VLIW Operation Refinement for Reducing Energy Consumption

Ulrich Hirnschrott and Andreas Krall
CD-Lab Compilation Techniques for Embedded Processors
Technische Universität Wien
Argentinierstraße 8,
A-1040 Wien, Austria
Email: {uli,andi}@complang.tuwien.ac.at
Telephone: +43 1 58801 {58520,18511}
Fax: +43 1 58801 18598

***Abstract*— The demand for mobile computer power has exploded in the recent years. Variable length VLIW processors offer the necessary performance at low power. Software optimizations are necessary to further decrease the energy consumption. In this article we present a compiler optimization which reduces the dynamic power dissipation resulting from the switching activities during instruction fetch.**

Energy consumption can be reduced by minimizing the Hamming distance between successively fetched instruction words. Using a dynamic programming approach we first compute a set of optimal instruction arrangements of the execution bundles in a basic block. These sets are used in an enumerative optimal algorithm and a genetic evolution, in order to minimize an objective function for the global Hamming distance. We evaluated our algorithms on different variable length VLIW architectures with 3 to 6 parallel functional units. On a large set of DSP benchmark programs the Hamming distance can be reduced by about 10% on average. Maximum reductions range up to 30%.

I. Introduction

As mobile computing is booming, reducing the energy consumption of applications has gained importance. Energy consumption can be reduced both by hardware and software techniques. This work is focused on energy reducing compilation techniques.

In digital CMOS circuits the dominant part of energy consumption is the switching activity [1]. Therefore, many techniques have been developed to reduce the amount of switching activity. Gray coding has been applied to reduce the number of bit switches in linear addressing and instruction scheduling has been used to reduce the number of bit switches during instruction fetch [2].

We propose an optimal operation refinement algorithm for variable length VLIW architectures. Related work is discussed in section II. The machine model is presented in section III-A and optimality is defined in section III-B. The global optimization algorithm is described in section IV. Finally we evaluate the algorithm for different VLIW architectures in section V.

II. Related Work

Tiwari et al. [3] give an overview of compilation techniques for reducing the energy consumption during program execution. Optimization methods which reduce the number of cycles, reduce the number of memory accesses, or increase locality also result in a reduction of energy consumption. Additional optimizations which reduce the switching activity on data or address buses reduce the energy consumption. Measurements of reorderings of instructions for a 486DX2 processor showed only differences in energy consumption below 2%.

Toburen et al. [4] implemented a list scheduling algorithm to limit the maximal energy consumption for the execution of an instruction group of an eight way VLIW architecture. After the dependence graph for a basic block is computed the list scheduler determines the ready list and begins scheduling of instructions based on the dependence height. When the energy threshold for an execution bundle is exceeded, the remaining instructions are scheduled in the next bundle. Experiments show that increasing the threshold from 10 to 12 nJ improves the run time for different benchmark programs from 1.3 to 10.2 %. Because of this unusual metric the results cannot be compared with any results from other work.

Parikh et al. [5] evaluated three different energy consumption aware scheduling algorithms for a scalar architecture. The first algorithm is list scheduling with the energy consumption as the first selection criterion. The second algorithm schedules instructions in reverse order. The third algorithm does a look-ahead of two levels. The algorithms have been evaluated with randomly generated dependence graphs. The three different algorithms perform similar regarding the energy consumption. The performance decrease is less than 6%. Three different algorithms which take performance as the first and energy as the second heuristic, energy as the first and performance as the second heuristic or the product of performance and energy as a heuristic behave similar, but do not have a performance penalty. The empirical evaluation is to artificial to be comparable with other work.

Lee et al. [6] presented a scheduling algorithm which does both a horizontal scheduling within an execution bundle and a vertical scheduling between bundles for a fixed length VLIW architecture. The horizontal scheduling is solved optimally using a multi-stage bipartite matching algorithm to minimize

the bus transition activities (the Hamming distance of the instruction encodings). Since vertical scheduling is NP-hard a heuristic algorithm based on weighted bipartite matching and allowable window for vertical scheduling is presented. Experimental results of horizontal scheduling show average 13.3% improvements with a 4-way issue architecture and average 20.15% improvements with an 8-way issue architecture. The additional enhancement from horizontal to vertical scheduling is 7.66% for 4-way issue and 10.55% for 8-way issue. The metric for the measurements in this work is the power consumption on the instruction bus.

Shin et al. [7] presented an algorithm which computes an optimal reordering of the operations within a VLIW instruction for a basic block. The Hamming distance between the instructions in a basic block is minimized. The problem is solved by transforming it into a shortest path problem and solving the shortest path problem. An empirical evaluation has been done on a variable length VLIW processor with eight functional units (256 bit VLIW length). The average reduction of switching activity on the instruction bus is 31.4%. Later Shin et al. [8] extended their algorithm to a heuristic global one. During transformation of the problem to a weighted graph, simplifications are necessary to limit computation and memory requirements. The global algorithm reduces switching activity by 34.3%.

Stouraitis [9] developed a post optimizer which reorders instructions and renames register operands for an ARM processor. The reordering is done using a list scheduler which uses energy reduction as the first heuristic. The effects have been measured with a wireless multimedia protocol on real hardware. The energy savings are 9.17%.

Choi and Chatterjee [10] solve the instruction scheduling for low-power problem as a precedence constrained Hamiltonian path problem for DAGs and the traveling salesman problem. Minimum spanning tree and simulated annealing are used for solving this NP-hard problem. For a five stage pipelined RISC processor energy savings between 2.68% and 29.19% are achieved.

III. Optimal Global Instruction Grouping

As former works have shown (see Section II), communication on core buses is a main contributor to the total energy consumption of a processor. Since instruction fetching is inherent in today's processor architectures, and instruction busses of VLIW architectures can become wide, there is a great potential in reducing the total system energy needs by reducing energy dissipation during instruction fetch.

The aim of our work is to refine VLIW operations in such a way, that fetching a function's instruction words from code memory needs as little energy as possible. As reported by others [6], [7], this goal can be reached by minimizing the Hamming distance of successively fetched words. Our method works as a post–pass optimization and has no runtime performance penalty on the compiled application.

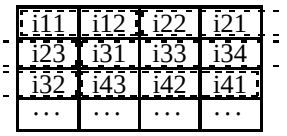

Fig. 1. Code image of variable length VLIW

TABLE I
NUMBER OF PARALLEL EXECUTION UNITS

	VLIW3	VLIW4c	VLIW4m	VLIW5	VLIW6
MOV	1	1	2	2	2
CMP	1	2	1	2	3
BR	1	1	1	1	1

A. Machine Model

The application area of our simulated architecture is digital signal processing. The raw processing power necessary for digital signal processing requires the exploitation of parallelism. Small code size is an important requirement and hardware costs shall be low. Therefore a variable length VLIW architecture was designed, where the order of operations within one VLIW is arbitrary. Figure 1 shows a possible code memory mapping of the following operation sequence.

$i_{11}|i_{13}$
$i_{21}|i_{22}|i_{24}$
$i_{31}|i_{32}|i_{33}|i_{34}$
$i_{41}|i_{42}|i_{43}$

(i_{xy} means instruction i is executed in cycle x at functional unit y)

Mind that there is an important difference between *fetch word* and *execution bundle*. A fetch word is the fixed amount of code memory that is fetched during one fetch cycle, depicted by one row of the code image in Figure 1. In opposite, an execution bundle is a varying number of operations which are executed in the same execution cycle. These are depicted by the dashed boxes in Figure 1.

For static evaluation of the improvements achieved by our refinement strategy, we modeled five different architectures. The maximum size for an execution bundle varies from 3 to 6 operations respectively, and each bundle can be built of operations from 3 different classes. These classes are:

- MOV (memory access)
- CMP (computation, ALU)
- BR (branches)

Operations are either coded as short operation (20 bit), or long operation (40 bit). A fetch word always consists exactly of 80 bit (e.g. four short operations). Table I shows these five architectural models.

B. Optimality

Equation 1 shows the objective function for a function's global Hamming distance which we used in our models.

$$Dist^{F}_{glob} = \sum_{b \epsilon B} f_b * \left(Dist^{int}_{b} + \sum_{s \epsilon S_b} p_{b \rightarrow s} * Dist^{ext}_{b \rightarrow s} \right) \quad (1)$$

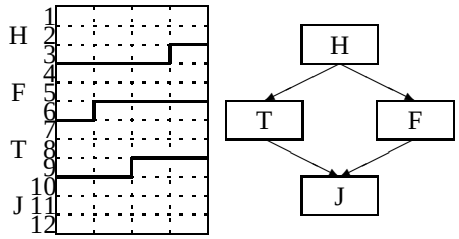

Fig. 2. Illustration of block alignment

where B is the set of F's basic blocks, f_b the execution frequency of block b, $Dist_b^{int}$ the internal Hamming distance of block b, S_b the successor blocks of b, $p_{b\to s}$ the probability of branching from b to s, and $Dist_{b\to s}^{ext}$ the so–called external Hamming distance of block b to block s. Execution frequencies and branch probabilities are estimated by the following heuristic functions:

$$f_b = 10^l \tag{2}$$

with l the loop nesting level of b, and

$$p_{b\to s} = \begin{cases} \frac{1}{|S_b|} & \text{for non–loop branches,} \\ 0.9 & \text{for the back–branch of a loop,} \\ 0.1 & \text{for the end–loop branch.} \end{cases} \tag{3}$$

The internal Hamming distance of a block b is calculated as follows:

$$Dist_b^{int} = \sum_{i=1}^{N_{FWord}^b - 1} Dist\left(FWord_i^b, FWord_{i+1}^b\right) \tag{4}$$

where N_{FWord}^b is the number of fetch words in block b, and $Dist\left(FWord_i^b, FWord_{i+1}^b\right)$ the Hamming distance of the ith and $(i+1)$th fetch words of block b.

Calculating the external Hamming distance of a block to its successors works mostly analogous. There is only a difference if the first or last execution bundles of a basic block are not aligned. This has to be taken into account correctly. Figure 2 depicts an example. For edge $H \to T$, the Hamming distance of fetch words 3 and 6 has to be calculated, For $H \to F$ 3 and 4, for $T \to J$ 9 and 10, and for $F \to J$ 6 and 9.

IV. Algorithm

The following refinement possibilities are taken into consideration:

- Permuting the operations of one execution bundle,
- Swapping operands of commutative operations.

Our optimization takes two steps. First a set of locally optimal operation refinements is computed. The second step then selects those local solutions that yield the minimal global Hamming distance as defined in Equation 1.

A. Local Optimization

In our model, the operations of one execution bundle are not bound to one fetch word, i.e. when choosing one reordering, an instruction might be placed into a different fetch word than it would be placed under another reordering. Therefore, the local reordering problem cannot be converted to a shortest path problem as proposed in [7]. We use an enumerative method with an optimization of runtime complexity by dynamic programming.

Actually, we have to enumerate all possible arrangements (including permutations and operand swapping) of each execution bundle's operations and find the combination which results in the minimal internal Hamming distance. This yields a not acceptable exponential runtime complexity. Our approach makes use of the circumstance, that only the first and the last fetch words of the blocks are needed for the global expansion step. So we only have to consider such combinations, that differ in their resulting first and last fetch words. All other combinations can be eliminated in each step of the dynamic program. This reduces runtime complexity so that the algorithm is reasonably fast for typical code and the used VLIW models. It also still yields the optimal result, because *all* [1] of the locally optimal solutions are generated and used in the global expansion.

B. Global Expansion

Local optimization yielded sets of local optima, i.e. operation arrangements with minimal Hamming distance for any possible combination of different first and last fetch words. Global expansion now tries to select one local optimum for each block in order to minimize the *global* Hamming distance as defined in Equation 1. Two different strategies were implemented The first one is a total recursive enumeration of all possible combinations of local optima, whereas the second strategy implements a genetic evolution for the global expansion step.

1) Total Recursive Enumeration: This strategy was meant to be a reference implementation that always finds the optimal solution of the global expansion. It loops through all the basic blocks and their corresponding local optima recursively. Every time the last block of the function is reached, the global Hamming distance is evaluated. If it is smaller than the currently known best solution, then the solution is memorized, otherwise it is discarded. This is continued until all combinations are checked.

2) Genetic Evolution: An individual of the population represents one possible combination of local optima. For each basic block the index of the chosen optimum is stored in an array. Crossover between two individuals is done by interchanging parts of two arrays and is performed during reproduction of a generation. The crossover points are chosen randomly. Mutation is done by changing one of the indices to another valid index and is done for a fixed ratio of randomly chosen genes. The fitness function of the individuals is defined by the global

[1] i.e. all combinations of different first and last fetch words ("borders")

TABLE II

GEOMETRIC MEANS OF IMPROVEMENTS

Model	LOC	GEN	TRE
VLIW3	6.1 %	7.7 %	7.8 %
VLIW4c	6.1 %	7.8 %	7.9 %
VLIW4m	7.3 %	10.0 %	9.7 %
VLIW5	7.6 %	10.4 %	9.8 %
VLIW6	8.3 %	11.1 %	10.4 %

TABLE III

OPTIMALITY OF GENETIC EVOLUTION

Model	sub–optimal	optimal
VLIW3	63	160
VLIW4c	68	144
VLIW4m	66	126
VLIW5	64	117
VLIW6	52	125

Hamming distance. Since we want to do minimization, we have to search for individuals with the least fitness. The size of the population depends linearly on the number of possible incarnations, and is currently bound to a maximum of 1000 (in case of more than 1.000.000 possibilities). The first population is generated randomly, but includes an individual made of the *absolute* local optima. Parents for the new generation are selected randomly within the fittest 50% of the population. The individual with best fitness survives unchanged (cloned reproduction). If evolution stagnates [2] for a certain number of generations, then we consider the global minimum to be found and stop the evolution.

V. EMPIRICAL EVALUATION

Static evaluation covered a total of 239 functions and was done for all different architecture models. An *enhanced full rate* coder application contributes 95 functions, 16 functions are taken from the DSP kernels of the DSPstone benchmark suite, the rest of our benchmarks are various vector operations, digital filters, algorithms like quicksort, bubblesort, md5, compress, etc.

The experiments started with an evaluation of each function's unoptimized Hamming distance. In the next step we applied the local optimization algorithm. Choosing the overall minimum out of the optima for each basic block yields the values for a only locally optimized Hamming distance without taking inter–block effects into account. After local optimization we applied the genetic evolution for global optimization. At last a total recursive enumeration for solving the global problem was done. Functions with more than a million different combinations of local optima were not enumerated.

Table II presents the geometric means of reduction of the global Hamming distance for all models. Table III shows a comparison of non–optimal and optimal solutions reached by genetic evolution. Table IV shows how many of the not enumerated functions were improved by the genetic evolution.

[2] fitness does not increase

TABLE IV

BENEFITS OF GENETIC EVOLUTION FOR NOT ENUMERATED FUNCTIONS

	Impact on unoptimized global HD	
Model	improved	no change
VLIW3	16	0
VLIW4c	27	0
VLIW4m	47	0
VLIW5	58	0
VLIW6	62	0

VI. CONCLUSION

In this work, we aim at operation refinement for variable VLIW processors in order to minimize energy consumption during instruction fetch. We proposed an optimal exhaustive algorithm and a genetic evolution for solving this problem. These algorithms are more general than other work since they can handle instruction bundles which cross a fetch word and also include operand swapping. We presented an extensive static experimental evaluation with a large set of benchmark programs for 5 different VLIW architectures. The Hamming distance can be reduced by about 10% on average, single functions show reductions of the global Hamming distance of up to 40%.

REFERENCES

[1] A. P. Chandrakasan, S. Sheng, and R. W. Brodersen, "Low power CMOS digital design," *IEEE Journal of Solid State Circuits*, vol. 27, no. 4, pp. 473–484, 1992.

[2] C.-L. Su, C.-Y. Tsui, and A. M. Despain, "Low power architecture design and compilation techniques for high-performance processors," in *Proceedings of IEEE CompCon'94*. IEEE, April 1994, pp. 489–498.

[3] V. Tiwari, S. Malik, and A. Wolfe, "Compilation techniques for low energy: An overview," in *Proceedings of the 1994 IEEE Symposium on Low Power Electronics*. IEEE, October 1994.

[4] M. C. Toburen, T. M. Conte, and M. Reilly, "Instruction scheduling for low power dissipation in high performance processors," in *Proceedings of the Power Driven Micro-architecture Workshop at ISCA'98*. ACM, June 1998.

[5] A. Parikh, M. T. Kandemir, N. Vijaykrishnan, and M. J. Irwin, "Instruction scheduling based on energy and performance constraints," in *Annual Workshop on VLSI (WVLSI'00)*. IEEE, 2000.

[6] C. Lee, J. K. Lee, and T. Hwang, "Compiler Optimization on Instruction Scheduling for Low Power," in *Proceedings of the 13th conference on International Symposium on System Synthesis*. ACM Press, 2000, pp. 55–60.

[7] D. Shin and J. Kim, "An operation rearrangement technique for low-power VLIW instruction fetch," in *Proceedings of Workshop on Complexity-Effective Design*, June 2000.

[8] D. Shin, J. Kim, and N. Chang, "An operation rearrangement technique for power optimization in VLIW instruction fetch," in *Proceedings of Design, Automation and Test in Europe, Date'01*. ACM, March 2001, pp. 809–817.

[9] T. Stouraitis, "Low Power Software Development for Embedded Applications," University of Patras, Tech. Rep., 2001.

[10] K. won Choi and A. Chatterjee, "Efficient instruction-level optimization methodology for low-power embedded systems," in *Proceedings of International Synmposium on System Synthesis ISSS 01*, October 2001, pp. 147–152.