Automated Instruction-Set Extension of Embedded Processors with Application to MPEG-4 Video Encoding

Nikolaos Kavvadias and Spiridon Nikolaidis
Section of Electronics and Computers, Department of Physics
Aristotle University of Thessaloniki, 54124 Thessaloniki, Greece
{*nkavv,snikolaid*}*@physics.auth.gr*

Abstract

A recent approach to platform-based design involves the use of extensible processors, offering architecture customization possibilities. Part of the designer responsibilities is the domain-specific extension of the baseline processor to fit customer requirements. Key issues of this process are the automated application analysis and candidate instruction identification/selection for implementation as application-specific functional units (AFUs). In this paper, a design approach that encapsulates automated workload characterization and instruction generation is utilized for extending processors to efficiently support embedded application sets. The method used for instruction generation is a highly parameterized adaptation of the MaxMISO technique, which allows for fast design space exploration. It is proven that only a small number of AFUs are needed in order to support the algorithms of interest (MPEG-4 encoding kernels) and that it is possible to achieve 2× to 3.5× performance improvements although further possibilities such as subword parallelization are not currently regarded.

1. Introduction

In the development of processors for consumer applications, closely matched design of instruction sets and micro-architectures is required, in respect to the application benchmarks, while being subject to several constraints. These constraints stem out of diverse and often conflicting requirements such as low power consumption, performance in a given application domain, code size and overall system cost [21].

The challenge of delivering the optimum balance between efficiency and flexibility can be met with the utilization of customizable processors. Most commercial offerings fall in the category of configurable and extensible processors [1],[19]. Configurability lies in tuning architecture parameters of synthesizable cores [2] or interfacing to application accelerators through the local bus. Extensibility of a processor usually refers to tightly-coupled modifications obtained by adding single-, multi-cycle or pipelined versions of complex instructions and the introduction of their associated functional units at the execution stage(s) of the processor pipeline.

The designer freedom in such configuration/extension scenarios has to be exploited for advantageous domain-specific specialization. While it has been argued that complete application characterization is a demand for inhibiting mismatches between expected and delivered performance [23], the established approach follows a two-level strategy of focusing on aggressively optimizing the application kernels and at a subsequent phase examining its effect on the entire application. For this reason, MediaBenchII [18], the awaited update to the popular MediaBench benchmark suite [24], will incorporate kernels such as motion estimation (key procedure in video coding standards) and wavelet filters.

In this paper, an application analysis and custom instruction generation prototype framework is presented, based on the MachSUIF compiler infrastructure [3] and a parameterizable instruction generation engine. Its features include support of two RISC-like instruction sets, built-in area and delay early estimators for the AFUs, classic compiler optimization passes, arithmetic optimizations, and a highly-controllable version of the MaxMISO instruction generation algorithm [27] that enables interesting multi-dimensional design space exploration possibilities. This framework has been utilized for identifying common instruction-set extensions for popular video encoding kernels.

The rest of this paper is organized as follows. Related work is summarized in Section 2. The instruction generation approach for customizing embedded ASIPs is detailed in Section 3. Section 4 discusses the application of the proposed approach on an MPEG-4 compliant shape encoder as a case study, and in Section 5 it is further applied on a set

of block-matching motion estimation algorithms. Finally, Section 6 summarizes the paper.

2. Related work

Methodical research efforts on application-specific extensions regard automating methods to explore the architecture design space [16],[30],[17],[28],[20]. A few open instruction generation frameworks that can be directly evaluated for the instruction-set extensions exist [16],[4]. An advantage of their approach is the usage of a pattern file format for storing, manipulating and exchanging instruction patterns. Some issues with the Pattlib approach regard the significant efforts for adapting the GCC compiler to emit information in "pattlib" format, and that the intermediate representation (IR) for their selected backend (SPARC V8) is not architecture-neutral. A disciplined approach to custom instruction generation for extensible processors is found in [20] where the Xtensa processor is augmented with application-specific instructions that may combine VLIW, SIMD or fused (chained) RTL operations. However, it is strongly oriented towards Xtensa, applying strict restrictions on the design space since it considers only 2-input MISO single-cycle instruction candidates. In addition to that, although they have included MPEG-4 video encoding as one of their benchmarks, details on the specifics for the motion estimation algorithms are omitted.

Close to our aims but from an empirical viewpoint, recent ARM architectures (ARM9 and later) can be optionally augmented with a low-power programmable coprocessor named MOVE [9], for accelerating the SAD operation in motion estimation. The plain SAD calculation without regarding data transfers requires 2 or 3 cycles given the operating mode. However, due to the effect of the limited bandwidth for accessing the pixel values, a $2\times$ speedup for MPEG-4 encoding over a software implementation is reported, although their use of SIMD instructions implies an 8-fold theoretical speedup limit (assuming transparent data updates for the pixel block buffer).

3. Custom instruction generation procedure

It is often at early stages in processor design, that the compilers and simulators for the design space of applicable processor architectures are not available. As a common estimation platform, we use the MachSUIF IR, which represents a generic RISC ISA named SUIFvm. Dynamic characterization is performed using the C backend *m2c* available in the MachSUIF distribution.

The instruction generation flow, which is shown in Fig. 1, is an enhancement of the work in [22]. On Step 1, the input C code for the application is processed by the SUIF

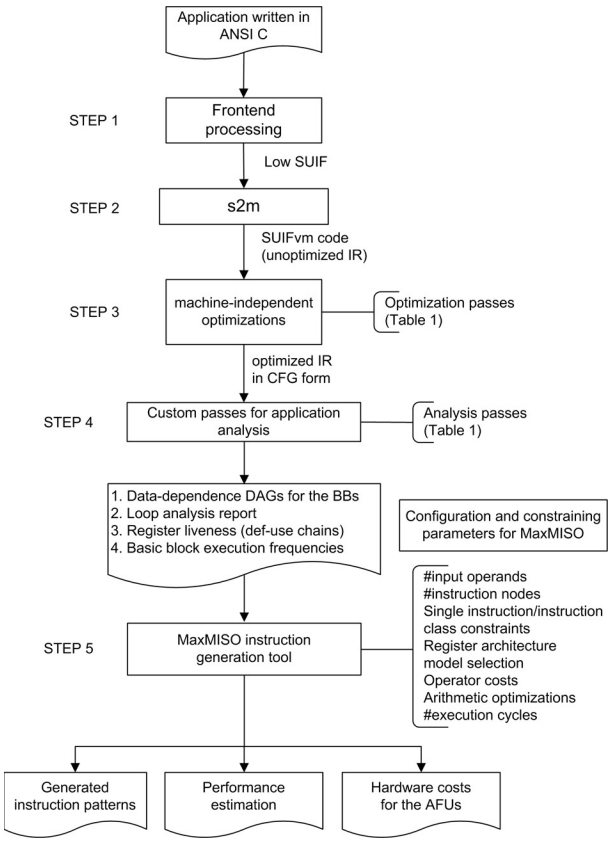


Figure 1. Application analysis and parameterised instruction generation flow.

frontend, to generate AST description with SUIF nodes for the application [5]. On Step 2, the resulted representation is fed to the *s2m* pass to emit SUIFvm assembly-like IR. The IR code is unscheduled while complete procedure entry and exit sequences have not been inserted at this stage, since stack frame layout is highly processor dependent. It is not meaningful to seek useful instruction extensions in stack manipulation code since in this case false dependencies within the data flow graphs of each basic block are created [17]. Step 3 performs architecture-independent optimizations on the IR most of them detailed in Table 1.

On Step 4, specific static and dynamic profiling information for the input application is gathered with the help of a set of analysis passes, accepting SUIFvm IR in CFG form. Table 1 gives compact descriptions of analysis and transformation passes that have been added to MachSUIF.

The instruction generation process takes place on Step 5. The instruction identification and generation engine currently implements the MaxMISO (maximal multiple-input single-output) algorithm [27], which identifies the maximal non-overlapping connected subgraphs of the data-dependence directed-acyclic graph (DAG) that produce a single computation result. In its original form, the only ap-

Table 1. Custom analysis and transformation passes for MachSUIF.

Pass name	Description	Pass type
dagconstruct	Construct DDGs for each basic block	Analysis
instrmix	Generates static instruction mix	Analysis
liveanalysis	Def-use chains/liveness analysis using the MachSUIF *cfa* library	Analysis
loopstr	Natural loop analysis using *cfa*	Analysis
m2c_bb	GNU patch and Perl scripts to correct and instrument *m2c* output	Analysis
buildcg	Call-graph constructor	Analysis
lcse	Local CSE optimization [6]	Optim.
if_conversion	If-conversion optimization [6]	Optim.
cplx_locate	Locates portions of SUIFvm code that can be replaced by uses of SUIFvm instructions: *abs*, *min*, *max*	Optim.
strength_reduct	Simple operator strength reduction for multiplication and division	Optim.

plicable constraints regard the maximum number of input operands that can be delivered to the AFU, but in our implementation is enhanced in order to be more suitable for performance tradeoff analysis. These MaxMISO parameters include:

1) The maximum number of primitive instruction nodes to be included in the MaxMISO.

2) The establishment of two types of node constraints that can be applied to any instruction class:

   a) Type-A or *boundary-node* constraint: Applying this constraint, prohibits growing an instruction cluster beyond the specified instruction. It has been observed that its application on data transfer instructions (load, store, memory-to-memory copy), forces the generation of complex addressing modes. This procedure automates a traditionally ad-hoc portion of the ASIP design flow, which is the identification of the most beneficial addressing modes for the processor's data transfer instructions.

   b) Type-B or *node-inclusion* constraint: A constraint of this type will not permit the inclusion of the specified instruction in the MaxMISO under build. It is usually applied to control-transfer instructions (*cti*) such as conditional/unconditional branch (*cbr*/*ubr*) and call/return. In the majority of extensible processors, the end-user is not permitted to alter the control transfer mechanisms i.e. to add complex instructions to the original ISA that modify the instruction fetch path, since its effect to the processor cycle time is less predictable than in the case that instruction extensions reside solely on the execution pipeline stage(s) of the processor.

3) Applying a limit on the maximum number of hardware cycles that is required for instruction execution. Single-cycle instructions require only minor modifications to the main instruction decode logic while multicycle instructions demand additional FSM control and possibly user-defined state registers [17],[12].

Additional features of our MaxMISO implementation include:

1) Support for SUIFrm (SUIF real machine), a close backend ISA to the SUIFvm IR that allows register allocation and code generation.

2) Single constant multiplication optimization according to Bernstein's algorithm [10],[13] but with exact delay costs instead of cycle costs.

3) Multi-operand addition optimizations [29],[11] that currently are applied after the main instruction generation procedure.

4. Motivational example: MPEG-4 context-based shape encoding

As demonstration application, we have selected a context-based shape encoder for MPEG-4 [8],[7]. This encoder allows for lossy decisions in the encoding of shape information, and is controlled by a motion vector prediction related (*motion_th*) and a lossy compression (*alpha_th*) parameter.

For the 16 run-cases defined by the following parameter space: {*alpha_th*, *motion_th*}={0, 32, 64, 256} it was derived that the most performance-critical basic blocks are BB #40 and #41 of the *find_vects* procedure (referred to as BB1 and BB2 in Table 2 respectively) where SAD computation takes place. Table 2 summarizes statistics for the generated custom instructions under 3 different constraint scenarios and for number of inputs given in ("#inputs" column). In column "#MaxMISO num./size", the number of static occurrences of custom instructions and their average size in number of primitive instructions are given. It is deduced that the constraint on the number of inputs has a dramatic impact on the size of the generated MaxMISO. The achievable speedups range from 1.07 to 3.50, the latter for the case with no restriction on the number of inputs. This case can only be realized with the use of state registers that significantly reduce the demand of input operands from the register file.

Fig. 2(a) shows the data-dependence graph of the maximal speedup MaxMISO identified in BB1 under node constraints: {Type-A/Type-B} = {*str*/*cal*}. Area and delay metrics in Fig. 2 are normalized to the values for a 32×32-bit single-cycle multiplier returning a 64-bit result (not truncated). By observation of the data flow graph in Fig. 2(a) we can extract some important remarks:

1) There exist 3 different instances of multi-operand addition with 3-, 4-, and 5- input operators, while larger addition structures could be devised by collapsing constant multiplications in the DFG. We have characterized multi-operand adders based on carry-save *n*:2 compressors, from a public VHDL library [11].

2) There is no variable-by-variable multiplication involved, but only multiplications-by-constant, where this constant is the picture width. In case a set of constants has to be supported, multiple constant multipliers should be used instead.

3) The 2 memory load operations (*lod*) do not correspond to zero-successor nodes of the DFG. Generally, load/store instructions infer a node-inclusion constraint, if the corresponding AFUs are not connected through parallel paths to the data memory.

4) An ILP (instruction-level parallelism) of only 1.6 is calculated for the DFG of Fig. 2(a) and of 2.18 for a 4-way processor configuration simulated with SimpleScalar 3.0d [15] for the entire application. It can be argued that low ILP, privileges architectures that exploit chained against VLIW operations. In contrast to VLIW architectures issuing simple independent operations each one occupying an instruction slot, processors with chained instruction extensions, exploit dependent operations. Notably, media-centric applications such as MPEG-4 visual (versions 1 and 2), and H.264/AVC have respectively low ILP of about 2 in average for both the encoding and decoding applications [18].

The final AFU hardware (Fig. 2(b)) for the MaxMISO of Fig. 2(a) employs both multi-operand addition and constant multiplication optimizations in addition to the manual introduction of state registers, motivated by analysis of register liveness results. The maximum speedup can be approached with 8 state registers as can be seen in Fig. 2(b). Also, selection of operator bitwidths has been performed with nodes *sub* and *abs* operating on 8-bit and the remainder nodes on 16-bit quantities. Overall, automated and manual optimizations result in 85% and 40% reduction against the unoptimized hardware for the derived MaxMISO for area and delay metrics, respectively.

Table 2. Detailed statistics for the generated custom instructions.

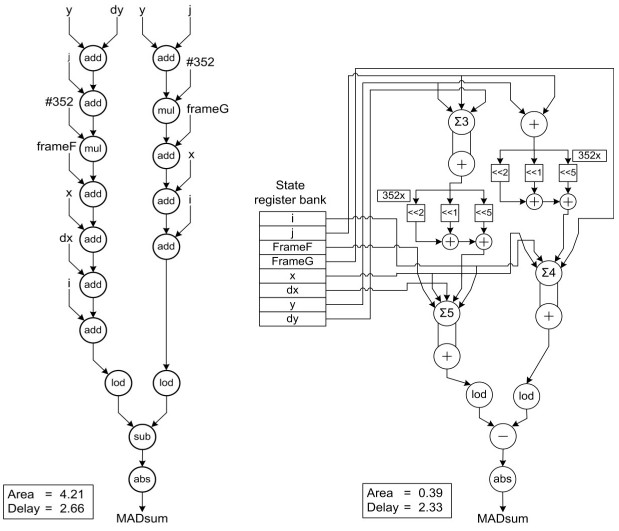
Basic block	#inputs	Constr. Type A/B	MaxMISO num./size	Rel. %cycles	Speedup
BB1	2	str/cal	5/2	73.68	1.36
BB2	2	str/cal	2/2		
BB1	2	str,lod/cal	5/2	73.68	1.36
BB2	2	str,lod/cal	2/2		
BB1	2	str,lod/cti	5/2	86.61	1.15
BB2	2	str,lod/cti	2/2		
BB1	4	str/cal	6/2.33	47.37	2.11
BB2	4	str/cal	2/3		
BB1	4	str,lod/cal	6/2.67	47.37	2.11
BB2	4	str,lod/cal	2/3		
BB1	4	str,lod/cti	6/2.67	65.22	1.53
BB2	4	str,lod/cti	2/2.5		
BB1	∞	str/cal	1/16	28.57	3.50
BB2	∞	str/cal	1/6		
BB1	∞	str,lod/cal	3/4.67	43.61	2.29
BB2	∞	str,lod/cal	2/3		
BB1	∞	str,lod/cti	3/4.67	48.78	2.05
BB2	∞	str,lod/cti	2/3		

5. Instruction extension generation for a motion estimation stressmark

At this point, we will evaluate the proposed approach for instruction-set extension of block-matching motion estimation algorithms, which are used in MPEG video compression systems. Table 3 presents the algorithmic kernels under investigation [26]. From left to right, the abbreviation, a short description and the number of executed instructions for processing a 10-frame sequence are given.

5.1. Design space exploration and tradeoff analysis

As a first task in tradeoff analysis, we identify the maximum performance increase that can be achieved by adding support for MaxMISO instruction clusters to the base microarchitecture. In Fig. 3, normalized execution cycles are plotted against the maximum number of inputs for given cycle constraints for the average case of all kernels. We can see that for up to 2-input instructions, a 2× speedup is expected, while for 4-input instructions it increases to 2.5 and for the asymptotical case of unconstrained number of inputs that could be approached by coarse-grain reconfigurable hardware, a theoretical maximum of 3.5 is observed. It should be noted, that bitwidth optimizations have not been considered but a number of algorithms for bitwidth analysis [14],[25] are currently under study for extending our framework. Last, it appears that constraining the number of cycles per instruction does not affect performance in terms of execution cycles for a small to moderate number of inputs.



(a) Initial DFG. (b) Utilization of multi-operand addition, constant multiplication and state registers.

Figure 2. Hardware optimization for the MaxMISO identified in find_vects.40.

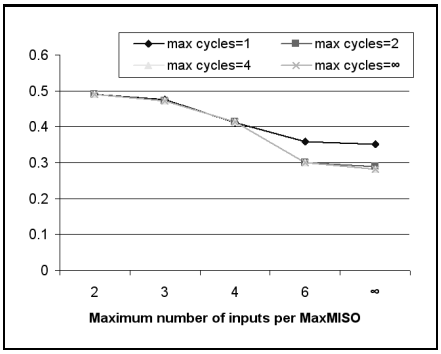


Figure 3. Normalized execution cycles for the average case of motion estimation algorithms.

At this point, we restrict our interest to the critical basic blocks of the examined MPEG-4 kernels, and to custom instructions of up to four input operands. It has been derived that significant performance increase can be achieved with only three additional functional units for supporting the shape encoder and a variety of motion estimation algorithms. DFG views for the two AFUs extracted from the stressmark are given in Fig. 4. Also, as in Section 4, arithmetic optimizations have been applied. The AFU in Fig. 4(a) implements the motion compensation task. The AFUs in Fig. 2(b), 4(a), and 4(b), are the suggested architectural extensions for the examined video encoding kernels that could be added to typical embedded processors, media-processing ASIPs or as custom logic to configurable processors already in the market (Altera Nios-II, ARC, Xtensa).

Table 3. Summary of the motion estimation algorithms.

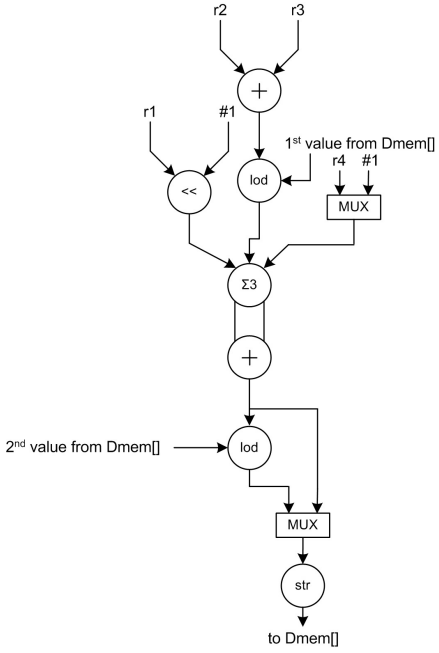
Abbrev.	Algorithm description	Exec. instr.
fs	Full-search	524006501
ofs	New full search	85480757
3ss	Three-step search	66407224
4ss	Four-step search	55404110
n3ss	New three-step search	67789322
bbgds	Block-based gradient descent search	47494185
coher	Region-based optical flow motion estimation	58004136
liu	Full-search with 4:1 alternate pel subsampling	110970080
pfs	Partial distortion full search	156713876
ucbds	Unrestricted center-biased diamond search algorithm	53705554

6. Conclusions

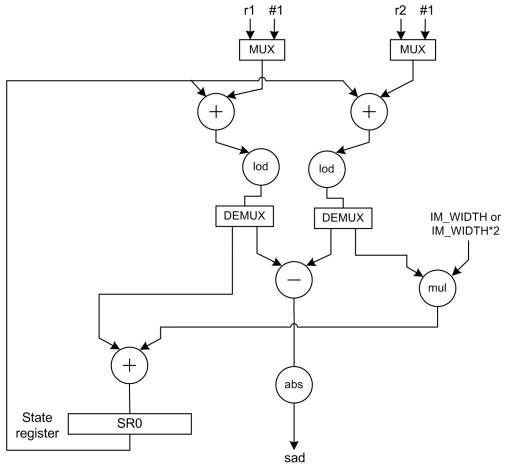
In this paper, an application analysis and instruction generation flow is used for automatically identifying beneficial instruction-set extensions of embedded processors. For this reason, a prototype instruction generation engine allowing multi-dimensional design space explorations of MaxMISO-generated custom instructions has been implemented. We have performed automated explorations regarding the maximum number of input operands, instruction latency, as well as node-inclusion and boundary-node constraints, constant multiplication and multi-operand addition in the search for optimal custom instructions and their resulting AFUs. It is concluded that only three AFUs are needed to accelerate a variety of motion-estimation related algorithms up to 3.5 times over the baseline processor.

References

[1] ARC Cores. http://www.arccores.com.
[2] Gaisler Research. http://www.gaisler.com.
[3] Machine-SUIF research compiler. http://www.eecs.harvard.edu/hube/research/machsuif.html.
[4] Pattlib. http://www.lsc.ic.unicamp.br/pattlib/.
[5] SUIF. http://suif.stanford.edu/suif/suif2/.
[6] Processor Architecture Laboratory at EPFL homepage. http://lapwww.epfl.ch/dev/.
[7] MPEG-4 shape encoder project. http://www.ece.cmu.edu/˜ece796/project99/12/final/.
[8] MPEG-4 video verification model version 18.0. MPEG-4 draft standard, International Organization of Standardization, Working Group on Coding of Moving Pictures and Audio, Pisa, Italy, Jan. 2001.
[9] ARM Ltd. *MOVE Coprocessor Technical Reference Manual*, Apr. 2002.
[10] R. Bernstein. Multiplication by integer constants. *Softw. - Practice and Experience*, 16(7):641–652, July 1986.



(a) DFG for the *res_image.9* basic block.



(b) Common single-cycle SAD implementation unit.

Figure 4. AFU hardware for the motion estimation stressmark.

[11] J.-L. Beuchat. *A VHDL Library for Integer and Modular Arithmetic - User Manual*, 0.1 edition, Sept. 2004. http://perso.ens-lyon.fr/jean-luc.beuchat/ArithLib.

[12] P. Biswas, V. Choudhary, K. Atasu, L. Pozzi, P. Ienne, and N. Dutt. Introduction of local memory elements in instruction set extensions. In *Proc. 41th Design Automation Conf.*, pages 729–734, June 7-11 2004.

[13] P. Briggs and T. Harvey. Multiplication by integer constants. Technical report, Rice University, July 1994.

[14] M. Budiu, M. Sakr, K. Walker, and S. C. Goldstein. BitValue inference: Detecting and exploiting narrow bitwidth computations. In *Proc. of the European Conf. on Parallel Processing*, pages 969–979, 2000.

[15] D. C. Burger and T. M. Austin. The SimpleScalar tool set, version 2.0. Technical report CS-TR-1997-1342, Computer Sciences Department, University of Wisconsin-Madison, June 1997.

[16] P. Castro, E. Borin, R. Azevedo, and G. Araujo. Looking for instruction patterns in the design of extensible processors. In *Proc. 3rd Wshp. on Application Specific Processors*, Sept. 2004.

[17] N. Clark, H. Zhong, W. Tang, and S. Mahlke. Automatic design of application specific instruction set extensions through dataflow graph exploration. *Int. J. Parallel Programming*, 31(6):429–449, Dec. 2003.

[18] J. Fritts. Mediabench II. In *2004 Workshop on Media and Signal Processors for Embedded Systems and SoCs*, pages 723–728, Sept. 2004.

[19] R. Gonzalez. Xtensa: A configurable and extensible processor. *IEEE Micro*, 20:60–70, Mar.-Apr. 2000.

[20] D. Goodwin and D. Petkov. Automatic generation of application specific processors. In *Proc. Int. Conf. on Compilers, Architectures and Synthesis for Embedded Systems*, pages 137–147. ACM Press, Oct. 30-Nov. 1 2003.

[21] M. Gschwind. Instruction set selection for ASIP design. In *Proc. 7th Int. Wshp. on Hardware/Software Codesign*, pages 7–11, May 3-5 1999.

[22] N. Kavvadias and S. Nikolaidis. Application analysis with integrated identification of complex instructions for configurable processors. In *Proc. 14th Int. Wshp. on Power and Timing Modeling, Optimization and Simulation*, pages 633–642, Sept. 15-17 2004.

[23] K. Keutzer, S. Malik, and A. R. Newton. From ASIC to ASIP: The next design discontinuity. In *Proc. 20th Int. Conf. on Comp. Design*, pages 84–90, Sept. 16-18 2002.

[24] C. Lee, M. Potkonjak, and W. H. Mangione-Smith. MediaBench: A tool for evaluating and synthesizing multimedia and communications systems. In *Proc. 30th IEEE/ACM Symposium on Microarchitecture*, pages 330–335, Dec. 1997.

[25] S. Mahlke, R. Ravindran, M. Schlansker, R. Schreiber, and T. Sherwood. Bitwidth cognizant architecture synthesis of custom hardware accelerators. *IEEE Trans. on Comput.-Aided Design Integrated Circuits*, 20(11):1355–1371, Nov. 2001.

[26] L. M. Po, C. K. Cheung, and C. H. Cheung. Various advanced motion estimation research development package, Aug. 2000. http://www.ee.cityu.edu.hk/˜lmpo/publications/.

[27] L. Pozzi, M. Vuletic, and P. Ienne. Automatic topology-based identification of instruction-set extensions for embedded processors. Technical Report CS 01/377, Swiss Federal Institute of Technology Lausanne, Processor Architecture Laboratory, Dec. 2001.

[28] F. Sun, S. Ravi, A. Raghunathan, and N. K. Jha. Custom-instruction synthesis for extensible-processor platforms. *IEEE Trans. on Comput.-Aided Design Integrated Circuits*, 23(2):216–228, Feb. 2004.

[29] S. Vassiliadis, J. Philips, and B. Blaner. Interlock collapsing ALUs. *IEEE Trans. Comp.*, 42(7):825–839, July 1993.

[30] P. Yu and T. Mitra. Characterizing embedded applications for instruction-set extensible processors. In *Proc. 41th Design Automation Conf.*, pages 723–728, June 7-11 2004.