van Praet, G. Goossens, D. Lanneer, and H. De Man, "Instruction set definition and instruction selection for ASIP's," in *Proc. 7th Int. Symp. High-Level Synthesis*, May 1994, pp. 10–16.

[7] C. Liem, T. May, and P. Paulin, "Register assignment through resource classification for ASIP microcode generation," in *Proc. Int. Conf. Computer-Aided Design*, Nov. 1994, pp. 397–402.

[8] R. P. Colwell, W. E. Hall, C. S. Joshi, D. B. Papworth, P. K. Rodman, and J. E. Tornes, "Architecture and implementation of a VLIW supercomputer," in *Proc. Supercomputing '90*, Branford, CT, Nov. 1990, pp. 910–919.

[9] P. Faraboschi, G. Desoli, J. A. Fisher, and G. Desoli, "Lx: A technology platform for customizable VLIW embedded processing," in *Proc. 27th Ann. Int. Symp. Computer Architecture*, Vancouver, BC, Canada, June 2000.

[10] P. Faraboschi, G. Desoli, and J. A. Fisher, "Clustered Instruction-Level Parallel Processors," Hewlett-Packard Co., HPL-98-204, 1998.

[11] J. Fritts, Z. Wu, and W. Wolf, "Parallel media processors for the billion-transistor era," in *Proc. Int. Conf. Parallel Processing*, Aizu, Japan, Sept. 1999.

[12] C. Basoglu, K. Zhao, K. Kojima, and A. Kawaguchi. (2000) The MAP-CA VLIW-Based Media Processor. Equator Technol. Inc. and Hitachi Ltd.. [Online]. Available: http://equator.com.

[13] *TMS320C6000 CPU and Instruction Set Reference Guide*, Texas Instruments Incorporated, 2000.

[14] ADSP-TS001M TigerSHARC DSP Product Description. Analog Devices. [Online]. Available: http://www.analog.com/products/descriptions/ADSP-TS001.html.

[15] M. F. Jacome, G. de Veciana, and V. Lapinskii, "Exploring performance tradeoffs for clustered VLIW datapaths," in *Proc. 2000 IEEE/ACM Int. Conf. Computer-Aided Design (ICCAD-2000)*, Nov. 5–9, 2000.

[16] D. Burger and J. R. Goodman, "Guest editors' introduction: Billion-transistor architectures," *IEEE Trans. Comput.*, vol. 30, pp. 46–48, Sept. 1997.

[17] P. Marwedel and G. Goossens, Eds., "Code generation for embedded processors," in *Kluwer International Series in Engineering and Computer Science*. Boston, MA: Kluwer, 1995.

[18] C. Liem, *Retargetable Compilers for Embedded Core Processors: Methods and Experiences in Industrial Applications*. Boston, MA: Kluwer, 1997.

[19] J. Llosa, E. Ayguad, and M. Valero, "Quantitative evaluation of register pressure on software pipelined loops," *Int. J. Parallel Programming*, vol. 26, no. 2, 1998.

[20] R. Govindarajan, E. R. Altman, and G. R. Gao, "Minimizing register requirements under resource-constrained rate-optimal software pipelining," in *Proc. 27th Ann. Int. Symp. Microarchitecture*, Nov. 1994, pp. 85–94.

[21] A. E. Eichenberger and E. S. Davidson, "Stage scheduling: A technique to reduce the register requirements of a modulo schedule," in *Proc. 28th Ann. Int. Symp. Microarchitecture*, Nov. 1995, pp. 180–191.

[22] C. M. Chu and J. M. Rabaey, "Hardware selection and clustering in the HYPER synthesis system," in *Proc. IEEE Eur. Conf. Design Automation*, Mar. 1992.

[23] J. L. Van Meerbergen, P. E. R. Lippens, W. F. J. Verhaegh, and A. Van Der Werf, "PHIDEO: High-level synthesis for high throughput applications," *J. VLSI Signal Processing*, vol. 9, no. 1/2, pp. 89–104, Jan. 1995.

[24] V. Lapinskii, M. F. Jacome, and G. de Veciana, "High-quality operation binding for clustered VLIW datapaths," in *Proc. 38th IEEE/ACM Design Automation Conf. (DAC-2001)*, June 18–22, 2001.

[25] V. Lapinskii, "Algorithms for compiler-assisted design space exploration of clustered VLIW ASIP Datapaths," Ph.D., Univ Texas at Austin, 2001.

[26] J. M. Rabaey and M. Pedram, Eds., *Low Power Design Methodologies*. Boston, MA: Kluwer, 1996.

[27] F. Catthoor, S. Wuyack, E. Degreef, F. Balasa, L. Nachtergaele, and A. Vandecappelle, *Custom Memory Management Methodology: Exploration of Memory Organization for Embedded Multimedia System Design*. Boston, MA: Kluwer, 1998.

[28] M. J. Wolfe, *High Performance Compilers for Parallel Computing*. New York: Addison-Wesley, 2000.
[29] P. R. Panda, A. Nicolau, and N. Dutt, *Memory Issues in Embedded Systems-on-Chip: Optimizations and Exploration*. Boston, MA: Kluwer, 1999.
[30] P. R. Panda, F. Catthoor, N. D. Dutt, K. Danckaert, E. Brockmeyer, C. Kulkarni, A. Vandercappelle, and P. G. Kjeldsberg, "Data and memory optimization techniques for embedded systems," *ACM Trans. Design Automation Electron. Syst.*, vol. 6, no. 2, pp. 149–206, Apr. 2001.
[31] C. Akturan and M. F. Jacome, "An effective software pipelining algorithm for clustered embedded VLIW processors," *Int. J. Design Automation Embedded Syst.*, 2002.
[32] C. Lee, M. Potkonjak, and W. H. Mangione-Smith, "Mediabench: A tool for evaluating and synthesizing multimedia and communications systems," in *Proc. 30th Ann. Int. Symp. Microarchitecture*, 1997, pp. 330–5.
[33] K. Dixit, "Performance SPECulations—Benchmarks, friend or foe," in *Seventh Int. Symp. High Performance Computer Architecture*, Monterrey, Mexico, Jan. 2001.
[34] G. J. Hekstra, G. D. La Hei, P. Bingley, and F. W. Sijstermans, "TriMedia CPU64 design space exploration," in *Proc. IEEE Int. Conf. Computer Design: VLSI in Computers and Processors*, Austin, TX, USA, Oct. 1999, pp. 599–606.
[35] Z. Wu and W. Wolf, "Data-path synthesis of VLIW video signal processors," in *Proc. 11th Int. Symp. System Synthesis*, Taiwan, R.O.C., Dec. 2–4, 1998, pp. 96–101.
[36] A. Capitanio, N. Dutt, and A. Nicolau, "Partitioned register files for VLIWs: A preliminary analysis of tradeoffs," in *Proc. 25th Ann. Int. Symp. Microarchitecture*, Portland, OR, Dec. 1992, pp. 292–300.
[37] J. A. Fisher, P. Faraboschi, and G. Desoli, "Custom-fit processors: Letting applications define architectures," in *Proc. 29th Ann. IEEE/ACM Int. Symp. Microarchitecture*, Paris, France, Dec. 1996.
[38] S. Chaudhuri, S. A. Blythe, and R. A. Walker, "An exact methodology for scheduling in a 3D design space," in *Proc. Eighth Int. Symp. System Synthesis*, Cannes, France, Sept. 13–15, 1995, pp. 78–83.
[39] D. S. Rao and F. J. Kurdahi, "Partitioning by regularity extraction," in *Proc. ACM/IEEE Design Automation Conf.*, June 1992.
[40] E. A. Rundensteiner, D. Gajski, and L. Bic, "Component synthesis from functional descriptions," *IEEE Trans. Computer-Aided Design*, vol. 12, pp. 1287–1299, Sept. 1993.
[41] B. Ramakrishna Rau and M. S. Schlansker, "Embedded computing: New directions in architecture and automation," Hewlett-Packard Co., HPL-2000-115, 2000.
[42] R. Schreiber, S. Aditya, B. Ramakrishna Rau, V. Kathail, S. Mahlke, S. Abraham, and G. Snider, "High-level synthesis of nonprogrammable hardware accelerators," in *Proc. IEEE Int. Conf. Application-Specific Systems, Architectures, and Processors*, Boston, MA, July 2000, pp. 113–124.
[43] S. Aditya and B. Ramakrishna Rau, "Automatic architecture synthesis and compiler retargeting for VLIW and EPIC processors," Hewlett-Packard Co., HPL-1999-93, 2000.
[44] S. Aditya, B. Ramakrishna Rau, and V. Kathail, "Automatic architectural synthesis of VLIW and EPIC processors," in *Proc. 12th Int. Symp. System Synthesis*, Nov. 1999, pp. 107–113.
[45] S. Abraham, B. Ramakrishna Rau, and R. Schreiber, "Fast design space exploration through validity and quality filtering of subsystem designs," Hewlett-Packard Co., HPL-2000-98, 2000.
[46] V. Kathail, M. S. Schlansker, and B. Ramakrishna Rau, "HPL-PD Architecture Specification: Version 1.1," Hewlett-Packard Co., HPL-93-80(R.1), 2000.
[47] S. Dutta and W. Wolf, "A circuit-driven design methodology for video signal-processing datapath elements," *IEEE Trans. Very Large Scale Integration Syst.*, vol. 7, pp. 229–240, June 1999.
[48] NOVA Project: ASIP's and retargetable compilers; CAD for embedded systems. Dept. ECE, Univ. Texas at Austin. [Online]. Available: http://horizon.ece.utexas.edu/nova/
[49] M. F. Jacome and G. De Veciana, "Design challenges for new application specific processors," *IEEE Design Test Computers*, vol. 17, pp. 40–50, Apr.–June 2000.

Viktor S. Lapinskii (S'96–M'01) received the Ph.D. and MSEE degrees from the University of Texas at Austin and the BSEE degree from the Moscow Institute of Radioengineering, Electronics, and Automation.

He is currently a postdoctoral Fellow in the Department of Electrical and Computer Engineering, University of Texas at Austin, and will soon be joining AmmoCore Technology. His research interests include high-level synthesis, compilation techniques for limited connectivity architectures, and VLSI information modeling at multiple levels of abstraction.

Dr. Lapinskii is a co-recipient of the IEEE/CAS William J. McCalla Best ICCAD Paper Award for 2000.

Margarida F. Jacome (S'92–M'94–SM'01) received the B.S. and the M.S. degrees from the Technical University of Lisbon, IST, in 1981 and 1988, respectively, and the Ph.D. degree in electrical and computer engineering from Carnegie Mellon University, Pittsburgh, PA, in 1993.

She is an Associate Professor of Electrical and Computer Engineering at The University of Texas at Austin. Her research interests include embedded computing, hardware-software codesign, and high-level synthesis.

Dr. Jacome is a member of the Executive Committee of the Design Automation Technical Committee (DATC) of the IEEE Computer Society. She has served on the technical program committee of a number of conferences, including the International Conference on Computer Aided Design (ICCAD), the Design Automation and Test in Europe (DATE), the International Symposium on Hardware/Software Codesign (CODES), the International Conference on Computer Design (ICCD), and the International Symposium on System Synthesis (ISSS). She has served as the General Co-Chair of the Workshop on Electronic Design Processes (EDP) in 1998 and 1999. She is a recipient of a 1996 National Science Foundation CAREER Award. She is a corecipient of the IEEE/ACM Design Automation Conference (DAC) Best Paper Award for 1992 and a corecipient of the IEEE/CAS William J. McCalla Best ICCAD Paper Award for 2000.

Gustavo A. de Veciana (S'88–M'94–SM'01) received the B.S., M.S, and Ph.D. degrees in electrical engineering from the University of California at Berkeley, in 1987, 1990, and 1993, respectively.

In 1993, he joined the Department of Electrical and Computer Engineering at the University of Texas at Austin where he is currently an Associate Professor. His research interests include the design and control of telecommunication networks and development of CAD tools and embedded processors for multimedia and signal processing applications. Specific interests include performance evaluation, resource management, routing in large-scale networks, efficient simulation, embedded clustered VLIW ASIPs and system software.

Dr. de Veciana is an Editor for the IEEE/ACM TRANSACTIONS ON NETWORKING. He is the recipient of a General Motors Foundation Centennial Teaching Fellowship in Electrical Engineering, 1996 National Science Foundation CAREER Award, and corecipient of the IEEE/CAS William J. McCalla Best ICCAD Paper Award for 2000.