DSP Address Optimization Using A Minimum Cost Circulation Technique

Catherine Gebotys
Department of Electrical and Computer Engineering,
University of Waterloo, Waterloo, Ont N2L 3G1 Canada

This paper presents a new approach to solving the DSP address assignment problem. A minimum cost circulation approach is used to efficiently generate high performance addressing code in polynomial time. Addressing code size improvements of up to 7 times are obtained, accounting for up to 1.6 times improvement in code size and performance of compiler-generated DSP code. Results also show that memory layout has a small effect on code size and performance when optimal addressing is used. This research is important for industry since this value-added technique can improve code size, power dissipation and performance, without increasing cost.

1. Introduction

As DSP applications are rapidly growing more complex, some designers are moving from full custom digital circuitry to programmable processors or in-house cores to obtain lower risk solutions. The DSP core is a DSP processor that can be reused and combined with program/data memory, dedicated logic, plus ASICs, and incorporated onto a large silicon chip, providing a cost efficient and flexible solution for many typical embedded applications requiring low power and high reliability. These systems demand small code size and high performance. Due to increasing complexities, high level compilation is a necessity. However the biggest drawback to both DSP processors or DSP core use is the code generation.

The use of conventional code generation techniques and even compilers specifically designed for commercial DSP processors produce very inefficient code [1,2]. There are many more limitations placed upon code generation for the DSP processor than for the general purpose processor. The difficulty arises from non-homogeneous register sets, small number of very specialized registers, very specialized functional units, restricted connectivity, limited addressing, and highly irregular datapaths[1].

Limited addressing modes and the use of address registers are also typical. For example many DSP processors assume auto-increment/decrement addressing modes for sequential accessing of data variables from memory will be used heavily. In particular there are a number of index registers which point to addresses in memory. The addresses in the index registers can be incremented or decremented at negligible cost. However adding or subtracting offsets (not equal to zero or one) to these index registers requires a specific instruction and therefore has a performance, code size, and energy dissipation cost associated with it. The auto-increment/decrement addressing subsumes address arithmetic instructions and requires shorter instructions[2] than other forms of addressing. Unfortunately it is assumed that efficient data memory layout has been performed to support this type of addressing. Given that these DSP processors must meet tight timing requirements using very small code space (all on chip ROM), the code generation problem is a very difficult one[2]. Typically DSP processors are difficult to use requiring long product development times (using a large number of assembler programmers) even though the program may be less than 1K program ROM. The need for decreasing time to market, development costs, and maintenance costs, demands the use of high level language compilation. All of these factors imply several challenges in writing efficient code generators for such DSP processors. This is even more difficult for in-house core instruction set architecture which requires retargetable compilation.

2. Problem Description and Related Work

The following problem, problem 1 given below, is an important part of the code optimization problem that will be studied in this paper. For simplicity let us assume that an algorithm to implement the application has already been assigned based upon accuracy required, low energy implementation, etc.. The algorithm is composed of basic blocks, which are given as a partially ordered list of code operations. For the problem definition below we assume that there is one target DSP processor or core defined with an instruction set architecture. The target processor supports indirect addressing, in particular there is one or more index registers which point to an address in memory and whose value can be incremented or decremented at negligible cost. Data layout is performed by using the approach described in [2] for single offset assignment layout or by a compiler or user.

Problem 1. Assume we are given initial code generated for the target processor and a memory layout (or sequence) of the data variables. Given the number of index registers in the target processor and the costs of loading an index register with a new value, and adding/subtracting a value to/from the index registers, the problem is to generate addressing code such that performance is maximized and the code size and estimated energy cost is minimum.

The performance and code size costs are exact measures of how many extra cycles or instructions will be required. By minimizing the number of instructions generated we are also minimizing energy dissipation. However the problem should also support instruction level power models[5]. Extensions to this problem also include supporting

0-89791-993-9/97 $10.00 © 1997 IEEE

addressing with fixed offset whose value is stored in a index register in addition to the auto-increment/decrement addressing.

Although many researchers have studied code generation for DSP processors or ASIPs [1], fewer have studied address generation. Researchers in [2] defined the single (and general) offset assignment problem and used a modification of Kruskal's algorithm[7] within a recursive algorithm (later extended by [2]), to modify memory layout for reduced code size. Researchers in [4] introduced a C code transformation approach to make better use of address calculation units. Address generation has been shown to account for a large percent of energy dissipation in [3] and researchers have studied instruction-level power models[5].

In this manuscript a new approach is presented to solve problem 1, DSP address code generation. Unlike previous research, we study the problem of given a data layout in memory, generate optimal addressing code to minimize code size, or maximize performance. This is a valued added approach, that can be used to improve any compiler generated DSP code by returning the code optimized for addressing or it can be used in combination with a data memory layout technique such as [2,9] to further optimize code generation. It supports cases where the data memory layout may be predefined at an interface with another processor or external I/O in the system. A minimum cost circulation technique is used to obtain optimal solutions in polynomial time.

The following terminology will be used in this paper: $AR=\{AR1,..ARn\}$ the set of n index registers. $|AR|=$ the number of index registers (n) in the processor. v_t means that data variable v is accessed at time t. This access may be a read or write. $d(v)$ is the memory address for variable v. $G=(V,A)$ is a graph G composed of vertices V and arcs A. $x_{i\to j}$ is the flow from vertex i to vertex j, where $i,j\varepsilon V$ and $i\to j\varepsilon A$. $c_{i\to j}$ is the capacity on arc $i\to j$. $e_{i\to j}$ is the cost of flow on arc $i\to j$. The next section will provide an introduction to minimum cost circulation. The following sections, will show how the DSP addressing code generation is mapped into a circulation problem and provide examples to illustrate the technique. Finally examples will be presented to explore the impact of data layout on addressing code generation.

3. Introduction to Circulations

To introduce the circulation problem, we will first discuss network flow and it's extension to circulations. The network flow problem is defined on a directed acyclic graph, G, composed of vertices and arcs, $G=(V,A)$, where each arc has a capacity associated with it. The capacity is a real valued quantity where arc $i\to j$, $i,j\varepsilon V$, has an associated capacity $c_{i\to j}$. Let the variable, $x_{i\to j}$, represent the flow in arc $i\to j$. We will call this the flow variable. For any vertex in the graph the flow into the vertex is equal to the flow out of the vertex (known as the conservation of flow [7]). The flow along any arc (in the same direction as the arc) must be positive valued (and also may be greater than or equal to a positive lower bound placed on that arc) but less than or equal to the capacity of that arc. There are two special vertices in this graph called vertex S and vertex T. The flow out of vertex S is equal to the flow into vertex T. Arcs incident to S only leave vertex S and arcs incident to vertex T only are directed into vertex T. The maximum flow problem [7] is to find the maximum amount of flow from vertex S to vertex T through the network graph, such that the conservation of flow is maintained.

To study the minimum cost circulation problem[7] we add an arc from vertex T to vertex S (that circulates the flow in the graph). Each arc has a cost which is also a real valued quantity where arc $i\to j$ has associated cost $e_{i\to j}$. The problem is to find the flow through this graph with the minimum cost such that the sum over all arcs of the multiplication of the flow in each arc and the cost of each arc is minimum. The following equations represent the formulation of the minimum cost circulation problem as a mathematical programming problem.

$$Minimize \sum_{i\to j} e_{i\to j}x_{i\to j} \; Subject\ to \sum_{i|i\to j} x_{i\to j} - \sum_{k|j\to k} x_{j\to k} = 0,\ \forall j,j\varepsilon V.$$

$$0\le l_{i\to j} \le x_{i\to j} \le c_{i\to j},\ \forall(i\to j)\varepsilon A,i,j\varepsilon V.$$

In the problem above, the capacities $c_{i\to j},\forall(i\to j)\varepsilon A$ and costs $e_{i\to j},\forall(i\to j)\varepsilon A$ are given. The problem is to solve for values of $x_{i\to j}$ that represent the flows in the network graph, $G=(V,A)$, such that the objective function is minimum. As long as the capacities and the lower bounds on flow ($l_{i\to j}$), are integer, we can be guaranteed of obtaining integer flows in the solution of this problem [7]. This problem can be solved in polynomial time using linear programming or more commonly by using faster and more efficient network algorithms[7].

4. Methodology and Modeling

This section will briefly describe the methodology for DSP address optimization, problem 1, and how the minimum cost circulation formulation is used to solve problem 1. First an algorithm or task flow graph is selected for the application based upon cost, performance and energy dissipation requirements and transformations are performed. Initial code generation is performed and a memory layout specified by the compiler or generated post process similar to the technique in [2,10] is performed. Finally the minimum cost circulation approach is applied to generate addressing code. The two sections below will describe in detail how the minimum cost network flow is used to solve problem 1.

To model problem 1 as a circulation problem, we first have to develop a graph, as shown in figure 1. The memory layout (used as x-axis in figure 1b)) along with the variable access sequence (the y-axis in figure 1b)) is used to form the graph. For example the x-axis in figure 1b) corresponds to storing data variables *bi*, *br* (shared with *ci*

), *ar* (shared with *cr*), and *ai* from left to right. The y-axis (from top to bottom) represents the access of data variables *ar*, *br*, *bi*, *ai*, *ar*, *cr*, *ai*, *br*, and *ci*, as defined in figure 1a) code. Each vertex in the graph (placed in the appropriate x-y coordinate in figure 1) represents a variable access in the initial code sequence (figure 1a)), $v_t \varepsilon V$, $t \varepsilon \{1,2,...x\}$. Vertices *S* and *T* are added to the graph representing times 0 and $x+1$ respectively. Arcs from the *S* vertex to all vertices in the graph (except vertex *T*) are added. Arcs from each vertex v_t to all other vertices accessed after time *t* (including vertex *T*) are also added. All of these arcs are not shown in figure 1b). Finally one more arc from vertex *T* to vertex *S* is added. Next, capacities and costs are assigned to each arc in the network flow graph and the minimum cost flow problem is solved. The capacity of all arcs is one except the arc from vertex *T* to *S* which has capacity equal to the maximum number of index registers in the target processor, $|AR|$.

The costs per arc can now be set up to reflect the actual costs of code size, performance, or energy dissipation. The path of each flow identifies a partition of accessed variables that will be assigned to one index register. As each unit of flow passes from *T* to *S* it accumulates a cost representing the instruction for loading of an initial value into the index register (*lark* in the C2x[6]). As each unit of flow passes through the vertices between *S* to *T* it accumulates any costs associated with adding or subtracting of offsets. For example if an offset is required, for example $|d(i)-d(j)| = offset > 1$ for arc $i \rightarrow j$, a cost of one would be required representing a separate instruction (for example *adrk* or *sbrk* in the C2x[6], see figure 1e)). In figure 1b), flows in the solution of the minimum cost circulation problem, providing optimal address generation for code in figure 1a), are shown. Only the arcs with non-zero flow in the solution appear for illustration purposes. The total flow from vertex *S* to *T* and vica versa is two, thus the solution requires two index registers in figure 1b) ($at0$,$ar1$). Figure 1c) provides the actual code corresponding to the solution shown in the graph in figure 1b). There is always an additional cost of one for the C2x processor in any solution of the minimum cost circulation problem, representing the instruction *larp*0 which identifies the current index register[6].

The formulation of the address generation problem as a minimum cost circulation problem is shown below. For illustration purposes the costs for the auto-increment/decrement addressing problem will be formulated below using the TMS320C2x DSP processor[6] (C2x). However in general other auto-increment/decrement addressing modes used in other DSP processors can also be supported. The cost for using an offset whose value is greater than one is one instruction in the C2x instruction set $e_{v_t \rightarrow u_{t2}}=1$, $\forall t \leq t2$, $v_t, u_{t2} \varepsilon V$, $|d(v)-d(u)|>1$. The cost for using each index register is the cost to load each index register with an initial value which again is one instruction in the C2x instruction set ($e_{T \rightarrow S}=1$). Equation (1) represents the

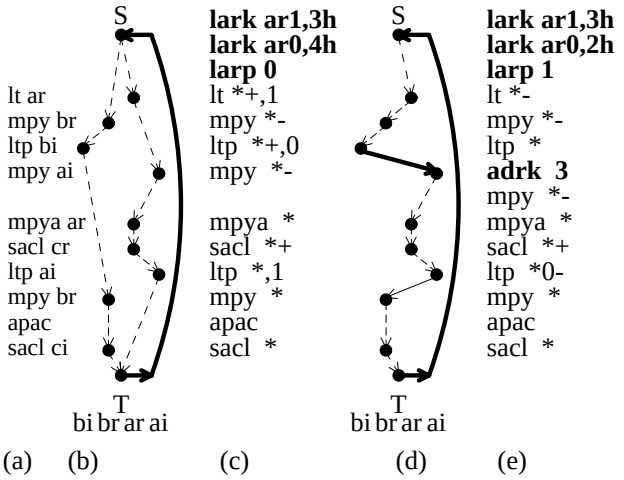

Figure 1. The code in a) is transformed into a flow graph with optimal min cost circulation in b) and C2x code in c). A circulation in d) with code in e) using offset register $ar0$ and *adrk*.

conservation of flow equation (including vertices *S*,*T*). Equation (2) ensure that the flow into and out of each data access variable, is equal to one. Finally equation (3) sets the arc capacities and lower bounds. Alternatively, equation (2) can be transformed into a pure circulation problem as presented in section 3. In this case each data access vertex in the graph is replaced by an arc whose lower bound is set to one (which has the same effect as equation (2)). This lower bound along with the conservation of flow inequality (1) is used along with inequality (3) to formulate the circulation problem.

$$Minimize \sum_{i \rightarrow j} e_{i \rightarrow j} x_{i \rightarrow j}$$

$$Subject\ to \sum_{i|i \rightarrow j} x_{i \rightarrow j} - \sum_{i|j \rightarrow i} x_{j \rightarrow i} = 0,\ \forall j \varepsilon V. \quad (1)$$

$$\sum_{i|i \rightarrow v_t} x_{i \rightarrow v_t} = 1,\ \sum_{j|v_t \rightarrow j} x_{v_t \rightarrow j} = 1,\ \forall v_t \neq S,T, v_t \varepsilon V. \quad (2)$$

$$1 \leq x_{T \rightarrow S} \leq |AR|,\ 0 \leq x_{i \rightarrow j} \leq 1,\ \forall (i \rightarrow j) \varepsilon A, (i \rightarrow j) \neq (T \rightarrow S). \quad (3)$$

This model can easily be extended to generate addressing code that additionally supports a fixed non-zero *offset* ($offset>1$) that is loaded into a index register and used to increment the address stored in any other index register by the *offset* amount (such as that capability provided in C2x processor with the $AR0$ register[6], $ARi+=AR0$, see figure 1e) *0+). In this case we extend the cost formulation as follows, for *offset*: $e_{v_t \rightarrow u_{t2}}=0$, $\forall t \leq t2$, $v_t, u_{t2} \varepsilon V$, $|d(v)-d(u)| \leq 1$, $or\ |d(v)-d(u)|=offset$. $e_{v_t \rightarrow u_{t2}}=1$, $\forall t \leq t2$, $v_t, u_{t2} \varepsilon V$, *otherwise.* Application to different types of data structures (arrays, pointers, etc) can also be supported. Although the extension of this circulation technique to loops and conditionals in general is NP-complete (ie. the multicommodity flow problem), an iterative application of the circulation techniques can be performed to handle these cases and achieve locally optimal solutions.

5. Experimental Results

Several DSP applications are used to illustrate this methodology. Code was generated using the Texas Instrument's (TI) C compiler[6] for the TMS320C2x and C3x DSP processors (which have very different addressing capabilities). The minimum cost circulation was solved on a Sun using a LP solver[8], although faster cpu times are possible using network solvers[7].

Table 1 illustrates the optimized results compared to the TI compiler generated addressing. Optimal addressing code (#instr) was generated for compiler generated DSP code using the same memory layout as the compiler. Results were compared with the initial code generated from TI compiler (CS, code size) which included the addressing code (#instr). Therefore the improvement (Impr) in performance and code size shown in table 1 is only due to optimal address generation. The cpu run times were all under 3 seconds. The last two rows in table 1 provided 1.5 times improvement in code size for the C3x processor where the optimized use of indirect addressing provided greater opportunity for implementation of parallel instructions.

Table 1. Optimized vs. Compiler-Generated Addressing Code

Ex	TI Compiler		Optimized		Impr
	CS	#instr	CS	#inst	
hp1	87	39	54	6	1.6
hp2	78	30	54	6	1.4
lms	79	36	49	6	1.6
fft	183	69	123	9	1.4
dct	210	72	149	11	1.4
fft/3x	93	34	60	10	1.5
dct/3x	69	34	46	15	1.5

Table 2. Different Memory layouts vs Address Generation

# of Index Regs	1	2	3	4
#instr for TI-mem	24	13	8	7
#instr for Krusk-mem	9	6	5	5

Table 2 presents the results of using different memory layouts (TI-generated, TI-mem, and as in [2], Krusk-mem) on the address generation problem for a variation of the least means square algorithm. For a fixed number of index registers, the optimal size of addressing code (#instr) is shown. To further analyze the impact of memory layout on optimal address code generation, the complement of the data access graph was used to obtain a poor memory layout. The optimal address generation technique was applied and the results for several examples (taken from table 1) were an additional cost of at most two instructions.

6. Discussions and Conclusions

In summary code size and performance savings from 1.3 to 1.6 (see table 1) were attained by optimizing the address generation code across several DSP examples. The technique presented in this paper performs optimal address code generation for a given memory layout. Since fewer instructions are used, a reduction in energy dissipated will also be obtained with this technique. Since there may be more than one solution which provides optimal performance and code size, the methodology would then solve for minimum energy using instruction-level models of estimated energy dissipation, as researched for general purpose processors in [5]. This problem can be solved in polynomial time using efficient network flow solvers. Since this address generation is dependent upon the memory layout, this could be used in conjunction with a search technique to find optimal memory layout and address generation. In combination with optimized code generation tools[10], larger savings in code size and improvements in performance are attainable.

In contrast to previous research[2,9,4] which examined the general offset assignment problem or other addressing techniques, we have presented a optimal polynomial technique which can work in conjunction with any data memory layout technique such as in[2] or with memory layout generated by a compiler. Results show that memory layout has a small effect on code size and performance when optimal addressing code is used. This may also be advantageous when memory layout is constrained by interfacing with external systems or when it is performed by an algorithm the user has selected and does not wish to change. It also allows a decomposition approach, or task by task approach to code generation since one can fix memory layout at the beginning of a task according to what was used in previous tasks in contrast to [2] which cannot support a fixed memory layout. This provides a value-added advantage where code can be quickly generated by a compiler and optimized for addressing without changing the memory layout. We have introduced a methodology for optimal address generation given memory layout that has provided significant improvements in performance and code size across several DSP applications. The author would like to thank Craig Ranta for his work. This research is supported in part by grants from NSERC and ITRC.

References

[1] P.Marwedel, G.Goossens Eds. **Code Generation for Embedded Processors**, Kluwer Acad Pub, 1995.

[2] S.Liao et al. "Storage Assignment to Decrease Code Size" ACM SIGPLAN Conf Prog. Lang Des and Impl, 1995.

[3] S.Wuytack et al. "Power Exploration for Data Dominated Video Applications", ISLPED, p359-364, 1996.

[4] C.Liem, P.Paulin, A.Jerraya "Address Calculation for Retargetable Compilation and Exploration of Instruction-Set Architectures" DAC, 1996.

[5] V.Tiwari, S.Malik,A.Wolfe, "Power Analysis of Embedded Software ;A First Step Towards Software Power Minimization, IEEE Trans on VLSI, Vol. 2, No. 4, Dec 1994, p437-445,

[6] **TMS320C2x User's Guide**, Texas Instruments Inc., 1993.

[7] E.Lawler **Combinatorial Optimization: Networks and Matroids** Holt, Rinehart and Winston, 1976.

[8] Brooke, Kendrick, Meeraus, "GAMS", Scientific Press, 1988.

[9] R.Leupers, P.Marwedel, "Algorithms for Address Assignment in DSP Code Generation", ICCAD 1996, 109-112.

[10] C.Gebotys, "An Efficient Model for DSP Code Generation: Performance, Code Size, Estimated Energy", Proc of Int'l Symp on Sys Synth, Sept 1997.