Hardware-Software Bipartitioning for Dynamically Reconfigurable Systems*

Daler N. Rakhmatov and Sarma B.K. Vrudhula
Center for Low Power Electronics
ECE Department, University of Arizona
Tucson, AZ 85721
daler/sarma@ece.arizona.edu

ABSTRACT

The main unique feature of dynamically reconfigurable systems is the ability to time-share the same reconfigurable hardware resources. However, the energy-delay cost associated with reconfiguration must be accounted for during hardware-software partitioning. We propose a method for mapping nodes of an application control flow graph either to software or reconfigurable hardware, explicitly targeting minimization of the energy-delay cost due to both computation and configuration. The addressed problems are energy-delay product minimization, delay-constrained energy minimization, and energy-constrained delay minimization. We show how these problems can be tackled by using network flow techniques, after transforming the original control flow graph into an equivalent network. If there are no constraints, as in the case of the energy-delay product minimization, we are able to generate an optimal solution in polynomial time.

Keywords: hardware-software partitioning, reconfigurable systems, network flows.

1. INTRODUCTION

To achieve greater flexibility, reduced costs and longer product life, the trend is to use configurable or alterable components in the design of embedded systems. To respond to changes in an application, such a system is reconfigured rather than redesigned and rebuilt. Tight coupling of the hardware and software in reconfigurable embedded systems requires extensive automated support for efficient hardware-software mapping of an application, with an explicit account for a new penalty metric associated with reconfiguration. In this paper we address the problem of hardware-software bipartitioning, which in our case is synonymous with hardware-software mapping of an application.

*This work was carried out at the National Science Foundation's State/Industry/University Cooperative Research Centers' (NSF-S/IUCRC) Center for Low Power Electronics (CLPE). CLPE is supported by the NSF (Grant EEC-9523338), the State of Arizona, and a consortium of companies from the microelectronics industry (visit the CLPE web site http://clpe.ece.arizona.edu).

CODES'02, May 6-8, 2002, Estes Park, Colorado, USA.

The assumed target architecture of a dynamically reconfigurable digital processor includes three key components: a software unit (a microprocessor), a dynamic hardware unit (a reprogrammable logic device), and a shared memory unit (communication link between software and hardware). The software can directly configure the hardware, which is partially reconfigurable. Partial reconfiguration allows for a selective change of hardware segments of arbitrary size at an arbitrary location, without disrupting the operation of the rest of the hardware space. Such a capability greatly reduces reconfiguration time and energy consumption, because the hardware updates are highly localized.

An application is represented by a control-flow graph (CFG), and our goal is to assign each graph node, or basic block, to either the hardware or the software. Usually, the hardware implementation is more energy-delay efficient than the software implementation; however, the former invokes a hardware reconfiguration penalty in terms of both energy and delay. In this paper, we address three types of problems: (1) energy-delay product minimization, (2) energy minimization under the delay constraint, and (3) delay minimization under the energy constraint. We show how to tackle these problems by using network flow techniques. The unconstrained minimization is solved optimally in polynomial time, and methods are presented for systematically exploring near optimal solutions for the constrained problems.

Although the close link between graph bipartitioning and network flows is well known, the applicability of network flows to our case is not obvious. Our cost function involve costs of all nodes and all edges in the CFG, and not just the edges in the cut-set separating software-mapped nodes and hardware-mapped nodes. Specifically, the cost of a node depends whether it is in software or in hardware, and the cost of an edge depends whether its origin node is in software or in hardware and whether its destination node is in software or in hardware.

2. PROBLEM DESCRIPTION

Our approach directly targets energy-delay minimization in the dynamically reconfigurable computing environment. The key assumptions are as follows: (1) the target architecture consists of a microprocessor core and a dynamically reconfigurable logic core, communicating through a shared memory core; (2) an application is represented by a control flow graph; (3) hardware-software bipartitioning is performed at the basic block level; (4) basic blocks are executed sequentially, in accordance with the control flow; (5) reconfiguration events take place during control transfers; (6) partial reconfigurability is utilized to reduce the amount of configuration; (7) the energy-delay penalties due to both computation and reconfiguration are taken into account; (8) the application delay is

defined as a weighted sum of computation delays of all the basic blocks and reconfiguration delays of all control transfers; (9) the application energy is defined as a weighted sum of computation energies of all the basic blocks and the reconfiguration energies of all control transfers; (10) the weights mentioned in (8) and (9) (for energy or delay) are execution frequencies, which are obtained from application profiling data.

Let $\mathcal{G}(V,E)$ denote the control flow graph subject to hardware-software bipartitioning. Nodes in V correspond to the basic blocks and the edges in E represent control transfers between the blocks. Associated with each node $i \in V$ is a partition variable defined as $x_i = 0,\ i \mapsto software$ and $x_i = 1,\ i \mapsto hardware$.

Each node $i \in V$ is associated with the cost $c_0(i)$ and the weight $w_0(i)$, if i is mapped to software, as well as the cost $c_1(i)$ and the weight $w_1(i)$, if i is mapped to hardware. Each edge $(i,j) \in E$ is associated with one of the following four costs and one of the following four weights, respectively, depending on the mapping of the nodes i and j:

$$\begin{array}{l} c_{00}(i,j) \text{ and } w_{00}(i,j), \text{ if } \overline{x}_i\overline{x}_j = 1; \\ c_{01}(i,j) \text{ and } w_{01}(i,j), \text{ if } \overline{x}_i x_j = 1; \\ c_{10}(i,j) \text{ and } w_{10}(i,j), \text{ if } x_i\overline{x}_j = 1; \\ c_{11}(i,j) \text{ and } w_{11}(i,j), \text{ if } x_i x_j = 1. \end{array}$$

The cost/weight can be either the energy or the delay or the energy-delay product of a node/edge, weighted by its execution frequency. We make two assumptions about the edge costs and weights. First, transferring control from a hardware block to a software block is more expensive than transferring control from a software block to a software block. This is realistic since a hardware-to-software transition may cause extra data traffic. Second, transferring control from a software block to a hardware block is more expensive than transferring control from a hardware block to a hardware block. This is also realistic since a hardware-to-hardware transition may be less expensive due to partial reconfigurability. These two assumptions are restated below:

$$\begin{array}{l} c_{10}(i,j) \geq c_{00}(i,j) \ \text{ and } \ w_{10}(i,j) \geq w_{00}(i,j); \\ c_{01}(i,j) \geq c_{11}(i,j) \ \text{ and } \ w_{01}(i,j) \geq w_{11}(i,j). \end{array} \quad (1)$$

We define the objective function F and the constraint function G, respectively, as follows:

$$F = \sum_{i \in V} f(i) + \sum_{(i,j) \in E} f(i,j) \ \text{ and } \ G = \sum_{i \in V} g(i) + \sum_{(i,j) \in E} g(i,j). \quad (2)$$

where
$f(i) = x_i c_1(i) + \overline{x}_i c_0(i)$,
$g(i) = x_i w_1(i) + \overline{x}_i w_0(i)$,
$f(i,j) = x_i x_j c_{11}(i,j) + x_i \overline{x}_j c_{10}(i,j) + \overline{x}_i x_j c_{01}(i,j) + \overline{x}_i \overline{x}_j c_{00}(i,j)$,
$g(i,j) = x_i x_j w_{11}(i,j) + x_i \overline{x}_j w_{10}(i,j) + \overline{x}_i x_j w_{01}(i,j) + \overline{x}_i \overline{x}_j w_{00}(i,j)$.

In the unconstrained hardware-software bipartitioning problem, the task is to find an assignment of each partitioning variable x_i, such that the sum of costs over all nodes and all edges (*the cost F of a bipartition*) is minimized. The energy-delay product minimization is the unconstrained bipartitioning problem, with the cost defined in terms of the energy-delay product.

The constrained hardware-software bipartitioning problem requires finding an assignment of each partitioning variable x_i such that the objective function F is minimized, and the sum of weights over all nodes and all edges (*the weight G of a bipartition*) does not exceed some given budget B. The delay-constrained energy minimization or energy-constrained delay minimization is the constrained bipartitioning problem with the cost and the weight defined in terms of energy or delay, accordingly.

3. RELATED WORK

In this section we briefly justify uniqueness of our problem and contribution of our approach, compared to previously published research in two related areas: circuit partitioning and hardware-software cosynthesis.

Circuit Partitioning: The circuit bipartition cost is usually defined as the number of edges crossing the cut that divides a circuit into two parts. Given an undirected graph with $|V|$ vertices, it is possible to find its optimal unconstrained bipartition in polynomial time by applying a max-flow algorithm at most $|V|-1$ times [1]. However, once we require the two circuit parts be of approximately equal size, the problem becomes hard, and we have to resort to heuristics [2, 3]. Note that circuit part size constraints are completely different from our constraint, while circuit bipartitioning cost is a special case of our cost function.

In [4], the authors applied network flows to circuit partitioning for reconfigurable FPGAs. The input to the problem is a directed acyclic graph (DAG), and the output is a graph partition such that graph parts can be sequenced in time without violating node dependencies, part sizes are balanced, and the number of cut edges is minimized. Even though our approach is similar (iterative application of a max-flow algorithm), our problem is different: we do not have precedence and balancing constraints for a control flow graph, and our cost function is not limited to cut edges.

Hardware-Software Partitioning: Application mapping can be performed either (1) at the task level (coarse-grain mapping), or (2) at the instruction cluster level (fine-grain mapping). The nature of the mapping problem is determined by the models of a system and an application. In [5, 6, 7] the focus is on synthesizing the architecture itself during coarse-grain partitioning. In [8, 9] the system consists of more than two processing units. Consequently, the mapping problem is different when compared to our case (e.g. scheduling becomes a part of it). The architecture model in [10] is similar to ours; however, the authors consider parallel task scheduling in addition to hardware-software assignment.

In [11, 12], the system architecture consists of a general purpose processor (software) coupled with a custom hardware chip, and the goal is to map program blocks, executed sequentially, to either software or hardware. Software-to-hardware and hardware-to-software communication delays as well as execution delays of software parts and hardware parts are taken into account, and the total hardware area is constrained. However, the extended Kernighan-Lin heuristic in [12] does not show us how to solve the unconstrained problem exactly, and the dynamic programming algorithm in [11] does not account for software-to-software and hardware-to-hardware communication, present in our case. Another work closely related to ours is the NIMBLE compiler [13]. It maps a set of candidate loops (kernels), extracted from an application code, onto the microprocessor core and the programmable logic core. The kernel profiling data (software execution time, hardware execution time, block execution frequencies, and hardware area) as well as the configuration time are incorporated into a global cost function that drives the partitioner toward the smallest total execution delay of all loops. Optimality of the solution is not guaranteed even though no constraints are included (e.g. total energy of all loops).

Our Contribution: For sequentially executed control flow graphs, we generalized the unconstrained bipartitioning problem so that the cost function accounts for computation costs of software and hardware blocks, as well as software-to-software, software-to-hardware, hardware-to-software, and hardware-to-hardware communication costs. Here, our contribution is to show how a CFG with node and edge costs can be transformed into a network, so that a minimum cut in the network corresponds to an optimal bi-

partition of the CFG.

For the constrained version, we define a constraint (weight) function similar to the cost function. Our goal is to map each node to either software or hardware (thus, changing costs and weights of nodes and edges) so that the overall cost is minimized and the overall weight does not exceed the budget. Our heuristic method is also based on network flows. We are able to systematically explore a polynomially bounded set of good solutions for the constrained problems.

Remark: Basic blocks in application control flow graphs can be of any granularity, provided that external control transfers take place only at the end of a basic block. At the lowest level, a basic block is comprised of primitive operations such as additions, multiplications, memory loads, etc., with (un)conditional branches allowed only at the end of the block. At the highest level, basic blocks are functions/procedures.

4. PROPOSED SOLUTION

In this section we show how the unconstrained and constrained bipartitioning problem can be solved by using network flows. First, the original CFG $\mathcal{G}(V,E)$ is transformed into a network $\tilde{\mathcal{G}}(\tilde{V},\tilde{E})$, as follows: (1) two additional vertices, s (source) and t (sink) are added to V obtain $\tilde{V}$, (2) in addition to edges in E, $\tilde{E}$ will have edges (j,i) if $(i,j) \in E$ and $(j,i) \notin E$, (3) for all $i \in V$, edges (s,i) and (i,t) are added to $\tilde{E}$.

Each edge in $\tilde{E}$ is assigned cost and weight capacities $\tilde{c}_F$ and $\tilde{c}_G$, respectively, as follows:

$$\begin{array}{rcl} \tilde{c}_F(s,i) & = & c_1(i) + \sum_{(j,i)\in E} c_{11}(j,i), \\ \tilde{c}_F(i,t) & = & c_0(i) + \sum_{(j,i)\in E} c_{00}(j,i), \\ \tilde{c}_G(s,i) & = & w_1(i) + \sum_{(j,i)\in E} w_{11}(j,i), \\ \tilde{c}_G(i,t) & = & w_0(i) + \sum_{(j,i)\in E} w_{00}(j,i), \\ \tilde{c}_F(i,j) & = & [c_{01}(i,j) - c_{11}(i,j)][(i,j)\in E] \\ & & +[c_{10}(j,i) - c_{00}(j,i)][(j,i)\in E], \\ \tilde{c}_G(i,j) & = & [w_{01}(i,j) - w_{11}(i,j)][(i,j)\in E] \\ & & +[w_{10}(j,i) - w_{00}(j,i)][(j,i)\in E]. \end{array} \quad (3)$$

CFG2Network (V,E)
  $\tilde{V} = V \cup \{s,t\}$, $S_V = \emptyset$, $S_E = \emptyset$
  FOR all $i \in V$
    $S_V = S_V \cup \{(s,i),(i,t)\}$
    $\tilde{c}_F(s,i) = c_1(i)$, $\tilde{c}_F(i,t) = c_0(i)$
    $\tilde{c}_G(s,i) = w_1(i)$, $\tilde{c}_G(i,t) = w_0(i)$
  FOR all $(i,j) \in E$
    $\tilde{c}_F(s,j) = \tilde{c}_F(s,j) + c_{11}(i,j)$, $\tilde{c}_F(j,t) = \tilde{c}_F(j,t) + c_{00}(i,j)$
    $\tilde{c}_G(s,j) = \tilde{c}_G(s,j) + w_{11}(i,j)$, $\tilde{c}_G(j,t) = \tilde{c}_G(j,t) + w_{00}(i,j)$
    IF $\{(i,j),(j,i)\} \notin S_E$
      $S_E = S_E \cup \{(i,j),(j,i)\}$
      $\tilde{c}_F(i,j) = c_{01}(i,j) - c_{11}(i,j)$, $\tilde{c}_F(j,i) = c_{10}(i,j) - c_{00}(i,j)$
      $\tilde{c}_G(i,j) = w_{01}(i,j) - w_{11}(i,j)$, $\tilde{c}_G(j,i) = w_{10}(i,j) - w_{00}(i,j)$
    ELSE
      $\tilde{c}_F(i,j) = \tilde{c}_F(i,j) + c_{01}(i,j) - c_{11}(i,j)$
      $\tilde{c}_F(j,i) = \tilde{c}_F(j,i) + c_{10}(i,j) - c_{00}(i,j)$
      $\tilde{c}_G(i,j) = \tilde{c}_G(i,j) + w_{01}(i,j) - w_{11}(i,j)$
      $\tilde{c}_G(j,i) = \tilde{c}_G(j,i) + w_{10}(i,j) - w_{00}(i,j)$
  $\tilde{E} = S_V \cup S_E$
  RETURN $(\tilde{V},\tilde{E})$

Figure 1: Network Construction.

Figure 1 shows the details of network construction. S_V denotes the set of all edges in $\tilde{E}$ that originate from s or terminate at t. Edges in S_V will be referred to as terminal edges. S_E denotes the set of all other arcs in $\tilde{E}$. Figure 4 shows an example of a control flow graph and the corresponding network.

Due to the assumption given by Equation (1), $\tilde{c}_F(i,j)$ and $\tilde{c}_G(i,j)$ are nonnegative and can indeed be interpreted as capacities of the edge (i,j). We consider only (s,t)-cuts, i.e., cuts that separate s and t in $\tilde{\mathcal{G}}(\tilde{V},\tilde{E})$ and partition the node set $\tilde{V}$ into two sets, denoted by S (a set containing s) and T (a set containing t). An example of such (s,t)-cut is shown in Figure 4. Associated with each node and a given (s,t)-cut is a cut variable y_i, such that $y_i = 0$ if node $i \in S$, otherwise $y_i = 1$.

A cut is associated with two capacities: one, denoted by C_F, is expressed in terms of the cost capacities, and the other, denoted by C_G, is expressed in terms of the weight capacities.

$$\begin{array}{rcl} C_F & = & \sum_{(i,j)\in\tilde{E} \mid i\in S,\ j\in T} \tilde{c}_F(i,j) = \sum_{(i,j)\in\tilde{E}} \bar{y_i} y_j \tilde{c}_F(i,j) \\ C_G & = & \sum_{(i,j)\in\tilde{E} \mid i\in S,\ j\in T} \tilde{c}_G(i,j) = \sum_{(i,j)\in\tilde{E}} \bar{y_i} y_j \tilde{c}_G(i,j) \end{array} \quad (4)$$

The following theorem relates our original objective function F to C_F and our original constraint function G to C_G.

THEOREM 1. *An (s,t)-cut in $\tilde{\mathcal{G}}(\tilde{V},\tilde{E})$ identifies a bipartition in $\mathcal{G}$, and the cut capacities C_F and C_G are equal to the cost F and the weight G of the bipartition, respectively.*

4.1 Unconstrained Bipartitioning Algorithm

In the unconstrained bipartitioning problem we want to minimize the cost function F without any restrictions on the constraint function G. To solve the unconstrained bipartitioning problem, we first construct the network from the control flow graph, as described above. Then, using a standard max-flow algorithm, we find a maximum flow ϕ_{max} and identify the corresponding cut with the minimum capacity $C_F = \phi_{max}$. Once the cut is known, a 0-1 assignment of the cut variables is straightforward. Upon this assignment, for each node i, except for s and t, we set $x_i = y_i$.

UnconstrainedBipartitioningProcedure (V,E)
  $(\tilde{V},\tilde{E}) = CFG2Network(V,E)$
  $\tilde{E} = SaturateNetwork(\tilde{V},\tilde{E})$, $CUT = FindMinCut(\tilde{V},\tilde{E})$
  FOR all $(i,j) \in CUT$
    $y_i = 0$, $y_j = 1$
  FOR all $i \in V$
    $x_i = y_i$
  RETURN $\{x_i \mid i \in V\}$

Figure 2: Unconstrained Bipartitioning Algorithm.

Figure 2 shows the details of the unconstrained bipartitioning algorithm. $SaturateNetwork(\tilde{V},\tilde{E})$ is used to saturate the network, and $FindMinCut(\tilde{V},\tilde{E})$ is used to identify the corresponding cut.

4.2 Constrained Bipartitioning Algorithms

In the constrained bipartitioning problem, we want to minimize the objective function F with a guarantee that the constraint function G does not exceed the budget B. A brute-force solution would be to consider all $2^{|V|}$ possible mappings $x_i \mapsto \{0,1\}\ \forall\, i \in V$, identify those that satisfy the weight constraint, and then among them select the one with the minimum cost. Such an approach is very expensive computationally. We propose two iterative methods that find a good solution in polynomial time; however, optimality is not guaranteed. The first method is cost-driven, i.e. the objective function F is systematically increased, starting from the minimum possible value, until either (1) the budget has been met, or (2) the number of iterations has exceeded a certain threshold. In this case, it is guaranteed that the minimum cost bipartition, which may or may not satisfy the constraint, will be considered. The second method is weight-driven, i.e. the objective function G is systematically increased, starting from the minimum possible value, until (1) the

budget can no longer be met, or (2) the number of iterations has exceeded a certain threshold. In this case, it is guaranteed that a feasible solution, if exists, will be considered, e.g. the minimum weight bipartition meets the budget; otherwise, no feasible solution exists. We advise to run both cost-driven and weight-driven algorithms in order to find a feasible solution of a good cost.

Cost-Driven Constrained Bipartitioning: After we construct and saturate the network, we find the corresponding min-cut CUT with the minimum cost capacity $C_F = \phi_{max}$. The cost and the weight of this initial solution are $F = C_F$ and $G = C_G$, respectively. If $G > B$, then at the next iteration we must find an alternative cut NEW with the lower weight capacity. (The cost capacity of such a cut may exceed the flow value ϕ_{max}.) To find NEW, we need to increase the cost capacity of some edges. Such selected edges must be excluded from alternative cuts of interest to maintain the equality between the objective function F and the cost capacity C_F of a cut. To achieve this, the selected cost capacities are set to infinity.[1] Note that any (s,t)-cut in the network contains exactly $|\tilde{V}| - 2 = |V|$ terminal edges (the edges with an endpoint at either the source or the sink) from the set S_V. We allow only terminal edges from S_V to be disabled (i.e. assigned to infinite cost capacity). Once as many as $|V|$ terminal edges have been disabled, there will remain only one cut not containing any of such edges (i.e. a cut with finite C_F). By disabling an additional terminal edge per iteration, we will need only $|V|$ iterations, thus ensuring the polynomial time complexity.

At each iteration, for a given CUT, we consider its terminal edges one by one. For each selected terminal edge in CUT, we set its cost capacity to infinity and saturate the network to find an alternative min-cut NEW. Before the next terminal edge from CUT is considered, we restore the original cost capacity of the currently excluded terminal edge. Thus, at a given iteration the number of disabled terminal edges remains the same. After each of $|V|$ terminal edges of CUT are individually considered, we let variable *LOCAL* hold the best min-cut among those found during the current iteration, and variable *GLOBAL* hold the best min-cut among all the iterations completed. For the next iteration, we set $CUT = LOCAL$, and restore the network that has *LOCAL* as its min-cut. Note that in this network, the number of disabled terminal edges is equal to the number of iterations completed. Once NEW is such that the constraint is satisfied, the local and the global solutions become the same, i.e. $GLOBAL = LOCAL$. There is no need for another iteration: we can only improve our solution during the current iteration.

THEOREM 2. *Let Z_∞ denote the set of terminal edges with the infinite cost capacity. If another terminal edge is added to Z_∞, the cost of a new min-cut will not improve.*

Thus, the iterative search terminates, once either (1) a feasible solution is found, or (2) $|V|$ iterations have been performed. Note that we have systematically examined only $O(|V|^2)$ cuts in the network. Figure 3 provides further details on the cost-driven algorithm, called $ConstrainedBipartitioningProcedure(\cdot)$.

Weight-Driven Constrained Bipartitioning: It is possible that none of the cuts considered by the cost-driven method has met the budget, i.e. we do not have a feasible solution. Therefore, running the weight-driven algorithm is necessary to guarantee that a feasible solution, if exists, is found. Figure 3 provides details on the weight-driven algorithm, called $ConstrainedBipartitioningProcedure2(\cdot)$. Saturating the network with respect to the weight capacities on edges is performed by $SaturateNetwork2(\cdot)$.

[1]Note that $\sum_{(i,j)\in\tilde{E}} \tilde{c}_F(i,j)$ is a trivial upper bound on flows in the cost-driven method. We let this sum serve as ∞, in this case.

To improve the cost of the minimum-weight solution, we use the same searching idea as in the cost-driven method.[2] The weight-driven algorithm terminates once either the number of iteration exceeds $|V|$, or the maximum flow at the current iteration exceeds the budget (next iterations will produce even greater flows, i.e. no other feasible solutions will be found). The output is the best-cost feasible solution found among $O(|V|^2)$ cuts considered.

5. ILLUSTRATIVE EXAMPLE

As an illustrative example, we take a control flow graph shown in Figure 4. It is the IDCT subroutine of the DJPEG program (decompressing JPEG file into an image) compiled for the C166/ST10 microprocessor [14]. Figure 4 also shows the number of memory access, ALU, and multiply instructions per block. We assume that (1) software (SW) and hardware (HW) operate at the same clock frequency with no pipelining; (2) one memory access takes 2 cycles and consumes 4 energy units, regardless whether it is issued from SW or HW; (3) one ALU operation takes 1 cycle in both SW and HW while consuming 2 energy units in SW and 3 energy units in HW; (4) one multiplication takes 4 cycles in SW and 1 cycle in HW while consuming 8 energy units in SW and 10 energy units in HW; (5) a single SW-to-SW control transfer takes 1 cycle and consumes 1 energy unit; (6) a single SW-to-HW control transfer takes 10 cycles and consumes 20 energy units; (7) a single HW-to-SW takes 2 cycles and consumes 4 energy unit; (8) a single HW-to-HW control transfer takes 1 cycle and consumes 1 energy unit; (9) the execution frequency for blocks $B0$, $B2$, $B4$, and $B9$ is 1, for blocks $B5$ and $B8$ is 64, for blocks $B6$ and $B7$ is 32 (50% probability), and for blocks (self-loops) $B1$ and $B3$ is 8; (10) the execution frequency for edges $(B0,B1)$, $(B1,B2)$, $(B2,B3)$, $(B3,B4)$, $(B4,B5)$, and $(B8,B9)$ is 1, for edges $(B5,B6)$, $(B6,B8)$, $(B5,B7)$, and $(B7,B8)$ is 32, and for edge $(B8,B5)$ is 64. Under these assumptions, the energy-delay specifications for nodes and edges are presented in Table 1.

Node	Energy		Delay		Children
	ε_0	ε_1	δ_0	δ_1	
B0	36	38	18	18	B1
B1	2672	3528	1336	952	B2
B2	2	3	1	1	B3
B3	2592	3392	1296	912	B4
B4	4	6	2	2	B5
B5	384	448	192	192	B6, B7
B6	64	96	32	32	B8
B7	64	96	32	32	B8
B8	896	1216	448	448	B5, B9
B9	30	31	15	15	-

Edge	Energy				Delay			
	ε_{00}	ε_{01}	ε_{10}	ε_{11}	δ_{00}	δ_{01}	δ_{10}	δ_{11}
(B0,B1)	1	20	4	1	1	10	2	1
(B1,B2)	1	20	4	1	1	10	2	1
(B2,B3)	1	20	4	1	1	10	2	1
(B3,B4)	1	20	4	1	1	10	2	1
(B4,B5)	1	20	4	1	1	10	2	1
(B5,B6)	32	640	128	32	32	320	64	32
(B5,B7)	32	640	128	32	32	320	64	32
(B6,B8)	32	640	128	32	32	320	64	32
(B7,B8)	32	640	128	32	32	320	64	32
(B8,B5)	64	1280	256	64	64	640	128	64
(B8,B9)	1	20	4	1	1	10	2	1

Table 1: IDCT Energy-Delay Specifications.

We consider three choices for the bipartitioning cost (energy, delay, and energy-delay product) and two choices for the bipartition-

[2]Note that $\sum_{(i,j)\in\tilde{E}} \tilde{c}_G(i,j)$ is a trivial upper bound on flows in the weight-driven method. We let this sum serve as ∞, in this case.

ConstrainedBipartitioningProcedure (V,E,B)
$(\tilde{V},\tilde{E}) = CFG2Network(V,E)$
$\tilde{E} = SaturateNetwork(\tilde{V},\tilde{E})$, $CUT = FindMinCut(\tilde{V},\tilde{E})$
$F = \sum_{(i,j)\in CUT} \tilde{c}_F(i,j)$, $G = \sum_{(i,j)\in CUT} \tilde{c}_G(i,j)$
IF $G > B$
$F_{global} = F$, $G_{global} = G$, $GLOBAL = CUT$, $LOCAL = CUT$
$\tilde{E}_{local} = \tilde{E}$, $iteration = 1$, $found$ = **FALSE**
WHILE $found$ = **FALSE**
$\tilde{E}_{copy} = SaveNetworkState(\tilde{E})$, $F_{local} = \infty$, $G_{local} = \infty$
FOR each $(i,j) \in CUT \mid \{i = s$ **OR** $j = t\}$
$\tilde{c}_F(i,j) = \sum_{(x,y)\in\tilde{E}} \tilde{c}_F(x,y)$
$\tilde{E} = SaturateNetwork(\tilde{V},\tilde{E})$, $NEW = FindMinCut(\tilde{V},\tilde{E})$
$F_{new} = \sum_{(i,j)\in NEW} \tilde{c}_F(i,j)$, $G_{new} = \sum_{(i,j)\in NEW} \tilde{c}_G(i,j)$
IF $found$ = **FALSE**
IF $F_{local} > F_{new}$
$F_{local} = F_{new}$, $G_{local} = G_{new}$, $LOCAL = NEW$
$\tilde{E}_{local} = SaveNetworkState(\tilde{E})$
IF $F_{global} > F_{new}$ **OR** $G_{new} \leq B$
$F_{global} = F_{new}$, $G_{global} = G_{new}$, $GLOBAL = NEW$
IF $G_{new} \leq B$
$F_{local} = F_{global}$, $G_{local} = G_{global}$, $LOCAL = GLOBAL$
$\tilde{E}_{local} = SaveNetworkState(\tilde{E})$, $found$ = **TRUE**
ELSE
IF $F_{global} > F_{new}$ **AND** $G_{new} \leq B$
$F_{global} = F_{new}$, $G_{global} = G_{new}$, $GLOBAL = NEW$
$F_{local} = F_{global}$, $G_{local} = G_{global}$, $LOCAL = GLOBAL$
$\tilde{E}_{local} = SaveNetworkState(\tilde{E})$
$\tilde{E} = RestoreNetworkState(\tilde{E}_{copy})$
$CUT = LOCAL$, $\tilde{E} = \tilde{E}_{local}$, $iteration = iteration + 1$
IF $iteration > |V|$ **BREAK**
$CUT = GLOBAL$
FOR all $(i,j) \in CUT$
$y_i = 0$, $y_j = 1$
FOR all $i \in V$
$x_i = y_i$
RETURN $\{x_i \mid i \in V\}$

ConstrainedBipartitioningProcedure2 (V,E,B)
$(\tilde{V},\tilde{E}) = CFG2Network(V,E)$
$\tilde{E} = SaturateNetwork2(\tilde{V},\tilde{E})$, $CUT = FindMinCut(\tilde{V},\tilde{E})$
$F = \sum_{(i,j)\in CUT} \tilde{c}_F(i,j)$, $G = \sum_{(i,j)\in CUT} \tilde{c}_G(i,j)$
IF $G \leq B$
$F_{global} = F$, $G_{global} = G$, $GLOBAL = CUT$, $LOCAL = CUT$
$\tilde{E}_{local} = \tilde{E}$, $iteration = 1$
WHILE $iteration \leq |V|$
$\tilde{E}_{copy} = SaveNetworkState(\tilde{E})$, $F_{local} = \infty$, $G_{local} = \infty$
FOR each $(i,j) \in CUT \mid \{i = s$ **OR** $j = t\}$
$\tilde{c}_G(i,j) = \sum_{(x,y)\in\tilde{E}} \tilde{c}_G(x,y)$
$\tilde{E} = SaturateNetwork2(\tilde{V},\tilde{E})$, $NEW = FindMinCut(\tilde{V},\tilde{E})$
$F_{new} = \sum_{(i,j)\in NEW} \tilde{c}_F(i,j)$, $G_{new} = \sum_{(i,j)\in NEW} \tilde{c}_G(i,j)$
IF $F_{local} > F_{new}$
$F_{local} = F_{new}$, $G_{local} = G_{new}$, $LOCAL = NEW$
$\tilde{E}_{local} = SaveNetworkState(\tilde{E})$
IF $F_{global} > F_{new}$ **AND** $G_{new} \leq B$
$F_{global} = F_{new}$, $G_{global} = G_{new}$, $GLOBAL = NEW$
$F_{local} = F_{global}$, $G_{local} = G_{global}$, $LOCAL = GLOBAL$
$\tilde{E}_{local} = SaveNetworkState(\tilde{E})$
$\tilde{E} = RestoreNetworkState(\tilde{E}_{copy})$
$CUT = LOCAL$, $\tilde{E} = \tilde{E}_{local}$, $iteration = iteration + 1$
IF $G_{local} > B$ **BREAK**
$CUT = GLOBAL$
FOR all $(i,j) \in CUT$
$y_i = 0$, $y_j = 1$
FOR all $i \in V$
$x_i = y_i$
RETURN $\{x_i \mid i \in V\}$

Figure 3: Proposed Constrained Bipartitioning Algorithms.

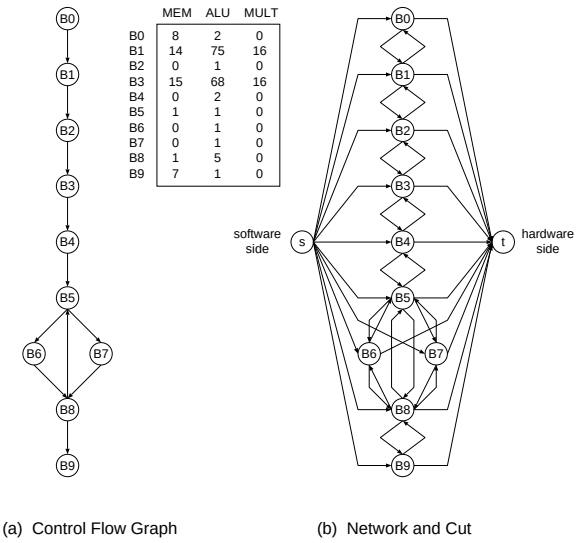

Figure 4: Example: IDCT subroutine (part of JPEG decompress).

ing weight (delay and energy). Table 2 presents the results of bipartitioning for the following cases:

- ε_{min}: unconstrained solution for min-energy,
- $\varepsilon_{min} \mid \delta_B$: delay-constrained solution for min-energy,
- δ_{min}: unconstrained solution for min-delay,
- $\delta_{min} \mid \varepsilon_B$: energy-constrained solution for min-delay, and
- $(\varepsilon \cdot \delta)_{min}$: unconstrained solution for min-energy-delay.

Table 2 shows costs and weights of hardware-software bipartitions (*HW* and *SW* columns) found by our approach and by an enumerative exponential-time algorithm *OPT*. In our example, the energy metric varies from 6942 to 9052 and the delay metric varies from 2803 to 3570. For case $\delta_{min} \mid \varepsilon_B$(c), our method generates a non-optimal cost solution; however, the deviation from the optimum cost is less than 1% (3196 vs 3187). For the other cases, our solution cost is the same the optimal solution cost.

6. CONCLUSION

In this paper we presented methods for the hardware-software mapping (bipartitioning) of an application control flow graph onto a dynamically reconfigurable system. We addressed the problems of (1) energy-delay product minimization, (2) delay-constrained energy minimization, and (3) energy-constrained delay minimization. We showed how to use network flow techniques to solve these problems. We proposed an efficient bipartitioning algorithm that finds an optimal solution for problem (1) and systematically searches for the best in a polynomially bounded set of good solutions for problems (2) and (3). Note that our problem formulation is not specific to energies and delays: costs and weights can be other design parameters.

7. APPENDIX

(A) See Theorem 1.

PROOF. A cut in $\tilde{\mathcal{G}}$ partitions the node set $\tilde{V}$ into two sets S and T. Since $V \subset \tilde{V}$, the cut identifies a bipartition of nodes in $\mathcal{G}$. Next, we show that the cut capacities in $\tilde{\mathcal{G}}$ are equal to the cost and the weight of a bipartition in $\mathcal{G}$: (1) $C_F = F$ and (2) $C_G = G$.

By construction of $\tilde{\mathcal{G}}(\tilde{V},\tilde{E})$, $i \in V \Leftrightarrow \{(s,i),(i,t)\} \in S_V$ and $(i,j) \in E \Leftrightarrow \{(i,j),(j,i)\} \in S_E$. Since $S_V \cup S_E = \tilde{E}$ and $S_V \cap S_E = \emptyset$,

Case	Cost		Weight		Budget
	Our	*OPT*	*Our*	*OPT*	
ε_{min}	6942	6942	3570	3570	-
ε_{min} δ_B(a)	7764	7764	3196	3196	3569
ε_{min} δ_B(b)	7803	7803	3187	3187	3195
ε_{min} δ_B(c)	8604	8604	2803	2803	3186
ε_{min} δ_B(d)	9052	9052	2802	2802	2802
δ_{min}	2802	2802	9052	9052	-
δ_{min} ε_B(a)	2803	2803	8604	8604	9051
δ_{min} ε_B(b)	3187	3187	7804	7803	8603
δ_{min} ε_B(c)	3196	3187	7765	7803	7803
δ_{min} ε_B(d)	3196	3196	7765	7764	7802
δ_{min} ε_B(e)	3196	3196	7764	7464	7764
δ_{min} ε_B(f)	3570	3570	6942	6942	7763
$(\varepsilon \cdot \delta)_{min}$	6940742	6940742	-	-	-

Case	Our Solution		OPT Solution	
	SW	*HW*	*SW*	*HW*
ε_{min}	B0,...,B9	-	B0,...,B9	-
ε_{min} δ_B(a)	B0,B1,B2,B4,...,B9	B3	B0,B1,B2,B4,...,B9	B3
ε_{min} δ_B(b)	B2,...,B9	B0,B1	B2,...,B9	B0,B1
ε_{min} δ_B(c)	B4,...,B9	B0,...,B3	B4,...,B9	B0,...,B3
ε_{min} δ_B(d)	-	B0,...,B9	-	B0,...,B9
δ_{min}	-	B0,...,B9	-	B0,...,B9
δ_{min} ε_B(a)	B4,...,B9	B0,...,B3	B4,...,B9	B0,...,B3
δ_{min} ε_B(b)	B3,...,B9	B0,B1,B2	B2,...,B9	B0,B1
δ_{min} ε_B(c)	B0,B1,B4,...,B9	B2,B3	B2,...,B9	B0,B1
δ_{min} ε_B(d)	B0,B1,B4,...,B9	B2,B3	B0,B1,B2,B4,...,B9	B3
δ_{min} ε_B(e)	B0,B1,B2,B4,...,B9	B3	B0,B1,B2,B4,...,B9	B3
δ_{min} ε_B(f)	B0,...,B9	-	B0,...,B9	-
$(\varepsilon \cdot \delta)_{min}$	B4,...,B9	B0,...,B3	B4,...,B9	B0,...,B3

Table 2: Generated Bipartitions of IDCT.

the sets S_V and S_E form a *partition* of the set $\tilde{E}$. Therefore, in equations (4) the sum $\sum_{(i,j)\in\tilde{E}}$ can be written as the sum of $\sum_{\{(s,i),(i,t)\}\in S_V}$ and $\sum_{\{(i,j),(j,i)\}\in S_E}$.

(1) First, we prove that $C_F = F$. The cut capacity C_F can be represented as follows:

$$\begin{aligned} C_F \;=\; & \textstyle\sum_{\{(s,i),(i,t)\}\in S_V} \left[\bar{y_s} y_i \tilde{c}_F(s,i) + \bar{y_i} y_t \tilde{c}_F(i,t)\right] + \\ & \textstyle\sum_{\{(i,j),(j,i)\}\in S_E} \left[\bar{y_i} y_j \tilde{c}_F(i,j) + \bar{y_j} y_i \tilde{c}_F(j,i)\right]. \end{aligned} \tag{5}$$

We can substitute the first sum by $\sum_{i\in V}$ and the second sum by $\sum_{(i,j)\in E}$. Obviously, $y_s = 0$ and $y_t = 1$. Thus,

$$C_F = \sum_{i\in V} \left[y_i \tilde{c}_F(s,i) + \bar{y_i} \tilde{c}_F(i,t)\right] + \sum_{(i,j)\in E} \left[\bar{y_i} y_j \tilde{c}_F(i,j) + \bar{y_j} y_i \tilde{c}_F(j,i)\right]. \tag{6}$$

After expressing edge capacities in terms of costs, we obtain:

$$\begin{aligned} & \textstyle\sum_{i\in V} \left[y_i \tilde{c}_F(s,i) + \bar{y_i} \tilde{c}_F(i,t)\right] = \\ & \textstyle\sum_{i\in V} \left\{ y_i \left[c_1(i) + \sum_{(j,i)\in E} c_{11}(j,i)\right] + \bar{y_i} \left[c_0(i) + \sum_{(j,i)\in E} c_{00}(j,i)\right] \right\}, \\ & \text{and} \\ & \textstyle\sum_{(i,j)\in E} \left[\bar{y_i} y_j \tilde{c}_F(i,j) + \bar{y_j} y_i \tilde{c}_F(j,i)\right] = \\ & \textstyle\sum_{(i,j)\in E} \left\{ \bar{y_i} y_j \left[c_{01}(i,j) - c_{11}(i,j)\right] + y_i \bar{y_j} \left[c_{10}(i,j) - c_{00}(i,j)\right] \right\}. \end{aligned} \tag{7}$$

Note that $\sum_{i\in V} y_i \sum_{(j,i)\in E} c_{11}(j,i) = \sum_{(i,j)\in E} y_j c_{11}(i,j)$, and $\sum_{i\in V} \bar{y_i} \sum_{(j,i)\in E} c_{00}(j,i) = \sum_{(i,j)\in E} \bar{y_j} c_{00}(i,j)$. Thus,

$$\begin{aligned} C_F = \; & \textstyle\sum_{i\in V} \left[y_i c_1(i) + \bar{y_i} c_0(i)\right] + \sum_{(i,j)\in E} \left[y_j c_{11}(i,j) + \bar{y_j} c_{00}(i,j) + \right. \\ & \left. \bar{y_i} y_j c_{01}(i,j) - \bar{y_i} y_j c_{11}(i,j) + y_i \bar{y_j} c_{10}(i,j) - y_i \bar{y_j} c_{00}(i,j)\right]. \end{aligned} \tag{8}$$

Since $y_i + \bar{y_i} = 1$,

$$\begin{aligned} C_F = \; & \textstyle\sum_{i\in V} \left[y_i c_1(i) + \bar{y_i} c_0(i)\right] + \sum_{(i,j)\in E} \left[y_j y_i c_{11}(i,j) \right. \\ & \left. + \bar{y_j} \bar{y_i} c_{00}(i,j) + \bar{y_i} y_j c_{01}(i,j) + \bar{y_j} y_i c_{10}(i,j)\right] \end{aligned} \tag{9}$$

To complete our proof, let $x_i = y_i$ for all $i \in V$. Now the terms of the sum $\sum_{i\in V}$ become $f(i)$, and the terms of the sum $\sum_{(i,j)\in E}$ become $f(i,j)$:

$$C_F = \sum_{i\in V} f(i) + \sum_{(i,j)\in E} f(i,j) = F. \tag{10}$$

(2) The proof of that C_G is equal to the weight G of a bipartition is analogous to (1). First we substitute C_F with C_G and the cost capacities $\tilde{c}_F(i,j)$ with the weight capacities $\tilde{c}_G(i,j)$. Then, we substitute the node costs $c_0(i)$ and $c_1(i)$ with the node weights $w_0(i)$ and $w_1(i)$, respectively, and the edge costs $c_{00}(i,j)$, $c_{01}(i,j)$, $c_{10}(i,j)$, and $c_{11}(i,j)$ with the edge weights $w_{00}(i,j)$, $w_{01}(i,j)$, $w_{10}(i,j)$, and $w_{11}(i,j)$, respectively. Finally, we substitute $f(i)$ with $g(i)$ and $f(i,j)$ with $g(i,j)$. After these changes, all the derivations are the same as in (1), and we obtain:

$$C_G = \sum_{i\in V} g(i) + \sum_{(i,j)\in E} g(i,j) = G. \tag{11}$$

□

(B) See Theorem 2.

PROOF. Let *CUT* denote the min-cut (with the cost F) in the network with $|Z_\infty|$ terminal edges whose cost capacities are set to infinity. Let CUT_+ denote the network min-cut (with the cost F_+) after some terminal edge (i,j) is added to Z_∞. Recall that the cost of a min-cut is equal to the maximum flow value in the network. Once the cost capacity of (i,j) is set to infinity, while the cost capacities of the other network edges remain the same (including those already set to infinity), it is clear that the max-flow in the new network can only increase or remain the same. Thus, $F_+ \geq F$. □

8. REFERENCES

[1] C. Cheng and T. Hu, "Maximum concurrent flows and minimum cuts," *Algorithmica*, vol. 8, 1992.

[2] C. Alpert and A. Kahng, "Recent directions in netlist partitioning: A survey," *Integration: VLSI J.*, vol. 19, 1995.

[3] S. Hauck and G. Borriello, "An evaluation of bipartitioning techniques," *IEEE Trans. CAD*, vol. 18, 1997.

[4] H. Liu and D. Wong, "Network flow based circuit partitioning for time-multiplexed FPGAs," *Proc. ICCAD*, 1998.

[5] R. Dick and N. Jha, "CORDS: Hardware-software co-synthesis of reconfigurable real-time distributed embedded systems," *Proc. ICCAD*, 1998.

[6] B. Dave, "CRUSADE: Hardware/software cosynthesis of dynamically reconfigurable heterogeneous real-time distributed embedded systems," *Proc. DATE*, 1999.

[7] U. Shenoy, P. Banerjee, and A. Choudhary, "A system-level synthesis algorithms with guaranteed solution quality," *Proc. DATE*, 2000.

[8] M. Wan, H. Zhang, V. George, M. Benes, A. Abnous, V. Prabhu, and J. Rabaey, "Design methodology of a low-energy reconfigurable single-chip DSP system," *J. VLSI Signal Processing*, 2000.

[9] S. Ogrenci, E. Bozorgzadeh, R. Kastner, and M. Sarrafzadeh, "A super-scheduler for reconfigurable systems," *Proc. ICCAD*, 2001.

[10] K. Chatha and R. Vemuri, "Hardware-software codesign for dynamically reconfigurable architectures," *Proc. FPL*, 1999.

[11] P. Knudsen and J. Madsen, "PACE: A dynamic programming algorithm for hardware/software partitioning," *Proc. CODES*, 1996.

[12] F. Vahid, "Modifying min-cut for hardware and software functional partitioning," *Proc. CODES*, 1997.

[13] Y. Li, T. Callahan, E. Darnell, R. Harr, U. Kurkure, and J. Stockwood, "Hardware-software co-design of embedded reconfigurable architectures," *Proc. DAC*, 2000.

[14] DJPEG Call Graph, *http://www.aisee.com/split/index.htm*, 2002.