CONSTRUCTING MEMORY LAYOUTS FOR ADDRESS GENERATION UNITS SUPPORTING OFFSET 2 ACCESS

Bernhard Wess and Martin Gotschlich

Institut für Nachrichtentechnik und Hochfrequenztechnik
Technische Universität Wien
Gusshausstrasse 25/389, A-1040 Vienna, Austria
bwess@email.tuwien.ac.at

ABSTRACT

We present an efficient memory layout generation algorithm for digital signal processors (DSPs) which takes advantage of indirect addressing modes with auto-modify operations. Previously proposed algorithms are optimized with respect to offset 1 access (auto-increment and decrement by 1). Our algorithm is based on a heuristic since the problem of generating optimum memory layouts is NP-complete. However, this algorithm produces optimum results if a bandwidth 2 layout exists for a given program variable access sequence. It is verified by experimental results that our technique achieves significant improvements over existing techniques.

1. INTRODUCTION

Modern digital signal processors (DSPs) provide dedicated memory address generation units (AGUs) supporting address computation in parallel to other machine operations. We focus on optimized memory layouts for AGUs supporting offset 2 access. Here it is assumed that memory locations can be referenced at no extra cost if the absolute value of the difference between the current and the next address is lower or equal 2. As an example, Motorola's DSP56k [1] and Analog Devices' ADSP-21xx [2] support offset 2 access. In case of the DSP56k, there is one modify register (designated as offset register N) associated with each address register. The address generation hardware allows address register updates by ± 1 or $\pm$N. The ADSP-21xx family provides four modify registers for each address register containing signed update values. For offset 2 access, the contents of the modify registers are -2, -1, 1, and 2.

In Section 2, we define the *address assignment problem* (AAP) and relate it to the *bandwidth minimization problem* (BMP). In Section 3, we discuss our heuristic offset 2 memory layout generation algorithm which produces optimum results if bandwidth 2 layouts exist. Section 4 presents experimental results, and conclusions are given in Section 5.

2. OPTIMUM ADDRESS ASSIGNMENT

Let V be the program variable set of an *access sequence* S. We define an undirected graph $G = (V, E)$ to represent the access transitions between program variables in S and call G the *access graph* of S. Each node in the graph corresponds to a unique program variable. For the rest of the paper, we use the notation $v \in V$ both for program variables of S and nodes of G. There is an undirected edge $e = (u, v) \in E$ in G with weight $w(e)$ if the program variables u and v are adjacent $w(e)$ times in S. Note that G is always a connected graph.

A *memory layout* of G is an injective function

$$f : V \rightarrow \{1, \ldots, N\} \text{ with } N \geq |V|$$

which assigns addresses to program variables. Equivalently, a layout can be regarded as a string φ of nodes and blanks with each node of V appearing exactly once. The correspondence between these two definitions is simply that $f(v) = k$ with $v \in V$ if and only if v is the k^{th} element of φ.

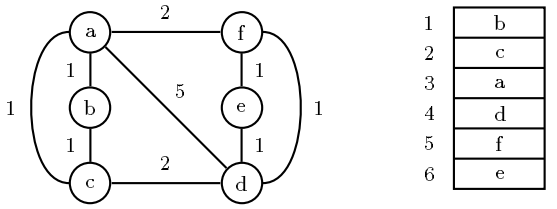
As an example, consider the program variable set $V = \{a, b, c, d, e, f\}$ which is accessed in the sequence

$$S = \{a, b, c, d, e, f, a, d, a, d, a, c, d, f, a, d\}.$$

Figure 1a shows the corresponding access graph G. Figure 1b defines a memory layout which can also be represented by the string

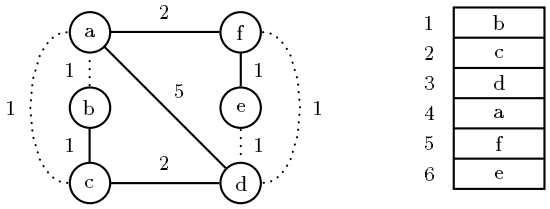
$$\varphi = bcadfe.$$

For the following discussion, we assume that the target processor provides an AGU with dedicated registers. After an indirect data access, the current address



(a) (b)

Figure 1: (a) Access graph, (b) memory layout.



(a) (b)

Figure 2: (a) Maximum-weighted Hamiltonian path, (b) memory layout.

register can be incremented by ± 1 or by the contents of a modify register. Operations of this type do not employ datapath resources and thus can be executed in parallel to other machine operations at no extra cost. In contrast, explicit address register and modify register load operations introduce both code size and speed overhead. Consequently, memory layouts should be constructed such that these load operations are minimized.

Suppose the AGU supports an auto-modify range $[-r, r]$ where r is any positive integer. Let S be an access sequence, $G = (V, E)$ the corresponding access graph, and f a memory layout for S. We define the memory layout cost as the sum of all access transitions where the absolute value of the difference between the current and the next address is larger than r,

$$c(f) = \sum_i w(e_i) \text{ with } e_i = (u, v) \in E,\ |f(u) - f(v)| > r.$$

Generating memory layouts for AGUs with $r = 1$ and a single address register (*simple offset assignment, SOA*) was first studied by Bartley [3] and Liao [4]. Liao showed that SOA is equivalent to searching for maximum-weighted Hamiltonian paths in the access graph.

As an example, Figure 2a shows the maximum-weighted Hamiltonian path for the access sequence defined above. Traversing this path gives the optimum memory layout f (Figure 2b) while the accumulated weights of edges not on the path define the layout cost $c(f) = 4$.

Liao showed that the SOA problem is NP-complete and proposed a heuristic algorithm. Leupers [5] refined Liao's SOA heuristic saving 3% addressing cost on average.

Additionally, Leupers proposed a technique for dynamically loading modify registers. This is done in a post-pass optimization step after memory layout generation. However, only for variable access sequences of length larger than 100, this technique results in significant cost reductions.

We propose to realize a larger auto-modify range by assigning static values to modify registers. In this case, the number of address offsets outside the specified auto-modify range should be minimized. For our access sequence, Figure 1b represents an optimum memory layout for auto-modify range $[-2, 2]$. Note that this layout does not correspond to a maximum-weighted Hamiltonian path in the access graph.

Now let us relate the AAP to the BMP of graphs. The bandwidth bw of f is defined as the maximum distance between the images under f of any two nodes that are connected by an edge,

$$bw(f) = \max\{|f(u) - f(v)| \mid (u, v) \in E\}.$$

Figure 1b shows a bandwidth 2 layout while Figure 2b gives an example for a layout with bandwidth 3. The bandwidth BW of G is defined as the least possible bandwidth for any layout of G,

$$BW(G) = \min\{bw(f) \mid f \text{ is a layout of } G\}.$$

As an example, $BW(G) = 2$ for our access graph in Figure 1a.

In contrast to the BMP, the AAP is not to find $BW(G)$ but to produce a layout f for a given modify range $[-r, r]$ such that the layout cost $c(f)$ becomes minimal. Obviously, if $BW(G) \leq r$ then all layouts f with $BW(G) \leq bw(f) \leq r$ lead to $c = 0$.

The general BMP of graphs is NP-complete [6]. However, Saxe [7] showed that the problem $BW(G) \stackrel{?}{\leq} k$ for some fixed constant k can be solved in polynomial time. Particularly, a linear-time algorithm exists for the problem $BW(G) \stackrel{?}{\leq} 2$ [8].

We say f is an *offset r layout* if it is optimized for modify range $[-r, r]$. Obviously, all layouts f with $bw(f) \leq r$ are offset r layouts. As mentioned above, generating optimum memory layouts is an NP-complete problem even for the specific case $r = 1$. In the next section, we discuss a heuristic algorithm which generates offset 2 layouts. Usually, such a modify range can be realized by assigning static values to modify registers in contemporary DSPs. Our algorithm produces

optimum memory layouts if $BW(G) \leq 2$ otherwise a heuristic is applied minimizing the layout cost.

3. OFFSET 2 LAYOUT GENERATION ALGORITHM

Our DSP memory layout generation algorithm is based on Garey's algorithm [8] for the problem $BW(G) \stackrel{?}{\leq} 2$. It starts with an *initial layout* consisting of a single program variable (node of the access graph). The algorithm recursively constructs a *complete layout* by adding program variables to the current *partial layout*.

A partial layout is an injective function f' that is defined only on a subset of the nodes,

$$f' : V' \rightarrow \{1, \ldots, N'\} \text{ with } V' \subset V \text{ and } N' \geq |V'|.$$

If f_1 is a partial layout defined on V_1 and f_2 is a partial layout on $V_2 \supseteq V_1$, we say that f_2 is an *extension* of f_1 if $f_2(v) = f_1(v)$ for all $v \in V_1$. Garey's algorithm constructs a complete layout if there is one containing the initial layout. It terminates as soon as it detects that the current partial layout cannot be extended to a complete layout.

We say $u \in V'$ is an *active node* of the partial layout f' if $(u, v) \in E$ with $v \in V \setminus V'$. The set of *successors* of an active node u is defined by

$$Q(u) = \{v | (u, v) \in E \text{ with } v \in V \setminus V'\}.$$

For the number of successors of an active node, we use the notation $n(u) = |Q(u)|$. If $A = \{u_1, \ldots, u_K\}$ is the set of active nodes, then the set of all successors is given by $B = \bigcup_{i=1}^{K} Q(u_i)$.

Garey's algorithm is based on an exhaustive list of actions for the different circumstances which can arise in the process of extending partial layouts. There are three types of partial layouts defined by a string φ:

(A) $\varphi = \alpha a b$ where at most a and b are active.

(B) $\varphi = \alpha \langle a_m b_m \rangle \ldots \langle a_1 b_1 \rangle$ for some $m \geq 1$ where at most a_1 and b_1 are active.

(C) $\varphi = \alpha \_ a_m \_ \ldots \_ a_1$ for some $m \geq 1$ where at most a_1 is active.

α represents inactive nodes and blanks (_) which have already been permanently placed. Type B defines two strings $\alpha a_m b_m \ldots a_1 b_1$ and $\alpha b_m a_m \ldots b_1 a_1$.

Let f_n be a partial layout of one of the three types. By looking at how the active nodes interact with their successors, it may be obvious that f_n cannot be completed with bandwidth 2. Otherwise the algorithm will find a sufficiently general extension f_{n+1} which can be completed with bandwith 2 whenever f_n can be. f_{n+1} is again of one of the three basic types. If any suitable extension is found, the string φ_n corresponding to f_n is replaced by the string φ_{n+1} corresponding to f_{n+1}. This process continues until either reaching an impasse or a complete layout.

Here is how the algorithm would proceed to search for a layout for the access graph in Figure 1a with the initial layout $\_b$:

n	Type	φ_n	
0	C	$\_b$	initial layout
1	B	$b\langle ac \rangle$	
2	A	$bcadf$	
3	A	$bcadfe$	complete layout

Garey's algorithm determines in $\mathcal{O}(n)$ steps with $n = |V|$ whether or not the access graph $G = (V, E)$ has a bandwidth 2 layout beginning with the initial layout $\_v$ where $v \in V$. By investigating all possible initial layouts, we have an $\mathcal{O}(n^2)$ algorithm for deciding whether or not $BW(G) \leq 2$.

It is obvious that a partial layout f' cannot be extended to a complete layout f with $bw(f) \leq 2$ in the following cases:

- f' is of type A with $n(a) > 1$ or $n(b) > 2$.
- f' is of type B with $|Q(a) \cup Q(b)| > 2$ or $n(a) = n(b) = 2$.
- f' is of type C with $n(a) > 3$.

Additionally, there are more subtle cases where partial layouts of type C with $n(a) = 3$ or $n(a) = 2$ cannot be completed. We refer to [8] for a detailed discussion.

For generating optimized offset 2 layouts, we make a heuristic extension step each time Garey's algorithm would terminate. We extend a partial layout f_n to f_{n+1} by adding a node set $C \subseteq B$ to f_n such that $\sum_i w(e_i)$ becomes a maximum where $e_i = (u, v)$ with $|f_{n+1}(u) - f_{n+1}(v)| \leq 2$, $u \in A \cup C$, and $v \in C$. Here A denotes the active nodes of f_n and B the set of all successors.

In each heuristic extension step $\varphi_n \rightarrow \varphi_{n+1}$, at most two nodes are added at the right end of φ_n. These nodes are new active nodes in φ_{n+1}. All nodes of φ_n that are still active in φ_{n+1} contribute to the layout cost. If φ_n represents a layout of type C, then additional nodes may fill some blanks.

For the access graph in Figure 3a, there is no bandwidth 2 layout since node a has degree 5.

Beginning with initial layout $\_b$, the layout generation algorithm would produce the offset 2 layout $bcadfe$ with cost 1.

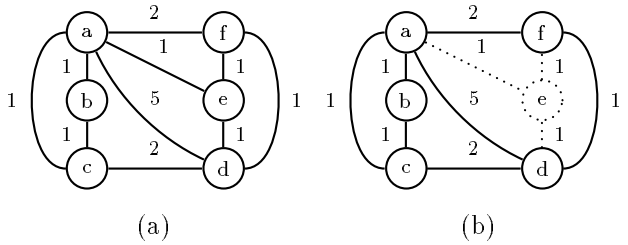


Figure 3: (a) Access graph, (b) heuristic extension of partial layout $\varphi_1 = b\langle ac\rangle$ to $\varphi_2 = bcadf$.

n	Type	φ_n	Cost	
0	C	$\_b$	0	initial layout
1	B	$b\langle ac\rangle$	0	heuristic extension
2	A	$bcadf$	1	
3	A	$bcadfe$	1	complete layout

φ_1 represents a partial layout of type B which is heuristically extended to φ_2 of type A. In this step, the set of active nodes is $A = \{a, c\}$ and the successor set is $B = \{d, e, f\}$. d and f become new active nodes in φ_2. a remains active because its successor node e has not been placed. Since node e cannot be placed within the modify range $[-2, 2]$ of a, the partial layout cost is increased by the weight of edge (a, e) (Figure 3b).

4. RESULTS

For unbiased comparison of techniques, we performed experiments on random access sequences. The random access sequences are defined by the number of program variables $|V|$ and the sequence length $|S|$. Table 1 summarizes the average addressing costs. On average, the proposed technique outperforms both Liao's and Leupers' SOA algorithm by 48% and 44%, respectively. Even for the largest example, the computation time is in the range of milliseconds on a Pentium PC.

$\|V\|$	$\|S\|$	Liao	Leupers	Proposed
5	10	2.2	2.1	0.1
5	20	5.4	5.3	0.8
15	20	5.1	4.7	1.9
10	50	23.1	22.7	14.0
40	50	13.6	12.1	7.2
10	100	51.0	50.7	39.1
50	100	52.5	47.9	39.6
80	100	28.0	24.6	18.4
100	200	110.9	101.0	86.0

Table 1: Comparison of layout generation algorithms.

5. CONCLUSIONS

We have related the AAP for DSPs to the BMP of graphs. These problems are known to be NP-complete. Contemporary DSPs support offset 2 layouts. For this case, we presented an efficient heuristic algorithm which produces optimum results if bandwidth 2 layouts exist. Experimental results show that significant improvements over existing techniques can be achieved.

6. ACKNOWLEDGEMENTS

The authors would like to thank Franz Rendl from TU Graz for helpful discussions. This work has been supported by ÖNB grant 5491.

7. REFERENCES

[1] Motorola, Inc., *DSP56000 Digital Signal Processor Family Manual*, 1992.

[2] Analog Devices, Inc., *ADSP-2100 Family User's Manual*, 1993.

[3] D. H. Bartley, "Optimizing stack frame accesses for processors with restricted addressing modes", *Software-Practice and Experience*, vol. 22, pp. 101–110, February 1992.

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

[5] R. Leupers and P. Marwedel, "Algorithms for address assignment in DSP code generation", *in Proceedings of the ACM/IEEE ICCAD'96*, pp. 109–112, San Jose, November 1996.

[6] C. H. Papadimitriou, "The NP-completeness of the bandwidth minimization problem", *Computing*, vol. 16, pp. 263–270, 1976.

[7] J. B. Saxe, "Dynamic-programming algorithms for recognizing small-bandwidth graphs in polynomial time", *SIAM J. Alg. Disc. Meth.*, vol. 1, pp. 363–369, December 1980.

[8] M. R. Garey, R. L. Graham, D. S. Johnson, and D. E. Knuth, "Complexity results for bandwidth minimization", *SIAM J. Appl. Math.*, vol. 34, pp. 477–495, May 1978.