OPTIMUM ADDRESS POINTER ASSIGNMENT FOR DIGITAL SIGNAL PROCESSORS

Bernhard Wess

Institute of Communications
and Radio-Frequency Engineering
Vienna University of Technology, Austria
bernhard.wess@nt.tuwien.ac.at

Thomas Zeitlhofer

Institute of Communications
and Radio-Frequency Engineering
Vienna University of Technology, Austria
thomas.zeitlhofer@nt.tuwien.ac.at

ABSTRACT

Generating optimum data memory layouts and address pointer assignments for digital signal processors are hard combinatorial optimization problems. In this paper, it is shown that for fixed memory layouts and in contrast to traditional heuristic approaches optimum address pointer assignments can be generated easily. The computational complexity depends exponentially just on the number of address pointers. The proposed technique is applied to a large benchmark suite. Experimental results for three address pointers show that optimum solutions can be generated in almost all cases (99.98%) within one second. Since a large number of address pointers may be intractable, an additional heuristic pruning technique with nearly optimum performance is proposed.

1. INTRODUCTION

Digital signal processors (DSPs) have dedicated address generation units (AGUs) that support address computation in parallel to data-path operations. AGUs support indirect addressing with address pointer updates by some fixed values without adversely affecting performance. Typical offset values for these zero-cost increment/decrement operations are ± 1. Some AGU architectures provide dedicated offset registers. Once such a register is initialized by a value m, address pointers can be updated by offset m at no extra cost. Minimizing addressing overhead requires to carefully place program variables in data memory and to look for optimized address pointer assignments. Unfortunately, these problems are NP-hard.

2. OFFSET ASSIGNMENT

Let V be a set of program variables. Each variable $v_i \in V$ is identified by index $i \in \{1, 2, \ldots, N\}$ with $N = |V|$. A variable access stream $S = [v_{s(1)}, v_{s(2)}, \ldots, v_{s(M)}]$ is defined by a function

$$s : \{1, 2, \ldots, M\} \rightarrow \{1, 2, \ldots, N\} \tag{1}$$

where M denotes the stream length and $N \leq M$. The image $s(\ell)$ of any $\ell \in \{1, 2, \ldots, M\}$ defines the program variable $v_{s(\ell)}$ on position ℓ in the access stream S.

A memory layout is a permutation

$$\pi : \{1, 2, \ldots, N\} \rightarrow \{1, 2, \ldots, N\} \tag{2}$$

that assigns addresses to all program variables which appear in the access stream S. Let us assume that an address register points to v_i and it should be used for accessing v_j. To this end, the address pointer has to be modified by address offset $\pi(j) - \pi(i)$. AGUs of DSPs support zero-cost address pointer updates for a limited set of offset values. Obviously, costs can be minimized by memory layout optimization. This problem is denoted as offset assignment (OA). OA for a single address pointer with zero-cost update range ± 1 is called simple offset assignment (SOA). Let c_{ij} with $i, j \in \{1, 2, \ldots, N\}$ be a cost value for redirecting an address pointer from address i to the new address j. For SOA, c_{ij} is defined by

$$c_{ij} = \begin{cases} 0 & \text{for } |j - i| \leq 1, \\ 1 & \text{otherwise.} \end{cases} \tag{3}$$

Additionally, let t_{ij}^S specify how often address j is accessed right after address i in S. For a single address pointer, the address computation costs can be expressed by

$$C_1 = \sum_{i=1}^{N} \sum_{j=1}^{N} c_{ij} t_{\pi(i)\pi(j)}^S. \tag{4}$$

Modifying π such that C_1 becomes a minimum in Equ. 4 is a quadratic assignment problem (QAP) [1]. Since it is NP-hard, optimum solutions can be found just for small values of N. An efficient SOA heuristic has been proposed

in [2] that looks for a minimum-weighted path in the access graph defined by the access stream S. Meanwhile several improvements have been proposed. As shown in [3], SOA combined with variable coalescing allows to produce best results compared to currently existing SOA techniques. An efficient OA algorithm for zero-cost offset range ± 2 is discussed in [4] that produces an optimum memory layout if there is a $C_1 = 0$ solution, otherwise a layout is generated with minimized C_1.

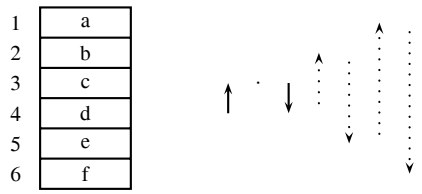

Fig. 1. a) Memory layout for $V = \{a, b, c, d, e, f\}$, b) address offsets for access stream $S = [d, c, c, d, b, e, a, f]$.

3. GENERAL OFFSET ASSIGNMENT

General offset assignment (GOA) is the problem of optimizing the memory layout for multiple address pointers. In case of GOA, both memory layout and address pointer assignment are optimized. For N program variables $N!$ different memory layouts exist. Assigning address pointers can be regarded as K-coloring the access stream S. For K homogeneous address pointers, an optimum solution has to be found out of K^{M-1} different colorings by simple exhaustive search. A coloring of S is a partition that decomposes $\{1, 2, \ldots, M\}$ into K disjoint subsets. Each subset defines an access stream

$$S_k = [v_{s_k(1)}, v_{s_k(2)}, \ldots, v_{s_k(M_k)}] \tag{5}$$

with

$$s_k : \{1, 2, \ldots, M_k\} \rightarrow \{1, 2, \ldots, M\} \text{ and } \sum_{k=1}^{K} M_k = M.$$

For each S_k, all elements appear in the same relative order as in S.

We calculate the overall address computation costs by

$$C_K = \sum_{k=1}^{K} \sum_{i=1}^{N} \sum_{j=1}^{N} c_{ij} t^{S_k}_{\pi(i)\pi(j)}. \tag{6}$$

The goal is to minimize C_K by optimizing the memory layout π and the partitioning of S. This optimization problem consists of K interdependent QAPs with a solution space size of $N!K^{M-1}$ which is far too big to be practicable even for small numbers of K, N, and M. In [5], simulated annealing (SA) is applied for solution space exploration. Heuristically reducing GOA to K SOA problems by decomposing variable set V into disjoint subsets are proposed in [2, 6]. These techniques are restricted to the offset cost function in Equ. 3.

4. OPTIMUM ADDRESS POINTER ASSIGNMENT

GOA consists of memory layout generation in conjunction with address pointer assignment (APA). For the following discussion we assume that the memory layout π has been optimized in a first step. Now our goal is to generate an optimum APA for a fixed layout π. We define the total addressing costs C as

$$C = I + (K + 1)\, C_K \tag{7}$$

where I are the initialization costs. We add 1 to C if a new pointer is initialized. C_K represents the number of address pointer reloads. The definition C in Equ. 7 ensures that a minimum number of address pointers and a minimum number of reloads are required.

We propose an algorithm that takes as input the variable access stream ,a fixed memory layout, and produces an optimum APA. This algorithm builds an assignment tree where all nodes at the same level correspond to different address pointer settings at some point in the access stream. Each path in the tree from the root to a leaf is a mapping of memory accesses to address pointers.

Fig, 2 shows the first three levels of the assignment tree for the variable access stream $S = [d, c, c, d, b, e, a, f]$ with $K = 2$ pointers and memory layout π that is shown in Fig, 1. Let us assume that c_{ij} is defined by Equ. 3. The

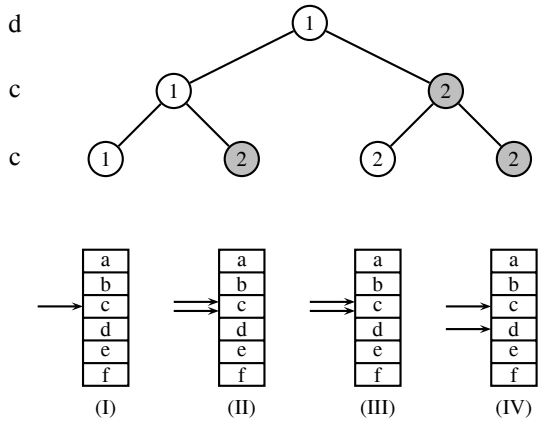

Fig. 2. Assignment tree pruning.

pointer assignment is indicated by the node color and the numbers inside the nodes are the total addressing costs defined by Equ. 7. For homogeneous address registers, the pointer settings (II) and (III) in Fig, 2 are equivalent. Within a group of equivalent nodes, only one lowest-cost node needs to be kept when continuing with the assignment tree construction. Ties are broken arbitrarily. The complete assignment tree for access stream $S = [d, c, c, d, b, e, a, f]$ with $K = 2$ pointers and memory layout π in Fig, 1 is shown in Fig, 3.

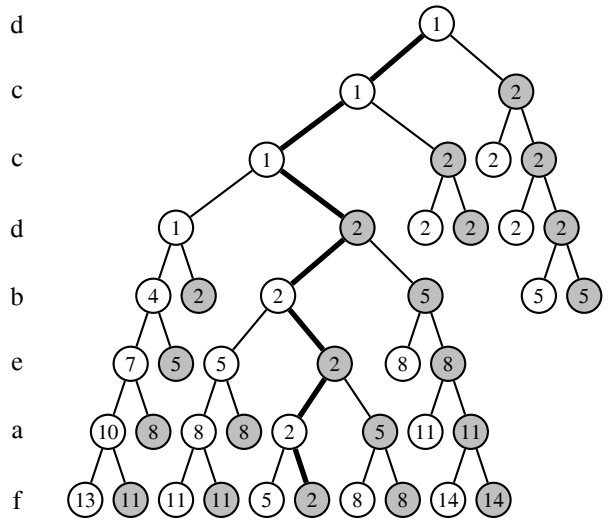

Fig. 3. Optimum assignment for two address pointers indicated by a minimum weighted path.

The coloring of the nodes on the path from the root to the lowest cost node yields the optimum address pointer assignment with

$$C_2 = \lfloor C/(K+1) \rfloor = 0$$

pointer reloads and

$$I = C \bmod (K+1) = 2$$

pointer initializations. The variable access stream S is partitioned into the sub-streams

$$S_1 = [d, c, c, b, a] \text{ and } S_2 = [d, e, f].$$

Note that in contrast to S, all address offsets in layout π are in the zero-cost range both for S_1 and S_2.

The maximum width w of the assignment tree is bounded by

$$w \leq \binom{N+K}{K} - 1 < N^K. \tag{8}$$

To verify this, consider all nodes at the same tree level, that is, at a given $\tilde{\ell}$ in the access stream. Each node corresponds to a specific address pointer setting where an address pointer is either directed to a program variable or is uninitialized ($N + 1$ possible assignments). The costs for succeeding accesses $\ell > \tilde{\ell}$ just depend on the contents of the address pointers at access $\tilde{\ell}$. Since we assume a homogeneous address pointer file, any two tree nodes are equivalent if their corresponding address pointer settings are equal except for permutations. So for K address pointers, the number of nodes that have to be distinguished is given by the number of K-combinations with repetition of $N + 1$ elements. Because we start with one pointer initialized, the combination of all pointers uninitialized does not appear as shown in Equ. 8.

A comparison of computational complexity for simple exhaustive search (Sec. 3) and our proposed approach is given in Fig, 4. The ratio $\frac{M}{N} = 1.52$ corresponds to the average ratio found in the *OffsetStone* benchmarks [7].

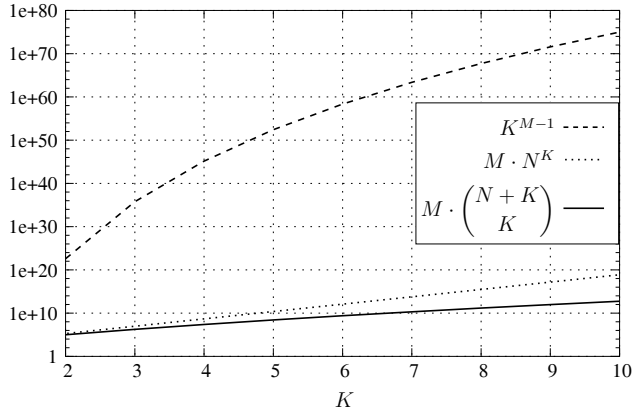

Fig. 4. Computational complexity bounds according to Equ. 8 for an access stream of length $M = 1.52 \cdot N$ and $N = 50$ variables.

5. EXPERIMENTAL RESULTS

We applied our optimum approach to all (55298) sequences of the *OffsetStone* benchmarks [7]. For $K = 3$ address pointers, we are able to compute optimum solutions for *all* sequences but one. The seven most complex examples are shown in Tab. 1. The computation of optimum address pointer assignments takes one second or less for each of the other examples. Note, that access streams as given in Tab. 1 are *not* typical as 99.98% of the *OffsetStone* examples are of less complexity. We just present the most complex examples to demonstrate the limits of our approach.

The computational complexity is mainly determined by the number of available pointers (Fig, 4). For a larger number of pointers (≥ 8) we found that access streams with

N	M	time [s][1]
94	189	5
161	322	31
243	506	130
388	776	461
432	864	564
441	873	675
678	3440	–[2]

Table 1. Most complex examples, $K = 3$.

$N \geq 25$ may become intractable. In this case we can apply a simple heuristic to achieve near-optimum results. The width of the search tree is pruned by considering only W nodes with minimum accumulated path costs.

N	M	opt.	$W = 100$		$W = 10$	
		C	C	time [s]	C	time [s]
94	189	187	187	< 1	187	< 1
161	322	1	1	1	1	< 1
243	506	83	83	1	127	< 1
388	776	267	267	1	267	< 1
432	864	91	91	1	91	< 1
441	873	91	91	2	91	1
678	3440	–	6867	3	7083	1

Table 2. Most complex examples, $K = 3$, heuristically pruned search tree width W.

The effect of heuristically pruning is shown in Tab. 2. Note, in these examples the memory layout has not been optimized prior to address pointer assignment. Therefore optimum cost values may be arbitrarily large. We applied rather strict bounds $W = 100$ and $W = 10$. Nevertheless, for $W = 100$ the algorithm still generates optimum solutions in all cases where the minimum total addressing costs are known. Even for $W = 10$ the total addressing costs are increased just in two cases.

6. CONCLUSIONS

We have presented an optimum approach for the address pointer assignment problem. This allows to find address pointer assignments with minimum cost given a fixed memory layout of program variables. An *optimum pruning* technique allows us to drastically reduce the search space compared to simple exhaustive search strategies. So in contrast to traditional approaches that apply heuristics, our new approach generates optimum solutions for typical problems. The proposed algorithm has been successfully applied to real-world examples from the *OffsetStone* benchmarks. For three address pointers all but one of these examples can be solved optimally. Additionally we presented a heuristic pruning technique that produces near-optimum results for problems of increased complexity.

[1] Experiments are computed on a PC with CPU frequency 1.4GHz.

[2] Tree size exceeded a reasonable maximum.

7. REFERENCES

[1] B. Wess and M. Gotschlich, "Optimal DSP memory layout generation as a quadratic assignment problem," in *Proc. IEEE Int. Symp. on Circuits and Systems*, Hong Kong, June 1997, vol. 3, pp. 1712–1715.

[2] S. Liao, S. Devadas, K. Keutzer, S. Tjiang, and A. Wang, "Storage assignment to decrease code size," in *Proc. ACM Conf. on Programming Language Design and Implementation*, June 1995, pp. 186–195.

[3] D. Ottoni, G. Ottoni, G. Araujo, and R. Leupers, "Improving offset assignment through simultaneous variable coalescing," in *Proc. 7th Int. Workshop on Software and Compilers for Embedded Systems*, Vienna, September 2003, pp. 285–297.

[4] B. Wess and M. Gotschlich, "Constructing memory layouts for address generation units supporting offset 2 access," in *Proc. IEEE Int. Conf. on Acoustics, Speech, and Signal Processing*, Munich, April 1997, vol. 1, pp. 683–686.

[5] B. Wess, "Minimization of data address computation overhead in DSP programs," *Kluwer Design Automation for Embedded Systems*, vol. 4, pp. 167–185, March 1999.

[6] R. Leupers and P. Marwedel, "Algorithms for address assignment in DSP code generation," in *Proc. IEEE Int. Conf. on Computer-Aided Design*, San Jose, November 1996, pp. 109–112.

[7] R. Leupers, "Offset assignment showdown: Evaluation of DSP address code optimization algorithms," in *12th International Conference on Compiler Construction (CC)*, Warsaw (Poland), April 2003, Springer Lecture Notes on Computer Science, LNCS 2622, http://www.address-code-optimization.org.