Compiler based Exploration of DSP Energy Savings by SIMD Operations

Markus Lorenz
Peter Marwedel
University of Dortmund
Dept. of Computer Science 12
44221 Dortmund, Germany
email: Markus.Lorenz@udo.edu
email: Peter.Marwedel@udo.edu

Thorsten Dräger
Gerhard Fettweis
Technische Universität Dresden
Vodafone Chair For Mobile
Communication Systems
01062 Dresden, Germany
email: draeger@ifn.et.tu-dresden.de
email: fettweis@ifn.et.tu-dresden.de

Rainer Leupers
Aachen University of Technology
Integrated Systems for Signal Processing
52056 Aachen, Germany
email: leupers@iss.rwth-aachen.de

Abstract— The growing use of digital signal processors (DSPs) in embedded systems necessitates the use of optimizing compilers supporting their special architecture features. Beside the irregular DSP architectures for reducing chip size and energy consumption, single instruction multiple data (SIMD) functionality is frequently integrated with the intention of performance improvement. In order to get an energy-efficient system consisting of processor and compiler, it is necessary to optimize hardware as well as software. It is not obvious that SIMD operations can save any energy: if n operations are executed in parallel, each of them might consume the same amount of energy as if there were executed sequentially. Up to now, no work has been done to investigate the influence of compiler generated code containing SIMD operations w.r.t. the energy consumption. This paper deals with the exploration of the energy saving potential of SIMD operations for a DSP by using a generic compilation framework including an integrated instruction level energy cost model for our target architecture. Effects of SIMD operations on the energy consumption are shown for several benchmarks and an MP3 application[1].

I. Introduction

Today, *digital signal processors* (DSPs) are frequently used in *embedded systems* to permit application specifications in software. Since performance, chip size and energy consumption are crucial points in embedded systems, special hardware features are implemented. For example, performance is increased by supporting *instruction level parallelism* (ILP) and *single instruction multiple data* (SIMD) operations. Chip size (and energy) is reduced by restricting the number of communication wires and by the use of special-purpose register files. Often it is necessary to find a trade-off between these optimization goals. Until now most research effort in reducing power/energy consumption has focused on the field of low-power design of integrated circuits (for an overview see e.g. [1]). However, to obtain an energy-efficient system, optimizing the software running on the hardware is essential, too. Since many programs are still written in assembly code, which is a very time consuming and error prone process leading to hardly portable code, there is a need for compilers supporting the special DSP architectural features and capable of generating efficient assembly code.

Obviously, the exploitation of SIMD operations bears a huge potential for reducing the execution time. For example making use of n parallel functional units in an SIMD operation, the benefit is $n : 1$ compared to a *single instruction single data* (SISD) operation. The benefit is potentially reduced if (un)packing of data words is required and/or additional loop transformations have to be applied in order to enable the use of SIMD operations (s. e.g. [2]).

If the execution of an SIMD operation consumes more energy than an SISD operation, the exploitation of SIMD operations can lead to a performance improvement but can also cause a more energy-intensive machine program. Energy optimizations are possible with compiler frameworks allowing a fast and precise evaluation of different machine programs in terms of energy consumption.

In this paper, the influence of compiler generated code containing SIMD operations on the energy consumption is investigated for the first time. In order to evaluate programs also w.r.t. energy consumption we have developed an instruction level energy cost model for our target architecture and have integrated it into our compiler and simulator. This allows well-founded statements about the energy saving potential for our target architecture. Due to the generic implementation of our tools, the energy exploration of other SIMD architectures can be done in a similar way.

The paper is structured as follows: In the next section an overview of related work in this area is given. The target architecture of the M3-DSP and its instruction level energy cost model is described in section III. After introducing our compilation framework in section IV, experimental results for several benchmarks and an MP3 application are presented. Finally we conclude the paper in section VI.

II. Related Work

In order to avoid the disadvantages of hand-crafted assembly code, different strategies have been proposed dealing with exploitation of SIMD operations within compilers (s. e.g. [3, 4, 5, 6]). Usually, loop transformations like *loop fission*, *strip mining*, *reduction recognition* or *scalar expansion* are performed in order to improve the exploitation of SIMD operations [7]. However, the influence of SIMD operations on

[1]This work has been sponsored by the German Research Foundation (DFG).

energy consumption is not reported.

In [2], a technique is proposed which aims at reducing the overhead of SIMD operations caused by (un)packing operations for the TriMedia TM1300 processor. The technique is manually applied to one benchmark on the source code level. In addition, the presented energy results for this benchmark are limited to the data memory.

Up to now, there are only a few compilers with integrated energy cost models (s. e.g. [8, 9, 10, 11]). However, none of these consider the exploitation of SIMD operations.

To the best of our knowledge, our compiler framework is the only one capable of making use of SIMD operations and allowing an evaluation of the energy consumption w.r.t. an energy cost model. In [12] we have published compiler optimization techniques with the aim of minimizing the energy consumption of embedded applications, containing only few results for small benchmarks concerning the exploitation of SIMD operations. In complement to that, this paper aims at investigating the influence of SIMD operations on energy consumption in more detail. For this reason, we have improved our vectorization technique so that our SIMD optimization technique is applicable to real-life applications including an MP3 application. Effects of loop transformations, often performed to increase the use of SIMD operations, are demonstrated too.

III. Target Architecture

A. M3-DSP

The M3-DSP is an instance of a scalable DSP platform for mobile communication applications [13]. The platform permits a fast design of DSPs adapted to special applications. In order to meet constraints with respect to real-time processing, chip area, and energy dissipation, the platform supports among others, the following features: There is a scalable number of data paths that enable processing either on a single data path or on all data paths in parallel according to the SIMD principle. In the case of the M3-DSP there are 16 data path slices. In order to provide an effective use of all data path slices in parallel, the memory is organized as an on-chip *group memory*: Addressing one 16-bit data word always means addressing an entire group of 16 such words. The addressed group is loaded into an intermediate register from which the values are distributed to the *group registers* in the data paths by an application-specific inter-communication network. The M3-DSP contains a 4-stage pipeline and is organized as a *very long instruction word* (VLIW) architecture.

B. Instruction Level Energy Cost Model

During the code generation process and for simulation it is necessary to evaluate different code sequences w.r.t. the energy consumption. However, repeated simulations or measurements would mean an unacceptable overhead. Thus, a suitable cost model permitting a precise and quick evaluation of arbitrary instruction sequences is essential. Tiwari [8] reports an instruction level energy model based on measurement of the energy consumption of a single instruction (*base energy cost*) and of the switching activities of successive instructions (*overhead energy cost*) meeting these requirements. Based on Tiwari's work, we have developed a modified instruction level energy cost model for the M3-DSP using measurements on the silicon chip [14]. A validation of our energy cost model shows a deviation of less than 2% compared to measurements on the real hardware when executing the whole program.

As expected, the energy cost model shows that load and store instructions have a more significant contribution to the energy consumption than SISD instructions (SISD = single instruction single data) of the data path. SIMD instructions show the highest energy consumption (4-5 times more compared to SISD instructions) but perform up to 16 useful computations in parallel. For this reason, if the overhead caused by (un)packing operations and loop transformations is small (or zero), executing SIMD instructions will potentially reduce the total energy dissipation of an application.

IV. Compilation Framework

The compilation process is started by transforming a given C source program into a machine independent representation of our *generic low-level intermediate representation* (GeLIR) [15] which serves as exchange format for all succeeding transformations and optimizations. In contrast to traditional intermediate representations it allows for storing machine dependent program and target architecture information. For instance, GeLIR supports the specification of irregular data paths containing special-purpose register resources, restricted parallelism and SIMD operations. Information about the energy consumption of processor instructions is also stored in the GeLIR data structures and permits an efficient evaluation of different code sequences. After applying the SIMD code generator, assembly code is emitted. In addition, the resulting GeLIR representation can be simulated. Due to the integrated energy cost model, information w.r.t. energy consumption is generated for the simulated application, too.

In the following, the main steps of our code generator including the SIMD optimization are described. Code generation is started after the source program is transformed into the GeLIR data structures. At this level of abstraction the source program is given by a set of *data flow graphs* (DFGs) which are then separately mapped to assembly code by performing the following steps:

1. Preprocessing
   The main task of this step is the generation of alternative machine programs (or solutions) of the source program. This is done by inserting the minimum number of graph nodes which are needed to generate all possible data transfer paths between two specific graph nodes and generating an initial covering of the graph nodes with processor resources. The example in figure 1 shows that for each graph node a set of machine operations `Op`, set of functional units `FU`, set of instruction types `IT`[2] and sets of resources for destination `Def` and arguments `Arg` are stored.

2. SIMD optimizations
   The exploitation of SIMD operations (or vectorization) is done by analyzing loops which can be vectorized

[2] Instruction types are used for specifying parallel execution possibilities of different machine operations. Thus, two operations having the same instruction type can potentially be executed in parallel.

(see e.g. [16]). If it is recognized that specific operations embedded in the analyzed loop can be vectorized, the set of operation alternatives is restricted. Communication with the code generator is done by performing restrictions with respect to the available machine operation alternatives which can cover a specific GeLIR graph node. This entails further restrictions e.g. with respect to the available register alternatives. It is thus possible to control the compilation process by preserving potential alternatives for example with respect to register files. In addition, there is no need for special source language extensions.

The result of this step is a DFG with restricted resource alternatives. For example if SIMD operations should be performed, the corresponding SISD resource alternatives are erased. In figure 1 these is demonstrated by crossed out resources.

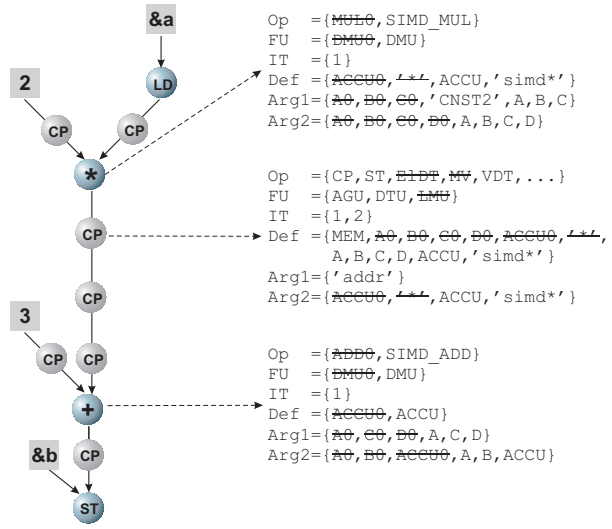

Fig. 1. DFG covering after SIMD optimization

3. Code generation
   In this step, the process of code generation aims at restricting the remaining set of resource alternatives by optimizing according to a specified cost function. In addition, every graph node has to be assigned to a specific control step. Since the generation of optimal assembly code means solving an NP-hard optimization problem, we are using an optimization algorithm based on a genetic algorithm. The result is a DFG with scheduled and parallelized (or compacted) graph nodes covered by processor resources.

4. Address code generation
   In this step, the address code for the given memory access sequence of a specific basic block is generated. The resulting sequential address code is inserted into the GeLIR-code and is then compacted by reusing the genetic algorithm driven code generator.

In order to increase the number of loops which can be vectorized, loop transformations are applied before starting the code generation process. This is done on source code level by using standard techniques [7]. The influence of commonly applied loop transformation to the energy consumption is shown in the following section.

V. Experimental Results

In this section, experimental results concerning the influence of SIMD operations to the energy consumption are presented. All data was generated by our compiler framework and the GeLIR simulator including the energy cost model of the M3-DSP. In the next subsection, the impact of loop transformations on the code quality is shown for an example program. After that, results for several DSP benchmarks and an MP3 application are presented.

A. Impact of Loop Transformations

In this subsection we will demonstrate the impact of frequently applied loop transformations to the code quality for the following `original` example program.

```
for( i=0; i<1024; i++ )
  for( j=0; j<40; j++ ) {
    if( cnd<42 ) {
      y[i] += x[i]*a[j];
      sum += z[i];
    }
    else
      y[i] += x[i]*b[j];
  }
```

We have applied the loop transformations *unswitching*, *interchange*, *split*, *reduction recognition* in this order. After applying a specific loop transformation we have compiled the generated source code by our compiler. The simulation results are presented in figure 2 and are all relative to the `original` program code (100%).

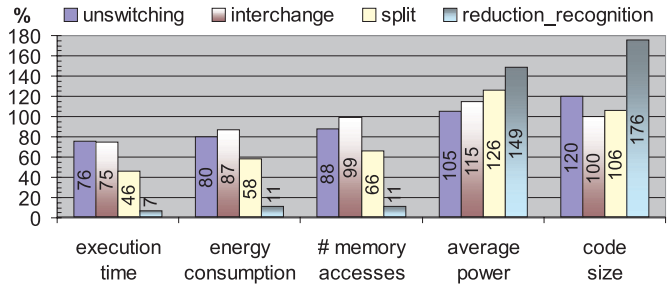

Fig. 2. Loop transformation (100% $\widehat{=}$ `original`)

The application of all loop transformations in total results in a reduction of the execution time by 93%, the energy consumption and the number of memory accesses by 89% compared to the original program. Due to the forced execution of energy-intensive SIMD operations the average power consumption is increased by 49%. The increase of code size by 76% for this program seems to be high, but usually only small program parts (mainly innermost loops) are subject to loop transformations. Otherwise a trade-off between performance/energy optimization and code size optimization has to be found.

B. Impact of SIMD Optimizations

In the following, results are presented for eight DSP routines and an MP3 application. The CPU time requirements of

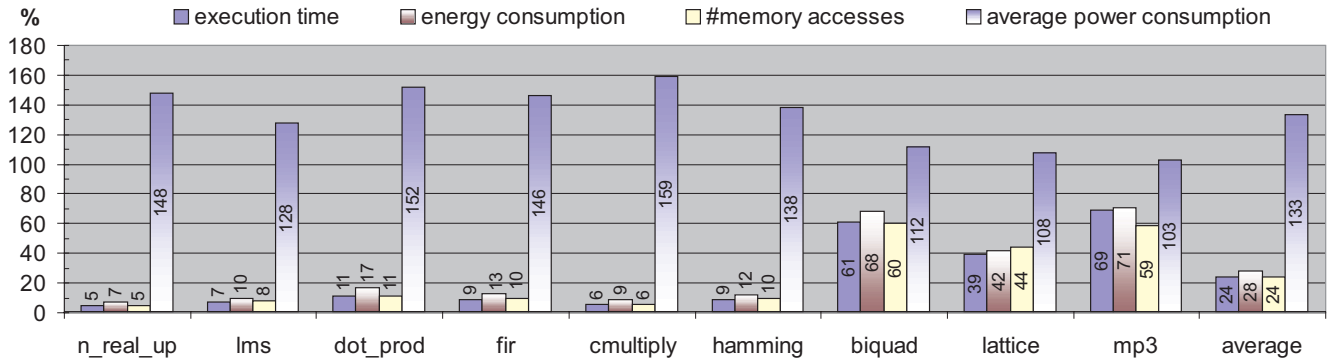

Fig. 3. Results for SIMD optimization: (100% $\widehat{=}$ SISD-mode)

our code generator in SISD-mode on a 2,66 GHz Intel Pentium 4-processor ranges between 23 seconds for the *n_real_update* routine and 1448 seconds for the MP3 application. The compilation times for the SIMD-mode are similar. Note that in most cases the best result is already found early in the optimization process. For this reason, the runtime of our code generator could be drastically reduced in most cases without code quality losings.

In figure 3, results are presented in terms of execution time, energy consumption, number of memory accesses and average power consumption. In order to show the effect of using SIMD operations, all results are related to results not using SIMD optimization.

The results show that the execution time is reduced between 31% for `mp3` and 95% for `n_real_up`. On the average, the number of execution cycles is reduced by 76% for all benchmarks.

Although an SIMD operation consumes 4-5 times more energy compared to the corresponding SISD operation, the energy consumption is reduced between 29% for `mp3` and 93% for `n_real_up`. On the average the energy consumption can be reduced by 72%, because our vectorization technique makes effective use of the 16 data paths by SIMD operations. The energy reduction is almost as large as the execution time. The number of energy-intensive memory accesses is reduced drastically.

Results concerning the power consumption confirm the previously made observations. As expected, the increased use of SIMD operations leads also to a higher average power consumption. For example the results for the `cmultiply` benchmark show that the average power consumption is only 59% higher although the execution time is reduced by 94%.

VI. Conclusions

The growing use of digital signal processors in embedded systems necessitates the use of optimizing compilers supporting the special architecture features. In order to improve performance of these systems, SIMD functionality is frequently involved. In this paper, the influence of compiler generated SIMD operations to energy consumption is investigated for the first time. This is done by using a generic compilation framework including an instruction level energy model. Results for our target architecture have shown that energy can nearly be reduced to the same extent as execution time, although the execution of an SIMD operation of the target architecture consumes 4-5 times more energy compared to the corresponding SISD operation. Making use of SIMD operations leads to an average reduction of 72% in terms of energy and 76% in terms of performance. Due to the generic implementation of our tools, the exploration of other SIMD architectures w.r.t. energy consumption can be done in a similar way.

References

[1] J.M. Rabaey and M. Pedram, editors. *Low Power Design Methodologies*. Kluwer Academic Publishers, 1996.

[2] P. Op de Beeck, C. Ghez, E. Brockmeyer, M. Miranda, F. Catthoor, and G. Deconinck. Low-Power Implementation of an OFDM Based Channel Receiver in Real-Time Using a Low-End Media Processor. In *Proc. of Workshop on Wireless Communications and Networking (WCN)*, 2002.

[3] D.J. DeVries. *A Vectorizing SUIF Compiler*. PhD thesis, University of Toronto, June 1997.

[4] N. Sreraman and R. Govindarajan. A Vectorizing Compiler for Multimedia Extensions. *Intern. Journal of Parallel Programming*, 28(4), 2000.

[5] S. Larsen and S. Amarasinghe. Exploiting Superword Level Parallelism with Multimedia Instruction Sets. In *Proc. of PLDI*, 2000.

[6] R. Leupers. Code Selection for Media Processors with SIMD Instructions. In *Proc. of DATE*, 2000.

[7] D.F. Bacon, S.L. Graham, and O.J. Sharp. Compiler Transformations for High-Performance Computing. *ACM Computing Surveys*, 26(4), 1994.

[8] V. Tiwari, S. Malik, and A. Wolfe. Power Analysis of Embedded Software: A First Step towards Software Power Minimization. In *IEEE Transactions on VLSI Systems*, 1994.

[9] H. S. Kim, N. Vijaykrishnan, M. Kandemir, and M. J. Irwin. A Framework for Energy Estimation of VLIW Architecture. In *Proc. of ICCD*, 2001.

[10] A. Bona, M. Sami, D. Sciuto, C. Silvano, V. Zaccaria, and R. Zafalon. Energy Estimation and Optimization of Embedded VLIW Processors based on Instruction Clustering. In *Proc. of DAC*, 2002.

[11] S. Steinke, N. Grunwald, L. Wehmeyer, R. Banakar, M. Balakrishnan, and P. Marwedel. Reducing Energy Consumption by Dynamic Copying of Instructions onto Onchip Memory. In *Proc. of ISSS*, 2002.

[12] M. Lorenz, L. Wehmeyer, T. Dräger, and R. Leupers. Energy aware Compilation for DSPs with SIMD Instructions. In *Proc. of LCTES/SCOPES*, 2002.

[13] G. Fettweis, M. Weiss, W. Drescher, U. Walther, F. Engel, and S. Kobayashi. Breaking new grounds over 3000 MOPS: A broadband mobile multimedia modem DSP. In *Proc. of ICSPAT*, 1998.

[14] T. Dräger and G. Fettweis. Energy Savings with Appropriate Interconnection Networks in Parallel DSP. In *Proc. of the DFG-Workshop "Grundlagen und Verfahren verlustarmer Informationsverarbeitung (VIVA)"*, 2002.

[15] GeLIR. http://ls12-www.cs.uni-dortmund.de/research/gelir/.

[16] H. Zima. *Supercompilers for Parallel and Vector Computers*. ACM Press, 1990.