MINIMIZATION OF DATA ADDRESS COMPUTATION OVERHEAD IN DSP PROGRAMS

Bernhard Wess and Martin Gotschlich

INTHFT, Vienna University of Technology
Gusshausstrasse 25/389, A-1040 Vienna, Austria
phone: +43 1 58801 3519, fax: +43 1 5870583
email: bwess@email.tuwien.ac.at

ABSTRACT

Digital signal processors (DSPs) provide dedicated data address generation units (AGUs) with multiple register files. These units allow data memory access by indirect addressing with automatic address modification. Typically, both linear and modulo addressing are supported. There is no address computation overhead if the next address is within the auto-modify range. Often, this range can be adapted to the application by assigning static values to modify registers.
In this paper, we discuss optimized data memory address generation in DSP programs. Here the goal is to minimize data address computation and register initialization costs by optimizing data memory layout, address register assignment, and auto-modify range. The investigated combinatorial optimization problems can have an extremely large solution space. However, experimental results indicate that random neighbourhood sampling by simulated annealing allows to produce highly optimized solutions.

1. INTRODUCTION

It has been shown in [1] that many high-level language compilers for *digital signal processors* (DSPs) produce very poor code. Often the only alternative is assembly-level programming which is both time-consuming and error-prone. To overcome this problem, new code optimization techniques are developed [2, 3]. As compared to compilers for general-purpose computers, lower compilation speed is acceptable and therefore more computation-time intensive algorithms may be applied.

In this paper, we focus on optimized data memory address generation in DSP programs. In section 2, we define a generic *address generation unit* (AGU) model which is consistent with current DSP architectures. The *address assignment problem* (AAP) and the *index allocation problem* (IAP) are discussed in section 3 and 4, respectively. Both AAP and IAP are combinatorial optimization problems of exponential complexity. However, experimental results in section 5 show that, for realistic problem sizes, highly-optimized solutions can be produced within a short time. Conclusions are given in section 6.

This work has been supported by the Fonds zur Förderung der wissenschaftlichen Forschung under research grant P10701-ÖTE.

2. DATA ADDRESS GENERATION UNITS

Typicallly, as shown in Fig. 1, AGUs of DSPs provide three register files: the *index register* file (IRF), the *modify register* file (MRF), and the *length register* file (LRF). The index registers contain the actual addresses for data memory access. When data is accessed in indirect mode, the address stored in the selected index register becomes the memory address. AGUs employ a post-modify scheme. After an indirect data access, the specified modify register is added to the selected index register. For circular buffers, the AGUs perform modulo address modification where the selected length register contains the buffer length.

In our AGU model, there are k *address pointers* (index registers) with an auto-modify range $[-n, p]$ where k, n, and p are any positive integers. For the specific case of auto-increment/decrement AGU architectures, both $n = 1$ and $p = 1$. Typically, a modify range $[-2, 2]$ can be realized in contemporary DSPs by assigning static values to modify registers. In general, the parameters n and p need not be equal. We can model both linear addressing

$$I + m \;\rightarrow\; I$$

and modulo addressing

$$(I + m - B) \bmod (L) + B \;\rightarrow\; I$$

with $m \in \{-n, \ldots, p\}$. I denotes the selected index register and L the buffer length. Without any loss of generality, we assume for the base address $B = 1$. For the following discussion, we assign zero cost to auto-modify operations and unit cost to register load operations. Note that our AGU model is consistent with

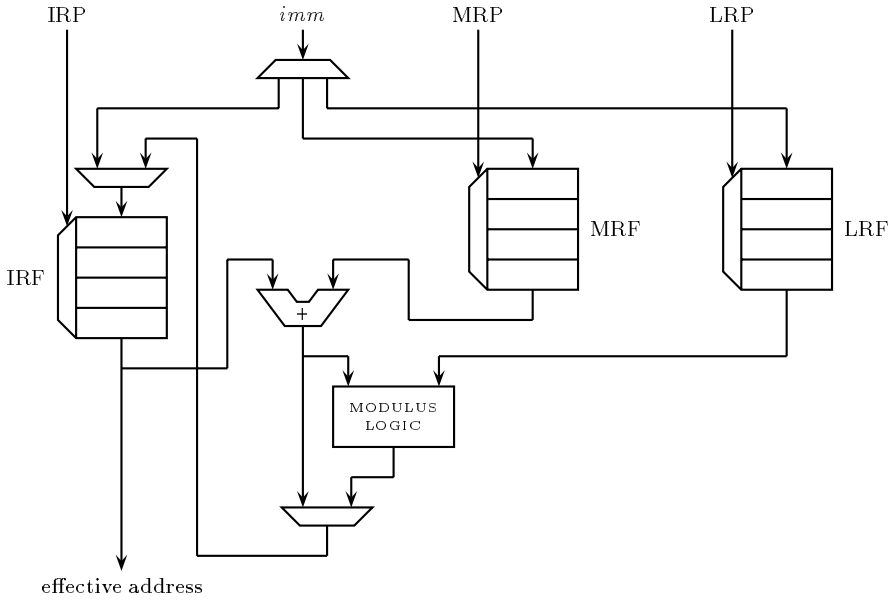

Figure 1: Typical architecture of an AGU.

contemporary DSP architectures such as Analog Devices' ADSP-2100 [4], Motorola's DSP56000 [5], and Texas Instruments' TMS320C5x [6].

3. ADDRESS ASSIGNMENT PROBLEM

The address assignment problem is to construct a data memory layout such that the number of accesses outside the auto-modify range is minimized. Here it is assumed that there is a set V of program variables which are accessed in the sequence S. Bartley [7] has proposed a heuristic algorithm for the specific case of linear addressing with auto-increment or decrement by 1. His work has been improved and generalized in [8, 9, 10, 11].

For the generic AGU model defined above, optimal memory layout generation can be formulated as a *quadratic assignment problem* (QAP) [12]. Let $A = (a_{ij})$ be an $|V| \times |V|$ matrix (*adjacency matrix*) where a_{ij} gives the number of access transitions between program variables i and j in S. We define a *cost matrix* $C = (c_{ij})$ which gives the addressing costs for all possible access transitions. For linear addressing with auto-modify range $[-n, p]$, C becomes

$$c(n,p)_{ij} = \begin{cases} 0 & \text{if } j = i + m \\ 1 & \text{otherwise,} \end{cases}$$

and in case of modulo addressing

$$c(n,p)_{ij} = \begin{cases} 0 & \text{if } j = (i + m - 1) \bmod |V| + 1 \\ 1 & \text{otherwise,} \end{cases}$$

with $m \in \{-n, \ldots, p\}$. With our definitions for the adjacency matrix A and cost matrix C, optimal memory layout generation is equivalent to finding a permutation π of set $N = \{1, \ldots, |V|\}$ such that the objective function

$$o(\pi, n, p) = \sum_{i=1}^{|V|} \sum_{j=1}^{|V|} c(n,p)_{ij} a_{\pi(i)\pi(j)}$$

becomes minimal. In this case, π gives the optimal addresses for the program variables $v \in V$ of S. The QAP is NP-hard but there are efficient heuristics leading to near-optimal solutions within short time.

Optimal address assignment for loops has to take into account access transitions between loop iterations. In this case, the objective function becomes

$$o(\pi, n, p) = \sum_{i=1}^{|V|} \sum_{j=1}^{|V|} c(n,p)_{ij} a_{\pi(i)\pi(j)} + c(n,p)_{lf} a_{\pi(l)\pi(f)}$$

where f and l denotes the first and the last program variable, respectively, in the access sequence.

Generating optimum memory layouts in the presence of k address pointers can be considered as a *k-coloring* problem. Here, a color is assigned to each element of S corresponding to the accessing address pointer. As a consequence, a program variable can be accessed by different address pointers at different times. The goal is to minimize the sum of the objective function values (index register reload costs irc) for the k subsequences of S which are defined by the colors,

$$irc = \sum_{i \in colors} o_i(\pi, n, p).$$

We take initialization costs of address registers into account by adding two terms to the objective function so that the total addressing costs tac are given by

$$tac = \underbrace{\sum_{i \in colors} o_i(\pi, n, p)}_{\text{index register reload costs}} + \underbrace{|colors|}_{\text{index reg. init. costs}} + \underbrace{n + p.}_{\text{modify reg. init. costs}}$$

In case of modulo addressing, we add an extra term for the length register initialization costs. If each index register has its own length register, these costs are $|colors|$. Let k be the maximum number of available address pointers and $[-n_0, p_0]$ be the largest possible auto-modify range. In general, we are interested in finding an optimized data memory layout π, an address pointer assignment with $|colors|$ address pointers, and an auto-modify range $[-n, p]$ that minimize tac with $|colors| \leq k$, $n \leq n_0$, and $p \leq p_0$.

4. INDEX ALLOCATION PROBLEM

Now, let us consider an access sequence to array elements. The problem of allocating address pointers to array accesses can be considered as a specific case of the AAP where the data memory layout is fixed. Let $ind(i)$ be the index of the array access i. We define the *indexing matrix* $M = (m_{ij})$ by

$$m(n,p)_{ij} = \begin{cases} 0 & \text{if } ind(j) - ind(i) \in \{-n, p\} \\ 1 & \text{otherwise.} \end{cases}$$

Let N be the length of the array access sequence and $X = \{1, \ldots, N\}$. Formally, k-coloring the array access sequence is equivalent to generating a *partition* $P = \{\{X_1\}, \{X_2\}, \ldots, \{X_k\}\}$ of X. Let $S_i = \{f(1), f(2), \ldots, f(L)\}$ be a sequence defined by X_i with $f(1) < f(2) < \ldots < f(L)$ and $L = |X_i|$. If each S_i represents the array accesses for one of the k address pointers, then the reload costs are given by

$$o_i(n,p) = \sum_{j=1}^{L-1} m_{f(j),f(j+1)}.$$

The goal is to find an optimum coloring (partition) such that the objective function value

$$o(n,p) = \sum_{i \in colors} o_i(n,p)$$

is minimized.

The goal of the IAP is to allocate address pointers to array accesses which are part of loop body statements. Araujo in [13] was the first to formulate this problem for index expressions of the type $c*i+d$ where c and d are integer quantities and i is the induction variable which is linearly updated by the integer quantity s (*loop step*). Araujo looks for a minimum number of address pointers avoiding reload operations. However, neither register initialization costs nor optimization across loop iterations are taken into account. A heuristic algorithm for inter-iteration optimizations is proposed in [14].

We investigate a related optimization problem. The goal is to minimize the total addressing costs for a given maximum number of address pointers. We take register initialization costs into account and do inter-iteration optimization.

Let $dist(i,j)$ denote the indexing distance between accesses i and j within a loop,

$$dist(i,j) = \begin{cases} ind(j) - ind(i) & \text{if } i \leq j \\ ind(j) - ind(i) + c*s & \text{if } i > j. \end{cases}$$

For the IAP, we define the indexing matrix by

$$m(n,p)_{ij} = \begin{cases} 0 & \text{if } dist(i,j) \in \{-n, p\} \\ 1 & \text{otherwise.} \end{cases}$$

For any subsequence S_i, the indexing costs are given by

$$o_i(n,p) = \sum_{j=1}^{L-1} m_{f(j),f(j+1)} + m_{f(L),f(1)}$$

where $m_{f(L),f(1)}$ represents the costs for redirecting the address pointer from $ind(f(L))$ (last access within the current loop iteration) to $ind(f(1)) + c*s$ (first access within the next iteration). Again, the goal is to find a coloring that minimizes the total addressing costs tac by taking address register initialization costs into account,

$$tac = \sum_{i \in colors} o_i(n,p) + |colors| + n + p.$$

5. EXPERIMENTAL RESULTS

The solution spaces of AAP and IAP instances can be extremely large. The number of solution space points for the AAP is

$$k^{|S|}|V|!$$

where $|S|$ is the access sequence length, $|V|$ the number of different program variables, and k the number of available address pointers.

We have applied a neighbourhood search technique to generate optimized solutions for the AAP and IAP. By repeatedly moving from the current solution to a neighbouring solution, a subset of feasible solutions is explored. Kirkpatrick [15] proposed to apply *simulated annealing* to escape from local minima in the search process. In contrast to descent strategies, the simulated annealing algorithm may accept neighbours giving rise to an increase in the cost function. The acceptance probability depends on a control parameter (*temperature*) and the magnitude of the increase. The simulated annealing algorithm can be stated as follows:

Objective function o, temperature reduction function α, and neighbourhood structure N.

```
select an initial solution s_0;
select an initial temperature t_0 > 0;
repeat
    repeat
        randomly select s ∈ N(s_0);
        δ = o(s) − o(s_0);
        if δ < 0 then s_0 = s
        else generate random x uniformly in [0, 1];
            if x < e^{−δ/t} then s_0 = s;
    until iteration_count = nrep;
    t = α(t);
until stopping_condition = true;
```

s_0 is the approximation to the optimal solution.

When generating neighbouring solutions for the IAP, random color changes are made. In case of the AAP, we additionally allow changes in the data memory layout by randomly swapping program variables. The number of colors and the size of auto-modify range are randomly modified both in case of the IAP and the AAP. These parameters determine the address register initialization costs.

We have made experiments with random access sequences for the AAP and random indexing sequences for the IAP. Highly optimized solutions have been obtained in at most a few seconds (for sequence lengths up to 50) on a Pentium PC. Our approach significantly outperforms the existing techniques in terms of total addressing costs. Typical savings are in the range of 30% to 40%.

6. CONCLUSIONS

Optimization of DSP programs with respect to data memory address generation is a combinatorial problem. We have defined a generic AGU model which is consistent with comtemporary DSP architechtures. Based on this model, the address assignment problem and the index allocation problem are introduced which are of exponential complexity. For realistic problem sizes, however, highly optimized solutions can be produced within a few seconds on a Pentium PC.

7. REFERENCES

[1] V. Zivojnovic, J. M. Velarde, C. Schläger, and H. Meyr, "DSPstone: a DSP-oriented benchmarking methodology", *in Proc. 5th Int. Conf. on Signal Processing Applications & Technology*, vol. 1, pp. 715–720, Dallas, October 1994.

[2] P. Marwedel and G. Goossens, Eds., *Code Generation for Embedded Processors*, Kluwer Academic Publishers, 1995.

[3] G. Goossens, J. V. Praet, D. Lanneer, W. Geurts, A. Kifli, C. Liem, and P. G. Paulin, "Embedded software in real-time signal processing systems: design technologies", *Proc. IEEE*, vol. 85, pp. 436–454, March 1997.

[4] Analog Devices, Inc., *ADSP-2100 Family User's Manual*, September 1995.

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

[6] Texas Instruments, Inc., *TMS320C5x User's Guide*, 1997.

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

[8] 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*, pp. 186–195, June 1995.

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

[10] 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*, vol. 1, pp. 683–686, Munich, April 1997.

[11] A. Sudarsanam, S. Liao, and S. Devadas, "Analysis and evaluation of address arithmetic capabilities in custom DSP architectures", *in Proc. 34th ACM/IEEE Design Automation Conf.*, Anaheim, June 1997.

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

[13] G. Araujo, A. Sudarsanam, and S. Malik, "Instruction set design and optimizations for address computation in DSP architectures", *in Proc. 8th Int. Symp. on System Synthesis*, La Jolla, November 1996.

[14] R. Leupers, *Retargetable Code Generation for Digital Signal Processors*, Kluwer Academic Publishers, 1997.

[15] S. Kirkpatrick, Jr. C. D. Gelatt, and M. P. Vecchi, "Optimization by simulated annealing", *Science*, vol. 220, pp. 671–680, May 1983.