LOW ENERGY REGISTER ALLOCATION BEYOND BASIC BLOCKS

Yumin Zhang, Xiaobo (Sharon) Hu, Danny Z. Chen

Department of Computer Science and Engineering
University of Notre Dame
Notre Dame, IN 46556, USA
{yzhang1, shu, chen}@cse.nd.edu

ABSTRACT

An approach of doing register allocation beyond basic blocks for low energy is presented in this paper. With careful analysis of boundary conditions between consecutive blocks, our approach achieves the allocation results benefiting the whole program. By allowing the allocation results to propagate down block by block without backtracking, we avoid excessive computational cost.

1. INTRODUCTION

Memory access is a major source of power consumption in many computer systems. For some data-intensive applications, energy consumption due to memory access can be more than 50% [6]. Since access to different types of memory components, such as register, cache, main memory and magnetic disk, differs dramatically in speed and power consumption [7], appropriately allocating data variables in a program to different memory elements can play an important role toward achieving high performance and low energy consumption [4]. In this paper, we study the problem of partitioning variables between registers and memory and mapping variables to registers (referred to as the register allocation problem) for the purpose of achieving both short execution time and low energy consumption.

Most existing register allocation techniques have focused on reducing program execution time on a given system (e.g., [3, 5]). However, an optimal-time register allocation does not necessarily consume the least amount of energy [8]. Some researchers have proposed approaches to deal with the optimal-energy register allocation problem [1, 2, 4, 10]. Chang and Pedram [2] gave an approach to find the lowest-energy assignment of variables to registers. Their techniques cannot be readily applied to the register allocation problem since they do not consider the partitioning between registers and memory. Gebotys [4] presented algorithms for determining both the partition between registers and memory and the assignment of variables to registers in order to minimize energy consumption. Nonetheless, these algorithms only consider variables within a piece of straight-line code called a basic block.

In this paper, we extend the work in [4] by investigating the register allocation problem for low energy beyond basic blocks. In particular, we consider the allocation of variables in a program with branches. As a fundamental structure, branches make up about 15% of instructions in a program [7]. Furthermore, loops and procedure calls can be considered as branches with additional features. We have developed algorithms for handling the boundary conditions between consecutive blocks. With our approach, one can greatly reduce the computational cost due to examining all possible paths of execution in a program and resolving any conflicting assignments that may arise. Furthermore, by carefully handling those variables used in multiple blocks, the resulting energy consumption is much lower than simply applying the algorithms in [4] separately to each basic block.

This work was supported in part by the National Science Foundation under Grant CCR-9623585 and MIP-9701416, and an External Research Program Grant from Hewlett-Packard Laboratories, Bristol, England.

2. PRELIMINARIES

Many optimization techniques carried earlier than register allocation in a compiler can affect the register allocation results. In this paper, we assume that such optimization techniques have been applied and a satisfactory schedule of the program is already given.

We assume that there are k registers in the system. Let $V = \{v_1, v_2, \ldots, v_n\}$ $(n > k)$ be the set of variables in a given program, and V_R/V_M represent the set of variables assigned to registers/memory. Furthermore, let $e^R_{r/w}$ (resp. $e^M_{r/w}$) be the energy consumed by reading/writing a variable from register (resp. memory). The total energy consumed by a program is the sum of energy consumed by register accesses and memory accesses. Then, the register allocation problem we consider can be formulated as one of partitioning V into V_R and V_M such that $V_R \cup V_M = V$ and mapping each variable $v \in V_R$ to one of the k registers to achieve the least energy consumption due to data accesses.

Different energy and data access models have been proposed in the literature [2, 4, 8]. The calculation of energy consumption depends on the particular models used. A simple model, called static energy model, assumes that references to the same type of storage component consume the same amount of energy regardless of the actual data. Under this model, optimizing energy consumption is equivalent to minimizing the number of accesses to memory [11], which is also equivalent to optimizing execution time.

A more sophisticated energy model, called activity based energy model, is proposed in [2, 8]. Under this model, the energy consumed by a program is dependent on the switched capacitance of successive accesses of data variables that share the same storage location. The Hamming distance between two subsequent data items in the same storage location has been used to compute the total switched capacitance between two consecutive accesses to the same location. Here we will only discuss the activity based energy model for assessing the register file in order to simplify the problem formulation. (Adopting the activity based model for memory is a simple extension to the algorithm discussed later in this paper.)

Let $\overline{H}(v_i, v_j)$ be the average Hamming distance between vari-

ables v_i and v_j, C^R_{rw} be the average switched capacitance per bit by the register access operations, and $\mathcal{V}$ be the operational voltage. Then, the total energy consumption of a program can be calculated as $E = N^M_r e^M_r + N^M_w e^M_w + \sum_{v_i \rightarrow v_j, v_i, v_j \in V_R} \overline{H}(v_i, v_j) C^R_{rw} \mathcal{V}^2$, where N^M_r and N^M_w are the total numbers of read and write accesses to memory, and $v_i \rightarrow v_j$ indicates that a register content switches from v_i to v_j.

Techniques for minimizing energy consumption based on the above model were proposed by Gebotys in [4]. Her approach transforms the problem to an instance of the network flow problem. However, only variables within a single basic block are considered in [4]. Programs from real application unavoidably contain branches and some other control structures. In this paper, we focus on solving the register allocation problem for programs containing branches. Though our approach does not cover all possible structures of programs, it is a basic step towards solving a general energy-optimal register allocation problem.

One way to solve the register allocation problem is simply applying the existing algorithms in [4] to each basic block in a program. However, without considering the relations between different blocks, such an approach may lead to many unnecessary memory/register references at blocks boundaries [11]. Another approach is to enumerate all possible execution paths in a given program and apply the algorithms in [4] to each path. However, the allocation results from different paths can potentially be different from each other for the same variable. Resolving these conflicts is not a trivial task as it requires examining the global structure again.

In the rest of the paper, we discuss an efficient heuristic which can find a nearly optimal solution for low energy register allocation. Our heuristic can work with both energy models discussed earlier. Due to the page limit, we will only discuss our results for the activity based energy model.

3. OUR APPROACH

To avoid the problems encountered by the known approaches discussed in the above section, we propose an iterative approach to handle programs containing branches.

A program with branches can be modeled as a tree T and each basic block as a node B in T. Let the parent block of B be B_p, and one of its children be B_c. We solve the register allocation problem for each B in the depth-first search order of T. A main challenge is how to handle those variables whose lifetimes extend beyond a basic block such that the overall energy consumption of the program is minimized. We have made several observations to help deal with such variables. The key idea is to set up appropriate boundary conditions between parent and child basic blocks. We use the allocation result from B_p as an initial assignment to guide the allocation of B. By allowing the allocation results to propagate down the tree T without back-tracking, we eliminate the excessive computational cost yet obtain superior allocations.

We first describe our graph model, which is an extension to the model introduced in [4]. For each basic block B in T, there is a generalized directed interval graph $G = (N, A)$ associated with it. Typically, there are three kinds of arcs in G. The first kind of arcs, $a(v)$, are those associated with each variable $v \in B$. (Note that v can be either a variable referenced in B or a variable which will be referenced by B's descendents.) We denote the two end nodes of $a(v)$ by $n_s(v)$ and $n_f(v)$, which correspond to the starting time $t_s(v)$ (when v is first written) and the finishing time $t_f(v)$ (when v is last read), respectively.

The second kinds of arcs are those going in between some variables. To define this kind of arcs, we first define some sets of variables. Let a critical set, C_i, be a set of variables with overlapping lifetimes to one another such that the number of variables in the set is greater than the number k of available registers. For each pair of C_i and C_{i+1} (the index ordering is based on scanning the variables from the beginning to the end of block B), let D_i be the set of variables whose lifetimes are in between the minimum finishing time of all variables in C_i and the maximum start time of all variables in C_{i+1}. Note that D_0 contains the variables whose lifetimes are in between the start time of B and the maximum start time of all variables in C_1, and that D_g is the set of variables whose lifetimes are in between the minimum finish time of all variables in C_g (the last critical set in B) and the finish time of B. Now, an arc, which belongs the the second kind, is introduced from $n_f(u)$ for each $u \in (C_i \cup D_i)$ to $n_s(v)$ for each $v \in (D_i \cup C_{i+1})$ such that $t_f(u) < t_s(v)$. Intuitively, these arcs represent allowable register sharing among subsequent data reference.

In [4], a source node, S, and a finish node, F, are introduced at the beginning and end of B, respectively. The third kind of arcs are used to connect S to $n_s(v)$ for each $v \in (D_0 \cup C_1)$ and connect $n_f(u)$ for each $u \in (C_g \cup D_g)$ to F. The nodes S and F can be considered as the registers available at the beginning and end of B.

To handle the allocation beyond a block boundary, our approach is to decide the register allocation for block B based on the allocation result from B's parent block, B_p. Depending on which variable is assigned to which register prior to entering B, the amount of energy consumption by each variable in B can be different. Therefore, simply using a single source S to represent the registers available at the beginning of B is no longer sufficient. We need to introduce a forth kind of arcs to represent the different register sources for the variables in $D_0 \cup C_1$.

The construction of graph G for B, where B has both a parent and at least one child, is generalized as follows. (The graphs for the root and leave blocks in T are simply special cases of G.)

For those variables that are referenced only in B, we introduce arcs and nodes associated with them in the same way as we discussed above. The start and finish nodes, S and F, for B are also maintained. Let the starting and finishing times of block B be $t_s(B)$ and $t_f(B)$, respectively. For each variable v in B whose starting (resp. finishing) time is earlier (resp. later) than the starting (resp. finishing) time of B, i.e., $t_s(v) < t_s(B)$ (resp. $t_f(v) > t_f(B)$), we still use two end nodes $n_s(v)$ and $n_f(v)$ in G for the associated arc $a(v)$ (which means that v is considered by all the graphs corresponding to the blocks with which the lifetime of v overlaps). The arcs between these nodes and S or F are defined in the same ways as discussed in the previous paragraphs. Furthermore, we introduce a register set, V_{RB}, which contains the variables residing in registers at the completion of the allocation process for block B. Note that V_{RB} becomes uniquely defined after the allocation process of B is completed, and that the size of V_{RB}, $|V_{RB}|$, is always less than or equal to k, the number of available registers. We expand G by adding a node $n_p(v)$ for each $v \in V_{RB_p}$, where B_p is the parent block of B. It is not difficult to see that the variables in V_{RB_p} are the only variables which have a chance to pass on their register locations to variables in B directly. Now, we insert an arc, which is the forth kind, from each $n_p(u)$ to $n_s(v)$ for each $v \in (D_0 \cup C_1)$ (where D_0 and C_1 for block B are as defined in the previous paragraphs). Comparing our generalized graph with a typical network flow graph, one can see that at most k additional nodes and $k \cdot |D_0 \cup C_1|$ additional arcs are used in

the generalized graph. Figure 1 shows an example graph G for the current block B assuming that there are three available registers.

Sometimes, a program may read a variable, v, more than once, In this case, we introduce an additional node $n_{r_i}(v)$ for each read of v except the last read. Additional arcs are also introduced to model possible register sharing between variables. Due to the page limit, we omit the discussion on this part.

Given the directed graph G for B, we are ready to construct the network flow problem associated with G. Let $x(n_f(u), n_s(v))$ and $c(n_f(u), n_s(v))$ be the amount of flow and the cost of one unit of flow on arc $a(n_f(u), n_s(v))$, respectively. Denote the total amount of energy consumed by B as E. The objective function of our network flow problem can be written as:

$$\text{Minimize: } E = \sum_{v \in B}(e_{rv}^M + e_{wv}^M) \quad - \sum_{v \in B | t_s(v) < t_s(B)} e_{wv}^M$$
$$- \sum_{v \in B | t_f(v) > t_f(B)} e_{rv}^M \quad - \sum_{a(p,q) \in A} c(p,q) \cdot x(p,q)$$

where A is the set of arcs in G, the first three terms are the amount of energy consumed by B if all the variables are assigned to memory, and the last term represents the energy saved by allocating certain variables to registers. The values of $x(p, q)$ are unknown and to be determined. If $x(p, q) = 1$ and arc $a(p, q)$ corresponds to a variable v, then v will be assigned to a register. The values of $c(p, q)$ are dependent on the types of arcs associated with them, and can be categorized into the following cases.

1) For an arc from a node of type n_f to another node of type n_s, i.e. $a(n_f(u), n_s(v))$, the cost associated to the arc is computed by $c(n_f(u), n_s(v)) = e_w^M + e_r^M - \overline{H}(u, v) C_{rw}^R \mathcal{V}^2 \; \forall u, v \in N$, where N is the set of nodes in G. This is the amount of energy saved by reading u from a register and writing v to the same register.

2) For an arc from a node of type n_p to another node of type n_s, i.e. $a(n_p(u), n_s(v))$, the cost associated to the arc is defined differently. There are a total of 7 cases to be considered.

2.1) If u is not in B(i.e., u's lifetime does not overlap with that of B), and v is written in B, $c(n_p(u), n_s(v)) = e_w^M - \overline{H}(u, v) C_{rw}^R \mathcal{V}^2$.

2.2) If u is not in B, and v has been assigned to a register in B_p, $c(n_p(u), n_s(v)) = -\overline{H}(u, v) C_{rw}^R \mathcal{V}^2$.

2.3) If u is not in B, and v has been assigned to memory during the allocation process of B_p, $c(n_p(u), n_p(v)) = -e_r^M - \overline{H}(u, v) C_{rw}^R \mathcal{V}^2$.

2.4) If u is in B, and v is written in B, the cost $c(n_p(u), n_s(v))$ is the same as defined for Case 2.2.

2.5) If u is in B, and v has been assigned to a register during the allocation process of B_p, $c(n_p(u), n_s(v)) = -e_w^M - \overline{H}(u, v) C_{rw}^R \mathcal{V}^2$.

2.6) If u is in B, and v has been assigned to memory during the allocation process of B_p, $c(n_p(u), n_p(v)) = -e_w^M - e_r^M - \overline{H}(u, v) C_{rw}^R \mathcal{V}^2$.

2.7) If u and v represent the same variable, the cost $c(n_p(u), n_s(v))$ is simply assigned to zero.

3) For an arc from start node S to another node of type n_s, i.e. $a(S, n_s(v))$ for $v \in D_0 \cup C_1$, we need to have three different cost functions.

3.1) If v is written in B, $c(S, n_s(v)) = e_w^M - \overline{H}(0, v) C_{rw}^R \mathcal{V}^2$,where $\overline{H}(0, v)$ is the average Hamming distance between 0 and a variable v, and is normally assumed to be 0.5.

3.2) If v has been assigned to a register during the allocation process of B_p, $c(S, n_s(v)) = -\overline{H}(0, v) C_{rw}^R \mathcal{V}^2$.

3.3) If v has been assigned to memory during the allocation process of B_p, $c(S, n_s(v)) = -e_w^M - \overline{H}(0, v) C_{rw}^R \mathcal{V}^2$.

4) For an arc from a node of type n_f to the finish node, F, i.e. $a(n_f(v), F)$ for $v \in D_g \cup C_g$, we need to have two different cost functions.

4.1) If v is read in B, $c(n_f(v), F) = e_r^M$.

4.2) If v is not read in B, the cost $c(n_f(v), F)$ is simply assigned to zero.

5) For an arc from a node of type n_s to another node of type n_f, which is the arc corresponding to the same variable, the cost associated to the arc is assigned to zero.

Using the above equations, the objective function for the network flow problem will be uniquely defined. The constrains include the capacity constraint and flow conservation for a network flow problem. Additionally, the total flow into the graph G should be less than the total number of available registers.

Applying a network flow algorithm such as the one in [9] to our network flow problem instance, we can obtain the value of each $x(p, q)$ in $O(kn^2)$ time for the block B, where k is the number of available registers and n is the number of variables whose lifetimes overlap with the lifetime of B. If $x(n_s(v), n_f(v))$ for v in B is one, the variable v is assigned to the appropriate register based on the flow information. The above formulation can then be applied to each basic block in the program tree in the depth-first search order as we discussed at the beginning of this section.

Here we should point out that our method can also be used if one wishes to explore program execution paths to determine the register allocation. For example, we can associate each execution path of a program with a priority. By applying our algorithm to different paths following the priority order, the variable allocation decisions made for the higher priority paths will not be changed by the allocation processes of the lower priority paths. Rather, the results from the higher priority paths are used to insure that good allocations are found for the lower priority paths based on these results. Hence, we have effectively eliminated the conflicting assignments that can arise when solving the allocation problem for each path separately. For more details, see [11].

4. EXAMPLE

Our algorithm is simple to implement and thus more efficient in both running time and space cost than other global register allocation algorithms. We have carried out several examples and the results are very promising. Due to the page limit, we only show a small example here.

The dataflow of an example program containing a branch is shown in Figure 2. Each line segment in the figure indicates the lifetime of a variable. Note that variables c and d are referenced in more than one block and c is referenced in two different execution paths. We assume that there are two general purpose data registers. Applying our algorithm, we begin with B_1, proceed to B_2 and then B_3. The allocation result is shown in Figure 2, where the variables represented by solid line segments are allocated to registers and the variables represented by the dashed lines are assigned to memory.

Execution Paths	Energy differences
100% B_1-B_2	$e_r^M + e_r^M$
100% B_1-B_3 only	-$(e_r^R + e_w^R)$
%20 B_1-B_2	0.2$(e_r^M + e_w^M)$-0.8$(e_r^R + e_w^R)$
%50 B_1-B_2	0.5$(e_r^M + e_w^M)$-0.5$(e_r^R + e_w^R)$
%80 B_1-B_2	0.8$(e_r^M + e_w^M)$-0.2$(e_r^R + e_w^R)$

Table 1: Energy consumption differences due to the exhaustive approach and our approach.

If the register allocation were done through enumerating all execution paths, we would need to perform allocation for each of the two possible paths separately. Unfortunately, the allocation result for c in different paths are different. In the path containing B_1 and B_2, c was assigned to a register, while c was assigned to memory in the other path. The conflicting results is not easy to resolve in a real program.

For the above example, we compare the energy dissipation based on our allocation algorithm and by enumerating all paths. We summarize in Table 1 the differences in the energy dissipation (due to accessing only data) during executing each path and during executing the entire program with varying probabilities of traversing each path. Three probability combinations are included in Table 1. By considering the fact that memory references consume much more energy than register references, the energy consumption based on our algorithm is less than that based on the exhaustive approach, except when the program mainly exercises B_1 and B_3. If this is the case, we can use the priority-based approach discussed at the end of the last section to eliminate the conflicting assignments. Hence, our algorithm is quite powerful for solving the register allocation problem for programs with branches.

5. CONCLUSION

In this paper, we proposed an algorithm for optimal energy register allocation beyond basic blocks. Our algorithm has much less time and space complexity than other approaches. The results so far are very promising and the result for the example in Section 4 shows this clearly. We are now investigating on how to deal with more complex control flow structures, such as loops and procedure calls.

6. REFERENCES

[1] A.P. Chandrakasan, M.P. Potkonjak, R.Mehra, J. Rabaey, and R.W. Brodersen, "Optimizing power using transformations," *IEEE Transactions on CAD of Integrated Circuits and Systems*, vol. 14, no. 1, January 1995, pp. 12-30.

[2] J. Chang and M. Pedram, "Register allocation and binding for lower power," *Design Automation Conference*, 1995, pp. 29-35.

[3] F.C. Chow and J.L. Hennessy, "The priority-based coloring approach to register allocation," *ACM Transactions on Programming Languages and Systems*, vol. 12, no. 4, October 1990, pp. 501-536.

[4] C.H. Gebotys, "Low energy memory and register allocation using network flow," *Design Automation Conference*, 1997, pp. 435-440.

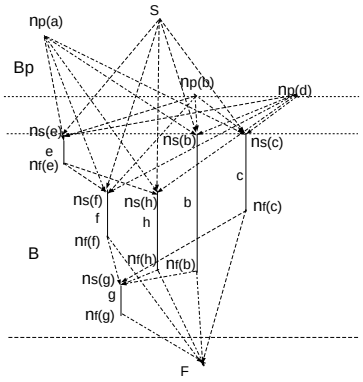

Figure 1: The graph G for block B based on the allocation of its parent block B_p.

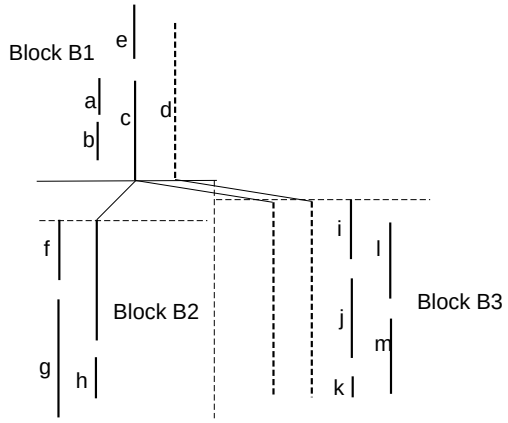

Figure 2: Global allocation result by our algorithm.

[5] U.I. Gupta, D.T. Lee, and J.Y. Leung, "An optimal solution for the channel-assignment problem," *IEEE Transactions on Computers*, vol. c-28, no. 11, November 1979, pp. 807-810.

[6] J. Henkel and Y. Li, "Energy-conscious HW/SW-partitioning of embedded systems: a case study on an MPEG-2 encoder" *Proceedings of Sixth International Workshop on Hardware/Software Codesign*, March, 1998, pp. 23-27.

[7] J.L. Hennessy and D.A. Patterson, *Computer Architecture: A Quantitative Approach*, 2nd edition, Morgan Kaufmann Publishers, 1996.

[8] P.E. Landman and J. Rabaey, "Activity-sensitive architectural power analysis," *IEEE Transactions on CAD of Integrated Circuits and Systems*, vol. 15, no. 6, June 1996, pp. 571-587.

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

[10] M. Pedram, "Power minimization in IC design: Principles and applications," *ACM Transactions on Design Automation of Electronic Systems*, vol. 1, no. 1, January 1996, pp. 3-56.

[11] Y. Zhang, X.S. Hu, and D.Z. Chen, "Low energy register allocation beyond basic blocks," Technical Report 98-22, Computer Science and Engineering, University of Notre Dame.