Design Challenges for New Application-Specific Processors

Margarida F. Jacome

Gustavo de Veciana
University of Texas at Austin

This article discusses challenges in developing retargetable compilers and synthesis tools for application-specific processor cores targeted at embedded portable digital communications and multimedia systems.

EMBEDDED SYSTEMS form a market that is already larger and growing more rapidly than that of general-purpose computers. In fact, real-time multimedia and signal processing embedded applications currently account for over 90% of all computer cycles.[8] Our focus in this article will be on an increasingly important set of embedded applications, consisting of portable systems in the areas of digital communications and multimedia consumer electronics (e.g., cellular phones, personal digital assistants, digital video cameras, and multimedia terminals). These complex systems rely on "power-hungry" algorithms for high-bandwidth wireless communications, video compression and decompression, handwriting recognition, speech and image processing, etc. The portability of these systems makes energy consumption a particularly critical design concern as it reduces battery life. Moreover, high power dissipation leads to more expensive packaging and decreases reliability. At the same time, levels of microelectronic integration continue to rise, enabling more integrated functionality on a single chip. Such integration has significant advantages from the point of view of performance, energy consumption, and reliability, but poses a basic challenge: how to effectively design "first-time-right" complex systems-on-a-chip that meet multiple stringent design constraints.

In order to successfully design complex systems within the short time-to-market windows characteristic of the embedded systems industry, it is important to maximize the flexibility or programmability of the target *system architecture.* In other words, it is desirable to move as much functionality as possible to embedded software. This minimizes or eliminates the need for application-specific hardware accelerators, which due to their lack of flexibility may compromise time-to-market. In particular, since the specification of such products frequently evolves over time, a programmable system architecture diminishes the impact of such changes on the design. Moreover, feature differentiation (within families of products) can be done in software, which greatly decreases development time and cost.

Unfortunately, the use of off-the-shelf embedded processor cores is often not viable for our applications, because general-purpose embedded processors (even top-of-the-line reduced-instruction-set computers and/or DSP cores) may not be able to deliver the performance required by the application and because they may be prohibitively expensive or inefficient,

0740-7475/00/$10.00 © 2000 IEEE

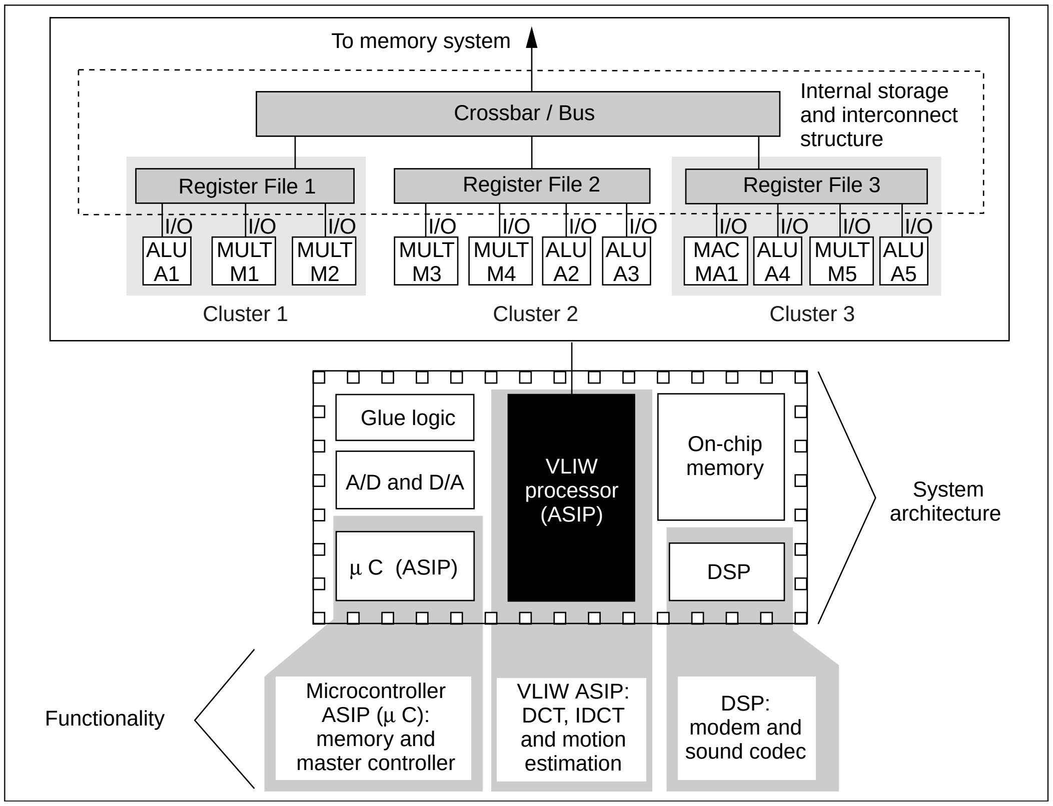


Figure 1. Block diagram of a multimedia system architecture containing a VLIW ASIP with a clustered datapath.

particularly with respect to energy consumption. Thus, the embedded systems industry has shown an increasing interest in Application-Specific Instruction-Set Processors (ASIPs), i.e., processors tailored or specialized to the needs of a specific product or family of products. By spending silicon where it truly matters, these processors are smaller and simpler than their general-purpose counterparts, are able to run at higher clock frequencies, and are more energy efficient.

The target applications in this article—digital communications and multimedia consumer electronics—often spend most of their cycles executing a few *time-critical code segments* with well-defined characteristics, making them amenable to processor specialization. Moreover, these computation-intensive components often exhibit a high degree of inherent *parallelism*, i.e., computations that can be executed concurrently. Very Large Instruction Word (VLIW) ASIPs are particularly effective in exploiting such fine-grained instruction-level parallelism. (See, e.g., http://www-us2.semiconductors.philips.com/trimedia/.) These processors (see Figure 1) comprise a large number of functional units (such as multipliers and arithmetic logic units) that can process multiple operations and data transfers or accesses simultaneously, thus enabling them to achieve performance commensurate with that of dedicated hardware accelerators.[4]

Consider the block diagram of a hypothetical programmable system architecture for a multimedia application shown in Figure 1. The VLIW ASIP performs number-crunching functions required by our hypothetical application, including discrete cosine and inverse discrete cosine transforms and motion estimation algorithms. An off-the-shelf DSP is used for the less computationally demanding modem and sound codec functions the application requires. A third programmable component, an application-specific microcontroller, is used to provide timely interleaving of memory and master control functions. Examples of embedded systems such as this one—wherein three or even more processor cores are instantiated (two ASIPs and one off-the-shelf DSP)—are not unusual in practice. However, current industry efforts to develop such system architectures, and the corresponding specialized processors, require excessive manpower and resources, resulting in unnecessarily high costs that only a few can afford.

In this article, we focus on VLIW ASIPs specialized to support the time-critical, processing-intensive components of our target embedded applications. Although these processors are potentially attractive to implement these system components, their promise can be fully realized only if methodologies and tools for the synthesis of specialized processors and associated high-quality retargetable compilers are developed. We argue here for the need to recognize the *complementarity* between processor design or specialization and retargetable compilation. (In simple terms, a compiler is said to be retargetable if it can generate "efficient" assembly code for various target processors. We discuss this in more detail below.) Indeed, observe that when considering processor specialization for a group of functions or algorithms, the benefits of a datapath structure and memory organization can be achieved only with an effective compiler. In turn, the effectiveness of a compiler depends on its ability to properly exploit the specialized features that make the processor suitable to the application at hand.

We will use this complementarity as a springboard to discuss the challenges associated with processor specialization. For concreteness, we discuss possible solutions based on our ongoing work (For information on the NOVA project, see http://horizon.ece.utexas.edu/~jacome/nova.) on a novel methodology that *jointly* addresses: the synthesis of specialized VLIW ASIPs and associated memory systems and the development of high-quality retargetable compilers for such specialized processors. As will be seen, the promise of this joint approach lies in enabling a systematic and aggressive *reuse-based, compiler-assisted* optimization of a processor's complex cost/efficiency trade-offs. As such, this is not a tutorial article, but rather presents our view of the various challenges lying ahead, based on past and ongoing work in this growing research area.

VLIW ASIPs: design space exploration supported by retargetable code generation

The datapath of a VLIW machine is an interconnection of multiple, horizontally microcoded, possibly pipelined functional units (e.g., multipliers, arithmetic logic units, and multiply-accumulate units) that are centrally controlled. Specifically, each functional unit has dedicated, fixed control fields in a "very long" machine instruction that can be independently set. VLIW machines are microcoded in that these long instructions execute in one cycle (i.e., every instruction word specifies all datapath and memory actions to be executed during that cycle). Thus, setting up and maintaining the instruction pipeline is the responsibility of the programmer or code generator; accordingly, the resulting pipeline schedule is fully visible in the machine code. A centralized controller issues a sequence of very long instructions during program execution.

VLIW machines can be seen as a hybrid between standard Single-Instruction, Multiple-Data (SIMD) and Multiple-Instruction, Multiple-Data (MIMD) parallel architectures. Indeed, a SIMD processor can be viewed as a collection of processing elements marching in lockstep under the orders of a centralized controller with each performing *identical* operations on different data elements. By contrast, in a VLIW architecture, *heterogenous* functional units can each be performing *different* operations on various data elements. Thus, the flexibility of a VLIW architecture is one step up from a SIMD/vector processor. A MIMD machine, on the other hand, has multiple, typically *identical* processing elements, each with its own thread of control. Thus, since a VLIW machine has only one thread of control, it can be viewed as a carefully "preplanned" MIMD machine with heterogenous processing elements.

Below, we will discuss the specialization "dimensions" that are worth pursuing for our target applications and present a compiler-assisted methodology for effectively exploring the VLIW ASIP design space.

Key specialization dimensions for VLIW ASIPs

In tuning the microarchitecture of a VLIW machine to an application's time-critical functions or components, three fundamental specialization dimensions must be considered:

- the number and type of functional units that should be instantiated in the machine's datapath
- the organization of internal storage (register files) and corresponding interconnect structure among such register files and to or from the memory system
- the organization of the memory system

Traditionally, datapaths have been based on a single register file shared by all functional units. This central register file provides internal storage as well as switching (i.e., interconnection among the functional units and to or from the memory system). Unfortunately, this simple organization does not scale well with the large number of functional units typical of a VLIW machine. Indeed, it has been shown that for N arithmetic units connected to a centralized register file, the area of the register file grows as N^3, the delays as $N^{3/2}$, and power requirements as N^3. See Rixner et al.[8] In short, as the number of functional units increases, internal storage and communication between functional units quickly become the dominant, if not the prohibitive factor in terms of area, delay, and power requirements. Thus, high-performance VLSI computing systems have become limited not by arithmetic capacity, but rather by communication bandwidth.[8] Indeed, as deep-submicron microelectronic technologies evolve, enabling very large numbers of functional units to be placed on small, inexpensive chips, the challenge is to devise microarchitectures capable of cost-effectively keeping these functional units busy.

A fundamental observation is that the area, delay, and power associated with the storage organization can be dramatically reduced by restricting the connectivity between functional units and registers, so that each functional unit can only read and write from or to a limited subset of registers. Thus, a key dimension of processor specialization to be explored is *clustering*—i.e., the development of datapaths comprising clusters of functional units connected to local storage (register files). A sketch of a VLIW processor with three such clusters is shown in Figure 1.

Although by moving from a centralized to a *distributed* register file organization one can reap *significant* delay, area, and power savings, this type of specialization may come at a cost. In particular, one may have to transfer data among these register files (i.e., clusters), possibly resulting in increased execution latency. Our premise is that by carefully considering the *specifics* of the target embedded application and by using powerful optimizing compilers, one can avoid these penalties and enjoy the benefits.

The memory system organization is also a major specialization dimension that should be carefully considered when designing a VLIW ASIP for the set of applications of interest. This is so because the large amounts of data parallelism typically exhibited by such applications provide major opportunities for performance enhancement. However, in order to effectively explore such parallelism, one needs to be able to stream data to the clusters or functional units at a sufficiently high rate (i.e., high memory bandwidth is required). The good news is that the algorithms of interest typically perform well-defined access patterns on simple, regular data structures (i.e., vectors and matrices) whose dimensions are known at compile time. This strongly suggests that a *distributed memory organization* (comprised of a number of small, fast memory banks) properly designed to exploit locality or regularity in these data access patterns can provide the required memory bandwidth at a reduced cost (i.e., power or energy, area, and delay). As in the previous case, though, the timely design of such specialized memory organizations (encompassing data partitioning, memory allocation and assignment, scheduling of data operations, loop transformations, and other complex tasks) requires the availability of effective memory synthesis systems working in conjunction with powerful optimizing compilers.

Finally, the interconnect structure (among

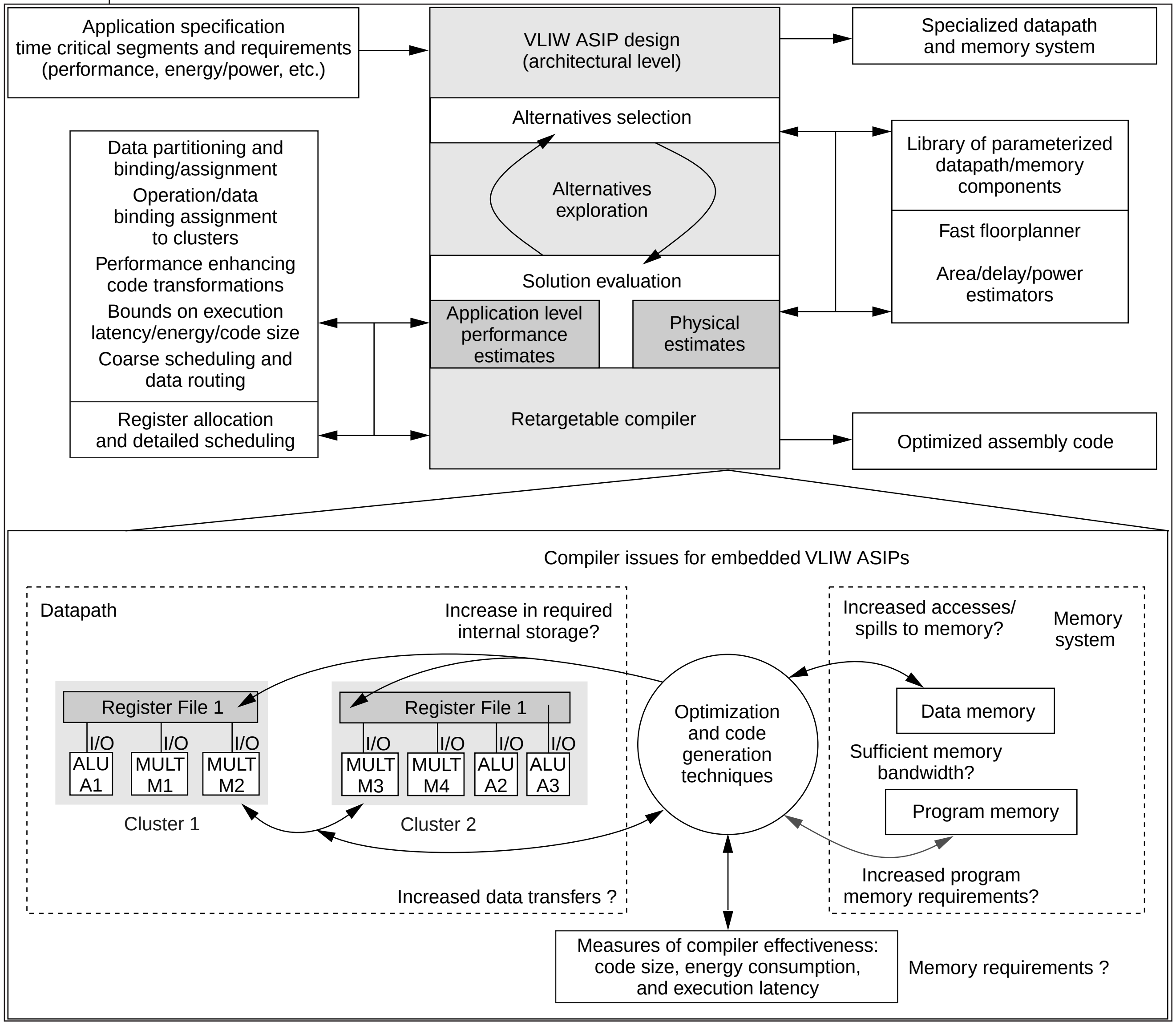


Figure 2. Research overview: VLIW ASIP design and retargetable compilers.

clusters and from clusters to memory) is a third major specialization dimension that should be explored. By choosing an appropriate interconnect structure (i.e., a bus or crossbar configuration matching the parallelism and access patterns in the algorithms of interest) and then appropriately binding the algorithms' operations to clusters, one can explore relevant cost/performance trade-offs.

In summary, the key specialization dimensions for the clustered VLIW machines we introduced above are as follows. First, one needs to define an effective organization for the memory system. Then, one needs to determine the number of clusters to be instantiated in the datapath and the interconnect structure among the clusters and to or from the memory system. Moreover, for each cluster, one needs to determine the number and type of functional units to be locally instantiated, as well as the capacity of the cluster's local register file. Devising an "optimal" datapath and memory system organization for a given application (i.e., one that meets the target performance while achieving low silicon cost and high energy efficiency) is an exceedingly complex, multiobjective optimization problem. In practice, this requires intensive design space exploration with trade-offs involving physical figures of merit (combinational delay, area, power dissipation) as well as application-level performance metrics (throughput or latency, code size, energy consumption). In the next subsection, we discuss these challenges and propose a reuse-based, compiler-assisted design space exploration methodology for specializing a VLIW ASIP's datapath and memory system to a given class of applications.

Methodology

As shown in Figure 2, given a characterization of the application of interest, our goal is to support both an iterative search (design space exploration) for "optimal" specialized memory organizations, datapaths, and interconnect structures (i.e., solutions that deliver the required performance at low silicon cost and high energy efficiency) and the generation of highly optimized assembly code. The key elements of the problem are as follows. To effectively explore the huge space of specialized datapaths and memory systems, one must have an infrastructure enabling a structured *selection* and *evaluation* process. To evaluate candidate solutions, one must have *reliable estimates* of the cost/efficiency *physical metrics* of a datapath, memory system, and interconnect configuration, as well as associated *application-level performance metrics*. Since these estimates will drive the search, they should be performed early (i.e., prior to the detailed datapath and memory system design and code-generation process).

To enable the selection of alternatives, we propose to structure the solution space by using a hierarchical parameterization of candidate solutions. The idea is to build a library of fundamental components parameterized with respect to relevant features (e.g., clusters parameterized by the number of functional units and register files) and memory banks parameterized by technology, speed, and access modes; size; and ports. Such components can then be composed on-the-fly to define the datapath, memory, and interconnect alternatives. Note that such an approach narrows the solution space along the critical specialization dimensions discussed above (i.e., supports the search for effective *distributed organizations* of computational resources, such as clusters, and data storage resources). Moreover, as shown on the right side of Figure 2, the proposed parameterization of components can facilitate early reliable estimation of physical figures of merit, such as delay, area, and power. Specifically, a "fast floorplanner" together with a database of statistically characterized clusters and memory banks can be used to derive estimates with improved reliability. This approach is a natural application of previous research on design reuse and early estimation (see, e.g., Jacome and Peixoto[6]).

To evaluate the suitability of a specialized memory organization and datapath to a target application, one must estimate the execution latency, code size, and energy consumption that can be achieved for the application's target code segments. As shown in Figure 2, the tasks required to derive such application estimates and the retargetable compilation problem are closely linked. The first such task is data partitioning and allocation or binding to memory banks so as to maximize raw memory bandwidth given the expected memory access patterns. Then, one needs to determine a binding or assignment of an application's operations to the datapath's clusters that is likely to reduce execution latency (i.e., what to execute where). As mentioned earlier, for datapaths with distributed register files, careful attention should be paid to penalties incurred by bindings that introduce data transfers among clusters, as well as significant imbalances between computation and memory operations. The third task is to determine optimal behavior preserving code transformations so as to enhance performance (e.g., increase parallelism or concurrency by pipelining loop iterations). Note that such optimizations should take into account both computation operations and the required memory access operations so as to properly balance the load on the datapath and on the memory system over time. As we discuss below, such transformations may significantly increase program size, a major drawback for memory-constrained embedded systems, as well as internal data storage requirements (i.e., register pressure) and energy consumption and thus must be carefully assessed for the class of embedded components of interest.

Given a set of bindings or assignments and parallelism-enhancing transformations, one needs to quickly generate bounds (estimates) for an application's execution latency and other application-specific metrics of interest. Such bounds may be analytical (e.g., Jacome and de Veciana[4]) and/or simulation-based, in which case the proper level of abstraction for such "fast" simulation must be devised (see http://www.trimaran.org/docs.html). Note that while the above tasks are required to select and evaluate candidate memory organizations and datapaths, they also represent the initial phases of the retargetable compilation process—thus our argument that the two problems are closely linked. The final research challenge is to enable the use of information generated at each iteration of this process to guide or assist the design space exploration process itself (i.e., help determine which alternative clustered datapath and memory system configurations are most promising and thus should be considered next).

Retargetable compilation for VLIW ASIP cores

Emerging retargetable compiler technology for clustered VLIW ASIPs has two challenging roles: as an essential component to assist the synthesis of specialized processors and as a means to produce high-quality embedded software for clustered VLIW machines. In this section, we argue that meeting these new challenges may require fundamental changes in the traditional compilation process and we discuss some research directions.

Traditional compilers typically include three main modules: a language-dependent front-end, an intermediate optimization stage, and a machine-dependent back-end.[1] The front-end module takes source code written in a high-level programming language (such as C or C++) and generates an internal or intermediate representation of the behavioral description. The intermediate optimization stage performs "machine-independent" powerful performance-enhancing transformations on this internal representation. Finally, the back-end generates machine code for the target processor architecture. Thus, in a traditional compiler, the specifics of the target machine (instruction set and structural details) are taken into account only during the last (code-generation) stage of the compilation process.

The broad aim of traditional compilers is to *quickly* produce fast code. Thus, the time complexity of compilation algorithms is a major concern. By contrast, when considering compilers for embedded processors, the quality of the produced machine code is much more important than the speed of the compilation process. Thus, the use of more powerful optimization algorithms, even if time-consuming, becomes necessary and justifiable. Moreover, for the embedded applications of interest, in addition to throughput, the code size and energy consumption are important for the generated code. Thus, the compilation process for embedded applications has a broader set of goals and constraints.

Most current compilers target a specific processor architecture, which is, for the most part, "hard-coded" in the code-generation module. However, in order to enable design space exploration for specialized VLIW machines, this target specificity is inadequate. Indeed, as we argued above, processor specialization can be explored only via an "automatically" retargetable compiler. A compiler is said to be automatically retargetable if the same compiler (i.e., same executable) can be used for a range (possibly limited) of target architectures. This is achieved by providing the compiler with a description of the target processor architecture using a special-purpose language.[7] (See, e.g., http://www.trimaran.org/docs.html.)

In summary, compiler technology for embedded VLIW ASIPs requires both retargetability as well as the ability to deal with a larger set of application-level performance issues. Moreover, when compilers are used to drive a datapath and a memory system synthesis or specialization process, they should encompass or provide a broader set of tools to assist the various phases of this process. Below, we discuss the impact of these new requirements on the traditional compilation framework.

Code generation: background and challenges posed by VLIW ASIPs

Traditional code generation. Traditional code generation consists of three main phases: instruction selection, register allocation and assignment, and scheduling and compaction. During the instruction selection phase, an intermediate representation of the source program is mapped to "atomic" machine operations, or "micro-operations," each typically specifying the transfer of a computed value to a register or memory location. Next, during the register allocation and assignment phase, program variables and intermediate results are mapped to sets of machine registers and then to specific physical registers. When scheduling is performed, a partial order for the execution of the previously obtained micro-operations is established that maintains the semantics of the original program. Finally, during compaction, micro-operations are mapped to (i.e., compacted into) actual machine instructions. Note that compaction takes place only for machines with some degree of parallelism (e.g., superscalar or VLIW processors).

These three phases are each exceedingly complex as well as mutually dependent, so addressing them sequentially can result in suboptimal code. This is known as the phase-coupling problem.[1] Although there is some consensus on the adequacy of the above phasing in the context of traditional compilers, the problem is far from solved for clustered VLIW ASIPs. Moreover, traditional algorithms or approaches to address compilation steps may not work well for such machines.

Code generation for embedded VLIW ASIPs. A key aspect driving the need to reevaluate the code-generation process for VLIW ASIPs is the hierarchical organization of the datapath into clusters (i.e., distributed "smaller datapaths" with limited communication bandwidth). Indeed, in this context, a particularly critical subproblem is the binding or assignment of operations to clusters, since it is crucial to effectively exploit the fine-grain parallelism present in the program. Although the cluster binding or assignment phase can be viewed as an "abstract form" of instruction selection, this critical step is not present in traditional code generation. The "quality" of a cluster binding or assignment depends on achieving a good trade-off between maximizing parallelism (i.e., allowing execution of as many concurrent operations as possible, even if that means "spreading" them across the clusters) and minimizing data transfers across clusters or to memory, since these may harm latency and energy consumption. In order to assess the quality of a binding, some (relaxed) form of early scheduling or compaction and data routing is required. Thus, there is a need to revise the manner in which the coupling among various compilation phases is handled in the context of clustered machines.

The clustered nature of these machines also impacts the effectiveness of traditional compilation algorithms. For example, coloring-based approaches have been successfully used for register allocation or assignment.[7,1] Unfortunately, such algorithms may perform quite poorly in the context of distributed register file organizations—in particular, when minimizing memory spills is important so as to reduce energy consumption. Hence, there is the need to develop approaches that consider datapaths with several register files and effectively route streams of data to or from and across register files (clusters). In doing so, one should attempt to minimize the need for spills to memory (to reduce energy consumption) as well as maximize the time windows for completing the required memory accesses (to avoid congestion on the interconnection structure). Along these lines, we propose[2] a technique that explores trade-offs between the size of potential prefetching windows and minimization of spills, given a maximally concurrent schedule of operations.

Taking a broader view of this problem, effective code generation for clustered machines with distributed storage requires not only revising the problem decomposition and associated algorithms but also taking a fresh look at the way dependencies among the resulting subproblems are handled. Specifically, the traditional sequential approach to code generation, using "monolithic" algorithms, needs to be replaced by a *hierarchical, iterative* compilation process. In such an approach, early compilation tasks (e.g., cluster binding or assignment) might be based on an abstract, coarse view of the datapath, permitting a rough assessment of quality without completely carrying out scheduling, register assignment, and data routing.

The subsequent compilation tasks would use increasingly precise models of the processor (datapath, memory system, and interconnect structure) to achieve high-quality code generation. Thus, a hierarchy is needed that enables one to easily move from coarse- to fine-grain models and back. Of particular importance is the ability to back-annotate high-level models with key information extracted from more-detailed ones. This not only allows a more-effective exploration of possible compiled code but also is compatible with the need to support the datapath and memory design process, as we discussed above.

We propose[4] and briefly illustrate below an algorithm and model that exhibits some of these desirable characteristics, while addressing the critical cluster binding or assignment problem. Given a time-critical loop body segment for which code is to be generated, the model and associated algorithm realize a careful decomposition and relaxation of the global scheduling problem, both in time (machine cycles) and space (clusters and interconnection network). This decomposition, called a window dependency graph, allows one to control complexity while reasoning about bindings. In particular, it is useful to explore trade-offs between achieving high instruction-level parallelism (ILP) versus data transfer penalties required to achieve such parallelism. The key idea is to judiciously relax both capacity and scheduling constraints so as to efficiently reason early on about binding. Thus, for example, one might initially assume unlimited local storage capacity but retain a finite (aggregated) capacity constraint for the interconnection network. This relaxed model for the datapath allows one to bring in both scheduling and a crude form of data routing, while handling important cluster assignment decisions. This model is currently being extended so as to support other phases of the code-generation process as well as data partitioning.

In addition to the code-generation issues discussed above, many of the so-called intermediate performance optimizations traditionally performed in a machine-independent fashion will have to be merged into the code-generation phase, at least during the generation of final or optimized machine code. Next, we discuss the rationale and challenges in this domain.

Performance-enhancing optimizations: background and challenges posed by VLIW ASIPS

As summarized in Figure 2, there are multiple issues specific to compilation for embedded clustered VLIW ASIPs that make the use of traditional *machine-independent* optimization techniques problematic. In particular, embedded applications may have stringent throughput as well as program memory and/or energy consumption constraints. When not used judiciously, these techniques may result in prohibitive increases in code size. Moreover, some such transformations may increase memory accesses, including spills due to insufficient local storage, which in turn may reduce throughput and/or increase energy consumption.

This suggests that during initial design space exploration or compilation steps, it may make sense to apply these techniques in a quasi-machine-independent context in order to quickly obtain upper bounds on achievable throughput and general guidance on the machine specializations that would be more favorable. Then, as the specifics of the target machine under construction are fleshed out, they should be incorporated into these algorithms, so as to provide more-precise trade-off information (e.g., tighter bounds on the application-specific metrics of interest). Eventually, when the microarchitecture of the VLIW ASIP is fully defined and the assembly code for the target machine is to be generated by the compiler, such algorithms should be moved into the first phases of the code-generation process and include latency, code size, and energy consumption constraints associated with the embedded application or system.

We will illustrate the inadequacies of machine-independent optimizations and drawbacks of focusing solely on latency by discussing software pipelining, a technique that has traditionally been used quite effectively to increase ILP for time-critical loops.[7] In software pipelining, direct data dependencies between operations are reduced (and thus ILP increased) by creating a modified loop body that pipelines operations from several loop iterations. Unfortunately, software pipelining may result in significant increases in code size. (Software pipelining requires an epilogue to fill in the iterations pipe and a prologue to empty the pipe.) Moreover, software pipelining usually increases internal storage requirements,[7,5] thus when applied to clustered machines with limited local storage, spills to memory and/or data transfers may be introduced, which compromise the gains expected from increased ILP. At the extreme, these additional spills and data transfers can result in decreased throughput, while unnecessarily increasing code size and energy consumption.

An example of an algorithm that captures machine specifics and incorporates constraints beyond latency is the software pipelining algorithm proposed in Jacome et al.[5] This algorithm, to be executed after cluster assignment, can minimize latency under *code size* and resource constraints and considers the effects on performance of memory operations. We are currently working on extensions that control the increase in the lifetime of data objects, since these can lead to costly spills to memory.

As alluded to previously, the applications of interest typically also exhibit large amounts of *data parallelism*, providing major opportunities for performance enhancement. To exploit data parallelism, one must adequately partition data and assign them to memory banks in the distributed memory system and adequately schedule the associated memory operations. This permits one to cost-effectively maximize the *rate* at which data can be streamed to the datapath, while *balancing* computation and memory operations. Although some work has been developed in the data-partitioning, assignment, and scheduling areas (see, e.g., Catthoor et al.[3]), robust optimization techniques to address these problems in the context of a distributed organization of computational resources (i.e., clusters) and data storage resources are still lacking. Due to the first-order impact on performance and power or energy consumption of memory accesses for our class of applications, the need for such techniques cannot be overemphasized.

In this article, we have identified the main challenges posed by clustered VLIW ASIPs, have discussed promising research directions, and have briefly presented some of our work in this area. We focused on clustered VLIW machines for two reasons. First, they constitute an important class of new machines that are particularly effective in the context of increasingly pervasive portable digital communications and multimedia consumer electronics. Second, they introduce a form of hierarchical aggregation that we believe not only is here to stay but also will be increasingly critical, as microelectronic technology enables increasing levels of integration. With the possibility of placing a very large number of processors on a single (billion-transistor) chip on the near horizon, we see a vast arena for research in application-specific, high-performance computing systems. Specifically, the future challenges might lie in the development of *single-chip* “special-purpose” networks of loosely connected supernodes (i.e., clusters or aggregates of specialized processors such as the VLIW ASIPs we discussed here), cooperating on a MIMD or quasi-MIMD schema and supported by an effective distributed main memory system that is partially or fully placed on a chip. Still, the starting point is to develop a solid understanding on how to control the complexity associated with designing and compiling for a single such specialized processor.

Acknowledgments

This work is supported by a National Science Foundation (NSF) Career Award MIP-9624231, NSF Award CCR-9901255, and Grant 003658-0649-1999 of the Texas Higher Education Coordinating Board Advanced Technology Program.

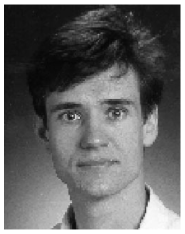
■ References

1. A. Aho, R. Sethi, and J. Ullman, *Compilers: Principles, Techniques and Tools*. Reading, Mass.: Addison-Wesley, 1988.
2. R. Anand, M.F. Jacome, and G. de Veciana, “Heuristic Tradeoffs Between Latency and Energy Consumption in Register Assignment,” *IEEE/ACM 8th Int’l Workshop Hardware/Software Codesign*, May 2000.
3. 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*. Kluwer Academic Publishers, 1998.

4. M. Jacome and G. de Veciana, “Lower Bound on Latency for VLIW ASIPs,” *Proc. ACM/IEEE Int’l Conf. Computer Aided Design (ICCAD)*, Nov. 1999.
5. M. Jacome, G. de Veciana, and C. Akturan, “Resource Constrained Dataflow Retiming Heuristics for VLIW ASIPs,” *7th Int’l Workshop Hardware/Software Codesign*, May 1999, pp. 12–16.
6. M. Jacome and H. Peixoto, “Design Reuse, How Far from Delivering the Promise,” *IEEE Design and Test of Computers*, to appear.
7. P. Marwedel and G. Goossens, eds, *Code Generation for Embedded Processors*. Kluwer Academic Publishers, 1995.
8. S. Rixner, W. Dally, B. Khailany, P. Mattson, U. Kapasi, and J. Owens, “Register Organization for Media Processing,” *Proc. 26th Int’l Symp. High-Performance Computer Architecture*, May 1999.

Margarida F. Jacome is an assistant professor in the Department of Electrical and Computer Engineering at the University of Texas at Austin. She received the BS and the MS degrees from the Technical University of Lisbon in 1981 and 1988, respectively, and the PhD degree in electrical and computer engineering from Carnegie Mellon University in 1993. Her research focuses on CAD for hardware/software codesign of embedded systems, application-specific high-performance programmable architectures, and retargetable compilers. In 1992, she was the recipient of the ACM/IEEE Design Automation Conference Best Paper Award. She is the recipient of a Halliburton Foundation Award of Excellence and a 1996 National Science Foundation Career Award.



Gustavo de Veciana received his BS, MS, and PhD 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 focuses on issues in the design and control of telecommunication networks and developing algorithms for CAD. He is an editor for the *IEEE/ACM Transactions on Networking*. He is the recipient of a General Motors Foundation Centennial Fellowship in Electrical Engineering and a 1996 National Science Foundation Career Award.

■ Direct comments and questions to Margarida Jacome, Department of Electrical and Computer Engineering, University of Texas, Austin, TX 78712; jacome@ece.utexas.edu.