Using Complementation And Resequencing To Minimize Transitions

Rajeev Murgai, Masahiro Fujita
Fujitsu Laboratories of America, Inc.

Arlindo Oliveira
Cadence European Labs./IST-INESC

Abstract

Recently, in [3], the following problem was addressed: *Given a set of data words or messages to be transmitted over a bus such that the sequence (order) in which they are transmitted is irrelevant, determine the optimum sequence that minimizes the total number of transitions on the bus.* In 1994, Stan and Burleson [5] presented the *bus-invert method* as a means of encoding words for reducing I/O power, in which a word may be inverted and then transmitted if doing so reduces the number of transitions. In this paper, we combine the two paradigms into one – that of sequencing words under the bus-invert scheme for the minimum transitions, i.e., words can be complemented, reordered and then transmitted. We prove that this problem DOPI – Data Ordering Problem with Inversion – is NP-complete. We present a polynomial-time approximation algorithm to solve DOPI that comes within a factor of 1.5 from the optimum. Experimental results show that, on average, the solutions generated by our algorithm were within 4.4% of the optimum, and that resequencing along with complementation leads to 34.4% reduction in switching activity.

1 Motivation

In several applications such as microprocessors and DSPs, switching activity on high-capacitance buses account for almost 30-40% of the power consumption [1]. Thus for reducing power dissipation, the problem of minimizing switching activity (i.e., the number of transitions) on a bus assumes great importance. In this context, consider the following problem. Given a set of n data words or messages, each of length k, to be transmitted over a bus (k-bit wide) such that the sequence in which they are transmitted is irrelevant, determine the optimum sequence that minimizes the total number of transitions on the bus. We call this the **Data Ordering Problem (DOP)**.

Definition 1.1 *If word w_r is transmitted, immediately followed by w_s, the total number of transitions is given by the number of bits that change. This is $d(w_r, w_s) = \sum_{j=1}^{k} w_{rj} \oplus w_{sj}$, also known as the* Hamming distance *between w_r and w_s. Here, w_{rj} denotes the j^{th} bit of w_r, and $\oplus$ denotes the EX-OR operation. For instance, $d(1001, 1110) = 3$.*

Word reordering can change the number of transitions significantly. For instance, let $n = 3$, $w_1 = 00011, w_2 = 10110$, and $w_3 = 01011$. So $k = 5$. If words are transmitted in the order w_1, w_2, w_3, the total number of transitions on the bus is $d(w_1, w_2) + d(w_2, w_3) = 3 + 4 = 7$. If transmitted in the order w_1, w_3, w_2, the number of bus transitions would be $d(w_1, w_3) + d(w_3, w_2) = 1 + 4 = 5$, 28% fewer.

The data ordering problem was addressed in [3]. It was shown that this problem arises in a wide variety of design tasks when power dissipation is the main concern: instruction scheduling, die testing, sequencing of test patterns in built-in self test (BIST) systems, cache write-back, and scheduling in high-level synthesis. It was shown in [3] that DOP is NP-complete. Three approximation schemes were proposed: *Double Spanning Tree (DST)* – which provably comes within a factor of two of the optimum solution, *Minimum Spanning Tree Maximum Matching (ST-MM)* – which comes within 1.5 (plus a small additive term) of the optimum, and *greedy* – which was empirically found to be the best.

In [5], Stan and Burleson presented the *bus-invert method* to reduce the number of transitions. It works as follows. An extra bus line, called *invert*, is used. The method looks at two consecutive data words on the bus. If the Hamming distance between the next word and the current transmitted word is at most $k/2$, the next word is sent as it is, with *invert* set to 0. Otherwise, each bit of the next word is complemented (inverted) and *invert* is set to 1, indicating that the word has been complemented. This scheme reduces the number of transitions on the bus. Let $\overline{w_i}$ denote the complement of w_i. E.g., $\overline{10110} = 01001$. In the last example, if we transmit w_1, w_3, and then $\overline{w_2}$, it results in $d(w_1, w_3) + d(w_3, \overline{w_2}) = 1 + 1 = 2$ transitions, 3 less than the uncomplemented case. If we count the extra transition on the *invert* signal, the total number of transitions is 3, still 2 less than 5.

In this paper, we combine the two paradigms into one: that of sequencing words under the bus-invert scheme, i.e., words can be complemented (if it helps), reordered and then transmitted. *Given n data words $w_1, w_2, \ldots, w_n$, the problem then is to determine i) for each word w_i, if it should be complemented or left uncomplemented (i.e., its* **phase assignment***), and ii) the order in which words should be transmitted, so as to minimize the transitions on the bus.* We call this problem **DOPI** – **Data Ordering Problem with Inversion**. The next section discusses the relevant background graph-theoretic material. We prove in Section 3 that DOPI is NP-complete. In Section 4, we devise approximation schemes for DOPI that guarantee bounds from the optimum solution. Experimental results are presented in Section 5.

2 Preliminaries

A **spanning tree** of a graph $G = (V, E)$ is a subgraph of G that is a tree and spans all the vertices of G. If the edges have weights, it makes sense to talk of the **minimum-weight spanning tree (MST)** of G, where the weight of a tree is the sum of the weights of the edges in the tree. It turns out that a simple, greedy algorithm that starts from an empty tree and repeatedly selects the minimum-weight edge, adding it to the partially generated tree if the tree still remains acyclic generates an MST. For the weighted graph of Figure 1 (A), the minimum-weight spanning tree is shown in Figure 1 (B). A **Hamiltonian tour** of a graph $G = (V, E)$ is a walk (simple path) with the same beginning and end points that visits each vertex of V exactly once. A **Hamiltonian path** of G is a walk with different beginning and end points that visits each vertex of V exactly once. For instance, 1, 2, 3, 4, 6, 5, 7, 1 is a Hamiltonian tour of the graph of Figure 1 (A), whereas 1, 2, 3, 4, 6, 5, 7 is a Hamiltonian path. If the edges have weights, we

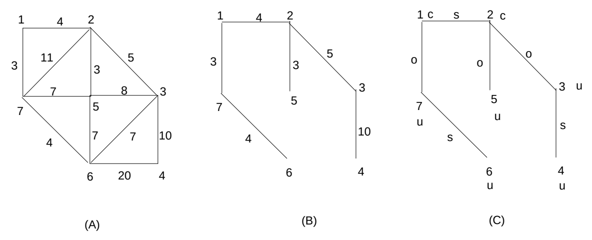

Figure 1: An example graph and its spanning tree

have the problems of finding a minimum-weight Hamiltonian tour (better known as the **traveling salesman problem** tour or TSP-tour) and minimum-weight Hamiltonian path (TSP-path). Both of these are known to be NP-complete, even for complete graphs.

A graph $G = (V, E)$ in which there is more than one edge joining a pair of nodes is called a **multigraph**. An **Eulerean cycle** of a multigraph is a walk with the same beginning and end points that contains each edge of the graph exactly once. E.g., in Figure 1(A), the tour 1, 2, 3, 4, 6, 3, 5, 6, 7, 5, 2, 7, 1 is an Eulerean cycle. It is well-known that *a multigraph G contains an Eulerean cycle if and only if each node of G is of even degree.* The following simple procedure finds an Eulerean cycle in such a multigraph G.

Procedure 2.1 (Finding an Eulerean cycle) *Start at any vertex v of G. Traverse any edge (v, w) whose removal does not disconnect the remaining (untraversed) graph. Delete (v, w). Repeat the procedure from w. Terminate when no more edges remain in G. This can happen only when at v. The order in which the edges are traversed yields the Eulerean cycle.*

Given a complete graph G and a spanning tree $T = (V, E')$ of G, the following procedure finds a Hamiltonian tour on G.

Procedure 2.2 (Constructing a Hamiltonian tour) *Construct the multigraph $\widehat{G}$ from T by duplicating each edge $e \in E'$. Since each node of $\widehat{G}$ is of even degree, $\widehat{G}$ contains an Eulerean cycle U. Construct U using Procedure 2.1. Delete all the node repetitions from U except for the final return to the first node. The resulting node sequence HT is a Hamiltonian tour on G.*

If we are interested in a minimum-weight Hamiltonian tour, we will start from an MST T and use Procedure 2.2.

Let each edge (v_i, v_j) of the graph have a non-negative weight $d(v_i, v_j)$. The weight $d(S)$ of a tour (path, tree) S is the sum of the weights of the edges in S. If the edge weights d satisfy **triangle inequality** (i.e., $d(v_1, v_2) + d(v_2, v_3) \geq d(v_1, v_3)$ for all v_1, v_2, v_3), it is easy to see that $d(HT) \leq d(U)$. Since $d(U) = 2d(T)$, we get

$$d(HT) \leq 2d(T) \qquad (1)$$

A **matching** in a graph $G = (V, E)$ is a subset $E' \subseteq E$ such that no two edges of E' are incident to each other. A **perfect matching** is a matching that is incident to each vertex in V. Clearly, $|V|$ must be even for a perfect matching to exist.

3 DOPI is NP-complete

The Data Ordering Problem with Inversion (DOPI) stated as a decision problem is:
INSTANCE: A set of n k-bit data words, $w_1, w_2, \ldots, w_n$, each $w_i \in \{0, 1\}^k$ – k a positive integer, and a positive integer M.
QUESTION: Are there phase assignment ρ and permutation δ of $w_1, w_2, \ldots, w_n$, such that

$$\sum_{i=1}^{n-1} d(\rho(w_{\delta(i)}), \rho(w_{\delta(i+1)})) \leq M? \qquad (2)$$

where $\rho(w_i) = w_i$ or $\rho(w_i) = \overline{w_i}$.

Since we know that DOP is hard – it was shown to be NP-complete in [4], it seems intuitive that DOPI, which has an extra degree of freedom of word-inversion, should be at least as hard. The following theorem shows that that is indeed the case.

Theorem 3.1 *DOPI is NP-complete.*

Sketch of Proof DOPI is easily seen to be in NP. To show NP-hardness of DOPI, we transform DOP to DOPI. DOP stated as a decision problem is:
INSTANCE: A set of N K-bit data words, $W_1, W_2, \ldots, W_N$, each $W_i \in \{0, 1\}^K$ – K a positive integer, and a positive integer L.
QUESTION: Is there a permutation σ of $W_1, W_2, \ldots, W_N$, i.e. a 1-1 function $\sigma : \{1, \ldots, N\} \rightarrow \{1, \ldots, N\}$ such that

$$\sum_{i=1}^{N-1} d(W_{\sigma(i)}, W_{\sigma(i+1)}) \leq L? \qquad (3)$$

Given an instance of DOP, we generate the following instance of DOPI. Let $n = N$ and $M = L$. For each i, form w_i by concatenating KN zeros after W_i, i.e., $w_i = W_i 000 \ldots 0$ (KN zeros). So, $k = K + KN$. It is easy to see that the transformation is polynomial-time. It can be shown that there exists a permutation σ of W_is such that (3) holds if and only if there exist ρ and δ of w_is such that (2) holds. Details can be found in [4]. ■

4 Solving DOPI

First, let us take care of the technicality of *invert* signal. It makes a transition when two successive words are in opposite phase. We should count also these transitions. This is easily handled in our formulation by appending a zero bit at the end of each word w_i. If the word is sent uncomplemented, the bit remains 0. Otherwise, this bit is set to 1. So, from now on, we will work on w_is augmented with the invert bit, but would still call them w_is.

There are 2^n possibilities, corresponding to two possible phase assignments ρ for each of the n words – for each word w_i, $\rho(w_i) = w_i$ or $\overline{w_i}$. Each phase assignment generates different words, which results in different edge weights and hence a different instance of DOP. One way to solve DOPI then is to generate these 2^n instances of DOP, solve them one by one (with DST, ST-MM, or greedy algorithm), and pick the best solution. Although feasible for small n, this scheme will clearly run out of steam for large n.

It turns out that it is not necessary to explicitly generate and solve 2^n instances of DOP. Due to special properties of the Hamming distance, *it is possible to come up with polynomial-time approximation algorithms (DST, ST-MM) for DOPI that guarantee the same performance bounds from optimality as the corresponding DOP algorithms and also have the same time complexity.*

4.1 DST Algorithm for DOPI

In the double spanning tree algorithm for DOP, the key idea for guaranteeing the upper bound was generating a minimum-weight spanning tree [3]. It turns out that the corresponding key concept in solving DOPI is *generating a minimum-weight spanning tree over all phase assignments.* Given n data words, create a multigraph G on n vertices – one vertex for each word. There are exactly two edges between each pair of vertices v_i and v_j, one labeled SAME (s) and the other OPPOSITE (o). The edge (v_i, v_j) labeled SAME corresponds to the case when the corresponding words w_i and w_j are either both complemented or both uncomplemented. The label OPPOSITE corresponds to when exactly one of w_i and w_j is complemented. These edges are assigned weights d_s and d_o respectively, as follows:

$$d_s(v_i, v_j) = d_s(w_i, w_j) = d(w_i, w_j) = d(\overline{w_i}, \overline{w_j})$$
$$d_o(v_i, v_j) = d_o(w_i, w_j) = d(\overline{w_i}, w_j) = d(w_i, \overline{w_j})$$

Consider $w_1 = 1000$ and $w_2 = 1111$. Then, $d_s(w_1, w_2) = d(w_1, w_2) = 3$, whereas $d_o(w_1, w_2) = d(\overline{w_1}, w_2) = 1$.

Construct a minimum spanning tree T of G. Some edges of T will be labeled s and the rest, o. To determine the phase in which each data word should be transmitted, we assign a label ℓ to each vertex. The **label $\ell(v_i)$ of a vertex** v_i is either COMPLEMENTED (c) or UNCOMPLEMENTED (u), implying that the corresponding data word w_i should be complemented and then sent or just sent uncomplemented respectively.

Definition 4.1 *Given a graph with all edges labeled s or o (edge labeling) and all vertices labeled c or u (vertex labeling ℓ). The vertex labeling ℓ is* **consistent** *with the edge labeling if for each edge (v_i, v_j) labeled s, $\ell(v_i) = \ell(v_j)$ (either both c or both u), and for each edge (v_i, v_j) labeled o, exactly one of v_i & v_j is labeled u.*

Proposition 4.1 *Given a spanning tree T with an edge labeling, there exists a vertex labeling consistent with the edge labeling.*

Proof The following procedure assigns labels to all the vertices of G (or T). Hang the tree T on any vertex, say v. So T becomes a rooted tree with root v. Let $\ell(v) = u$ (arbitrarily).[1] Assign vertex labels from the root to the leaves of T. For an edge $(w, x) \in T$, where w is x's parent, we can assume that w has already been labeled. If (w, x) is labeled s, assign $\ell(x) = \ell(w)$. Otherwise, (w, x) is labeled o, and assign $\ell(x) \neq \ell(w)$, i.e., if $\ell(w) = c$, assign $\ell(x) = u$, and vice-versa. It is easy to see that this procedure labels vertices consistent with the edge labeling. ■

Consider the tree T of Figure 1 (B). Assume edge-labels as in Figure 1 (C). To see that a vertex labeling consistent with this edge labeling exists, hang T on (arbitrarily chosen) vertex 6. $\ell(6) = u$. Since edge (6, 7) is labeled s, $\ell(7) = \ell(6) = u$. This implies $\ell(1) = c$, which implies $\ell(2) = c$. Both 5 and 3 are assigned phases opposite to that of 2. So $\ell(5) = \ell(3) = u$. Finally, $\ell(4) = u$.

Thus, from T and its edge labeling, a consistent vertex labeling is determined, from which the codes ρ that will be transmitted are computed. If $\ell(v_i) = c$, $\rho(w_i) = \overline{w_i}$. Otherwise, $\rho(w_i) = w_i$. The complete DST algorithm is as follows:

1. Form the multigraph G as explained above, with edges labeled s and o appropriately.
2. Find an MST T of G. Using the edge labels in T, determine vertex labels ℓ (c or u) – as in Proposition 4.1. From these labels, determine the phase assignment ρ.
3. Apply Procedure 2.2 to find a Hamiltonian tour HT. This procedure first duplicates each edge of T, finds an Eulerean cycle U using Procedure 2.1, and then constructs HT from U. Care must be taken while constructing HT from U. For instance, if (v_4, v_1) and (v_1, v_6) are being replaced by (v_4, v_6), appropriate edge (v_4, v_6) must be added, since there are two edges between v_4 and v_6. The edge consistent with the labels $\ell(v_4)$ and $\ell(v_6)$ should be added. For instance, if $\ell(v_4) = \ell(v_6) = c$, add the edge labeled SAME.
4. Delete the longest edge e in HT to get a Hamiltonian path HP. The sequence of vertices in this path corresponds to the desired permutation.

Theorem 4.2 *DST algorithm for DOPI yields a Hamiltonian path HP and phase assignment ρ such that $d(HP_\rho) \leq \frac{2(n-1)}{n} d(HP^*_{\rho^*})$, where $HP^*_{\rho^*}$ is the optimum solution to DOPI.*

[1]For $\ell(v) = c$, another labeling will be obtained.

Sketch of Proof The key observation is that T is in fact the MST over all possible phase assignments, i.e., if we generate 2^n graphs each corresponding to a different phase assignment (and hence different edge weights), then determine the MST for each assignment, and pick from all these trees the one with the minimum weight, it will have the same weight as T.

Consider the (complete) graph G^* corresponding to the phase assignment ρ^*. Let a minimum-weight spanning tree in G^* be $T^*_{\rho^*}$. Then, $d(HP^*_{\rho^*}) \geq d(T^*_{\rho^*}) \geq d(T) \geq 0.5 d(HT_\rho)$. The last inequality is from (1). The factor $\frac{n}{n-1}$ comes from deleting the longest edge in HT while obtaining HP. ■

Modified DST It turns out that to find T, it is not necessary to form a multigraph G. It suffices to have a simple graph $\widetilde{G}$ that has only one edge between any vertex pair. This edge is the one that has lower weight among the two edges labeled s and o in the multigraph G. This is because the minimum spanning tree algorithm will always select the lower-weight edge. It can be shown that the edge weights of $\widetilde{G}$ satisfy triangle inequality [4]. Then DST can be modified as follows. *Find MST T in $\widetilde{G}$. Use T to find a Hamiltonian tour HT. Delete the longest edge e in HT to get a Hamiltonian path HP. HP has labels s and o on the edges. Obtain phase assignment for all the vertices from Proposition 4.1, since HP is a spanning tree.* This algorithm also comes within $2\frac{(n-1)}{n}$ of the optimum [4].

4.2 ST-MM Algorithm

ST-MM is a theoretical improvement over DST, since it guarantees a bound within a factor of 1.5 from the optimum solution (plus an additive term). The algorithm is as follows. Find MST T in $\widetilde{G}$. Identify the set O of vertices in the tree T with odd degrees ($|O|$ must be even). Find a minimum-weight perfect matching M on $G(O) = (O, E(O))$, the subgraph of G induced on O.[2] Consider the graph $T \cup M$. Since each edge of M connects two different odd-degree vertices of T, $T \cup M$ is a connected graph with all vertices having even degrees. Then, there exists an Eulerean cycle U in $T \cup M$ – construct U using Procedure 2.1. From U, construct a Hamiltonian tour HT by Procedure 2.2. Delete the longest edge e in this tour to get a Hamiltonian path HP. The phase assignment ρ is determined exactly as in the modified DST.

Theorem 4.3 *Let $HP^*_{\rho^*}$ be the optimum solution for DOPI. Then, $d(HP_\rho) \leq \frac{n-1}{n}[\frac{3}{2} d(HP^*_{\rho^*}) + \frac{k}{2}]$, where k is the word size and HP_ρ is the Hamiltonian path produced by the ST-MM algorithm.*

Although T and M may correspond to different phase assignments, it does not matter since the final phase assignment is decided at the very end – on HP.

4.3 Greedy Heuristic

It works on $\widetilde{G}$ and selects the minimum-weight edge, say (i, j), of $\widetilde{G}$. The path P after this step is (i, j). Next, the minimum-weight edge incident on an end-point of P (i.e., on i or j) is selected. Let it be (i, k). Thus P becomes (k, i, j). Thus, at each step, it greedily selects the minimum-weight edge incident on an end-point of P that is not incident on an internal vertex of P. The heuristic terminates when all the vertices of $\widetilde{G}$ are in P, in which case a Hamiltonian path has been constructed. The phase assignment ρ is determined as in modified DST.

[2]$G(O)$ is a subgraph of G on the vertices O and has exactly those edges of G that have both their end-points in O.

5 Experimental Results

We generated a set of uniformly distributed n k-bit data words, for different values of n and k. Then we applied the following schemes: `random`, `greedy`, DST, and ST-MM. A subset of results is shown in Table 1 – for the complete set, please see [4]. In DOP, only resequencing is allowed. In DOPI, both complementation and resequencing are done. Each entry in the table denotes the total number of transitions for transmitting the n words. In `random` (`rand` in the table), words are transmitted in the same order as they were generated (which was random). This scheme models no resequencing. In DOPI `random`, a word is complemented if it reduces the number of transitions with respect to the last word sent. Other schemes were discussed in Section 4. We also provide a lower bound on the optimum solution (column ℓ-`bnd`) – which is the weight of the MST.

First we compare columns DOPI `greedy` – `grdy` in the table (which gives the best results) with DOPI ℓ-`bnd` to check how close our algorithms come to the optimum. On average, the `greedy` heuristic is within 4.4% of the ℓ-`bnd`, and hence the optimum. The standard deviation of the difference between `greedy` and the ℓ-`bnd` is merely 4.6%. Thus, most of the time, the greedy solution is very close to the optimum.

How much do we benefit by resequencing and complementation? For that, we compare columns DOP `random` and DOPI `greedy`. DOP `random` represents the case when the data words are transmitted uncomplemented and without resequencing. The average percentage reduction in transitions is 34.4%. The maximum reduction was as high as 80% (for $n = 40, k = 5$).

How useful is complementation alone? We make two comparisons. *In the absence of resequencing*: comparing columns `random` under DOP and DOPI (`random` models no resequencing), we see that complementation yields, on average, an 11.3% reduction in transitions, a significant number. *With resequencing*: on comparing resequencing with resequencing and complementation (columns `greedy` under DOP and DOPI), we find that complementation and resequencing yield, on average, an improvement of 3.9% over resequencing. The best improvement is 33%. Also, as k or n increase, the effectiveness of *bus-invert* decreases. This indicates that although complementation by itself is good, when combined with resequencing its impact is significant only for small values of n and k. This is still fine, since in most applications such as instruction sequencing, high-level scheduling, cache write-back, etc., n is not so large and k is at most 64 (machine's word-length).

Finally, we see that the `greedy` scheme is the best. DST results are omitted for lack of space; they were almost always inferior to `greedy` and ST-MM. DST and ST-MM are important nevertheless, since they guarantee bounds with respect to the optimum.

n	k	DOP		DOPI			
		rand	**grdy**	**ℓ-bnd**	**rand**	**grdy**	**ST-MM**
5	5	12	8	7	10	7	7
5	16	30	27	27	28	27	27
5	32	64	58	55	60	55	55
20	3	29	7	7	23	7	8
20	8	70	34	28	61	32	35
20	16	146	97	83	126	92	93
40	5	99	21	19	83	20	24
40	40	784	567	523	690	543	555
100	10	496	174	141	423	172	189
100	80	4019	3049	2896	3621	2987	3101
200	20	1998	957	848	1702	922	1011
200	150	15001	12407	11522	14115	11771	12182
400	100	20116	15419	13943	18517	14377	15090
400	200	39846	33392	31378	37811	32014	32999

Results for cache write-back: $k = 32$

n	cache size	DOP		DOPI			
		rand	**grdy**	**ℓ-bnd**	**rand**	**grdy**	**ST-MM**
4	1024	1	1	1	1	1	1
18	1024	191	148	137	184	143	153
10	1024	113	89	80	110	89	85
10	1024	66	27	25	61	25	25
617	4096	7851	3582	3260	7745	3582	3915
348	4096	4205	2076	1887	4168	2076	2254
373	4096	4699	2370	2163	4617	2358	2553
138	4096	1712	991	914	1679	979	1043
965	65536	11854	5270	4749	11676	5273	5679
756	65536	9329	4218	3827	9232	4218	4610
560	65536	7066	3394	3062	6960	3384	3678
181	65536	2222	1248	1136	2194	1243	1305

Table 1: Effect of Data Resequencing and Inversion on Number of transitions

Cache Write-back Application In Table 1, we also present results for a real cache write-back application. Consider a computer system with a main memory and a cache. On a context switch, often the cache is "flushed," i.e., the dirty data lines in the cache are written back to main memory. The order in which lines are written back is often irrelevant. So we can use our data ordering and/or complementation methods to reduce transitions on the data and address buses. For these experiments, the data on cache flushes was obtained from code that performs LU-decomposition of a 30x30 matrix. This code was compiled using `gcc` and its execution was simulated in a system with the following characteristics: Harvard architecture with separate address and instruction buses, cache block size of 4 bytes, and write policy of write-back and write-allocate. For simulation, we used a modified version of the `DineroIII` cache simulator [2] and `dlxsim`, a simulator for the DLX processor. Three experiments were conducted, with unified caches of sizes 1024, 4096, and 65536 bytes, respectively. We assumed that a process switch took place every 2.5×10^5 instructions, causing a cache flush of all dirty blocks. Table 1 shows that resequencing and complementation reduce the number of transitions by 42.7%. We also note that using both complementation and resequencing does not buy us much over using resequencing alone.

In the future, we would like to study resequencing in conjunction with encoding schemes other than complementation. It would also be interesting to study the interaction among resequencing, encoding, and compression. This has interesting applications in network and satellite communication.

References

[1] Experienced Motorola Designer. Personal comm., Apr. 95.

[2] J. L. Hennessy and D. A Patterson. *Computer Architecture: A Quantitative Approach.* Morgan Kaufmann, 96.

[3] R. Murgai, M. Fujita, and S. C. Krishnan. Data Sequencing For Minimum-transition Transmission. In *VLSI'97*, Brazil.

[4] R. Murgai, M. Fujita, and A. Oliveira. Using Complementation And Resequencing To Minimize Transitions. In *Internal Report, Fujitsu Labs of America, Inc.*, Nov. 97.

[5] M. R. Stan and W. P. Burleson. Limited-weight Codes for Low-power I/O. In *Int. Work. on Low Power Design*, Apr. 94.

DAC 98, June 15-19, 1998, San Francisco, CA USA
ISBN 1-58113-049-x/98/06…$5.00