論文 / 著書情報
Article / Book Information

Title	Memory Allocation Method for Indirect Addressing DSPs with ±2 Update Operations
Authors	Nakaba Kogure, NOBUHIKO SUGINO, Akinori Nishihara
Citation	IEICE Trans. Fundamentals., Vol. E81-A, No. 3, pp. 420-428
Pub. date	1998, 3
URL	http://search.ieice.org/
Copyright	(c) 1998 Institute of Electronics, Information and Communication Engineers

IEICE TRANS. FUNDAMENTALS, VOL. E81-A, NO. 3 MARCH 1998

PAPER Special Section of Selected Papers from the 10th Karuizawa Workshop on Circuits and Systems

Memory Allocation Method for Indirect Addressing DSPs with ±2 Update Operations

Nakaba KOGURE†, *Nonmember*, Nobuhiko SUGINO†, *and* Akinori NISHIHARA†, *Members*

SUMMARY Digital signal processors (DSPs) usually employ indirect addressing using an address register (AR) to indicate their memory addresses, which often introduces overhead codes in AR updates for next memory accesses. In this paper, AR update scheme is extended such that address can be efficiently modified by ±2 in addition to conventional ±1 updates. An automatic address allocation method of program variables for this new addressing model is presented. The method formulates program variables and AR modifications by a graph, and extracts a maximum chained triangle graph, which is accessed only by AR ±1 and ±2 operations, so that the estimated number of overhead codes is minimized. The proposed methods are applied to a DSP compiler, and memory allocations derived for several examples are compared with memory allocations by other methods.

key words: *DSP compiler, memory addressing, code optimization*

1. Introduction

Digital signal processors (DSPs) are widely used to implement various real-time applications, such as filters, FFTs, CODECs, MODEMs, etc. The programming of such real-time algorithms on DSPs, however, is cost and time consuming work; programmers are required to have enough knowledge of both the processor architecture and the processing algorithm to write an efficient (short in execution time) program/code.

In order to reduce the heavy programming load, software tools such as high-level languages and its compilers are very important. Many programming tools and compilers are presented by venders and researchers [1]–[5]. In Ref. [4], [5] DIMPL (Digital network IMPlementation Language) and its compiler have been proposed. In these compilers, an efficient program benefits from effective use of registers. Efficient access of memory in a program, however, is as important as effective use of registers in order to derive an efficient program.

In many DSPs, memory content is accessed indirectly through address registers (ARs). Although AR must be updated before memory access, simple AR update operations such as increment or decrement (AR ±1) operations can be concurrently performed in one instruction cycle besides data operations such as addition, subtraction and date movement. When AR update amount is beyond these simple AR operations, additional one instruction cycle is required for substituting memory address for AR (immediate AR load operation), which becomes overhead in a program. In order to derive an efficient program, reduction in the number of such immediate AR loads is very important.

In Refs. [6]–[11], for a DSP with these indirect memory addressing operations, methods to derive a memory access pattern with less overhead codes have been studied. These methods model program variables and AR operations by an access graph (AG). Liao [6] proposed a heuristic method to search a maximum-weighted Hamiltonian path in the AG. Leupers [7] presented AR ±1 and ±k operations, and gave a method to utilize additional registers. Wess [8] represented memory address allocation problem in a matrix form.

In this paper, a DSP with extended indirect memory addressing operations, where AR can be increased or decreased by one (AR ±1) or two (AR ±2) besides arithmetic operation or data movement in one instruction cycle, is assumed. Although additional hardware is required for these new addressing operations, further reduction in overhead codes is expected. For hardware implementation of AR ±2 operations, techniques in n-bit up-down counter is applicable, so that additional hardware amount is expected to be small.

For this new DSP model, sophisticated memory allocation is necessary to achieve overhead code reduction. Although the memory allocation method in Ref. [8] deals with more generalized memory addressing model, it is easily estimated that it requires large amount of computational period with a set of parameters. Meanwhile, this paper presents a heuristic method based on the graph representation and it gives an efficient memory allocation with less computational cost.

2. Memory Addressing

When reading a datum from memory or writing a datum into memory (in general, accessing a datum in memory), memory address corresponding to the datum must be pointed by some ways, which are called as memory addressing modes. In general purpose processors, several memory addressing modes such as immediate addressing, indirect addressing, indexed addressing mode, and so on, are provided, so that programmers easily implement various data structures. These addressing modes are also utilized in compilers for general purpose pro-

Manuscript received June 30, 1997.
Manuscript revised September 19, 1997.

†The authors are with Department of Physical Electronics, Faculty of Engineering, Tokyo Institute of Technology, Tokyo, 152–0033 Japan.

cessors. As for DSPs, memory addressing modes are rather inferior to general purpose processors in return for high performance in computation. Almost all the DSPs have indirect addressing modes. They have so called address registers (ARs) to point the memory addresses to be accessed.

In this paper, a DSP with the following indirect addressing mode is assumed. AR update range of this DSP is depicted in Fig. 1.

Memory addressing mode of a new DSP model

- Memory addresses are pointed by an address register (AR).
- In one instruction cycle, AR can be increased or decreased by one (±1) or two (±2) (hereafter AR ±1·±2) besides arithmetic operation or data movement.
- Every AR can be simultaneously updated by ±1 or ±2 in one instruction cycle, whether one of the ARs is refered for memory access or not. Updated AR is valid after this cycle (post-modification).
- An "AR load" operation, which directly substitutes a memory address to an AR, costs one instruction cycle only by itself.

The third feature shows an apparent difference from the DSP model used in [9]–[11]. When a code requires to read a datum from memory or to write a datum into memory, an AR must be updated before this cycle. For the conventional DSP model, an AR load is required when this AR update amount is beyond ±1, and this becomes an overhead in codes. Meanwhile, an AR load is required when the AR update amount is beyond ±2, for the new DSP model. Additional hardware required for these new addressing operations is expected to be small.

Let us think of memory address and AR operations with a simple example, signal flow graph of a second order IIR filter shown in Fig. 2. After an appropriate computational ordering, intermediate codes are derived as shown in left hand side of Table 1, where A denotes the intermediate results kept in the register.

The right hand side in Table 1 shows memory access sequences and AR operations. For the conventional DSP model, one may decide the addresses of variables as they appears in the program, and it is shown in "Appearance Order" column of Table 1. By this method, 4 immediate AR loads between codes 4~6, 7~8, 8~9, 10~11, which are denoted as "*LD*" in Table 1, are required. Meanwhile, if memory address is appropriately decided as shown in "Conventional DSP model" column of the same table, only 3 AR loads between codes 1~2, 3~4, 11~12 are required, and one immediate AR load can be saved. For the new DSP model, AR can be updated by ±2 in one instruction cycle, so that no

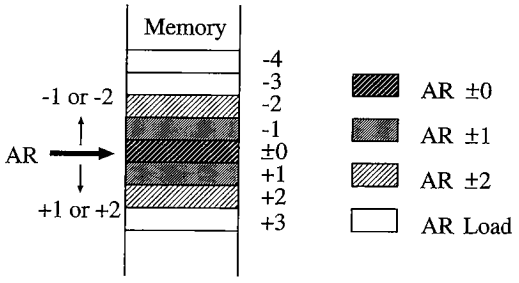


Fig. 1 AR update range of this DSP.

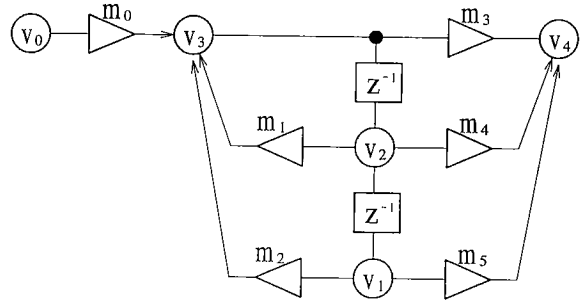


Fig. 2 SFG of 2nd order IIR filter.

Table 1 Intermediate code for Fig. 2 and AR operations.

Instruction Cycle	Intermediate Code	Memory Access Sequence	AR operation		
			Appearance Order	Conventional DSP Model [10]	New DSP Model
1	$A \leftarrow v_0 \times m_0$	v_0	0) +1	0) *LD*	0) +2
2	$A \leftarrow A + v_1 \times m_2$	v_1	1) +1	2) +1	2) +1
3	$A \leftarrow A + v_2 \times m_1$	v_2	2) +1	3) *LD*	3) −2
4	$v_3 \leftarrow A$	v_3	3) *LD*	1) +1	1) +1
5	$A \leftarrow A \times m_3$	none			
6	$A \leftarrow A + v_1 \times m_5$	v_1	1) +1	2) +1	2) +1
7	$A \leftarrow A + v_2 \times m_4$	v_2	2) *LD*	3) +1	3) +1
8	$v_4 \leftarrow A$	v_4	4) *LD*	4) −1	4) −1
9	$A \leftarrow v_2$	v_2	2) −1	3) −1	3) −1
10	$v_1 \leftarrow A$	v_1	1) *LD*	2) −1	2) −1
11	$A \leftarrow v_3$	v_3	3) −1	1) *LD*	1) +2
12	$v_2 \leftarrow A$	v_2	2	3	3
Number of immediate AR loads			4	3	0

AR load is required as shown in "New DSP model" column, when the same memory allocation is used for the conventional DSP model.

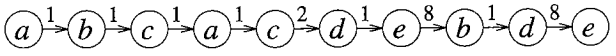
In the case of the new DSP model, appropriate memory address allocation is also very important as well as the case of the conventional DSP model. An appropriate memory address allocation can utilize ± 1 and ± 2 AR operations as much as possible and can save immediate AR address loads. The problem to find such an efficient memory address allocation is called as "memory allocation issue" in this paper. Moreover, DSPs usually have multiple ARs and utilization of these ARs can improve memory access. The problem to find an appropriate memory accesses assignment into multiple ARs is called as "AR assignment issue." In this paper, however, only memory allocation issue is discussed.

The optimum memory allocation is given by examining all the possible combination of memory address and variables. However that method costs a plenty of time and is not appropriate for practical use [11]. Therefore heuristic methods like the methods in Refs. [6]–[11] are employed in this paper.

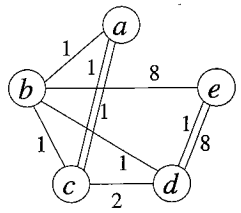
3. Memory Allocation Method

3.1 Access Graph

A given memory access sequence is modeled by an access graph (AG), where each vertex and edge denote the variable to be accessed and the required AR update, respectively [9]–[11]. A unique AG is derived for a given memory access sequence and a memory allocation method is applied for this AG. After a memory allocation, a memory address is decided for each vertex, and AR update operations are assigned to each edge. Note that the number labeled on each edge is the address update length which indicates the number of possible AR ± 1 or ± 2 operations for the corresponding AR update. For example, the AG of the memory access sequence in Fig. 3 (a) is shown in Fig. 3 (b), where the number labeled on each arrow in Fig. 3 (a) and each edge in Fig. 3 (b) denote the length. In this paper, all the edges connected between two vertices are handled as an edge group.



(a) Memory access in 5th order Wave Digital Lattice Filter



(b) The AG for the memory access in (a)

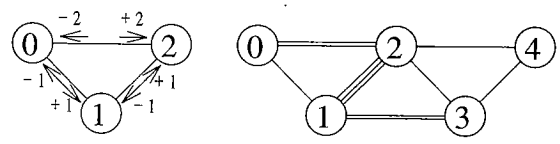
Fig. 3 An AG example.

3.2 Triangle Graph and Chained Triangular Graph

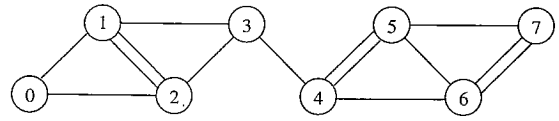
For the above mentioned new DSP model, AR is updated within ± 2 in one instruction cycle, so that no immediate AR load is required for a triangle shaped AG as shown in Fig. 4 (a). For a chain of triangle graphs as shown in Fig. 4 (b), no immediate AR load is required, and memory addresses for variables are easily decided. In this paper, such a graph is called as Chained Triangular Graph (CTG). CTG is a kind of 2 path outer-plane graphs (or 2 path maximum outer-plane graphs) in the graph theory. In the case that two CTGs are linked by edge groups as shown in Fig. 4 (c), no AR load is required. Such a graph is called as quasi-CTG in this paper. A quasi-CTG linked with a CTG by an edge group also forms a quasi-CTG. A CTG itself is also a class of quasi-CTGs.

Generally, an AG includes some edges in addition to a CTG or a quasi-CTG. Therefore, memory allocation method proposed in this paper extracts a quasi-CTG in a given AG, since the quasi-CTG gives a memory allocation without need of any AR load. As a result of extraction, some edges in a given AG are removed. At these removed edges, immediate AR loads are required according to their length and the decided memory allocation.

For example in Fig. 5, an edge between vertices 0 and 3 is removed to form a CTG, where the length is labeled on each edge. For the case of Fig. 5 (a), an immediate AR load is required at the removed edge as marked by $\times$. Meanwhile, no immediate AR load is

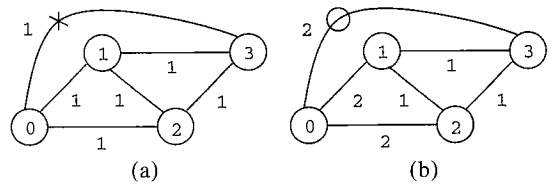


(a) A triangle graph (b) A CTG



(c) A quasi-CTG

Fig. 4 Classes of AG.



(a) (b)

Fig. 5 Length of edge.

required, for the case of Fig. 5 (b), because the removed edge allows 2 AR update operations (AR ±4). In order to evaluate edge length, a cost function mentioned in the following section is introduced. By use of the cost function, the method extracts a quasi-CTG of the maximum cost, so that total cost of removed edges is minimized.

3.3 Cost of Edge Group

In order to minimize the number of AR loads for a memory allocation, it is preferable to extract a quasi-CTG by removing the minimum number of edges. When all the edge lengths are 1 or the memory allocation method does not consider AR update operations at the codes without any memory accesses, the number of AR loads equals to the number of removed edges. In this case, the extracted quasi-CTG of the maximum edges gives the most efficient memory allocation. However, the exact number of AR loads for edges with lengths is not known until the memory allocation completes, because it is counted according to the placement of variables in memory space. Therefore, the cost function $w(u, v)$ estimates the number of AR loads for each edge group (u, v) in terms of the number of edges in (u, v) and their length.

Suppose variable b is accessed l cycles after an access of variable a as shown in Fig. 6 (a), where the length of this AR update is labeled on the edge. Note that AR can be updated by $2l$ at an edge of length l, when the new DSP model with AR ±2 operations is assumed. When N variables are included in a given memory access sequence, the number of possible address allocations is

$$N!.$$

In some of $N!$ address allocations, b is located in the next address to a as shown in Fig. 6 (b), and the number of these address allocations is given by

$$2 \cdot (N-1) \cdot (N-2)!,$$

where 2 means that variables a and b are positioned in 2 ways. Similarly, the number of possible allocations in which a and b are located in 2 address distance as shown in Fig. 6 (c) is

$$2 \cdot (N-2) \cdot (N-2)!.$$

In general, the number of possible allocations in which a and b are located in k address distance as shown in Fig. 6 (d) is

$$2 \cdot (N-k) \cdot (N-2)!.$$

The number of possible allocations in which address distance between the two variables is more than $2l$ as shown in Fig. 6 (e) is given by

$$\sum_{k=2l+1}^{k=N} 2 \cdot (N-k) \cdot (N-2)! = (N-2l-1)(N-2l) \cdot (N-2)!,$$

and these allocations require an AR load at this AR update. Note that $2l < N$ is assumed, because AR update of $2l \geqq N$ is easily realized by use of AR ±1 and ±2 operations. When edges between variable a and b are removed in the graph triangulation, variables a and b are not located within 2 address distance. The number of possible allocations in which a and b are not located within 2 address distance is given by

$$N! - 2 \cdot (N-1) \cdot (N-2)! - 2 \cdot (N-2) \cdot (N-2)! = (N-2) \cdot (N-3) \cdot (N-2)!.$$

Consequently, the ratio of the number of possible allocations in which address distance of the two variables is more than $2l$ to the number of overall possible allocations in which the two variables are not located within 2 address distance is given by

$$P(N,l) = \frac{(N-2l-1)(N-2l) \cdot (N-2)!}{(N-2) \cdot (N-3) \cdot (N-2)!} = \frac{(N-2l)(N-2l-1)}{(N-2)(N-3)}.$$

$P(N, l)$ estimates the number of AR loads at the edge of l length, when this edge is removed.

For edge group (u, v), which contains multiple edges of different lengths, the cost or weighting function $w(u, v)$ is given by

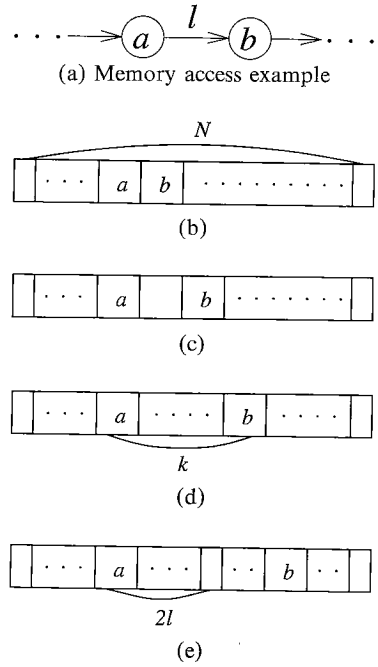


Fig. 6 Memory locations and edge lengths.

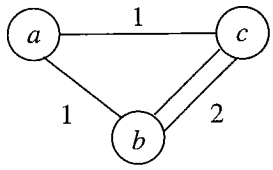


Fig. 7 Triangle graph $(a, b.c)$.

$$w(u,v) = \sum_{l=1}^{N-1} a_l(u,v) \cdot P(N,l), \tag{1}$$

where $a_l(u,v)$ denotes the number of edges with length l connected between vertices u and v.

3.4 Graph Triangulation

To minimize the number of AR loads, the proposed memory allocation algorithm extracts a quasi-CTG in a given AG according to the cost in the previous subsection. Extraction procedures are the followings.

3.4.1 Triangle Graph

In the first stage, costs of all the edge groups in the AG are evaluated. And then, all the triangle graphs involved in the AG are extracted. For each triangle graphs, its cost is evaluated by the sum of their edge group costs.

Let us explain the procedure for evaluating triangle graphs by using an example. Fugure 7 shows a triangle graph, where the numbers labeled on edges denote their costs. The first triangle cost function $T_1(a,b,c)$ of the triangle graph (a,b,c) in Fig. 7 is given by

$$\begin{aligned} T_1(a,b,c) &= \sum_{(u,v)\in E_T} w(u,v) \\ &= w(a,b) + w(b,c) + w(c,a) \\ &= 4, \end{aligned} \tag{2}$$

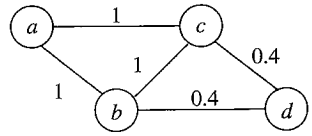
where E_T denotes all the edge groups in the triangle graph (a,b,c).

This triangle graph cost gives the number of AR loads, if all the edge groups of this triangle graph are removed in the graph triangulation procedure.

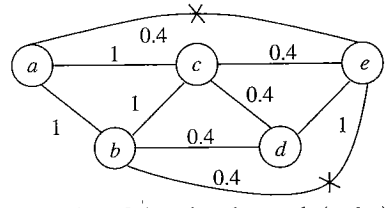
3.4.2 CTG

In the second stage, triangle graphs are jointed to form CTGs. At first, a triangle of the maximum cost is chosen as a CTG core. Then, the CTG is iteratively expanded by jointing a triangle graph of the maximum cost among triangle graphs adjacent to the CTG. These procedures are repeated until no triangle graph is left. Finally, several number of CTGs are derived.

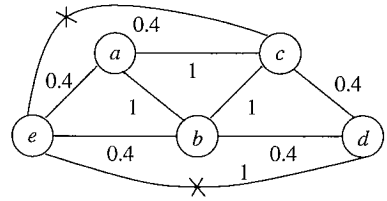
Procedures to form a CTG are illustrated by an example CTG in Fig. 8 (a). Assume we have triangle graphs (c,d,e) and (a,b,e) adjacent to this CTG. Figures 8 (b) and (c) show CTGs after joint of these two triangle graphs, respectively. In the figures, the numbers labeled on edges denote their costs.



(a) CTG (a,b,c,d)



(b) Joint of the triangle graph (c,d,e)



(c) Joint of the triangle graph (a,b,e)

Fig. 8 Joint of triangle graphs.

The triangle cost function $T_2(c,d,e)$ of the triangle graph (c,d,e) in Fig. 8 (b) is given by

$$\begin{aligned} T_2(c,d,e) &= T_1(c,d,e) - \sum_{(u,v)\in E_C} w(u,v) \\ &= T_1(c,d,e) - (w(a,e) + w(b,e)) \\ &= 1, \end{aligned} \tag{3}$$

where E_C denotes all the removed edge groups to derive the CTG (a,b,c,d,e).

By similar procedure, the cost function $T_2(e,a,b)$ of a triangle graph (e,a,b) in Fig. 8 (c) is given by

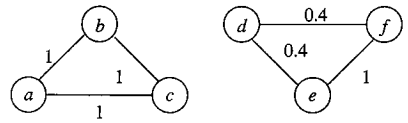
$$\begin{aligned} T_2(e,a,b) &= T_1(e,a,b) - \sum_{(u,v)\in E_C} w(u,v) \\ &= T_1(e,a,b) - (w(c,e) + w(d,e)) \\ &= 0.4, \end{aligned}$$

where E_C denotes all the removed edge groups to derive the CTG (e,a,b,c,d). According to the cost, the adjacent triangle graph (c,d,e) is jointed to the CTG in Fig. 8 (a), and Fig. 8 (b) is derived in this example.

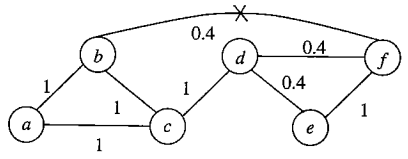
3.4.3 Quasi-CTG

In the final stage, CTGs are linked together by appropriate edge groups, and a quasi-CTG is derived. These edge groups are selected by evaluating their cost.

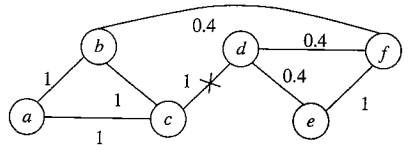
Procedures to link CTGs are explaind by an example in Fig. 9 (a). Assume we have edge groups (c,d) and



(a) CTGs (a, b, c) and (d, e, f)



(b) Link by the edge group (c, d)



(c) Link by the edge group (b, f)

Fig. 9 Link of CTGs.

(b, f) between two CTGs (a, b, c) and (d, e, f). CTGs linked by edge groups (c, d) and (b, f) form quasi-CTGs depicted in Fig. 9 (b) and (c), respectively, where the numbers labeled on edges denote their costs. The edge cost function $E_1(c, d)$ of the edge group (c, d) in Fig. 9 (b) is given by

$$\begin{aligned} E_1(c,d) &= w(c,d) - \sum_{(u,v)\in E_Q} w(u,v) \\ &= w(c,d) - w(b,f) \\ &= 0.6, \end{aligned} \tag{4}$$

where E_Q denotes all the removed edge groups to derive the quasi-CTG (a, b, c, d, e, f).

By similar evaluation, the cost function $E_1(b, f)$ of the edge group (b, f) in Fig. 9 (c) is given by

$$\begin{aligned} E_1(b,f) &= w(b,f) - \sum_{(u,v)\in E_Q} w(u,v) \\ &= w(b,f) - w(c,d) \\ &= -0.6, \end{aligned}$$

where E_Q denotes all the removed edge groups to derive the quasi-CTG (a, c, b, f, e, d). Finally, according to the cost, the edge group (c, d) is chosen to link two CTGs, and a quasi-CTG in Fig. 9 (b) is derived.

3.5 Memory Allocation Algorithm

By use of the edge group cost and the graph triangulation procedures, the memory allocation algorithm called as ALOMA-CTG (Address Load Operation Minimization Algorithm by use of CTG) is summarized as the followings.

ALOMA-CTG

Input AG $G = (V, E)$
Memory area $M = \{x \mid x \in \boldsymbol{N};\ 0 \leqq x < |V|\}$

Output Memory allocation $(m : V \rightarrow M)$ to minimize the cost function

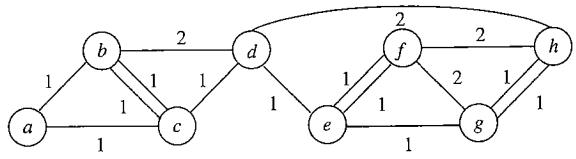
Cost function The number of immediate AR load operations required in G

1. For all the edge groups (u, v) in G, evaluate cost $w(u, v)$ by Eq. (1).
2. Extract all the triangle graphs in G. For each triangle, evaluate cost T_1 by Eq. (2).
3. Select a triangle graph of the maximum cost T_1 as a core of CTG.
4. Extract all the adjacent triangle graphs to the core of CTG. For each triangle, evaluate cost T_2 by Eq. (3).
5. Joint adjacent a triangle graph with the maximum cost T_2 to the CTG to derive the extended CTG.
6. Repeat 4 $\sim$ 5 until no adjacent triangle graph is left.
7. Repeat 3 $\sim$ 6 until no core triangle graph is left.
8. Extract all the edge groups to link two of the derived CTGs. For each edge group evaluate cost E_1 by Eq. (4).
9. Link two of the derived CTGs by the edge group of the maximum cost E_1.
10. Repeat 8 and 9 until no edge group between two CTGs is left.
11. For the vertices in the derived quasi-CTG, memory addresses are decided.

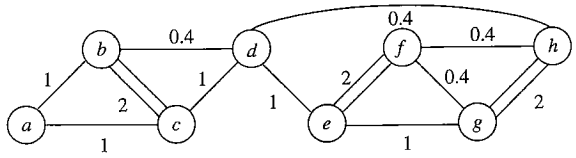
Computational cost of this algorithm is evaluated for three procedures separately. When a given graph has v vertices, t triangles and c CTGs, computational costs for triangle extraction procedure (step 1, 2), CTG forming procedure (step 3 $\sim$ 7) and CTG linkage procedure (step 8 $\sim$ 10) are given by $O(t)$, $O(t^2 \cdot c)$ and $O(v^2 \cdot c)$, respectively. Totally, computational cost of the whole algorithm is given by $O((t^2 + v^2) \cdot c)$.

Let us explain procedures of the proposed method by using an example. An example of the whole procedure of memory allocation is shown in Fig. 10. Figure 10 (a) shows an initial AG with 8 vertices, where the numbers labeled on edges denote memory access lengths.

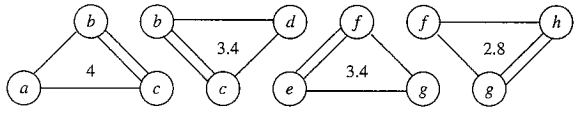
At first, the cost of all the edge groups in the AG are evaluated and they are labeled on every edge group in Fig. 10 (b). According to the 2nd procedure of ALOMA-CTG, all the triangle graphs involved in the AG are extracted as shown in Fig. 10 (c), where the



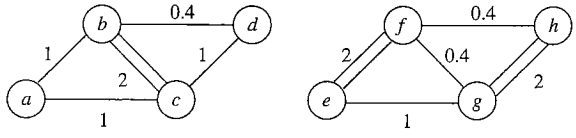
(a) Initial AG



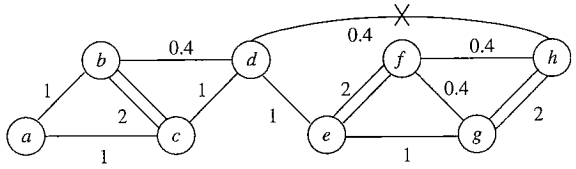
(b) Cost of each edge group



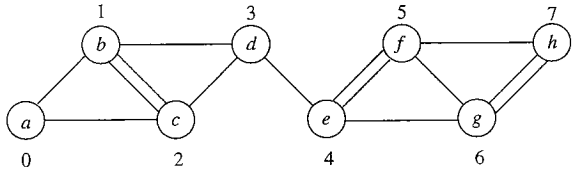
(c) Extraction of triangle graphs



(d) Joint of triangle graphs



(e) Link of CTGs



(f) Decision of memory addresses

Fig. 10 Procedure of memory allocation.

number labeled on every triangle graph denotes its cost T_1.

Among these triangle graphs, the triangle graph (a, b, c) of the maximum cost T_1 is selected first. Then, a triangle graph (b, c, d) adjacent to the triangle graph (a, b, c) is jointed together, and the CTG (a, b, c, d) is derived. Similar procedure is started from the triangle graph (e, f, g), and the CTG (e, f, g, h) is derived. The derived CTGs are shown in Fig. 10 (d).

The derived two CTGs can be linked by edge groups (d, e) or (d, h) as shown in Fig. 10 (e). By evaluating the costs E_1 of two edge groups, edge group (d, e) of the larger cost is selected, and the quasi-CTG with maximum cost is derived(Fig. 10 (f)). Finally, memory addresses of vertices a through h are decided for the derived quasi-CTG, and they are labeled on vertices in Fig. 10 (f).

4. Address Allocation Results

The above proposed methods are applied to the DIMPL compiler [4], [5] for the above mentioned DSP model and codes for several examples are generated. In these examples, Wave Digital Filters (WDFs) include rather complicated memory access sequences, and they are appropriate for the comparison in memory addressing methods. The numbers of their program steps, variables, accesses are shown in Table 2. Table 3 shows the comparison of the proposed methods to the existing method [11] and the optimal memory location, in terms of the number of AR loads in the generated codes and execution time which is in the range of seconds on a

Table 2 # of Steps, Variables, and Accesses.

	# of Steps	# of Variables	# of Accesses
WDF(5)	71	14	33
WDF(7)	113	23	53
WDF(9)	145	32	75
WDF(11)	183	40	93
WDF(17)	298	64	159
FFT(2^3)	90	8	56
FFT(2^4)	250	16	152

WDF(): Wave Digital Filter (Order)

Table 3 Comparison of memory allocation methods (# of AR loads and Execution Time (seconds)).

	Conventional DSP				A New DSP Model				
	Appearance Order	ALOMA-L [11]	Optimal/ Suboptimal	Execution Time	Appearance Order	ALOMA-L(±2)	ALOMA-CTG	Optimal/ Suboptimal	Execution Time
WDF(5)	7	8	5	0.06	4	2	3	0	0.10
WDF(7)	16	15	11*	0.07	7	9	4	3*	0.16
WDF(9)	28	22	15*	0.09	14	17	7	6*	0.36
WDF(11)	41	29	23*	0.14	19	22	14	12*	0.75
WDF(17)	73	64	58*	0.45	46	53	37	33*	5.20
FFT(2^3)	26	25	23	0.06	19	15	11	9	0.10
FFT(2^4)	90	88	86*	0.09	59	71	59	56*	0.52

*: The minimum number of AR loads derived until now.

JCC JS20/762. The column "ALOMA-L(±2)" shows the result derived by the method for the new DSP model with AR ±2 operations in the ALOMA-L [11].

Memory allocation results derived by the proposed methods need less AR loads than results derived by the simple allocation method. They are near to optimal/sub-optimal memory allocation results derived by methods based on exhaustive and iterative search methods.

5. Conclusion

In this paper, AR update operations by ±2 in indirect memory addressing mode are newly considered, and a method to derive an efficient memory allocation utilizing both AR ±1 and ±2 update operations is proposed. In this method, a given memory access sequence is modeled by a graph, and it is triangulated by joint of CTGs with considering the number and the length of edges. The proposed methods are applied to the DSP compiler, and resultant memory allocations for several examples are near to the optimal/sub-optimal memory allocation. The proposed methods simply input an access sequence and output an address location, so that they are applicable to other compilers.

Methods to utilize multiple ARs (AR assignment issue) are under investigation now. For an indirect addressing mode with AR $\pm 2^n$ operations, which is easily realized by n-bit up-down counter technique, methods to derive an efficient memory address allocation will be studied in future. Although computational complexity of the algorithm increase, memory allocation methods suitable for this DSP, which reduce overhead codes with small additional hardware, are expected.

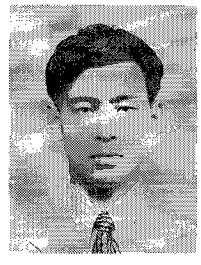
Acknowledgement

The authors are grateful to Prof. Fujii and Assoc. Prof. S. Takagi of Tokyo Institute of Technology for the valuable discussions. This work is supported by the Research Body of CAD21 (Computer Aided Design for 21th Century) in Tokyo Institute of Technology.

References

[1] R. Simar, Jr. and A. Davis, "The application of high-level languages to single chip digital signal processors," 1988 ICASSP, pp.1678–1681, 1988.
[2] J. Hartung, S.L. Gay, and S.G. Haigh, "A practical C language compiler/optimizer for real-time implementations on a family of floating point DSPs," 1988 ICASSP, pp.1674–1677, 1988.
[3] E.A. Lee, W.-H. Ho, E. Goei, J. Bier, and S. Bhattacharyya, "Gabriel: A design environment for DSP," IEEE Trans. ASSP, vol.37, no.11, pp.1751–1761, Nov. 1989.
[4] N. Sugino, A. Toshikiyo, E. Watanabe, and A. Nishihara, "Computational ordering of digital signal processing networks and its application to compilers for signal processors," IEICE Tarns., vol.J71-A, pp.327–335, 1988.
[5] N. Sugino, S. Ohbi, and A. Nishihara, "Computational ordering of digital network under the pipeline constraints and its application to compiler for DSPs," Proc. ECCTD '89, pp.395–399, Sept. 1989.
[6] S. Liao, S. Devadas, K. Keutzer, S. Tjiang, and A. Wang, "Storage assignment to decrease code size," ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI), pp.186–195, June 1995.
[7] R. Leupers and P. Marwedel, "Algorithms for address assignment in DSP code generation," ACM/IEEE ICCAD, pp.109–112, Nov. 1996.
[8] B. Wess, "Automatic code generation for integrated digital signal processors," Proc. ISCAS 1996, pp.33–36, June 1996.
[9] N. Sugino, S. Iimuro, A. Nishihara, and N. Fujii, "DSP code optimization utilizing memory addressing operation," IEICE Trans. Fundamentals, vol.E79-A, no.8, pp.1217–1224, Aug. 1996.
[10] N. Sugino, H. Miyazaki, S. Iimuro, and A. Nishihara, "Improved code optimization method utilizing memory addressing and its application to DSP compiler," Proc. ISCAS 1996, pp.249–252, May 1996.
[11] N. Sugino and A. Nishihara, "Memory allocation methods for a DSP with indirect addressing modes and their application to compilers," Proc. ISCAS 1997, pp.2585–2588, June 1997.

Nakaba Kogure was born in Tokyo, Japan on December 12, 1973. He received B.E. degrees in electrical and electronic engineering from Tokyo Institute of Technology in 1997. He is now a postgraduate student of Department of Physical Electronics, Faculty of Engineering, Tokyo Institute of Technology. His main research interests are in software for digital signal processing.



Nobuhiko Sugino was born in Yokkaichi, Mie, Japan on November 19, 1964. He received B.E., M.E. and Dr.Eng. degrees in physical electronics from Tokyo Institute of Technology in 1987, 1989 and 1992, respectively. Since 1992, he has been with Tokyo Institute of Technology, where he is now a lecturer of department of information processing, interdisciplinary graduate school of science and engineering. His main research interests are in hardware and software for digital signal processing, especially in software development tools for digital signal processors. Dr.Sugino is a member of IEEE.

Akinori Nishihara was born in Fukuoka, Japan on February 26, 1951. He received B.E., M.E. and Dr.Eng. degrees in electronics from Tokyo Institute of Technology in 1973, 1975 and 1978, respectively. Since 1978, he has been with Tokyo Institute of Technology, where he is now Professor of the Center for Research and Development of Educational Technology. His main research interests are in filter design, 1D and multiD signal processing, and educational technology. From 1990 to 1994 he served as an Associate Editor of the IEICE Trans. Fundamentals, and is now serving as an Associate Editor of the IEICE Trans. Circuits & Systems II. He was 1995–96 Student Activities Committee Chair, IEEE Region 10 (Asia Pacific Region). Dr.Nishihara is a member of IEEE, EURASIP, ECS and JET.