DSP DATA MEMORY LAYOUTS OPTIMIZED FOR INTERMEDIATE ADDRESS POINTER UPDATES

Bernhard Wess and Stefan Fröhlich

INTHFT, Vienna University of Technology
Gusshausstrasse 25/389, A-1040 Vienna, Austria
phone: +43 1 58801 38919, fax: +43 1 5870583
email: bwess@email.tuwien.ac.at

ABSTRACT

Dedicated address generation units (AGUs) in modern digital signal processors (DSPs) support data memory access by indirect addressing with subsequent address pointer modification in parallel to other machine operations. In this paper, we present an integrated data memory layout and address register assignment optimization procedure. This technique allows to reduce both execution time and code size of DSP programs.

Our generic AGU model is consistent with AGUs of contemporary fixed-point DSPs. It captures important addressing capabilities of DSPs such as linear addressing, modulo addressing and auto-modifying within a given auto-modify range. There is no address computation overhead if the next address is within the auto-modify range. We exploit multiple address pointer update opportunities between data memory accesses. Experimental results demonstrate that the proposed technique significantly outperforms existing optimization strategies.

1. INTRODUCTION

Many high-level language compilers for digital signal processors (DSPs) produce very poor code that is unacceptable with respect to code density and performance [1]. In most real-time applications 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, 4]. As compared to compilers for general-purpose computers, lower compilation speed is acceptable and therefore more computation-time intensive algorithms may be applied.

One architectural feature that is typical for DSPs is the dedicated hardware for data memory address computation. DSPs provide address generation units (AGUs) which allow to perform indirect address computation in parallel to the execution of other data path operations. We present an algorithm that attempts to maximize the benefit of this architectural feature. In other words, our optimization technique tries to minimize the data address computation overhead in DSP programs. To this end, we construct data memory layouts and assign address registers such that zero-cost AGU operations are exploited. Our technique is based on a parameterized AGU model which is consistent with modern fixed-point DSP architectures.

The paper is organized as follows. In Section 2, we give a short overview of related work. A generic model of an address generation unit which captures typical addressing capabilities of DSPs is given in Section 3. In Section 4, we introduce our optimization technique based on our AGU model. Experimental results are presented in Section 5 and conclusions are given in Section 6.

2. RELATED WORK

Address assignment for AGUs with a single address register (*simple offset assignment, SOA*) supporting modify range $[-1, 1]$ was first studied by Bartley [5] and Liao [6]. Liao showed that this problem is NP-hard and also proposed a heuristic algorithm for the *general offset assignment* problem (GOA). Here it is assumed that the AGU provides any fixed number of address registers. Leupers [7] refined Liao's SOA algorithm and proposed a new algorithm for the GOA problem outperforming Liao's algorithm by 22% on average. An efficient layout generation algorithm for auto-modify range $[-2, 2]$ is proposed in [8] and address assignment for any symmetric auto-modify range is investigated in [9]. For the generic AGU model defined above, optimal memory layout generation can be formulated as a quadratic assignment problem (QAP) [10].

Some DSPs support address pointer modifications in parallel to arithmetic or logic operations even though no memory access is done. This leads to a larger possible modify range for the variable coming next in the access se-

This work was supported by ÖNB grant 6867 and the Fonds zur Förderung der wissenschaftlichen Forschung under research grant P10701-ÖTE.

quence. None of the previously cited papers exploits this dynamic modify range. In [11, 12] heuristic data memory layout generation algorithms are proposed which take advantage of these additional update opportunities. However, these techniques use a very restricted AGU model, as only a limited address pointer update range is supported and modulo addressing can not be exploited. Although these algorithms are based on heuristics, their execution times grow exponentially with the number of program variables.

3. GENERIC AGU MODEL

Most DSPs include one or more special units that are dedicated to address calculation. AGUs can perform operations without using the data path of the processor. This allows address calculations to take place in parallel with arithmetic operations on data.

Most AGUs employ a post-modify scheme, that is, an increment is added to the index register value after the address is used to access data in memory. The most common increment is plus or minus one. However, the auto-modify range can often be enlarged by assigning static values to dedicated registers. Typically, circular buffer management is supported by modulo addressing where the selected length register contains the buffer size.

Our generic AGU model provides c index registers with an auto-modify range $[-n, p]$ where c, 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 set up 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. We assume that auto-modify operations can be carried out in parallel to any data-path operation even if there is no data memory access. That is, the effective zero-cost modification range for any address pointer is not constant since it depends on the number of updates between memory accesses.

4. PROPOSED APPROACH

Let V be a set of program variables. Each program variable $v_i \in V$ is identified by a unique $i \in \{1, 2, \ldots, |V|\}$. An access sequence $S = v_{s(1)}, v_{s(2)}, \ldots, v_{s(|S|)}$ is defined by a function

$$s : \{1, 2, \ldots, |S|\} \;\rightarrow\; \{1, 2, \ldots, |V|\} \tag{1}$$

where $|S|$ denotes the sequence length. The image $s(i)$ of any $i \in \{1, 2, \ldots, |S|\}$ defines the program variable $v_{s(i)}$ on position i in the access sequence.

A memory layout is a permutation

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

which assigns addresses to all program variables that appear in an access sequence S. The image $\pi(i)$ is the address of variable v_i.

We say there is an access transition from program variable v_i to variable v_j if v_j succeeds v_i in S. There are $|S| - 1$ access transitions in S which are defined by ordered pairs (i, j) with $i, j \in \{1, 2, \ldots, |V|\}$. We define the distance between any two adjacent program variables $v_{s(i)}$ and $v_{s(i+1)}$ by

$$d(i) = \pi\left(s(i+1)\right) - \pi\left(s(i)\right). \tag{3}$$

Suppose all variables $v_i \in V$ of access sequence $S = v_{s(1)}, v_{s(2)}, \ldots, v_{s(|S|)}$ are referenced indirectly by an address pointer with auto-modify range $[-n, p]$. Our goal is to minimize the number of access transitions (i, j) in S that are outside the auto-modify range, $\pi(j) - \pi(i) \notin \{-n, \ldots, p\}$. Let $u(i) > 0$ be the number of possible zero-cost address pointer updates between the accesses to $v_{s(i)}$ and $v_{s(i+1)}$. We define a cost function $c(i)$ that returns zero if the address pointer can be redirected from $\pi\left(s(i)\right)$ to $\pi\left(s(i+1)\right)$ with zero-cost operations and one otherwise. For linear addressing the cost function is defined by

$$c_i = \begin{cases} 0 & \text{for } -n \le \frac{d(i)}{u(i)} \le p \\ 1 & \text{otherwise} \end{cases} \tag{4}$$

and for modulo addressing

$$c_i = \begin{cases} 0 & \text{for } -n \le \frac{d(i)+m}{u(i)} \le p \\ 1 & \text{otherwise} \end{cases} \tag{5}$$

with m equal to $|V|$, $-|V|$, or 0. The objective function

$$o(s, \pi) = \sum_{i=1}^{|S|-1} c_i \tag{6}$$

gives the number of address pointer reload operations.

Since the problem under discussion has an extremely large solution space, we make a statistical solution space exploration. We apply a neighborhood search technique to generate optimized solutions for the address assignment

problem. By repeatedly moving from the current solution z_0 to a neighboring solution $z \in N(z_0)$, a subset of feasible solutions is explored.

Kirkpatrick [13] 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 neighbors z giving rise to an increase in the objective function $o(z)$. The acceptance probability depends on a control parameter t (*temperature*) and the magnitude of the increase δ. Our simulated annealing implementation can be stated as follows:

```
Select randomly an initial solution z0;
  t := |S|;                                   /* select initial temperature */
  nrep := 2|V|;                               /* select initial number of iterations */
  repeat
    repeat
      randomly select z ∈ N(z0);              /* random move */
      δ := o(z) − o(z0);
      if δ < 0 then z0 := z
      else generate random x uniformly in [0, 1];
      if x < e^(−δ/t) then z0 := z;
    until iteration_count = nrep;
    t := α ∗ t;                               /* reduce temperature */
    nrep := β ∗ nrep;                         /* increase number of iterations */
  until stopping_condition = true;
```

The parameter t, initialized by the access sequence length $|S|$, is decreased gradually until no changes in the objective function value occur during a complete cycle of random moves at constant t. We use a geometric reduction function αt where $0 < \alpha < 1$. The number of iterations $nrep$ at each temperature varies from temperature to temperature. The value of $nrep$ is initialized by $2|V|$ and increased geometrically by multiplying by a factor $\beta > 1$.

We explore the neighborhood structure N by allowing random transpositions in a given layout π, a permutation of set $Z = \{1, 2, \ldots, |V|\}$. Generating optimum memory layouts in the presence of any fixed number of address pointers can be regarded as a coloring problem [10]. Here a color is assigned to each program variable in S representing the accessing address pointer.

5. EXPERIMENTAL RESULTS

For an unbiased comparison of techniques, we performed experiments on random access sequences. These sequences differ in length, the number of distinct variables, and the number of additional update opportunities between the individual accesses. To provide a good basis for a comparison of the algorithm in [11] and our technique, sequences with similar characteristics as in [11] have been chosen. Additionaly, results for sequences with less update opportunities are given.

For our technique, we have selected the parameters

Table 1: 71 steps, 14 variables and 33 accesses.

Algorithm	$[-1, 1]$	$[-2, 2]$, modulo	$[-3, 3]$
Sugino et al. [11]	10.0	–	–
Proposed	8.5	0	0

Table 2: 113 steps, 23 variables and 53 accesses.

Algorithm	$[-1, 1]$	$[-2, 2]$, modulo	$[-3, 3]$
Sugino et al. [11]	22.0	–	–
Proposed	20.2	8.2	3.6

Table 3: 145 steps, 32 variables and 75 accesses.

Algorithm	$[-1, 1]$	$[-2, 2]$, modulo	$[-3, 3]$
Sugino et al. [11]	34.0	–	–
Proposed	33.5	19.5	13.2

Table 4: 183 steps, 40 variables and 93 accesses.

Algorithm	$[-1, 1]$	$[-2, 2]$, modulo	$[-3, 3]$
Sugino et al. [11]	38.0	–	–
Proposed	37.4	23.5	17.4

Table 5: 298 steps, 64 variables and 159 accesses.

Algorithm	$[-1, 1]$	$[-2, 2]$, modulo	$[-3, 3]$
Sugino et al. [11]	85.0	–	–
Proposed	84.0	61.7	51.9

such that an optimized solution is generated in approximately 10 seconds on a Pentium PC. Slightly better solutions can typically be found by performing several restarts or by spending more time for the calculation. The values quoted are the average result of 50 iterations on the same problem. Typically, the deviation is in the range of +/- 5 %. Our implementation of Sugino's algorithm [11] needs significantly more CPU time to generate a solution. Even worse, the complexity of the algorithm is close to exponential behaviour so that solutions can only be found within reasonable time for relatively small examples. Regarding the quality of the solutions found, we have not been able to observe significant differences between these algorithms. However, the proposed algorithm is able to handle more general AGU models and thus is applicable for a larger class of DSP architectures. The algorithm is not only suitable for any auto-modify range (including asymmetric ranges) but also handles modulo-adressing. As can be seen by Tables 1 to 5, these architectural features allow to drastically reduce addressing costs. In [11], the mincut-algorithm is applied to handle more than one address register. However, the CPU times needed by the mincut algorithm are quite exhaustive specifically for more than two address pointers. In contrast, integrating color moves into our simulated annealing implementation is very simple and causes just a modest increase in run-time.

6. CONCLUSIONS

The proposed technique can be applied to improve the quality of high-level language compilers for DSPs or to support assembly code synthesis by hand. Compared to existing algorithms, the new technique uses a more general AGU model and shows a better run-time behavior. The simulated annealing parameters can be either chosen to obtain good solutions in short time or high-quality solutions if longer run-times are acceptable. The proposed technique offers a faster and more flexible approach to optimized data memory layout generation and address pointer assignment.

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

[4] C. Liem, *Retargetable Compilers for Embedded Core Processors*, Kluwer Academic Publishers, 1997.

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

[6] 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.

[7] 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.

[8] 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.

[9] 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.

[10] 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.

[11] N. Sugino and A. Nishihara, "Memory allocation methods for a DSP with indirect addressing modes and their application to compilers", *in Proc. IEEE Int. Symp. on Circuits and Systems*, vol. 4, pp. 2585–2588, Hong Kong, June 1997.

[12] N. Kogure, N. Sugino, and A. Nishihara, "Memory address allocation method for a DSP with ± 2 update operations in indirect addressing", *in Proc. Europ. Conf. on Circuit Theory and Design*, Budapest, September 1997.

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