# An Automatic System for Application-Specific Instruction Format Design and Code Generation for VLIW and EPIC processors

Shail Aditya Scott Mahlke B. Ramakrishna Rau
{aditya,mahlke,rau}@hpl.hp.com

Hewlett-Packard Laboratories
1501 Page Mill Rd. MS 3L-5
Palo Alto, CA 94304

Richard Johnson
rjohnson@transmeta.com

Transmeta Corporation
3940 Freedom Circle
Santa Clara, CA 95054

**Introduction.** Whereas the workstation and personal computer markets are rapidly converging on a small number of similar architectures, the embedded systems market is enjoying an explosion of architectural diversity. This diversity is driven by demands for higher performance at a lower cost and power consumption, and is propelled by the possibility of designing application-specific instruction-set processors (ASIPs) that are optimized for particular application domains. To achieve high performance, embedded VLIW (Very Long Instruction Word) and EPIC (Explicitly Parallel Instruction Computing) processors have begun to establish themselves as the processors of choice. However, the sheer complexity and the huge initial investment needed for the design of an application-specific, embedded VLIW processor, along with a compiler for it, can be prohibitive. The goal of our *Program-In-Chip-Out* (PICO) design project is to automate this process completely, so that a family of optimized VLIW processor designs may be generated automatically that are all customised to a given application program or domain and exhibit various cost-performance trade-offs.

For each processor design point, we create an abstract architecture specification which consists of the opcode repertoire of the target machine, its register files, and a specification of the instruction-level parallelism (ILP) desired. This is then used to drive the design of the instruction format, the datapath, the instruction fetch pipeline and the associated instruction decode logic. The system also automatically retargets our VLIW compiler, Elcor, to the target processor by extracting a high-level machine description (mdes) from the processor design. In this paper, we will focus on the automatic instruction format design and the code generation system used in PICO and show how they work in tandem to produce high-quality, application-specific designs. The quality of the generated formats is a function of both the instruction length, which affects code size, and its distribution and decode complexity, which affects the processor cost and performance. To illustrate the effectiveness of the system, a set of experiments are presented that measure the benefit of customizing the instruction format to a particular application and illustrate the design trade-off between code size and hardware complexity.

**Instruction format design.** The code generator for VLIW processors is typically faced with varying degrees of ILP both due to the compiler's limited ability to find it in the given application and the constraints present in the architecture. In order to optimize the overall code size, therefore, PICO generates variable-length, multi-template instruction formats. Each instruction template consists of one or more concurrent operation slots each of which encodes one or more mutually exclusive operations using multiple instruction fields. Each instruction field provides control information to the corresponding control port in the datapath. The same operation may appear in multiple templates which allows the code generator to select the minimum sized template for a given set of co-scheduled operations. A novel aspect of our design is that this *logical* instruction format is implemented by an optimized *physical* instruction format which is geared more towards compressing the template size and reducing the instruction distribution and decode hardware complexity rather than human readability or convenience.

The process of instruction format design consists of the following major steps. First, a set of logical templates are generated covering the given opcode repertoire of the architecture keeping in mind the ILP constraints among them. Our strategy is to first partition all operations into equivalent sets called *super groups* and then to treat each concurrent clique of super groups as a logical instruction template.

Next, the instruction fields in the various templates are assigned physical bit positions. Two instruction fields are said to *conflict* if they can be present simultaneously in the same logical template, and therefore must be assigned disjoint bit positions. Conversely, non-conflicting fields may be assigned the same or overlapping bit positions. The allocation algorithm we use is a variant of Chaitin's graph coloring algorithm where instruction bits are resources and each requesting instruction field may request multiple bits. We use various heuristics to reduce the overall template width and the decode complexity, such as packing the instruction fields to the left (leftmost allocation), assigning contiguous bit positions to multi-bit fields (contiguous allocation), and aligning instruction fields corresponding to the same control port to the same bit position (affinity allocation).

**Retargetable code generation.** The code generation system maps a machine-independent intermediate representation (IR) used by the compiler into the machine-dependent binary form as specified by the instruction format. PICO's code generation system consists of the following components: literal code generator, operation scheduler, register allocator, assembler, and linker.

The machine-independent IR assumes any operand can be a literal and there are no restrictions on its size. The actual machine, however, has specific restrictions on porting and sizes of literals as specified in the architecture specification. The literal code generator is responsible for identifying literal operands that are not directly representable as immediate operands of operations and generating code to materialize them into registers. The scheduler and register allocator then map the resulting IR to the physical processor resources while performing sophisticated ILP optimizations. They are targeted to a specific processor by the high-level machine description extracted automatically from the processor datapath.

The assembler, and the linker make specific use of the instruction format. The assembler is responsible for mapping the set of operations concurrently scheduled on each cycle into an appropriate instruction template. The smallest sized template that can accommodate the set of operations is chosen. In addition, the assembler is responsible for inserting any no-ops to fill empty cycles left by the scheduler and for aligning branch target instruction to a *packet*[1] boundary. The latter avoids instruction cache fetch stalls for branch targets at the expense of slightly larger code size. The output of the assembler is a binary representation as specified by the instruction format for each procedure in the application. Finally, the linker combines the individual procedure binaries into an executable program. The linker is responsible for interprocedural instruction address assignment and code layout.

**Custom template design.** Normally the instruction format is designed to cover all and only the concurrency relations implied by the high-level architecture specification. However, this often leads to many templates that are either not used or not fully utilized and hence a wastage of code space. The solution to this problem is to customize the templates to the application and processor of interest. To facilitate this, the PICO design system uses the following design flow.

First, the literal code generator, the scheduler, and the register allocator are used to translate the machine-independent IR into fully-bound, assembly-level code. The operation schedule is used to extract information about which combinations of operations are most frequently scheduled together. This information is then used to generate a set of custom templates that, together with the set of original templates, can be used to more efficiently represent the application. Finally, the assembler uses this extended set of templates to produce object code, thereby reducing the requisite code size.

**Experiments.** We illustrate the overall operation of the PICO instruction format design and code generation system with a set of experiments. One unique feature of this system is the ability to automatically customize the instruction format for a particular application. The effectiveness of our customization procedure over the baseline instruction format is measured for a set of embedded applications. A second feature of the system is the ability to design instruction formats that tradeoff decode complexity versus code size. A set of measurements illustrating this design space tradeoff is also presented.

[1] A packet consists of the bits fetched from the instruction cache in a single cycle.