Cluster-Based Architecture, Timing-Driven Packing and Timing-Driven Placement for FPGAs

by

Alexander R. Marquardt

A thesis submitted in conformity with the requirements
for the degree of Master of Applied Science
Department of Electrical and Computer Engineering
University of Toronto

Abstract

Cluster-Based Architecture, Timing-Driven Packing and Timing-Driven Placement for FPGAs

Master of Applied Science, 1999
Alexander R. Marquardt
Department of Electrical and Computer Engineering
University of Toronto

As process geometries shrink into the deep-submicron region, interconnect resistance and capacitance account for an increasingly significant portion of the delay of circuits implemented in Field-Programmable Gate Arrays (FPGAs). One way to improve FPGA speed is to employ logic-cluster-based architectures which have high-speed local connections among groups of logic elements. In this work we show what size logic-cluster results in the best area-speed trade-off.

To obtain the best choices for a cluster-based architecture, we use computer aided design (CAD) tools to experimentally evaluate architectures with different sized logic clusters. As part of this CAD flow, we develop a timing-driven algorithm that packs logic elements into these clusters. In addition, we develop a timing-driven placement algorithm that results in significant improvements in FPGA speed over existing non-timing-driven algorithms.

Acknowledgments

I would like to thank my advisor Jonathan Rose for providing direction, motivation, and advice throughout this work. He has taught me a great deal about FPGA research.

I would also like to give thanks to Vaughn Betz. He and I spent many hours discussing FPGA architecture and CAD, and each discussion we had was educational.

I would also like to thank the students in Jonathan's research group, Yaska, Jordan, Khalid, Rob, and Paul. Through our weekly meetings, and through other informal meetings, we have shared many ideas.

I am grateful to my parents for giving me constant support and encouragement throughout my life and always having faith in me.

Table of Contents

List of Tables

List of Figures

CHAPTER 1 *Introduction*

Field-Programmable Gate Arrays (FPGAs) have become one of the most popular implementation media for digital circuits, and since their introduction in 1984, FPGAs have become a multi-billion dollar industry. The key to the success of FPGAs is their programmability, which allows any circuit to be instantly realized by appropriately programming an FPGA.

FPGAs have some compelling advantages over Standard Cells or Mask-Programmed Gate Arrays (MPGAs): faster time-to-market, lower non-recurring engineering costs (NRE), and easier debugging. Additionally, FPGAs offer designers the ability to fix errors or to add features to systems that have already been manufactured. FPGAs are also useful for implementing designs that are low volume or are required immediately, since they do not require extensive manufacturing like Standard Cells or MPGAs.

The benefits offered by FPGAs come at a price — FPGAs are at least three times slower, and require at least ten times the area of MPGAs [Brow92]. This loss in speed is mainly due to the fact that logic in FPGAs is connected via programmable switches, while in Standard Cells or MPGAs, logic is directly connected with metal wires. The programmable switches in FPGAs have high resistance and capacitance compared to the metal wiring in Standard Cells or MPGAs, and therefore reduce circuit speed. Interconnect delay is more significant (a larger proportion of circuit delay) in FPGAs than it is in MPGAs or Standard Cells, and consequently it is more important to minimize the interconnect delay in FPGAs than it is in MPGAs or Standard Cells.

Another important factor affecting circuit delay is the process used in the manufacture of an FPGA. As process geometries shrink into the deep-submicron region, interconnect[1] resistance and capacitance become increasingly significant — smaller processes which result in improvements in logic speed do not result in similar improvements in interconnect speed. The result of this is that as processes shrink, interconnect delay accounts for an increasing proportion of total circuit delay. Clearly each process shrink makes interconnect delay more and more significant, and it must be minimized to achieve the best possible circuit performance.

The quality of the computer-aided design (CAD) tools used to map circuits into an FPGA and the quality of the FPGA architecture can have a significant impact on the FPGA's performance. It is clear that interconnect delay is an increasingly important factor in the overall performance of an FPGA, so it is crucial that FPGA CAD tools and FPGA architectures minimize this delay. Our research focuses on the following two areas

1. Exploring FPGA logic block architectures to minimize interconnect delay, and
2. Developing CAD tools that minimize interconnect delay.

It is important that FPGA architecture and CAD be studied in concert, since architectural features must be properly utilized by CAD tools to be of any benefit, and CAD tool enhancements cannot be properly evaluated without a good architecture. In this thesis, we are concerned with improving FPGA performance without sacrificing large amounts of area. To accomplish this we investigate three promising aspects of FPGA architecture and CAD: Logic-cluster based FPGA architectures, timing-driven packing, and timing-driven placement. These three areas are described in the following sections.

1. Interconnect is the wiring and switches that connect logic elements.

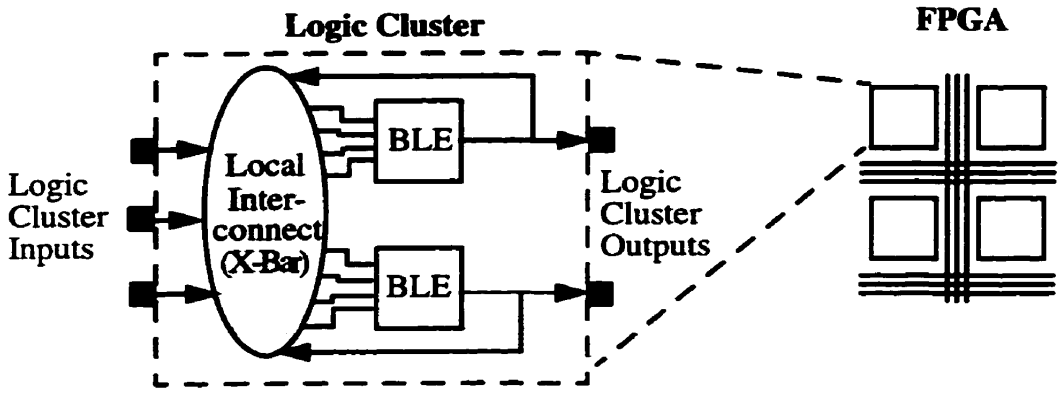


FIGURE 1.1 Example logic cluster containing two LUTs [BETZ99]

1.1 Cluster-Based Logic Blocks

An important factor affecting the area and speed of an FPGA is the logic block (logic cluster) architecture used within the FPGA. In general a logic cluster consists one or more "basic logic elements" (BLEs) connected by fast local interconnect [Betz98b, Betz99], where the BLE (described fully in Section 2.1.1) that we use consists of a 4-LUT and a register. Figure 1.1 shows an example logic cluster consisting of two BLEs and local interconnect. The size of the logic cluster (number of BLEs it contains) used in an FPGA architecture can have a dramatic effect on its area and performance. Previous work [Betz98b] demonstrated the effect of cluster size on area efficiency. Also, in [Betz98b] it was speculated that as cluster size is increased, circuit speed would be improved. As cluster size is increased, two things happen

1. More critical path connections are able to use the fast local interconnect rather than using slow inter-cluster (between cluster) interconnect, but this local interconnect becomes slower.
2. More connections are completely absorbed within clusters so less inter-cluster routing is required (reducing area), but the local interconnect area per cluster is growing quadratically (increasing area).

We are concerned with determining the effect of logic cluster size on circuit speed as well as area and finding what size logic cluster has the best area-delay trade-off. To our knowledge no work has been done which simultaneously investigates logic clusters with respect to both area and speed.

1.2 Timing-Driven Packing

To fairly evaluate different size logic clusters with respect to speed, it is important that the CAD tools take advantage of the fast local interconnect within the clusters in order to minimize the critical path delay. A packing algorithm selects how BLEs in a circuit are to be mapped into logic clusters, while a "timing-driven packing" algorithm attempts to map BLEs along the critical path into the fewest number of clusters so that many critical path connections use fast local interconnect. We give a more formal definition of packing in Section 2.2.2.

1.3 Timing-Driven Placement

Placement involves selecting the coordinates in the FPGA where each logic cluster will be mapped to. A timing-driven placement algorithm attempts to map logic clusters that are on the critical path into physical locations that are close together so as to minimize the amount of interconnect through which the critical signal must travel. Previous work [Betz99, Betz98b] has done a good job considering timing during routing, but it did not consider timing during placement. While there is evidence that timing-driven placement improves speed for standard cells, there has been no clear quantification of how much the improvement is for FPGAs. A goal of this work is to determine what improvements can be obtained with timing-driven placement. Placement is formally defined in Section 2.2.3.

1.4 Thesis Organization

This thesis is organized as follows. Chapter 2 describes FPGA architecture and CAD, and gives an overview of existing CAD tools. Chapter 3 introduces a new timing-driven logic block packing algorithm. Chapter 4 describes architecture experiments that evaluate different size logic clusters with respect to area and speed. Chapter 5 describes a new timing-driven placement algorithm. Finally, in Chapter 6 we present our conclusions and suggestions for future work.

CHAPTER 2 *Background and Previous Work*

In this chapter, we first give an overview of FPGA architecture with a focus on logic block architecture. After this we discuss the CAD flow used to map circuits into FPGAs including an introduction to timing analysis, and a detailed review of logic cluster packing, placement, and timing-driven placement.

2.1 Overview of FPGA Architecture

In general, an FPGA consists of logic blocks, I/O blocks, and programmable routing as shown in Figure 2.1. To implement a circuit in an FPGA, each of the logic blocks in the FPGA are appropriately programmed to perform a small portion of the functionality of the desired circuit, and each of the I/O blocks is programmed to be an input pad or an output pad as required by the circuit. Then these functional portions and I/Os are all appropriately connected through the programmable routing. The logic block used in an FPGA can have a significant impact on the performance of an FPGA, and since we are interested in determining the effects and trade-offs of cluster-based logic blocks, we describe cluster-based logic blocks below.

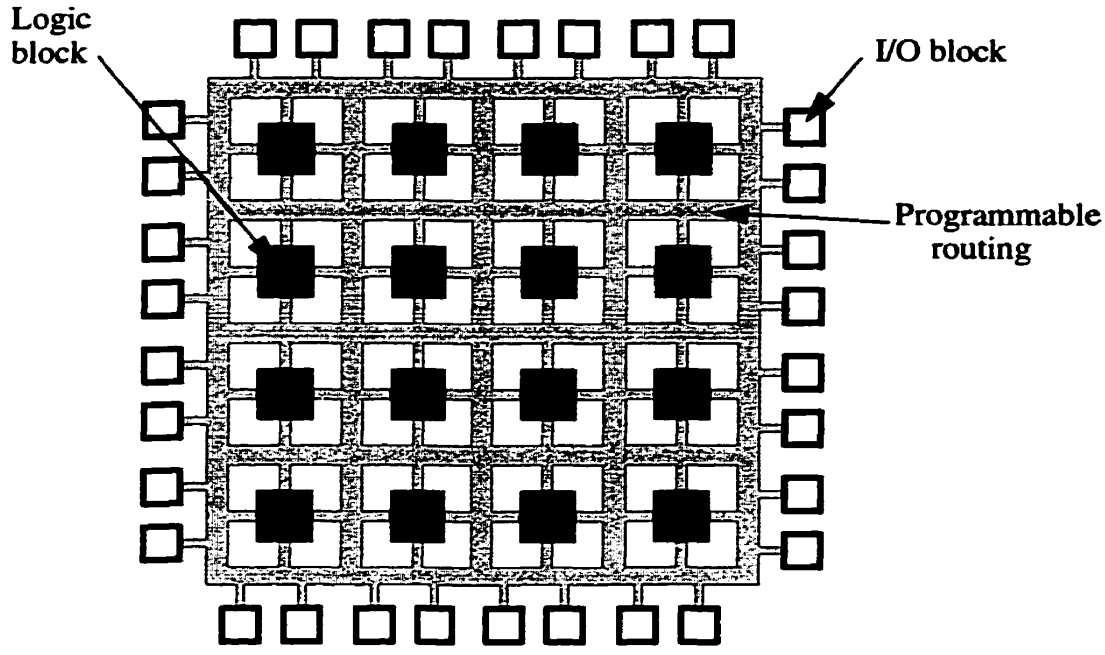


FIGURE 2.1 A generic FPGA [Brow92]

2.1.1 Cluster-Based Logic Blocks

We are interested in studying logic blocks that consist of a grouping of *basic logic elements* (BLEs) connected with fast local interconnect. In general, a BLE is a small indivisible unit combining sequential and combinational logic, while the BLE that we study consists of a 4-LUT and a flip-flop as shown in Figure 2.2-b. A logic block combining many BLEs is known as a *logic cluster* [Betz99, Betz98b]. An example of a logic cluster is the Logic Array Blocks used in Altera's FLEX 6K, FLEX 8K, and FLEX 10K parts [Alte98a], as well as the Configurable Logic Blocks used in the Xilinx 5200 [Xili97] and Virtex [Xili98] parts. Figure 2.2-a shows the structure of a logic cluster that consists of one or more BLEs and the routing required to connect them together.

The clusters that we study are *fully-connected*, meaning that any BLE input can connect to any cluster input or any BLE output. Since the cluster is fully connected it is possible to bring a net into the cluster on a single cluster input, and route this net to many BLEs within the cluster via the local routing. This allows the number of nets brought into the cluster (number of cluster inputs

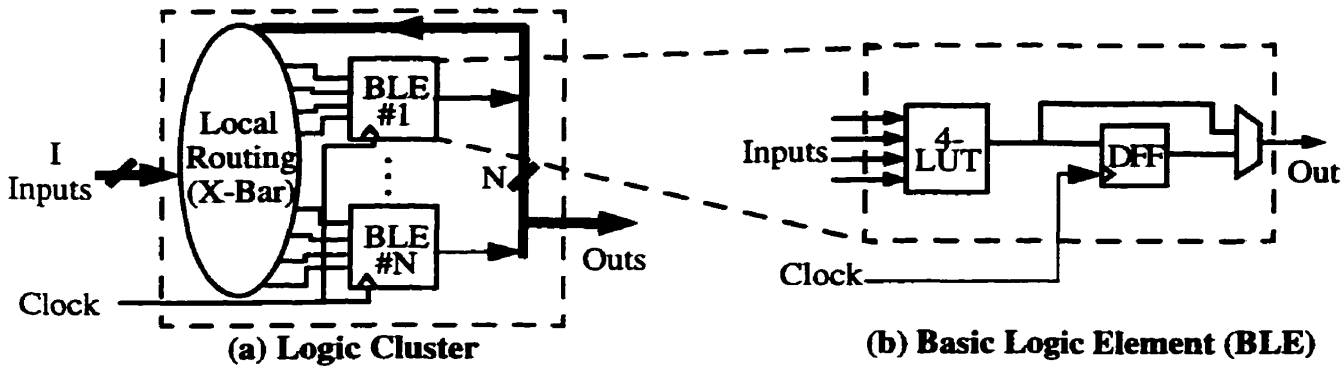


FIGURE 2.2 Logic cluster and basic logic element (BLE)

used) to be less than the total number of BLE inputs within the cluster. Another benefit of fully connected clusters is that CAD tools are simplified since all BLEs within the cluster are logically equivalent.

A logic cluster consisting of BLEs is described with the following four parameters [Betz99, Betz98b]:

1. The size of (number of inputs to) a LUT (K),
2. The number of BLEs in a cluster (N),
3. The number of inputs to the cluster for use as inputs by the LUTs (I), and
4. The number of clock inputs to a cluster (for use by the registers), *Mclk*.

The work of [Betz99, Betz98b] focused on logic clusters in which the LUT size, K, is 4 and the number of clock pins on a cluster, M_{clk}, is 1 — this is the case shown in Figure 2.2. The total number of BLE inputs is $K \cdot N$, however, only I inputs are brought into the cluster. [Betz98b] showed that a good rule of thumb[1] is to design logic clusters with $I = 2 \cdot N + 2$. Also shown was that FPGAs composed of logic clusters of size 1-10 BLEs (with the exception of size 2) have the best area efficiency. This research did not consider the effect of cluster size on circuit speed, however, it was speculated that larger cluster sizes would have a positive impact on FPGA performance.

1. This rule of thumb applies to the case when the LUT size, K, is 4. An interesting direction for future research would be to study the interactions between LUT size, K, the number of inputs to a cluster, I, and the number of BLEs in a cluster, N, and determine the best combination of these parameters.

2.2 CAD for FPGAs

Figure 2.3 illustrates the CAD flow that is used to evaluate FPGA architectures and CAD algorithms. This CAD flow mirrors the actual CAD flow employed by FPGA and ASIC designers. Each circuit we use is logic-optimized by SIS [Sent92] and then technology-mapped into 4-LUTs by FlowMap [Cong94]. VPack [Betz98b] is then used to group the LUTs and registers into logic clusters[1] of the desired size. Finally, we use VPR [Betz98b, Betz99] to place (determine the x, y position of each cluster in the FPGA) and route (connect the wires) each circuit. VPR's timing-driven router extracts the elmore delay [Elmo48] of each routed net, and performs a path-based timing analysis to determine the delay of the circuit's critical path. Finally, VPR uses a transistor-based area model [Betz98b, Betz99] to estimate the total layout area required by this FPGA to implement each circuit.

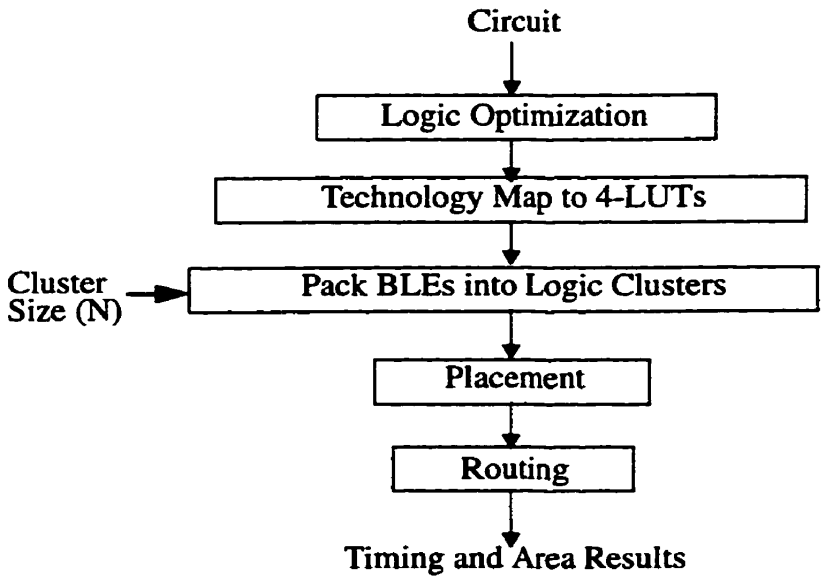


FIGURE 2.3 CAD flow

1. Note, following the convention of [Betz98b] our CAD flow shows packing and placement as two separate steps. After packing, we treat a logic cluster as an indivisible unit which is then placed. This division is not always necessary (depending on the CAD flow used), but we impose it in order to simplify the CAD tools. Another approach would be to eliminate packing, and allow the placement algorithm to move LUTs and registers freely between different clusters. This approach to placement would considerably increase the computational complexity of the placement algorithm, but would likely produce better results.

In this section we first describe how timing analysis is used to evaluate a circuit's speed, and how it guides timing-driven algorithms. Then we discuss two packing algorithms VPack and RASP. After this we discuss placement, and give an overview of Simulated Annealing and VPR's placement tool, and we discuss several timing-driven placement approaches.

2.2.1 Timing Analysis

Timing analysis [Hitc83] has two main purposes:

1. To determine the final maximum speed that a circuit implementation can achieve.
2. To determine the delay of all the paths and connections in a circuit during placement and routing, and use these as a guide to reduce the total circuit delay.

To perform a timing analysis, we must first represent the circuit as a directed graph. Nodes in the graph represent input and output pins of circuit elements such as LUTs, registers, and I/O pads. Connections[1] between these nodes are modeled with edges in the graph. These edges are annotated with a delay corresponding to the physical delay between the nodes.

To determine the delay of the circuit, a breadth first traversal is performed on the graph starting at sources (input pads, and register outputs). Then we compute the *arrival time*, $T_{arrival}$, at all nodes in the circuit with the following equation

$$T_{arrival}(i) = Max_{\forall j \in fanin(i)}\{T_{arrival}(j) + delay(j, i)\} \quad \textbf{(2.1)}$$

Where node i is the node currently being computed, and delay(j,i) is the delay value of the edge joining node j to node i. The delay of the circuit is then the maximum arrival time, D_{max}, of all nodes in the circuit.

1. In a graph representation of the circuit we define a "connection" to be an edge between a net driver and any of its terminals.

To guide a placement or routing algorithm, it is useful to know how much delay may be added to a connection before the path that the connection is on becomes critical. The amount of delay that may be added to a connection before it becomes critical is called the *slack* [Hitc83] of that connection. To compute the slack of a connection, we must compute the *required arrival time*, $T_{required}$, at every node in the circuit. We first set the $T_{required}$ at all sinks (output pads and register inputs) to be D_{max}. Required arrival time is then propagated backwards starting from the sinks with the following equation

$$T_{required}(i) = Min_{\forall j \in fanout(i)}\{T_{required}(j) - delay(i, j)\} \quad \textbf{(2.2)}$$

Finally, the slack of a connection driving node, i, is defined as:

$$Slack(i, j) = T_{required}(j) - T_{arrival}(i) - delay(i, j) \quad \textbf{(2.3)}$$

2.2.2 Packing Algorithms for Cluster-Based FPGAs

A packing algorithm takes a netlist consisting of LUTs and registers and produces a netlist consisting of logic clusters. This involves combining the LUTs and registers into BLEs, and then grouping the BLEs into logic clusters (Figure 2.4).

There are two main constraints that packing algorithms must meet:

1. The number of BLEs must be less than the cluster size, N.
2. The number of distinct inputs generated outside the cluster and used as inputs to BLEs within the cluster must be less than or equal to the number of cluster inputs, I.

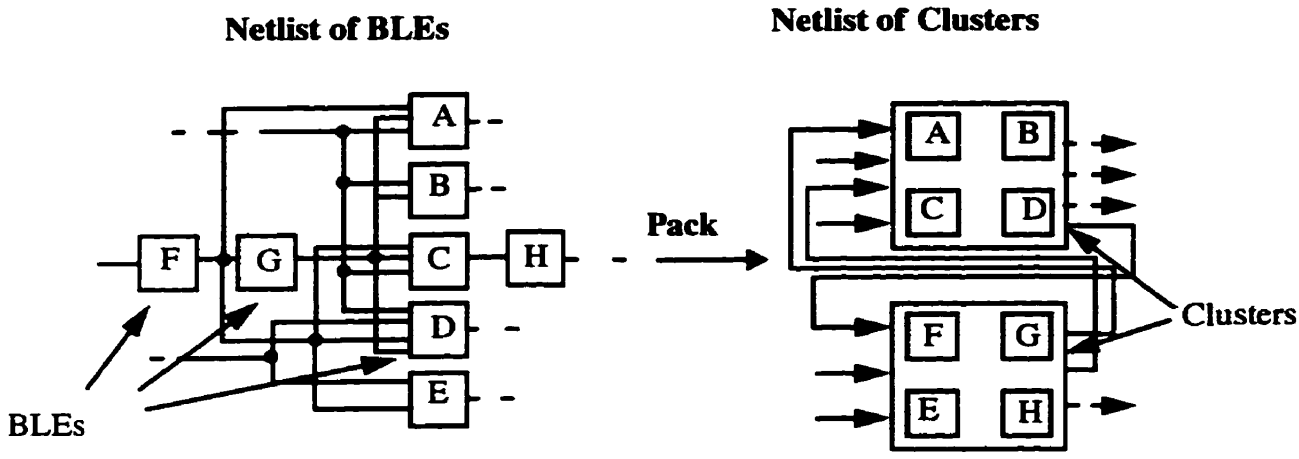


FIGURE 2.4 Packing example

Altera has an in-house tool [Alte95] that targets cluster-based logic blocks, and Xilinx has an in-house tool targeting the "cluster-like" logic blocks of the 5200 [Xili97] and Virtex [Xili98] FPGAs, however to our knowledge, this work has not been made publicly available. In this section we discuss two publicly available packing algorithms, VPack [Betz98b] and RASP [Cong96].

2.2.2.1 The VPack Logic Cluster Packing Tool

VPack [Betz98b, Betz99] takes a netlist of LUTs and registers, and produces a netlist of logic clusters. All parameters relating to the logic clustering (N, I, K, M_{clk}) are specified at run-time. VPack first groups LUTs and registers into BLEs, and then packs the BLEs into logic clusters. The pseudo-code for the VPack algorithm is given in Figure 2.5 [Betz98b, Betz99].

The VPack algorithm has two optimization goals. The first is to pack each logic cluster to its capacity to minimize the number of clusters needed. The second goal is to minimize the number of inputs to each cluster in order to reduce the number of connections required between clusters.

```
Let: UnclusteredBLEs be the set of BLEs not contained in any cluster
     C be the set of BLEs contained in the current cluster
     LogicClusters be the set of clusters (where each cluster is a set of
BLEs)

UnclusteredBLEs = PatternMatchToBLEs (LUTs, Registers);
LogicClusters = NULL;

while (UnclusteredBLEs != NULL) {    /* More BLEs to cluster */
     C = GetBLEwithMostUsedInputs (UnclusteredBLEs);
     while (|C| < N) {   /* Cluster is not full */
          BestBLE = MaxAttractionLegalBLE (C, UnclusteredBLEs);
          if (BestBLE == NULL)  /* No BLE can be added to cluster */
               break;
          UnclusteredBLEs = UnclusteredBLEs - BestBLE;
          C = C ∪ BestBLE;
     }
     LogicClusters = LogicClusters ∪ C;
}
```

FIGURE 2.5 Pseudo-code for VPack [Betz98b, Betz99]

Vpack uses a greedy algorithm to construct each cluster sequentially. At the start of each cluster operation, VPack selects as a "seed" an unclustered BLE with the most used inputs, and then places this "seed" into a cluster *C*. Then VPack selects a new BLE, *B* to pack into *C* based on the *attraction* that *B* has to *C*. Attraction is determined by the number of inputs and outputs that *B* and *C* have in common:

$$Attraction(B) = |Nets(B) \cap Nets(C)| \quad \textbf{(2.4)}$$

BLEs are added to the current cluster until it cannot fit any more, at which point packing begins on a new cluster. The process terminates when there are no more unclustered BLEs left.

The time complexity of this algorithm is $O(k_{max} \cdot K \cdot n)$ which is a result of the fact that when each BLE is clustered (n BLEs) we must examine all of the nets attached to the BLE (K nets), and we must examine all BLEs that each net fans out to (maximum fanout = k_{max}). This results in an execution time of about four seconds to pack the largest MCNC[1] circuit (clma) [Yang91] on a 296 MHz UltraSPARC-II processor.

2.2.2.2 RASP

In [Cong96] the RASP logic block packing tool is described. This tool is capable of mapping circuits represented as a network of LUTs into several different types of logic blocks. This algorithm uses a "closeness" cost function to weigh the desirability of mapping LUTs into the same logic block. This closeness cost function can be set up to prefer to minimize delay or area, or to maximize routability. The closeness of two LUTs is marked on an edge in a "compatibility graph" if it is allowable to pack the two LUTs into one logic block. If the LUTs cannot be packed together (i.e. they violate some hard constraint such as number of inputs or BLEs allowed) then there is no edge put into the compatibility graph. The packing step selects LUTs to pack together by performing a maximum weighted matching on the compatibility graph. The complexity of this algorithm is O(nm) where n is the number of LUTs, and m is the number of edges in the compatibility graph. With the logic blocks used in our research, the number of edges, m, in the compatibility graph is $O(n^2)$, which leads to an algorithm complexity of $O(n^3)$.

2.2.3 Placement

Placement is the process by which a netlist of circuit blocks (I/Os or logic clusters) is mapped into physical locations in an FPGA. The locations that blocks are mapped to can significantly affect the performance of the FPGA. There are three main goals that placement algorithms may attempt to satisfy:

1. We give a brief overview of the 20 largest MCNC circuits in Appendix A.

1. To minimize the amount of wiring required, which we refer to as wirelength-driven placement.
2. To balance the wiring density across the FPGA, called routability-driven placement.
3. Minimize the delay of the critical path(s), called timing-driven placement.

Placement algorithms may simultaneously satisfy one or more of these goals.

In the remainder of this section we review the Simulated Annealing algorithm that is commonly applied to placement problems. Then we discuss the Simulated Annealing-based placer built into VPR [Betz98b, Betz99] which we call VPlace. After this we review various timing-driven placement approaches.

2.2.3.1 Simulated Annealing

The Simulated Annealing placement algorithm mimics the annealing process used to gradually cool molten metal to produce high-quality metal structures [Kirk83]. A Simulated Annealing-based placer initially places logic clusters and I/Os (circuit blocks) randomly into physical locations in an FPGA. Then the placement is iteratively improved by randomly swapping blocks and evaluating the goodness of each swap with a cost function. If the move will result in a reduction in the placement cost, then the move is accepted. If the move would cause an increase in the placement cost, then the move may still be accepted even though it makes the placement worse. The purpose of accepting some "bad" moves is to prevent the Simulated Annealing-based placer from becoming trapped in a local minimum.

The probability of accepting a "bad" move is given by $e^{-\Delta C/T}$, where ΔC is the positive change in cost function that acceptance of the move would result in, and T is a parameter called temperature that controls the likelihood of accepting each move. Initially, a Simulated Annealing-based placer starts at a high temperature, so that almost all moves are accepted, then the temperature is gradually reduced so that the probability of accepting moves that make the placement worse becomes very low. In the final stages of placement only moves that decrease the placement cost are accepted.

```
S = RandomPlacement ();
T = InitialTemperature ();
R_limit = InitialR_limit ();

while (ExitCriterion () == False) {        /* "Outer loop" */
    while (InnerLoopCriterion () == False) {    /* "Inner loop" */
        S_new = GenerateViaMove (S, R_limit);
        ΔC = Cost (S_new) - Cost (S);
        if (ΔC < 0) {
            S = S_new /*Move is good, accept*/
        }
        else {
            r = random (0,1);
            if (r < e^(-ΔC/T)) {
                S = S_new; /*Move is bad, accept any way*/
            }
        }
    }          /* End "inner loop" */
    T = UpdateTemp ();
    R_limit = UpdateR_limit ();
}   /* End "outer loop" */
```

FIGURE 2.6 Pseudo-code of a generic Simulated Annealing-based placer [Betz98b, Betz99].

In the final (low temperature) stages of the placement, if all blocks in the FPGA are considered for swapping, most swaps will be rejected because they result in large positive changes in the cost function. To increase the number of accepted moves at low temperatures, only blocks that are close together should be considered for swapping since "local swaps" tend to result in relatively small changes in the placement cost. Accordingly, a Simulated Annealing-based placer uses a parameter called R_{limit} ("range limiter") that controls how close together circuit blocks must be to be considered for swapping. Initially, R_{limit} spans the entire FPGA which means that blocks on opposite sides of the FPGA may be considered for swapping. As the placement proceeds, R_{limit} is decreased, so that in the final stages of placement, only blocks that are close together are considered for swapping.

In Figure 2.6 we show the pseudo-code for a generic Simulated Annealing-based placer, as presented in [Betz98b, Betz99].

2.2.3.2 The VPR Placement Tool (VPlace)

In this document we will refer to the placement algorithm used within VPR as VPlace. VPlace is a Simulated Annealing-based placement algorithm that attempts to minimize the wirelength of the resulting circuit by placing circuit blocks that are on the same net close together. To accomplish this, VPlace uses a bounding-box based "linear congestion" [Betz98b, Betz99] cost function to estimate wirelength requirements. The VPlace algorithm follows the format of the pseudo-code shown in Figure 2.6.

The linear congestion cost function has the following functional form [Betz98b, Betz99]

$$Cost_{linear\ congestion} = \sum_{i=1}^{N_{nets}} q(i) \cdot [bb_x(i) + bb_y(i)] \quad \textbf{(2.5)}$$

where there are N_{nets} in the circuit. The cost of each net, i, is determined by its horizontal span, $bb_x(i)$, and its vertical span, $bb_y(i)$. The q(i) factor compensates for the fact that the bounding box wire length model underestimates the wiring necessary to connect nets with more than three terminals. The values used for q(i) were obtained from [Chen94] so that q(i) is set to 1 for nets with 3 or fewer terminals, and it slowly increases to 2.79 for nets with 50 terminals. Beyond 50 terminals, the q(i) function linearly increases at the rate of

$$q(i) = 2.79 + 0.02616 \cdot (Num\_Terminals - 50). \quad \textbf{(2.6)}$$

The complexity of this algorithm is $O(n^{4/3})$ where n is the number of blocks in the circuit.

2.2.3.3 Timing-Driven Placement

Placement algorithms that attempt to minimize the critical path delay of the resulting circuits are called timing-driven. There are different approaches to minimizing critical path delay in timing-driven placement algorithms. One approach which we call "path-based" timing-driven placement computes path delays at every stage of the placement, and uses theses delays in its cost function. This path-based approach is computationally expensive since path delays must be continuously re-computed. Another approach is "connection-based" timing-driven placement, which involves

performing a path-based timing analysis and assigning slacks to each connection in the circuit. Then during placement, more attention is paid to connections with low slack, but the more global view of the complete path delay is not used. It is also possible to combine connection-based and path-based timing-driven placement by periodically performing a full path analysis based on the current placement, and then updating the slacks on individual connections.

In this section we discuss the existing timing-driven placement algorithms that are most relevant to our work.

TimberWolfSC

The TimberWolfSC timing driven placement algorithm for row-based standard cell ICs is presented in [Swar95]. This algorithm uses a Simulated Annealing approach to placement. In this algorithm, net delay is computed as

$$Net\ Delay = T_{driver} + R_{driver} \cdot (C_{net} + C_{gates}) \tag{2.7}$$

Where T_{driver} is the intrinsic delay of the driver, R_{driver} is the resistance of the driver, C_{net} is the estimated capacitance of the net, and C_{gates} is the gate input capacitance of all sinks on the net. The arrival time at the sink of a path is the summation of all of the net delays along that path. This formulation of delay assumes that the driver resistance is much larger than the wiring resistance (so that it can ignore wiring resistance). The fact that wiring resistance is ignored likely makes these net delays optimistic, especially for circuits implemented in deep-submicron processes where wiring resistance and delay is significant.

The cost function used in this algorithm penalizes any paths where the arrival time is greater than the required (user defined) arrival time with the following:

$$Penalty = T_{arrival} - T_{required} \tag{2.8}$$

The total timing penalty P_t is the sum of all critical path penalties.

$$P_t = \sum_{\forall paths} Penalty \tag{2.9}$$

The cost function consists of two terms, a wire length term represented by W, total timing penalty, P_t, and a trade-off variable λ that trades off between the two terms

$$Cost = W + \lambda \cdot P_t \tag{2.10}$$

The authors of [Swar95] found that setting

$$\lambda = 3 \cdot \frac{\overline{\Delta W}}{\overline{\Delta P_t}} \tag{2.11}$$

gave the best results, where $\overline{\Delta W}$ is the average change in wire length and $\overline{\Delta P_t}$ is the average change in the timing penalty measured during the first "outer loop" iteration of a Simulated Annealing algorithm. This implies that changes in the timing penalty are three times as important as changes in the wire length.

The authors presented results for three MCNC standard cell circuits, for which timing information was previously available. Compared to the previous results they reduced delay by 28% - 50% at an area cost of between 2.5% and 6%. It is not clear from the paper how the previous timing results were obtained. This algorithm is path based, so the computational complexity is likely quite high, but is not revealed in the paper.

PROXI

In [Nag95] a performance-driven simultaneous place and route algorithm (PROXI) is presented.

After each placement perturbation in the anneal, a small subset of relevant nets (previously unroutable and newly disturbed nets) is ripped up and rerouted with a fast maze router. As the placement evolves the critical path is evaluated. The cost function used in this algorithm is

$$Cost = W_r \cdot R + W_t \cdot T \tag{2.12}$$

Where R is the number of unrouted nets and T is the critical path. W_r and W_t are weights that are determined adaptively at runtime so as to normalize the components of the cost function so that each term contributes equally to the cost function. This algorithm is unique in that it performs placement and routing simultaneously — most place and route software does placement first, and then routes the placed circuit. Performing placement and routing in one stage should theoretically give better results than a two stage (place then route) algorithm, however it is much more computationally expensive.

This algorithm achieves 8% - 15% improvement in delay when compared to the Xilinx XACT5.0 place and route system. This algorithm, however, has a significant disadvantage in CPU compile time compared to the XACT5.0 tool, ranging from 6 times for the smallest design (12x12 array), to 11 times for the largest design (16x16 array).

2.3 *Summary*

In this chapter we presented an overview of FPGA architecture including a description of cluster based logic blocks [Betz99, Betz98b]. Then we discussed CAD for FPGAs. This included discussions of timing analysis, packing algorithms, and placement.

CHAPTER 3 *Timing-Driven and Connection-Driven Packing*

In this chapter we first discuss the experimental methodology that we use to evaluate different CAD algorithms and FPGA architectures. Then we introduce two new packing algorithms that are extensions to the VPack [Betz98b, Betz99] algorithm. The first is a timing-driven packing algorithm that we call T-VPack, and the second is a connection-absorption-driven packing algorithm that we call C-VPack. We then compare the results of both of these algorithms to the results of VPack.

3.1 Experimental Methodology

The CAD flow that we use to evaluate different CAD algorithms and FPGA architectures is the same as in [Betz98b, Betz99], and is given in Figure 3.1. First each circuit is logic-optimized by SIS [Sent92] and technology mapped into 4-LUTs by FlowMap [Cong94]. T-VPack (described in Section 3.2) is then used to group the LUTs and registers into logic clusters of the desired size with the desired number of inputs. Then VPR is used to place and route each circuit. The placement algorithm in VPR is simulated annealing based and optimizes the final placement to minimize the required routing area. The router in VPR is fully timing-driven and attempts to minimize the critical path delay (given the current placement). After placement and routing, we

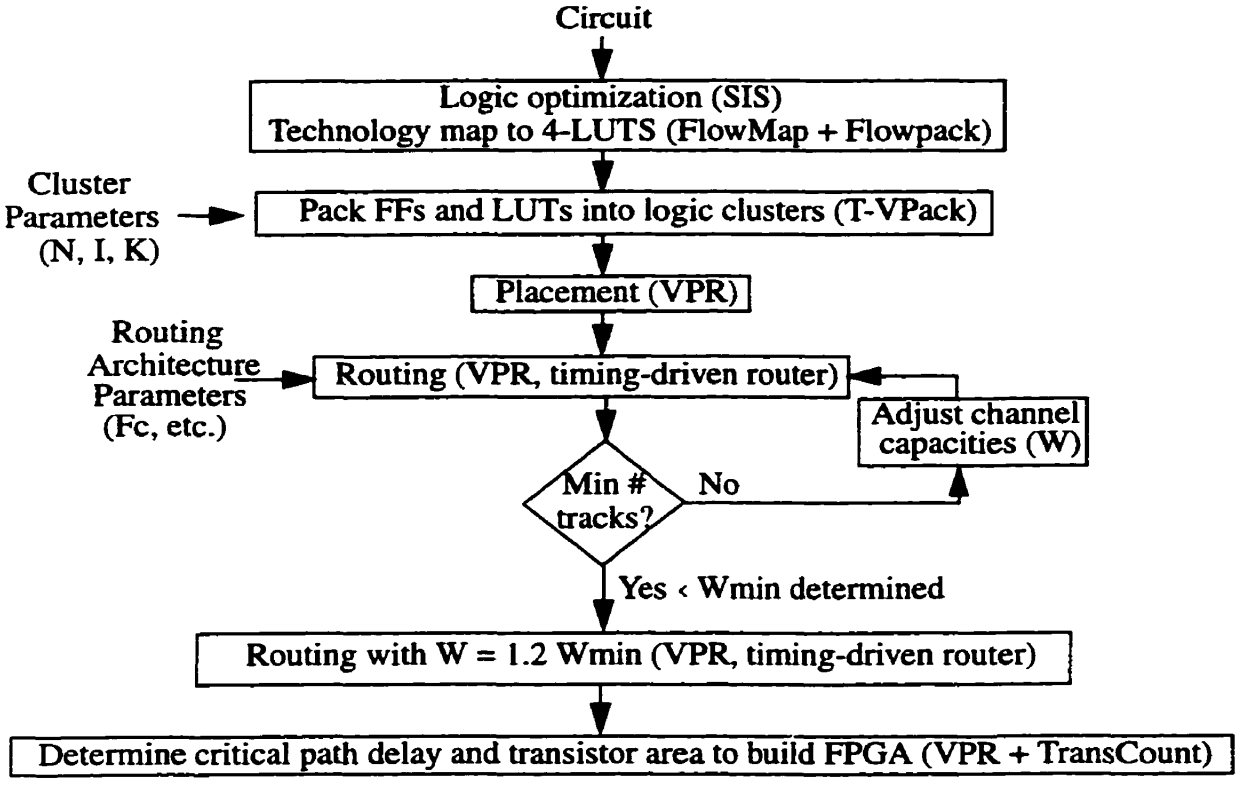


FIGURE 3.1 Architecture evaluation CAD flow [Betz98b, Betz99].

know the estimated area and track width required to implement each circuit and the estimated critical path delay, where area and delay values are computed using the area and delay models described in the next chapter.

Figure 3.1 shows how VPR computes the minimum number of tracks in which a circuit will route, which we refer to as a *high-stress* routing. Basically VPR repeatedly routes each circuit with different channel widths (number of tracks per channel), scaling the FPGA's architecture until it finds the minimum number of tracks in which the circuit will route. We define a *low-stress* routing (as does [Swar98a]) to occur when an FPGA has 20% more routing resources than the minimum required to route a given circuit. We feel that low-stress routings are indicative of how an FPGA will generally be used (it is rare that a user will utilize 100% of all routing and logic

resources), so many of our delay results are based on low-stress routings. We also present results that are based on an infinite[1] number of routing resources. These infinite routing results tell us the best possible router-achievable speed of a circuit given the current packing and placement of that circuit. We feel that is a useful indicator of how well a packing or placement algorithm performs with respect delay.

By allowing the channel width to vary, and searching for the minimum routable width, we can detect small improvements in FPGA architectures or CAD algorithms that might otherwise go unnoticed. Compare this to mapping a circuit into a fixed size FPGA — this would only tell us if the circuit fit or not. A "binary" result like this makes it is difficult to draw conclusions about new architectures or CAD algorithms.

3.2 *Timing-Driven Packing: T-VPack*

Our timing-driven logic block packing algorithm, T-VPack, attempts not only to pack each logic block to capacity and minimize the number of cluster inputs used, but also to minimize the number of inter-cluster (between cluster) connections on the critical path(s). The local routing within clusters is faster than the general-purpose routing between logic clusters, so reducing the number of inter-cluster connections on the critical path(s) reduces circuit delay. The basic operation of the algorithm is the same as that of the VPack algorithm described in Section 2.2.2.1 with a few modifications. We show the pseudo-code for the T-VPack algorithm in Figure 3.2.

T-VPack first performs a timing analysis (defined in Section 2.2.1) to determine the critical path(s) of the circuit. Then T-Vpack finds a "seed" BLE by selecting a BLE on the critical path(s) rather than selecting a BLE with the most used inputs. BLEs are then added to the current cluster

1. Infinite routing resource results are delay results from the router when it ignores congestion, i.e. the router is allowed to use a single resource for multiple un-related connections. This allows the router to allocate the fastest possible resource for every connection in the circuit. See [Betz98b, Betz99] for a detailed description of how the router in VPR works.

```
Let: UnclusteredBLEs be the set of BLEs not contained in any cluster
     C be the set of BLEs contained in the current cluster
     LogicClusters be the set of clusters (where each cluster is a set of BLEs)

UnclusteredBLEs = PatternMatchToBLEs (LUTs, Registers);
LogicClusters = NULL;

ComputeCriticalities();
BLEsSinceLastCriticalityRecompute = 0;

while (UnclusteredBLEs != NULL) {    /* More BLEs to cluster */

    C = GetMostCriticalBLE (UnclusteredBLEs);
    BLEsSinceLastCriticalityRecompute ++;

    while (|C| < N) {   /* Cluster is not full */

        if (BLEsSinceLastCriticalityRecompute >= RecomputeInterval) {
            ComputeCriticalities();
            BLEsSinceLastCriticalityRecompute = 0;
        }

        BestBLE = MaxAttractionLegalBLE (C, UnclusteredBLEs);
        if (BestBLE == NULL)   /* No BLE can be added to cluster */
            break;
        UnclusteredBLEs = UnclusteredBLEs - BestBLE;
        C = C ∪ BestBLE;
        BLEsSinceLastCriticalityRecompute ++;

    }
    LogicClusters = LogicClusters ∪ C;
}
```

FIGURE 3.2 Pseudo-code for T-VPack

based on the attraction they have to the current cluster, where the attraction function is modified to prefer to absorb connections along the critical paths(s). After each cluster is full, packing begins on a new cluster.

In this section we first discuss timing-analysis and delay modeling within T-VPack. Then we give details of the algorithm implementation. After this we provide an analysis of T-VPack to see the effect of various parameters within T-VPack. Finally after this we analyze the complexity of the algorithm.

3.2.1 Timing Analysis and Delay Models

To minimize the number of inter-cluster connections on the critical path(s), T-VPack first needs to determine which connections are on the critical path(s). Accordingly, T-VPack performs a timing analysis to determine the slack of each connection between BLEs. The timing analyzer within T-VPack models three types of delay: the delay through a BLE, or *LogicDelay*, the connection delay between blocks within the same cluster or *IntraClusterConnectionDelay*, and the connection delay between blocks that are in different clusters, or *InterClusterConnectionDelay*. The delay of a connection between two BLEs in different logic clusters is not known until after a circuit has been placed and routed, so T-VPack approximates the delay between clusters as a constant InterClusterConnectionDelay. Note that this leads to some inaccuracy in T-VPack's estimate of where the critical path(s) lies, so that sometimes T-VPack will be attempting to shorten a path that will not be part of the post-place-and-route critical path(s). The performance of T-VPack is not very sensitive to the exact values chosen for these three delay parameters. Throughout this work we set LogicDelay to 0.1, IntraClusterConnectionDelay to 0.1 and InterClusterConnectionDelay to 1.0. Note that the timing analysis can be performed as often as the user specifies, i.e. a timing analysis can be performed after each BLE is clustered, or at the other end of the spectrum timing analysis may be done once at the beginning of the algorithm execution and never again. The effect of this *recompute interval* is discussed in Section 3.2.3.

3.2.2 Timing-Driven Packing Description

After a timing analysis is complete, we are able to begin packing. This section describes how we determine which BLE will be selected as a "seed" for each cluster, and how BLEs to be added to each cluster are selected. We first define many sub-equations that are used in selecting a cluster seed and in the attraction function. After these preliminaries, we finally present how we select a cluster seed, and our new attraction function.

3.2.2.1 Preliminary Definitions

We define the criticality of a connection, i, to be

$$ConnectionCriticality(i) = 1 - \frac{slack(i)}{MaxSlack} \quad (3.1)$$

where *MaxSlack* is the largest slack amongst all connections in the circuit.

The *Criticality* of a BLE is computed as follows. Let us first define the base criticality of an unclustered BLE, or *BaseBLECrit(B)*. BaseBLECrit is defined slightly differently depending on whether we are choosing a seed BLE for a new cluster or computing the attraction of a BLE to the current cluster:

1. When we are choosing a seed BLE, BaseBLECrit(*B*) is the maximum ConnectionCriticality value amongst all of BLE *B*'s connections; or
2. When we are computing the attraction of a BLE to the current cluster, BaseBLECrit(*B*) is the maximum ConnectionCriticality value amongst all the connections joining BLE *B* to BLEs within the cluster currently being packed, *C*. If a BLE does not have any connections to *C* then its base criticality score is zero.

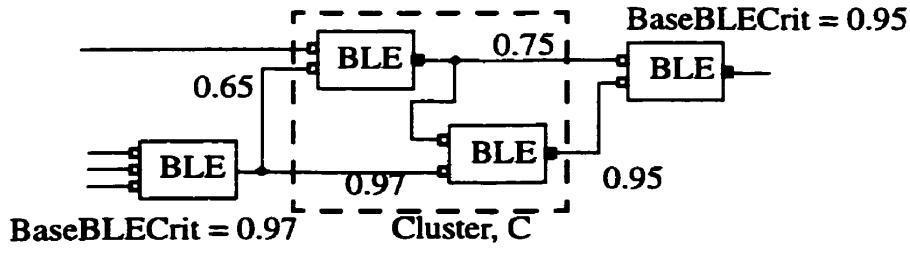


FIGURE 3.3 Determining BaseBLECrit from connection criticalities.

In Figure 3.3 we illustrate how the BaseBLECrit values are assigned when we are computing the attraction of a BLE to the current cluster. Each connection between unclustered BLEs and BLEs within the cluster C is labelled with its ConnectionCriticality value. Notice how the base criticality of each BLE is set to the highest criticality amongst the connections between it and the cluster being packed.

During packing, multiple BLEs often have the same base criticality value. In this case, we use a tie-breaker mechanism to select which BLEs are the most beneficial to pack. This mechanism is designed to choose (from the BLEs tied with the highest base criticality value) the BLE whose packing would reduce the length of the largest number of critical paths. This is best illustrated by an example.

In Figure 3.4 we have darkened connections and BLEs on the critical paths. Notice that when selecting which BLEs to place into a cluster, it is more beneficial to absorb certain critical BLEs over other critical BLEs. In this case, absorbing BLEs X, Y, and Z would be much more beneficial than absorbing BLEs Q, T, and V. We can see that absorbing X, Y, and Z affects three partially-overlapping critical paths, and will shorten the lengths of the critical path(s) that seven other BLEs (Q, R, S, T, U, V and W) are on. On the other hand, absorbing Q, T, and V affects only one critical path, and will not reduce the criticality of any other BLEs, since all the other critical BLEs would still lie on a critical path after the packing of Q, T, and V into a single cluster. Clearly it is best to cluster BLEs that reduce the criticalities of the most other BLEs.

We define three variables that keep track of the number of critical paths that each BLE in the circuit affects. First we define *InputPathsAffected* as the number of critical paths between timing path sources (primary inputs or register outputs) and the BLE currently being labelled. Next we define *OutputPathsAffected* as the number of critical paths between the BLE currently being labelled and the timing path sinks (primary outputs or register inputs). Finally, we define *TotalPathsAffected* as the sum of the previous two variables. The calculation of these variables is explained below.

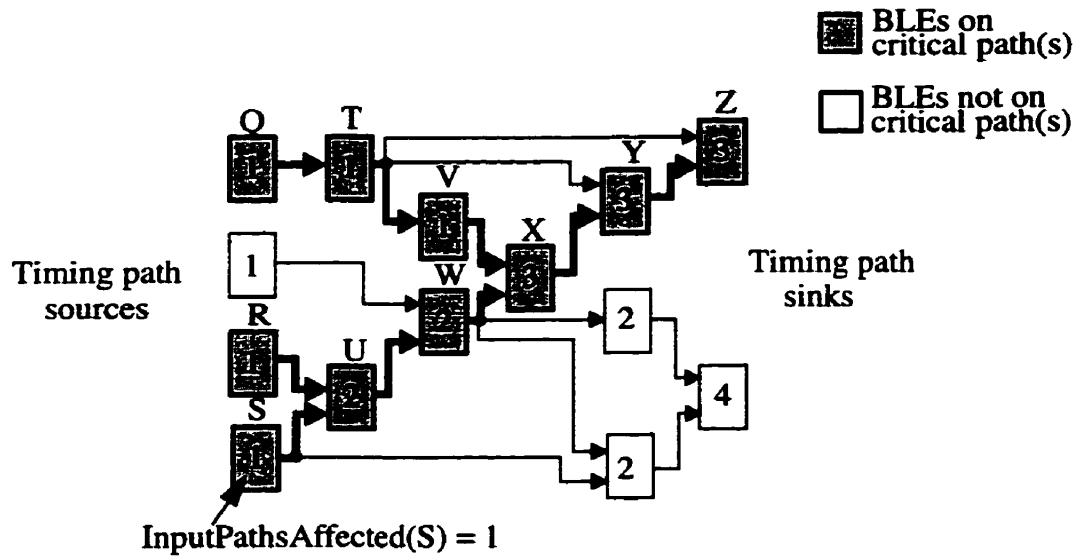


FIGURE 3.4 Example of first criticality tie-breaker.

An example of the computation of InputPathsAffected is shown in Figure 3.4. In this figure each BLE is labelled with its InputPathsAffected value. We assign all timing path source nodes an InputPathsAffected value of one. Then we perform a breadth-first traversal of the circuit starting at the sources, and define the InputPathsAffected value as

$$InputPathsAffected(B) = \sum_{\forall D \in \textit{most critical inputs}(B)} InputPathsAffected(D) \qquad (3.2)$$

where *most critical inputs(B)* refers to all of the BLE(s) driving the connections on B's input(s) that have a criticality value equal to the largest criticality of any input on B. The OutputPathsAffected variable is calculated in the same manner, but it starts at timing path sink nodes and works back toward the timing path sources.

$$OutputPathsAffected(B) = \sum_{\forall D \in \textit{most critical outputs}(B)} OutputPathsAffected(D) \qquad (3.3)$$

TotalPathsAffected is then simply

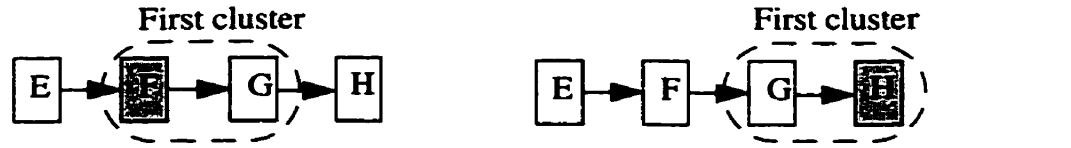
$$TotalPathsAffected(B) = InputPathsAffected(B) + OutputPathsAffected(B) \qquad (3.4)$$

When two BLEs have the same BaseBLECrit, we break the tie by choosing to insert the BLE with the higher TotalPathsAffected value in the cluster. While this breaks most ties, it does not resolve all of them. Consider the simple circuit shown in Figure 3.5, and the selection of a seed BLE for the first cluster. There is only one path through the circuit, so all 4 BLEs are on the critical path, and have a BaseBLECrit value of 1. Similarly, all 4 BLEs have a TotalPathsAffected value of 2, so we have a four-way tie for the best BLE to use as a seed. Figure 3.5(a) shows a potential outcome if we randomly choose one of the four tied BLEs as the cluster seed when the cluster size is 2. If we choose BLE F as the cluster seed, and then BLE G is chosen as the second BLE in this cluster, we have "marooned" BLEs E and H — it is not possible to pack either E or H with its fan-in or fan-out. If instead, we choose BLE H as the seed of the first cluster (as Figure 3.5(b) shows), the first cluster would contain G and H, and E and F could still be packed together into one cluster. Clearly, the clustering shown in Figure 3.5(b) is preferable to that of Figure 3.5(a).

We use a second tie-breaker mechanism to break ties not resolved by the first tie-breaker to reduce the likelihood of "marooning" BLEs. We always choose to pack the tied BLE that is the farthest from the timing path sources (i.e. is the closest to the timing sinks).[1] In Figure 3.5, for example, this second tie-breaker causes T-VPack to always choose BLE H as the seed of the first cluster, so the superior clustering solution of Figure 3.5(b) is achieved.

The criticality of a BLE is its BaseBLECrit slightly adjusted by these two tie-breakers:

$$Criticality(B) = BaseBLECrit(B) + \varepsilon \cdot TotalPathsAffected(B) + \varepsilon^2 \cdot D_{source}(B) \quad \textbf{(3.5)}$$



(a) BLE F chosen as seed of first cluster (b) BLE H chosen as seed of first cluster

FIGURE 3.5 Example of second criticality tie-breaker.

1. Note that choosing to always pack the tied BLE closest to the timing path sources would work just as well. The key is simply to ensure that one consistently chooses BLEs from one end of a chain of tied BLEs, rather than from the middle or from a mix of the two ends.

where ε is a small number (e.g. 0.01) to ensure that the second two terms function only as tie-breakers, and $D_{source}(B)$ is a BLE's distance, or level, from the timing path sources. We show the effectiveness of incorporating these tie-breakers in Section 3.2.3.

3.2.2.2 Seed Selection and Attraction Function

Now that we have all of the preliminary definitions out of the way, we can explain how we select a cluster seed and define our attraction function.

The seed BLE packed into a new cluster is the unclustered BLE with the highest *Criticality*. Once a seed is chosen, the attraction function used to determine the next unclustered BLE, *B*, to be added to the current cluster, *C*, is:

$$Attraction(B) = \alpha \cdot Criticality(B) + (1 - \alpha) \cdot \frac{|Nets(B) \cap Nets(C)|}{MaxNets} \quad \textbf{(3.6)}$$

Notice that the second term in (3.6) is essentially the attraction function from the original VPack algorithm. The MaxNets factor in the denominator of this term is the maximum number of nets that could connect to any BLE ($I + N + M_{clk}$) and simply normalizes the magnitude of the second term. The first term in (3.6) promotes the grouping of BLEs with high criticalities (defined below) into the current cluster to minimize delay. α is a parameter that controls the trade-off between net sharing and delay minimization. If α is 0, we have an algorithm that focuses solely on minimizing the number of used inputs to a cluster, and is equivalent[1] to the basic VPack algorithm described in Section 2.2.2.1. If α is 1, the algorithm focuses solely on minimizing the delay of a circuit, with no regard for how many nets are shared by the BLEs within a cluster. Using the CAD flow described in Section 2.2 we have experimentally determined that any α value in the range from 0.4 to 0.9 produces the best quality packings in terms of both post-place-and-route delay and routing area (channel width). The trade-off curves for channel width and critical path delay vs. α

1. The attraction function is equivalent to the VPack attraction function if α is 0, but the seed selection is still based on the maximum criticality.

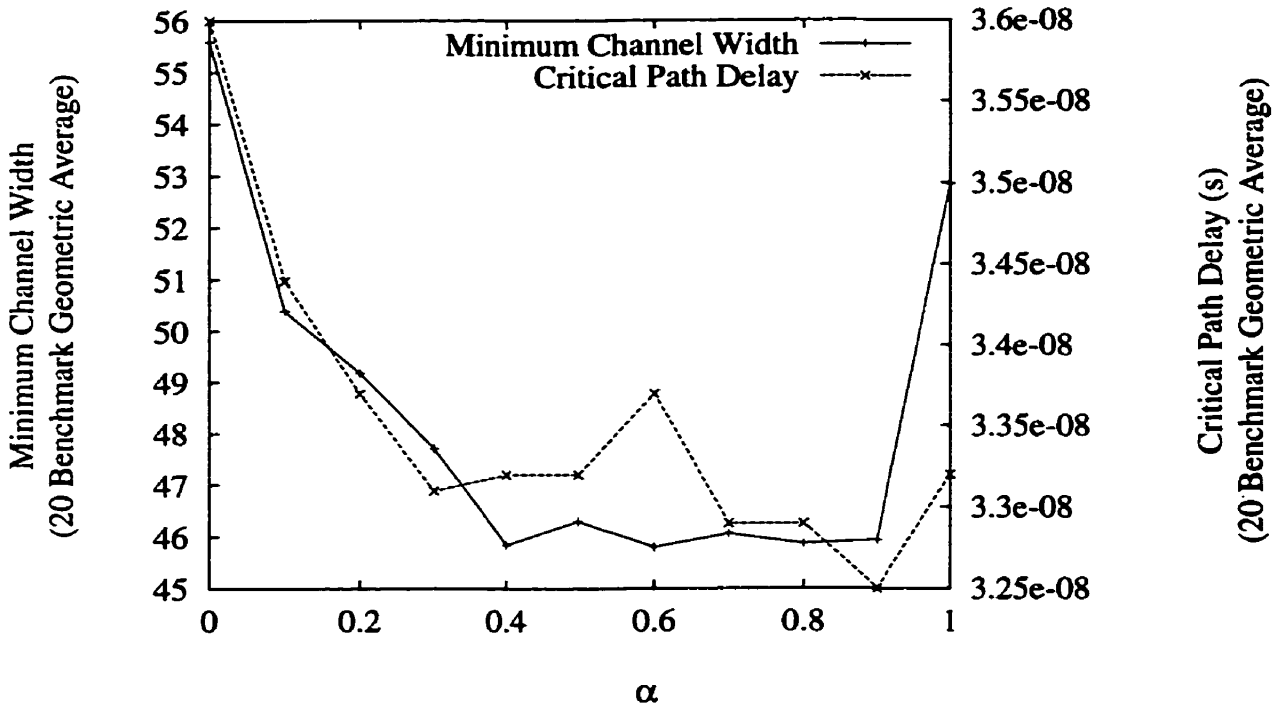


FIGURE 3.6 Post place and route T-VPack alpha trade-off curves.

for an architecture using size 8 clusters are shown in Figure 3.6. In this figure, the critical path delay is computed based on an architecture with infinite routing resources. Throughout our work we set α to 0.75.

3.2.3 Algorithm Analysis

In this algorithm, there are two factors that may impact the quality of the packing solutions. First, what is the effect of the tie-breaker enhancements on the overall quality. Second, the complexity of T-VPack depends on how often timing analysis is performed on the circuit as BLEs are packed into logic clusters, so we are interested in how often we should recompute timing information to get the best results. We call this the *recompute interval*. We perform a new timing analysis if the number of BLEs packed since the last timing analysis is greater than or equal to the recompute interval. The pseudo-code for T-VPack was given in Figure 3.2.

In Table 3.1 we show the effect of the tie-breakers and the recompute interval. This table shows that in the infinite routing case, the tie-breakers improve the post-place-and route speed by about 4% compared to not using these tie-breakers. It also shows that computing the critical path only once during the algorithm execution and using tie-breakers results in a 2% speed degradation compared to recomputing the timing information after each BLE is clustered and using tie-breakers. The low stress routing case shows that the three cases have roughly the same critical path delay. We feel that the infinite routing results are a good indicator of how well the packing algorithm performs with the various options since these infinite routing results reduce variations due to the router.

It is likely that the reason that the recompute interval has such a small effect on the quality of the resulting circuits is because the T-VPack predicted critical path (which uses the same delay for each inter-cluster connection) and the post-place-and-route critical path are different enough that there is no benefit to updating connection criticalities as more information about the circuit packing becomes available. Since the post-place-and-route quality is not dramatically affected by the recompute interval, we use a recompute interval of ∞ (compute criticality once) for the results presented in the remainder of this thesis. This dramatically reduces the execution time of our algorithm as we discuss below.

3.2.4 Computational Complexity

The computational complexity of T-VPack depends on how often timing analysis is performed on the circuit as BLEs are packed into logic clusters. If timing analysis is performed after each BLE is packed (a recompute interval of 1), the algorithm has the most up-to-date view of the criticality of each BLE, but the algorithm is $O(n^2)$, where n is the number of BLEs in the circuit. This complexity is the result of O(n) BLEs to pack, and each timing analysis being an O(n) operation. In this case T-VPack requires about fifteen minutes on a 300 MHz UltraSparc workstation to pack the largest MCNC benchmark circuit (clma) [Yang91], which contains 8383 BLEs.

TABLE 3.1 Effects of using tie-breakers, and the recompute timing interval (cluster size = 8).[a]

Circuit	Minimum Channel Width for Successful Routing (W_{min})			Post-Place-and-Route Critical Path (ns) $W = \infty$[b]			Post-Place-and-Route Critical Path (ns) $W = W_{min} + 20\%$		
	With Tie Breakers and Criticality Always Up to Date	No Tie Breakers and Criticality Always Up to Date	With Tie Breakers and Compute Criticality Once	With Tie Breakers and Criticality Always Up to Date	No Tie Breakers and Criticality Always Up to Date	With Tie Breakers and Compute Criticality Once	With Tie Breakers and Criticality Always Up to Date	No Tie Breakers and Criticality Always Up to Date	With Tie Breakers and Compute Criticality Once
alu4	43	42	42	25.2	27.2	25.6	29.9	28.3	28.0
apex2	50	58	54	30.2	32.6	35.6	35.9	34.3	33.1
apex4	60	54	58	27.1	29.8	25.5	31.5	31.4	30.1
bigkey	28	25	26	15.7	21.4	16.4	17.2	21.7	17.7
clma	68	68	66	63.6	80.3	64.0	67.3	82.1	66.0
des	29	31	31	28.1	28.2	27.5	28.7	28.4	28.3
diffeq	34	33	34	30.2	30.0	26.5	34.6	31.2	30.1
dsip	23	23	24	17.9	18.1	20.2	20.5	18.0	21.9
elliptic	47	48	55	40.1	40.5	40.5	52.9	48.8	54.7
ex1010	63	59	62	41.4	49.1	42.8	44.9	50.8	49.3
ex5p	59	56	58	28.9	29.1	28.4	29.7	3.32	33.2
frisc	55	55	58	55.0	54.6	55.9	65.0	62.2	63.1
misex3	47	48	48	23.1	25.3	25.2	27.3	33.2	31.4
pdc	76	82	78	47.3	48.6	52.5	72.8	52.5	63.3
s298	31	32	32	48.7	53.4	49.5	57.6	59.4	67.0
s38417	44	44	44	39.3	43.4	41.6	40.8	43.9	42.7
s38584.1	43	44	44	35.5	32.0	29.3	36.6	33.9	38.1
seq	54	54	51	25.9	30.3	26.9	29.0	32.6	30.6
spla	69	69	68	38.9	42.1	41.0	59.0	48.2	46.3
tseng	29	29	36	28.9	29.6	29.0	32.1	33.7	30.1
Geom. Av.	45.1	45.1	46.1	32.5	35.0	33.0	37.7	37.8	37.6

a. The results shown in Table 3.1 were computed with an older version of VPR than the results displayed in Table 3.2. Also, the architecture used for this experiment has a lower F_c value than the architecture used to generate Table 3.2. For these reasons, the numbers shown here (for the tie-breakers enabled and compute criticality once case) do not exactly match the size 8 cluster T-VPack results which we present in Table 3.2 (although they are quite close). Since all results shown in *this* table are generated with the *same* router and F_c values, it provides an accurate comparison of tie-breakers and the recompute interval. However, comparisons should not be made between Table 3.1 and Table 3.2.

b. Note that the congestion-oblivious (infinite routing) delay is *not* a lower bound on the achievable delay. In fact there is currently no known algorithm to route a net with guaranteed minimum Elmore delay, short of exhaustively searching all possibilities [Boes93].

We demonstrated in the previous section that it is not necessary to always have an up to date view of the critical path, and in fact never recomputing the timing information (recompute interval = ∞) results in an average degradation of only 2%. If timing analysis is performed only after every P BLEs are packed, the algorithm complexity is $O(n^2/P)$. If it performs timing analysis only once before any BLEs have been packed, the algorithm has the same complexity as VPack: $O(k_{max} \cdot K \cdot n)$ where k_{max} is the maximum number of terminals of any net and K is the number of inputs to each BLE. As with VPack, this complexity is the result of the fact that after each BLE is clustered (n BLEs) we must examine all of the nets attached to the BLE (K nets), and we must examine all BLEs that each net fans out to (maximum fanout = k_{max}). In this timing analyze only once case, T-VPack requires only a few seconds to pack the largest MCNC circuit (clma). We feel that this time-quality trade-off is the best for our purposes, so the remainder of our T-VPack results are based on packing solutions in which the recompute interval is ∞.

3.3 Connection-Driven Packing: C-VPack

As we will show in Section 3.4, our T-VPack algorithm on average requires fewer post-place-and-route tracks to implement the benchmark circuits than VPack. This is a result of the fact that a side effect of T-VPack is that many connections (and hence low fanout nets) are completely absorbed into logic clusters. This occurs because the attraction function used in T-VPack prefers to pack a BLE with its fan-in or fan-out BLEs, rather than packing it with BLEs that it has many nets in common with. Because of the realization that absorbing connections is good for area, we decided to design an algorithm with this as a goal.

Our connection-absorption-driven logic block packing algorithm, C-VPack, attempts not only to pack each logic block to capacity and minimize the number of cluster inputs used, but also to minimize the number of inter-cluster (between cluster) connections in the resulting circuit. This

algorithm is *not* timing driven, so it has no concept of where the critical path lies, or what BLEs are critical. The basic operation of the algorithm is the same as that of the VPack algorithm described in Section 2.2.2.1: C-VPack first chooses an unclustered BLE as the seed of a new cluster, and then sequentially adds unclustered BLEs with the greatest attraction to the current cluster until the cluster is full. Then this process is repeated until all BLEs have been packed. The seed is selected in the same manner as in VPack, however the C-VPack algorithm differs from VPack in the attraction function it uses.

3.3.1 Attraction Function

C-VPack packs clusters together in a manner that minimizes the number of inter-cluster connections in the resulting circuit. With this goal in mind, the attraction function used to select the unclustered BLE, B, to add to the current cluster, C, is:

$$Attraction(B) = \alpha \cdot ConnectionGain(B) + (1 - \alpha) \cdot |Nets(B) \cap Nets(C)| \quad (3.7)$$

The first term in (3.7) promotes the grouping of BLEs with high *ConnectionGains* (the number of point to point connections between B and C) into the current cluster to minimize the number of inter-cluster connections. α is a parameter that controls the trade-off between net sharing and connection absorption. If α is 0, we have an algorithm that focuses solely on minimizing the number of used inputs to a cluster, and is basically equivalent to the basic VPack algorithm described in Section 2.2.2.1. If α is 1, the algorithm focuses solely on minimizing the number of inter-cluster connections remaining, with no regard for how many nets are shared by the BLEs within a cluster. Using the CAD flow described in Section 2.2 we have experimentally determined that any α value in the range from 0.7 to 0.9 produces the best quality packings in terms of both post-place-and-route delay and routing area (channel width). The trade-off curves for channel width and critical path delay vs. alpha for an architecture using size 8 clusters are shown in Figure 3.7. In this figure, the critical path delay is computed based on an architecture with infinite routing resources. In the remainder of this thesis, C-VPack has α set to 0.75.

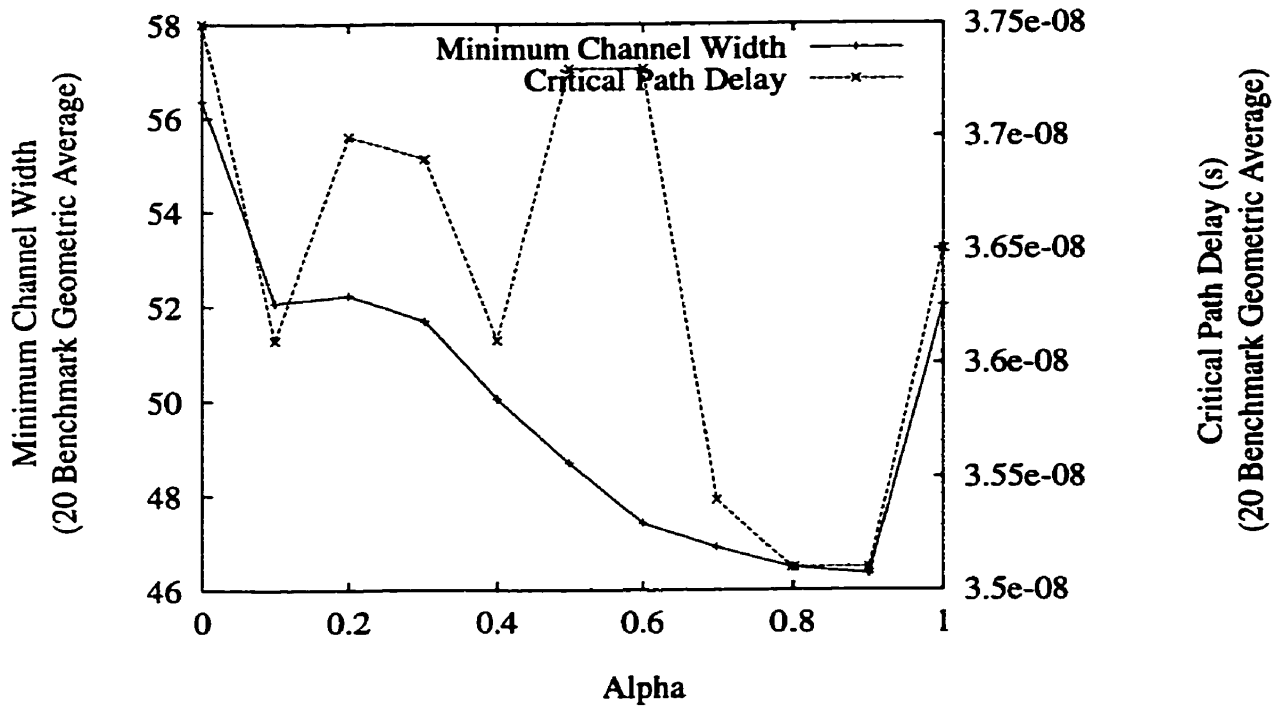


FIGURE 3.7 Post place and route C-VPAck alpha trade-off curves.

3.3.2 Time Complexity

The time complexity of the C-VPack algorithm is the same as the original VPack algorithm, $O(k_{max} \cdot K \cdot n)$ where k_{max} is the maximum number of terminals of any net, K is the number of inputs to each BLE, and n is the number of BLEs in the circuit. As with VPack, this complexity is the result of the fact that after a BLE is put into a cluster (n BLEs) we must examine all of the nets attached to the BLE (K nets), and we must examine all BLEs that each net fans out to (maximum fanout = k_{max}).

3.4 Result Quality of T-VPack, C-VPack, and VPack

Table 3.2 summarizes the performance of the basic VPack algorithm and the enhanced, T-VPack and C-VPack algorithms for the 20 largest MCNC[1] [Yang91] benchmark circuits. The logic cluster targeted in this experiment contains 8 BLEs. The first column gives the circuit names. The second, third and fourth columns compare the number of inputs (I) required to achieve 98% logic utilization where logic utilization is defined as

$$utilization = \frac{\left\lceil \frac{num\ logic\ blocks}{cluster\ size} \right\rceil}{num\ clusters\ used} \tag{3.8}$$

Notice that compared to the original VPack algorithm, T-VPack requires an average of 8% fewer cluster inputs to achieve 98% utilization, and C-VPack requires 10% fewer cluster inputs. The reason for this surprising result is that the criticality term in the T-VPack attraction function (3.6) makes T-VPack favour clustering a BLE with its fan-in or fan-out vs. clustering it with BLEs with which it shares inputs. The ConnectionGain term in the C-VPack attraction function (3.7) has a similar effect. Other researchers have also found that grouping circuit blocks with their fan-in or fan-out tends to be an effective clustering technique [Sauc93].

The remaining columns in Table 3.2 compare the post-place-and-route performance of circuits packed with the three different algorithms. The CAD flow used to generate these results is the same as the CAD flow described in Section 3.1. To generate the results listed in these columns, the number of inputs per cluster was set to 18. The architecture used in the experiments that produced these results is described in Section 4.2.

Three columns in Table 3.2 list the minimum number of tracks per channel (W_{min}) required to successfully route the packed circuit produced by each algorithm. Compared to VPack, T-VPack results in circuits that require 16% fewer tracks for successful routing, while C-VPack results in circuits that require 19% fewer tracks. To understand the reason for this surprising result, one

1. We give a brief summary of the 20 largest MCNC circuits in Appendix A.

TABLE 3.2 Comparison of VPack, T-VPack, and C-VPack result quality (Cluster Size = 8).

Circuit	Cluster Inputs (I) Required for 98% Logic Utilization			Minimum Channel Width for Successful Routing (W_{min})			Post-Place-and-Route Critical Path (ns) $W = \infty$[a]			Post-Place-and-Route Critical Path (ns) $W = W_{min} + 20\%$		
	VPack	T-VPack	C-VPack	VPack	T-VPack	C-VPack	VPack	T-VPack	C-VPack	VPack	T-VPack	C-VPack
alu4	20	16	17	55	39	40	28.1	25.1	28.8	30.7	27.9	30.7
apex2	19	19	18	58	55	52	37.5	32.7	34.7	37.9	34.5	37.2
apex4	19	20	19	53	52	51	28.5	25.5	28.3	34.5	32.9	31.4
bigkey	15	12	13	41	27	28	17.4	16.6	17.8	18.7	17.5	25.5
clma	17	17	16	75	64	65	82.9	64.7	71.9	84.3	77.1	79.4
des	19	17	17	36	29	31	28.8	27.4	26.6	29.9	29.4	28.4
diffeq	16	15	14	37	33	28	38.5	26.6	34.4	41.3	31.6	40.8
dsip	28	13	13	41	23	24	19.2	20.2	16.4	23.1	22.2	21.1
elliptic	16	17	16	57	49	43	50.2	40.1	56.2	60.5	49.1	69.5
ex1010	20	20	19	61	58	54	45.4	42.7	46.1	53.0	56.3	52.2
ex5p	19	20	20	55	53	52	27.6	28.1	27.6	31.9	30.9	33.1
frisc	16	16	17	57	58	50	75.0	56.4	61.2	80.2	65.3	68.8
misex3	18	18	18	49	43	44	25.5	25.3	27.0	29.1	30.6	29.6
pdc	20	18	18	82	76	72	56.7	52.7	54.3	57.9	81.8	62.0
s298	18	15	15	48	28	31	49.9	49.7	58.1	58.0	63.3	70.3
s38417	14	14	14	47	42	39	51.2	41.6	46.1	59.7	45.0	53.7
s38584.1	13	12	12	43	44	38	39.6	29.3	37.0	40.2	30.3	41.3
seq	18	17	17	57	47	49	30.2	27.2	27.9	31.5	37.0	30.1
spla	19	18	18	76	59	64	47.0	41.0	40.8	48.3	47.0	49.6
tseng	17	18	14	39	33	26	37.1	29.0	36.2	38.2	33.8	41.0
Arith. Av.	18.1	16.6	16.3	53.4	45.6	44.1	40.8	35.1	38.9	44.5	42.2	44.8
%Diff w.r.t VPack	—	-8.3%	-10%	—	-14.6%	-17.4%	—	-14.0%	-4.7%	—	-5.2%	+0.6%
Geom. Av.	17.8	16.4	16.1	51.9	43.4	42.0	37.7	33.0	36.1	41.2	38.9	41.7
%Diff w.r.t VPack	—	-7.9%	-9.6%	—	-16.3%	-19.1%	—	-12.5%	-4.2%	—	-5.6%	+1.2%

a. Note that the congestion-oblivious (infinite routing) delay is *not* a lower bound on the achievable delay. In fact there is currently no known algorithm to route a net with guaranteed minimum Elmore delay, short of exhaustively searching all possibilities [Boes93].

must compare the structure of the packed circuits produced by the three algorithms. Since T-VPack and C-VPack prefer to cluster a BLE with BLEs in its fan-in or fan-out, rather than with other BLEs that share inputs with it, these algorithms produce circuit packings in which many low-fanout nets have been completely absorbed into logic clusters. Overall, the output of T-VPack and C-VPack have fewer nets to route between clusters than the output of VPack, but the average fanout of each inter-cluster net is higher (more cluster inputs are used) with these algorithms than with VPack (see Table 3.3). The net result is that the output of T-VPack and C-VPack is somewhat

TABLE 3.3 Net absorption and inputs used (cluster size 8)

Algorithm	Average Percentage of Nets Absorbed	Average Number of Cluster Inputs Used
VPack	16.4%	12.09
T-VPack	40.6%	13.56
C-VPack	41.2%	13.41

easier to route than the output of VPack.[1] An example of why net absorption is good is given in Figure 3.8 — this figure demonstrates that each net to be routed requires its own track, and multiple point to point connections that are on the same net are able to share a track. An important factor in reducing the minimum required channel width is to minimize the number of nets remaining in the circuit (by absorbing many nets into clusters).

1. This result shows the importance of using a full CAD flow, including placement and routing, to evaluate many FPGA issues. It would have been difficult to guess that the output of T-VPack and C-VPack would be easier to route than the output of VPack without actually placing and routing the outputs of all of the packing algorithms. In fact, since the circuit packings produced by T-VPack and C-VPack have more point-to-point connections to route between clusters (despite having fewer nets) one would likely guess that T-VPack and C-VPack generated circuits would be more difficult to route.

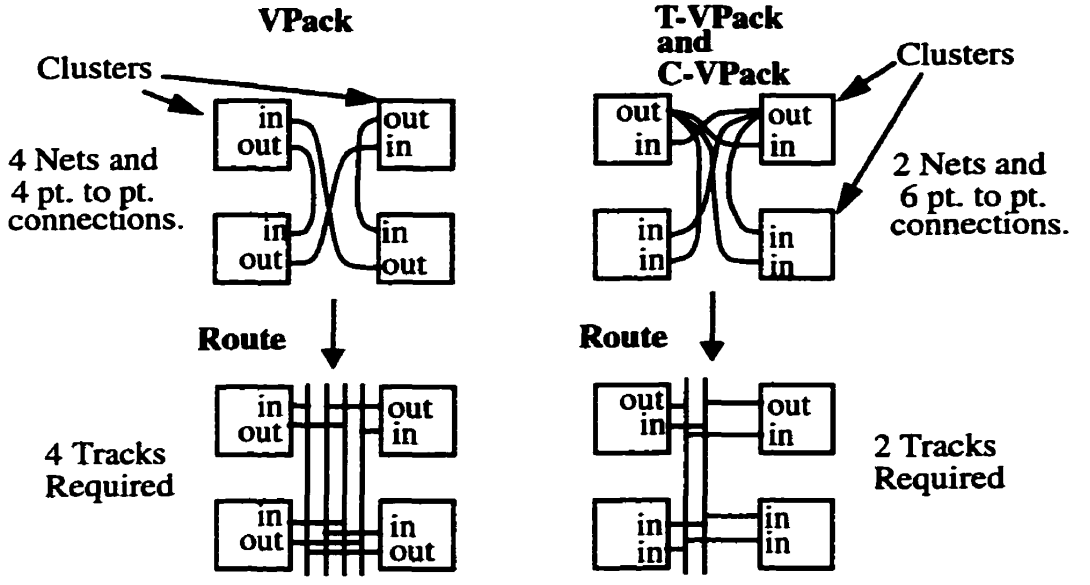


FIGURE 3.8 Why reducing the number of nets in a circuit is good

The post-place-and-route critical path columns in Table 3.2 compare the relative speeds of circuits implemented with the three different packing algorithms. One set of post-place-and-route speed results assumes that the circuits have essentially an infinite amount of general purpose routing available. In this case, the router is able to focus entirely on speed optimization, rather than congestion avoidance, so we obtain a good estimate of the speed difference between the two packings when the circuits are mapped into routing-rich FPGAs. The $W = W_{min} + 20\%$ speed results, on the other hand, show the speed difference between the two packings when an FPGA has a more limited amount of interconnect — only 20% more than the minimum required by each circuit packing. Remember that T-VPack and C-VPack produce circuits that require fewer tracks to route than circuits generated by VPack. Since our low stress delay results are based on this minimum width + 20%, a low stress routing is more difficult for T-VPack and C-VPack circuits vs. VPack circuits since the router has fewer tracks to select from. We therefore think that the infinite routing delay results give a better (more fair) comparison of the algorithms.

As one would expect, T-VPack decreases delay vs. VPack — by 12.5% (a 14% increase in speed) for the unlimited interconnect case, and 5.6% (a 6% increase in speed) for the limited interconnect case, on average. Note also that in the limited interconnect case, the T-VPack circuits are faster than those of VPack despite the fact that the router is being given significantly (16%) fewer tracks to route them than it is given for the VPack-generated packings.

The C-VPack algorithm reduces delay by 4.2% (a 4.4% increase in speed) for the unlimited interconnect case compared to VPack. In the limited interconnect case, C-VPack circuit delays are only 1.2% more (a 1.2% reduction in speed) than VPack delays, despite the fact that the router has been given significantly (19%) fewer tracks to route them than it is given for the VPack-generated packings.

3.5 Summary

In this chapter we introduced two new packing algorithms, T-VPack, and C-VPack. Overall, it is clear that T-VPack is better than VPack in terms of both circuit speed and routing area required. C-VPack on the other hand, is better than VPack in terms of area, but has worse speed for the low stress case and has better speed for the unlimited interconnect case.

CHAPTER 4 The Effect of Cluster Size on FPGA Speed and Density

In this chapter we investigate the speed and area-efficiency of FPGAs employing *logic clusters* (described in Section 2.1.1) as their logic block. We are particularly interested in the effect that cluster size has on FPGA speed and density. In the next section we discuss the trade-offs involved in selecting the proper cluster size for a cluster-based FPGA. After this, Section 4.2 explains how we model the area and speed of various FPGA architectures. Section 4.3 describes various architectural parameters that define the FPGAs used in our experiments. Section 4.4 describes the *area-delay product* that we use to evaluate the quality of different FPGA architectures. In sections 4.5 and 4.6 we explore key architectural questions concerning how circuit speed, FPGA area-efficiency, and compile time are affected by the size of the logic cluster used. Finally, Section 4.7 summarizes our results.

4.1 Trade-offs in Cluster-Based FPGAs

Much of the speed and area-efficiency of an FPGA is determined by the logic block it employs. In a cluster-based FPGA (described in Section 2.1.1), there are clear trade-offs between cluster size and FPGA speed and area. If a very small logic cluster is used (few BLEs per logic cluster), many logic blocks are required to implement each circuit, and many connections must be routed between the numerous logic blocks. Since routing consumes most of the area and accounts for most of the delay in FPGAs, a small logic block often results in poor area-efficiency and speed

due to the excessive routing required to connect all the logic blocks. If, on the other hand, a very large logic block is employed (many BLEs per logic cluster), fewer logic blocks are required to implement each circuit, but the logic block area and delay may become excessive, again resulting in poor area-efficiency and speed. Choosing the best size for an FPGA logic block therefore involves balancing complex trade-offs.

We are interested in determining the best cluster size for cluster-based architectures (described in Section 2.1.1). This style of logic block is of interest for several reasons. First, the Altera Flex series FPGAs [Alte98], the Xilinx 5200 and Virtex FPGAs [Xili97, Xili98], the newest Actel [Acte99], and the Vantis VF1 FPGAs [Vant99] all employ cluster-based logic blocks, so research concerning the best size of logic clusters is of clear commercial interest. Second, prior research [Betz98b] has shown that the area-efficiency of large logic clusters is quite competitive with that of FPGAs using single look-up table (LUT) logic blocks. Third, an FPGA composed of large logic clusters requires fewer logic blocks to implement a circuit than an FPGA using a more fine-grained block. This reduces the size of the placement and routing problem, and hence design compile time — an increasingly important concern as the logic capacity of FPGAs rises. Finally, cluster-based logic blocks can improve FPGA speed compared to single-BLE logic blocks by reducing the number of connections on the critical path that must be routed between logic blocks.

We are interested in two aspects concerning the design of cluster-based logic blocks. First, how many LUTs should be included in a cluster to create FPGAs with the best combination of speed and area-efficiency? Second, how is the time required to compile a circuit affected by the size of logic cluster used?

4.2 Architecture Modeling

In this section we first describe the area and delay models that we use to evaluate the various FPGA architectures. After this we describe the effect that varying cluster size has on segment lengths, and transistor sizing.

4.2.1 Area Model

The area model that we use is based on counting the number of *minimum-width transistor areas* required to implement each FPGA architecture, which is the same model as was used in [Betz98b, Betz99]. A minimum-width transistor area is simply the layout area occupied by the smallest transistor that can be contacted in a process, plus the minimum spacing to another transistor above it and to its right [Betz98b]. By counting the number of minimum-width transistor areas required to implement an FPGA, rather than the number of square microns that these transistors would occupy, we obtain a process-independent estimate of the FPGA area. The area model that we use is described in detail in [Betz98b, Betz99].

We use a program called *TransCount* [Betz99], to determine the area of a cluster-based logic block (including the local cluster routing) with any values of N, I, K, and M_{clk}. This program models such effects as buffer resizing as a function of the fanout of the connections within a logic block, and builds multi-stage buffers when high drive strengths are required. Since the area of an FPGA includes both logic block area and routing area, we use VPR to determine the transistor-count of the area taken by the routing for each FPGA of interest, and by adding this area to the logic block area we obtain the total FPGA area.

4.2.2 Delay Model

The delays of the connections within logic clusters were found by performing SPICE simulations using TSMC's 0.35 µm process for each structure in the cluster. Figure 4.1 shows the major structures and speed paths in a logic cluster. Important delay values through this cluster are shown in Table 4.1, while some delays cannot be listed because the process information is proprietary and was obtained under a non-disclosure agreement. The architectures corresponding to the numbers shown in this table had the number of inputs per cluster set to the number of inputs required for an average utilization (defined in Section 3.4, Equation 3.8) of 98% (which is shown in Section 4.3.2) when the circuits are packed with T-VPack.

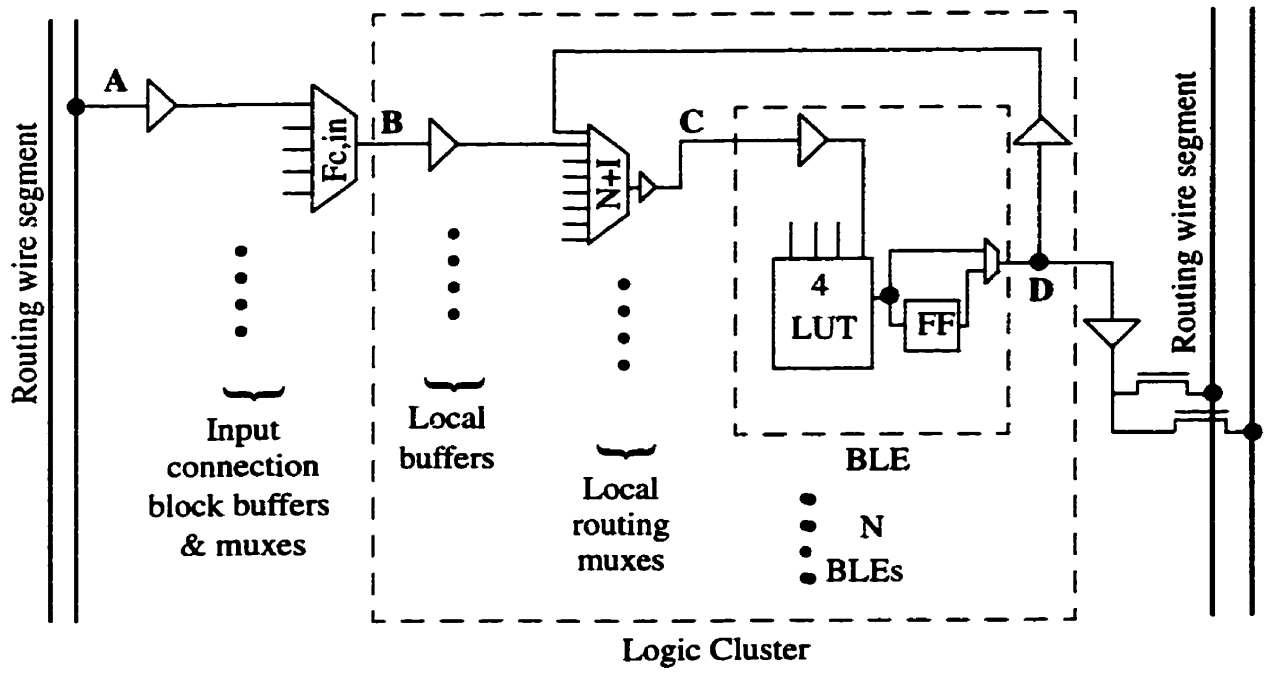


FIGURE 4.1 Structure and speed paths of a logic cluster.

TABLE 4.1 Important intra-cluster delays in TSMC's 0.35 µm CMOS process.

Cluster Size (*N*)	A to B (ps)	B to C and D to C (ps)	C to D (ps)	B to D (ps)
1 (No local routing muxes)	760	140 (and no D to C path)	379	519
2	760	687	379	1066
4	760	761	379	1140
8	760	902	379	1281
16	760	1054	379	1433
20	760	1081	379	1460

VPR has a built in delay estimator that uses a *modified* Elmore delay [Elmo48] model to estimate the delay of each connection in the routing. The modifications to the Elmore delay are described in [Okam96], and are such that it can be used to estimate delay of circuits containing buffers, resistors, and capacitors. After every connection's delay in the circuit has been computed, VPR

performs a path-based timing analysis using these inter-cluster connection delay values (Elmore delay) and intra-cluster delay values (Table 4.1). A full description of the timing-analyzer used in VPR is available in [Betz98b] or [Betz99].

4.2.3 Effect of Cluster Size on the Physical Length of FPGA Routing Segments

As we increase the cluster size, both the logic area per cluster and routing area per cluster grow. Figure 4.2 demonstrates how a tile (a logic block plus its associated routing) grows as cluster size is increased. This increased tile size results in routing segments with the same logical length having different physical lengths for logic clusters of different sizes.

We define the measured length of a routing segment as its physical length. The resistance and capacitance of a routing segment grow linearly with the segment's physical length. We have experimentally determined the average rate at which the FPGA tiles grow with cluster size, and have used this information to appropriately scale the routing segment resistance and capacitance values for the various cluster sizes. The increase in the resistance and capacitance of routing segments as the size of the FPGA logic block increases is an important effect that has often been

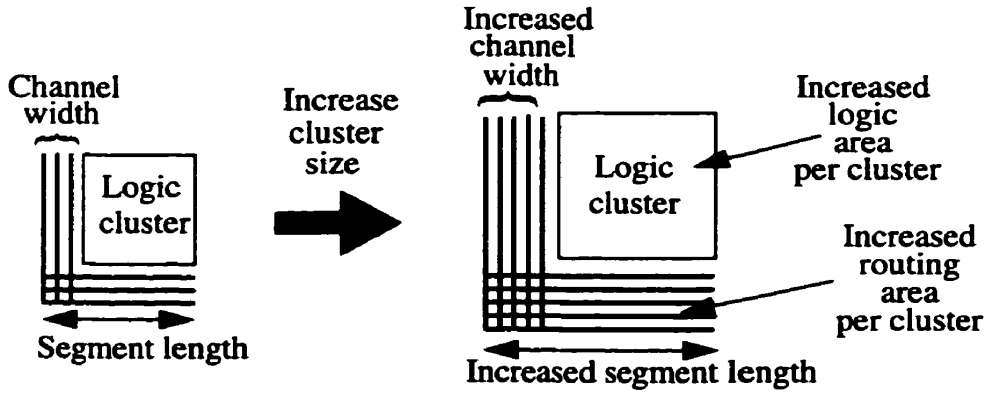


FIGURE 4.2 Effect of cluster size on physical length of routing segments.

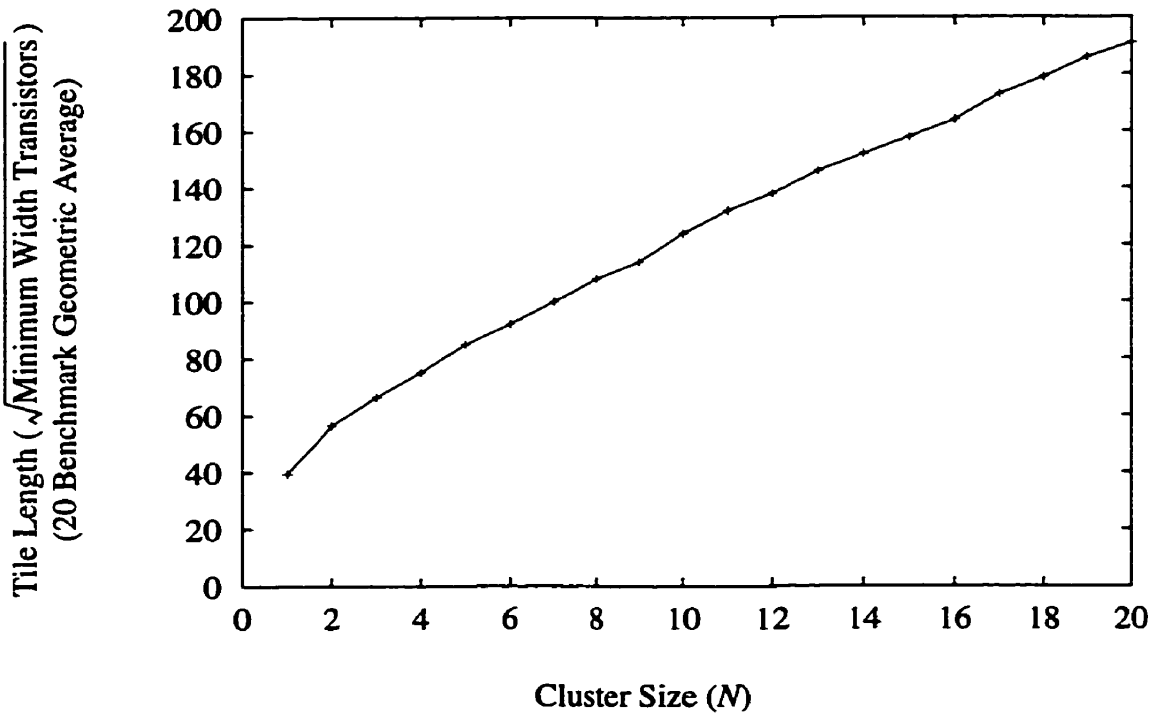


FIGURE 4.3 Effect of cluster size on tile length

neglected in prior FPGA architecture research. In Figure 4.3 we show experimental results showing the effect of cluster size on tile length. We use the ratio of the actual values in this curve to accurately scale routing segment lengths for different cluster sizes.

4.2.4 Sizing Routing Transistors to Compensate for Different Physical Segment Lengths

To compensate for differences in the capacitance and resistance of routing segments in FPGAs using different sizes of logic clusters, we scale the routing pass transistors and buffers. All of our pass transistor and buffer scaling is in relation to a base architecture that has been area-delay optimized for clusters of size four. From this base architecture, we linearly scale routing buffers and pass transistors depending on the relation between the new segment lengths and the base segment length. For example, in an FPGA with size 16 clusters, the physical segment length is

approximately two times longer than in an architecture with size 4 clusters. To maintain roughly the same speed per routing segment, we increase the size of the routing switches connecting to each wire by a factor of 2. In Section 4.5 we verify that this linear scaling of buffers and pass-transistors with physical segment length provides good results.

VPR models changes in delay caused by resizing buffers and pass-transistors in the routing, and it also accurately models the area required for different sizes of routing pass-transistors and buffers.

4.3 FPGA Architectural Parameters

To evaluate the speed and area of an FPGA employing logic clusters for its logic blocks, we must choose not only the logic block architecture, but also a routing architecture, transistor sizes, and the flexibility of the logic block to routing interface. The following sections detail the architectural parameters used in our experiments.

4.3.1 Basic Architecture

We investigate *island-style* FPGAs in which each logic cluster is surrounded by routing channels on all four sides with the logic cluster input and output pins evenly distributed around the logic cluster perimeter. This type of FPGA was described in Section 2.1. For our experiments each circuit is mapped to the smallest square FPGA with enough logic clusters and I/O pads to accommodate it.

For our experiments, we vary the number of I/O pads per row or column depending on the cluster size. Since a large cluster size requires fewer clusters to implement a given circuit, we require more I/O pads per row or column. We set the number of I/O pads per row or column to

$$Pads = \lceil 2 \cdot \sqrt{Cluster\_Size} \rceil \quad (4.1)$$

Setting the number of I/O pads per row or column with the above equation keeps the total number of I/O pads roughly the same for each FPGA architecture, independent of the cluster size that is used.

Recall from Section 2.1.1 that we can describe a logic cluster with four parameters: the number of logic inputs (I), the number of BLEs (LUTs and registers) in a cluster (N), the number of clock inputs (M_{clk}), and the number of inputs to each LUT (K). In this chapter we fix the number of clocks per cluster at one for all our experiments, since the MCNC benchmark circuits we use to evaluate architectures all have only one clock. We set the number of inputs to each LUT, K, to 4, since previous research has shown LUTs of this size are the most area-efficient [Rose90], and because this is the LUT size used in most commercial FPGAs. We describe how we set the number of inputs, I, in the next section.

4.3.2 Inputs Required vs. Cluster Size

Previous work [Betz98b] has examined the issue of how many cluster inputs are required for 98% utilization (defined in Section 3.4, Equation 3.8) of the logic clusters. This research, however, used VPack to map logic into the clusters. Since we are using our new T-VPack algorithm for packing in our cluster-based logic block experiments, and because we showed in Section 3.4 that T-VPack has better utilization than VPack, it is prudent to re-run these experiments with T-VPack. Figure 4.4 and Table 4.2 show the number of inputs required to achieve an average utilization of 98% vs. cluster size for both VPack and T-VPack[1]. We use the T-VPack results of this experiment to set the number of inputs per cluster for the remainder of our architecture studies.

1. This shows that T-VPack reduces the number of inputs required vs. VPack for 98% utilization at large cluster sizes. The fact that the two curve have different requirements for the number of inputs is an example of the dependencies between FPGA architecture and CAD.

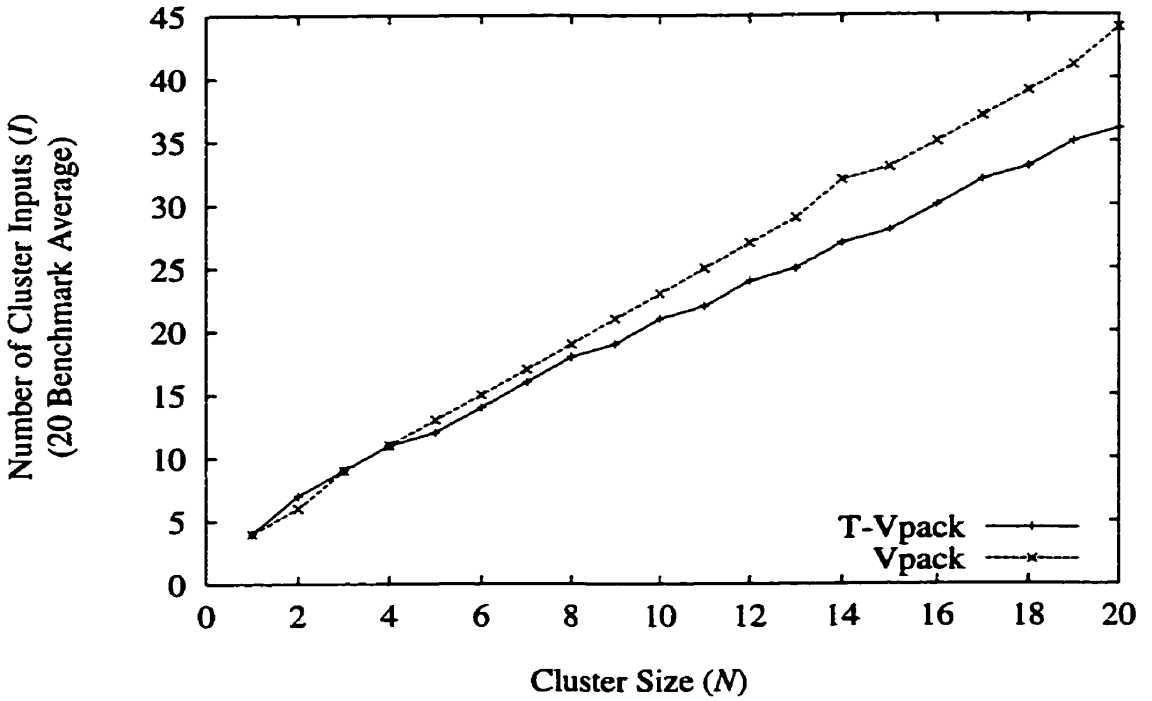


FIGURE 4.4 Inputs required for 98% utilization vs. cluster Size

4.3.3 Routing Architecture

We define the number of logic blocks that a routing segment spans as the logical length of that segment. In [Betz98b, Betz99] it is shown that an architecture in which routing segments have a logical length of four, with 50% of the segments connected by tri-state buffers and 50% connected by pass-transistors, provides good area-efficiency and speed for FPGAs containing logic clusters of size four. This routing architecture is shown in Figure 4.5. We implicitly assume that this routing architecture is good for architectures containing logic clusters of all sizes, and we use this routing architecture in all of our experiments. Ideally, one would find the best routing architecture for each FPGA employing a different cluster size, but this would require a huge amount of effort. By basing all of our experiments on this routing architecture, we may slightly favor architectures with size four clusters over other architectures.

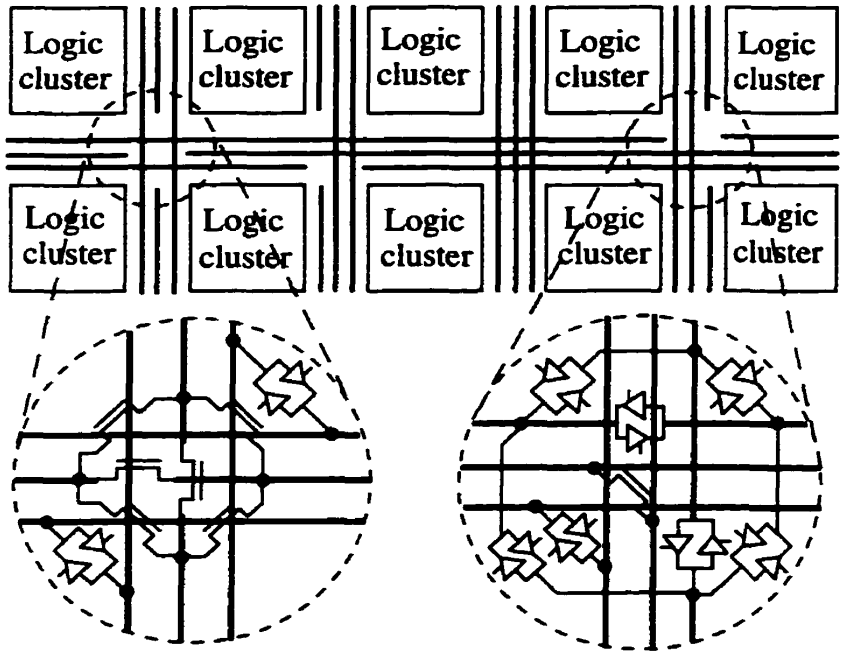


FIGURE 4.5 FPGA with length 4 segments, 50% buffered and 50% pass transistor switches.

4.3.4 Flexibility of Logic Block to Routing Interconnect vs. Cluster Size

For a cluster of size 1 [Rose91] showed that a good value of F_c (the number of routing tracks to which each logic block pin can connect) is W (the total number of tracks in a channel); This value of F_c means that each logic block pin can connect to any routing track in an adjacent channel. However, for large clusters, setting F_c to W provides far more routing flexibility than is required, wasting area.

[Betz98b] found that a more appropriate level of routing flexibility results when the F_c value for logic block output pins, $F_{c,output}$ is set to W/N, so all the experiments in the next section use this value. This choice of $F_{c,output}$ ensures that all the routing tracks in each channel can be driven by at least one output from each cluster.

TABLE 4.2 Inputs required for 98% utilization for VPack and T-VPack

Cluster Size	VPack	T-VPack
1	4	4
2	6	7
3	9	9
4	11	11
5	13	12
6	15	14
7	17	16
8	19	18
9	21	19
10	23	21
11	25	22
12	27	24
13	29	25
14	32	27
15	33	28
16	35	30
17	37	32
18	39	33
19	41	35
20	44	36

Choosing the appropriate value for $F_{c,input}$ involves finding the best trade-off between track width and area per track as follows

1. As $F_{c,input}$ is increased, fewer tracks are required to implement a given circuit since the router has more choices of which track each input can connect to.
2. Each track takes more area as $F_{c,input}$ is increased since there are more switches on each track (Note that routing area is determined by transistor area, not wiring area [Betz98b, Betz99]).

Therefore, we must determine the point at which the best trade-off occurs. We have run experiments on size 4, 8, 14, and 20 clusters to determine the best $F_{c,input}$ values as shown in Table 4.3, and have linearly interpolated between these results for other cluster sizes. Note, for these experiments, we have noticed that the critical path is not affected by the $F_{c,input}$ values chosen, so we choose the $F_{c,input}$ value based only on the area results.

TABLE 4.3 Routing area vs. $F_{c,\,input}$ for various cluster sizes[a]

$F_{c,\,input}$	Routing Area for various cluster sizes (in millions of minimum-width transistors)			
	4	8	14	20
0.1	—	—	—	1.51
0.2	—	—	1.38	1.41
0.3	—	1.29	1.34	1.41
0.4	1.47	1.27	1.34	1.42
0.5	1.45	1.28	1.37	1.46
0.6	1.44	1.30	—	—
0.7	1.45	—	—	—
0.8	1.49	—	—	—
0.9	1.50	—	—	—
Best $F_{c,input}$ value	0.6	0.4	0.3	0.2

a. The MCNC circuits used for these experiments are the 10 smallest circuits of the 20 circuits shown in Appendix A.

4.4 *Architecture Evaluation Metric: Area-Delay Product*

In this section we define the *area-delay product* metric, which we feel is useful for evaluating different architecures with respect to both speed and area. This is a reasonable architecture metric for two reasons:

1. Intuitively, we want to find the point at which we are sacrificing the least amount of area for the most improvement in speed. Given that we can always trade area for speed (see below), and speed for area, it makes sense to combine these two factors into one curve to see where the best trade-off occurs.
2. The computational throughput of an FPGA (on a parallel algorithm) is simply the number of functional units multiplied by the clock speed. Another way of looking at this is, *throughput = (1/area per functional unit) · (1/delay)*. Therefore by minimizing the area-delay product, we maximize throughput.

There are two main factors that can affect the area-delay product of an FPGA: transistor sizing and the FPGA architecture. In general, the speed of an FPGA can be increased (to a point) by sizing up the buffers and transistors within the FPGA, but this increases area. Alternatively, the FPGA can be made smaller by sizing down the buffers and transistors, but this degrades the FPGA performance.

Throughout this chapter, we will size the transistors in each FPGA architecture to minimize the FPGA's area-delay product. Only by resizing transistors appropriately for each architecture in this way can we fairly compute the speed and area-efficiency of FPGAs with different logic block architectures.

4.5 Speed and Area-Efficiency vs. Cluster Size

In this section we study the effect that varying cluster size has on the area and delay of implementations of the benchmark circuits. To obtain our results, we use the experimental flow given in Section 3.1 for 20 MCNC benchmark circuits, and we present the geometric averages of the results for these circuits. For all of the experiments in this section we set the number of inputs, I, for a cluster of size N to the minimum value that allows T-VPack to achieve 98% logic utilization (as shown in Figure 4.4). This value of I allows full utilization of our logic clusters, while keeping the complexity (hence area) of the clusters to a minimum.

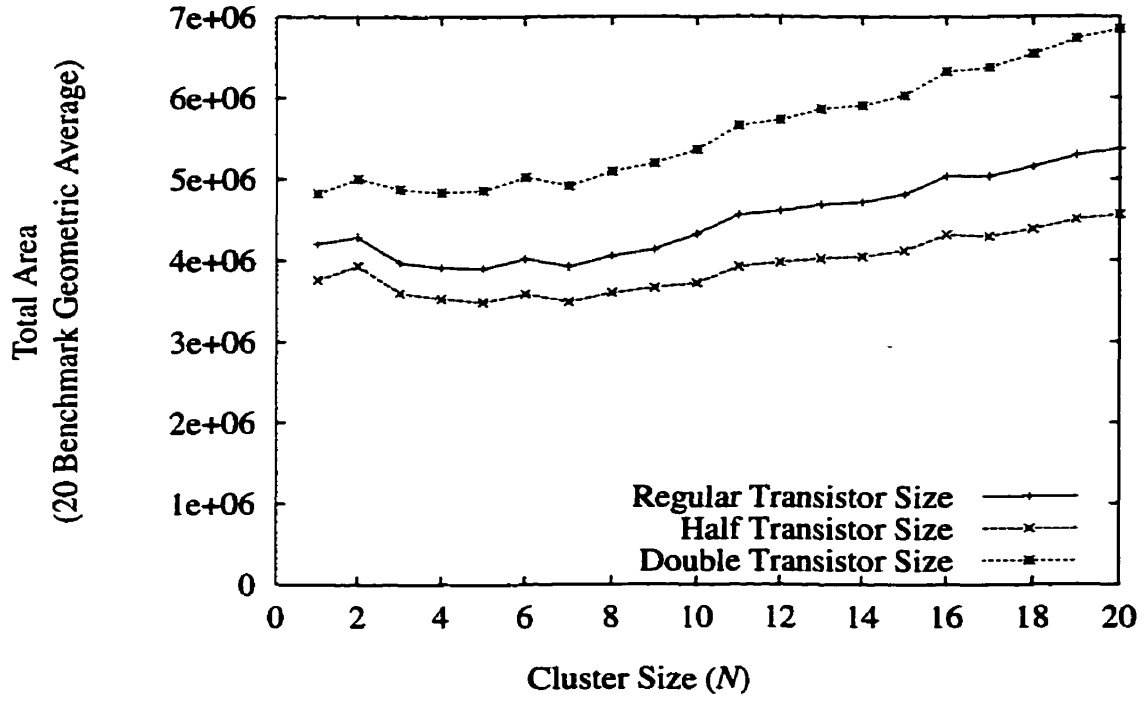


FIGURE 4.6 Total area vs. cluster size.

In Figures 4.6 and 4.8, we show the geometric average over the benchmark circuits of the total FPGA area required and the critical path delay, respectively. Note that we are showing three different routing transistor sizings in each of these graphs to ensure that we do not unfairly penalize any architecture with an inappropriate transistor sizing. The solid curves show the area and delay when we use the "normal" transistor sizing described in Section 4.2.4, while the dashed curves show the results when we use transistors that are one-half or double the size of those in the "normal" case. Notice that we can indeed trade speed for area by resizing routing transistors — the half transistor size results have less area, but greater delay, while the double transistor size results have less delay, but greater area.

Figure 4.7 shows the effect that varying cluster size has on the area required to implement the benchmark circuits. Area is affected by two factors.

1. As we increase cluster size up to about size 9, the routing requirements between clusters is reduced since many connections are completely absorbed within the clusters. After size 9, the routing area begins to increase. We believe that the reason for this increase is because large clusters make it difficult for the placer to do a good job minimizing wirelength. This is the result of each cluster being connected to so many nets that it is sharing nets with essentially every other cluster. It is therefore likely that when the placer moves these large clusters to improve the wire-length of some nets, this same move will increase the wire-length of many other nets.
2. As we increase cluster size, the total area taken by the multiplexers within each cluster grows quadratically, but the number of clusters required to implement a circuit is decreasing. This results in a linear increase in the total area taken by all the logic clusters. For sufficiently large clusters, the area reductions in the routing are overtaken by the increased area required to implement the larger clusters.

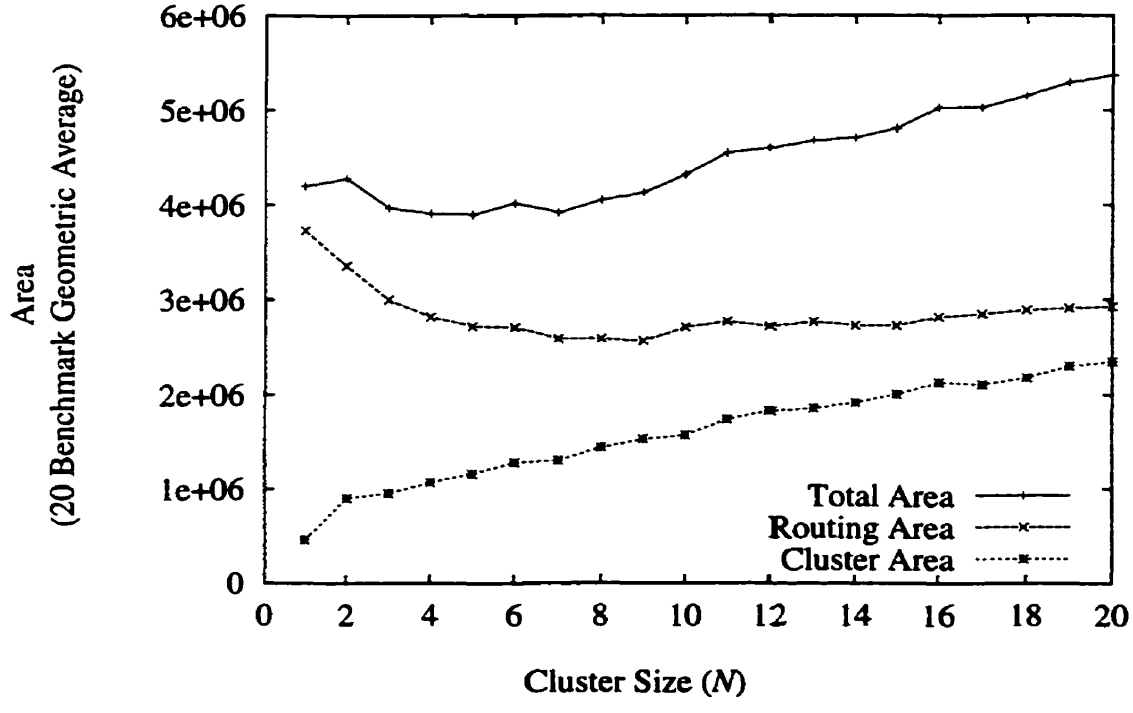


FIGURE 4.7 Area components vs. cluster size.

Figure 4.8 shows that circuit speed increases significantly as we increase the cluster size. As one increases the cluster size from size 1 to 7, the circuit speed rapidly increases — with the "normal" transistor sizing, a size 7 logic cluster leads to circuits which are 51% faster than those implemented with a size 1 cluster. Increases in cluster size past $N = 7$ produce smaller incremental speed gains. For example, with the "normal" transistor sizing, a cluster of size 20 is 7% faster than a cluster of size 7.

In Figure 4.9, we show how the geometric average of the area-delay product achieved by the benchmark circuits varies with cluster size, again for three different transistor sizings. Notice that the "normal" transistor sizing provides the best area-delay product for all the architectures except a cluster size of 1, indicating that linearly scaling routing transistor size with the length of a layout tile is a good method to size transistors. For a cluster of size 1, however, the normal transistor sizing is smaller than optimal, and the double transistor size FPGA has a 7.5% lower area-delay product than the normal transistor size FPGA. There is a broad minimum in the area-delay

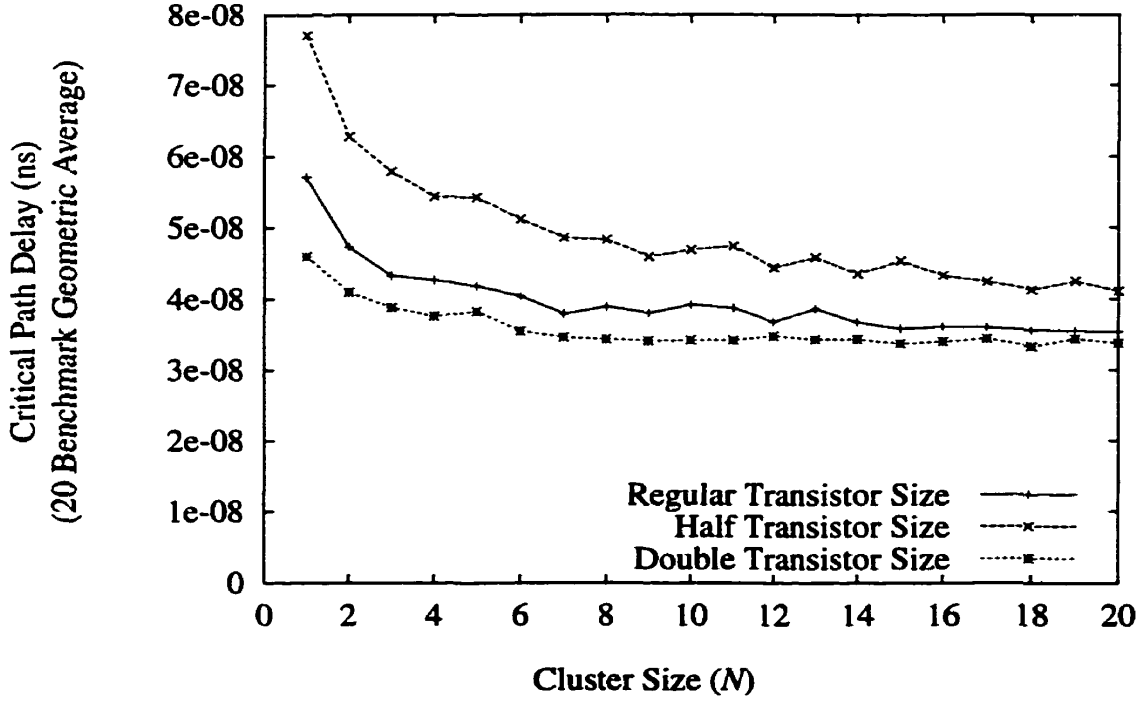


FIGURE 4.8 Critical path delay vs. cluster size.

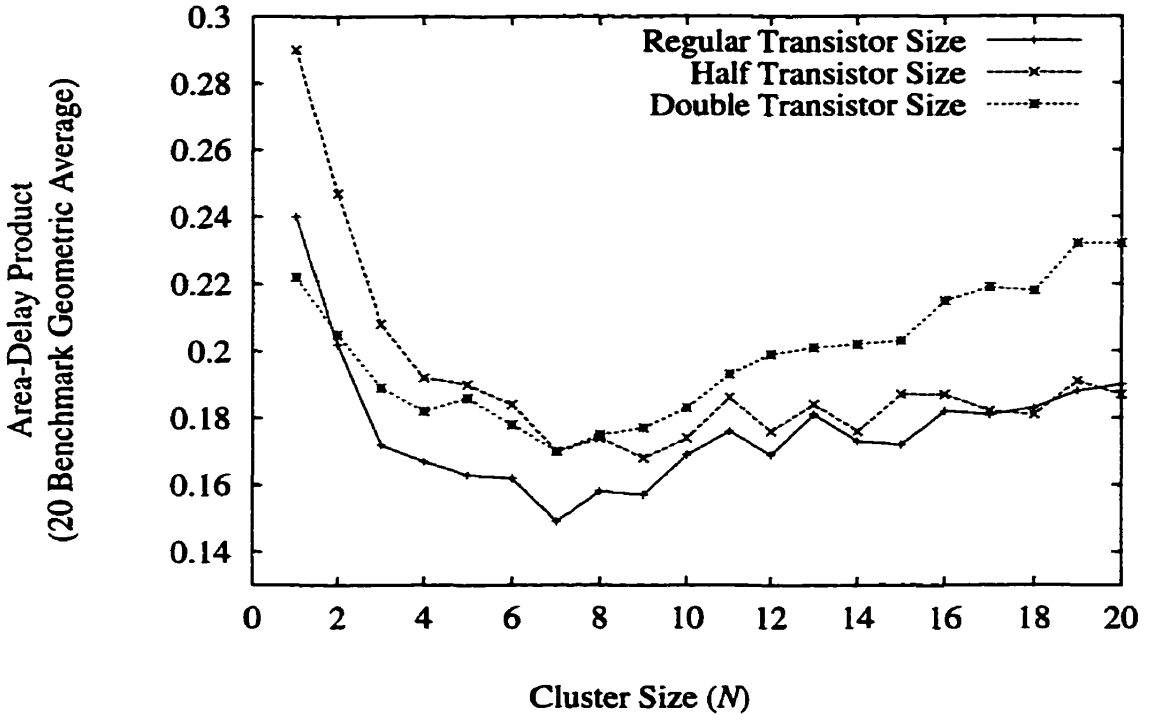


FIGURE 4.9 Area-delay product vs. cluster size.

product for cluster sizes from 4 to 10. A cluster of size 7 has the lowest area-delay product, but any cluster size between 4 and 10 is within 12% of the minimum, and hence would be a reasonable choice. Notice that moderate-size logic clusters significantly improve the area-delay product of an FPGA vs. using a single BLE logic block. Comparing a size 7 logic cluster (with the normal transistor sizing) to a size 1 logic cluster (with double-sized transistors — the best for this cluster size), one sees that the size 7 logic cluster has an area-delay product that is 33% lower than that of a size 1 cluster. An FPGA using a size 7 logic cluster is simultaneously 21% faster (a 17% delay reduction), and requires 19% less area than an FPGA using a size 1 logic cluster.

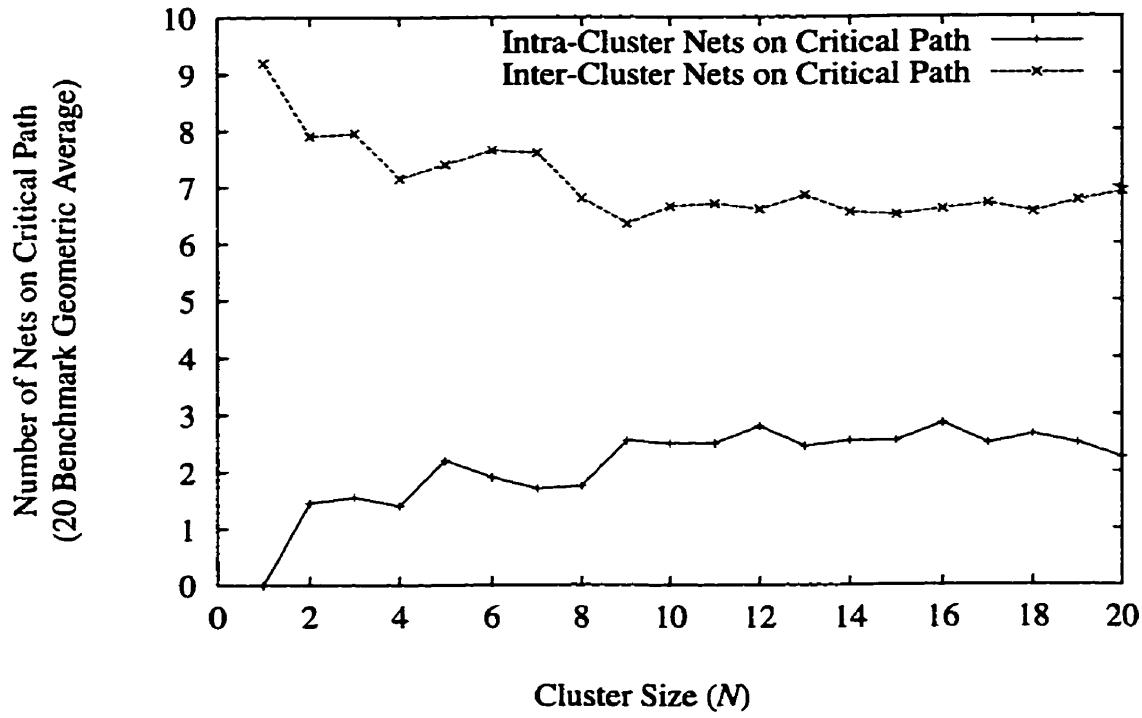


FIGURE 4.10 Inter-cluster and intra-cluster nets on the critical path.

4.5.1 Discussion of Delay vs. Cluster Size Results

In Figure 4.10 we show the relationship between the number of intra-cluster (fast) and inter-cluster (slower) connections on the critical path as a function of cluster size. As cluster size is increased, the number of intra-cluster connections on the critical path increases, and the number of inter-cluster connections decreases. This provides a circuit speedup since intra-cluster connections are faster than inter-cluster connections.[1]

Interestingly, the number of inter-cluster nets on the critical path does not decrease as much with cluster size as the inter-cluster delay decreases with cluster size (see Figure 4.11). From size 2 to size 20 we have a reduction in the number of inter-cluster nets on the critical path of 13%

1. As cluster size is increased, intra-cluster multiplexer, buffer and wiring delays increase. If we were to increase the size of cluster to very large values, this effect would eventually result in intra-cluster delays becoming large enough that any gains obtained by making connections local to the cluster would be lost.

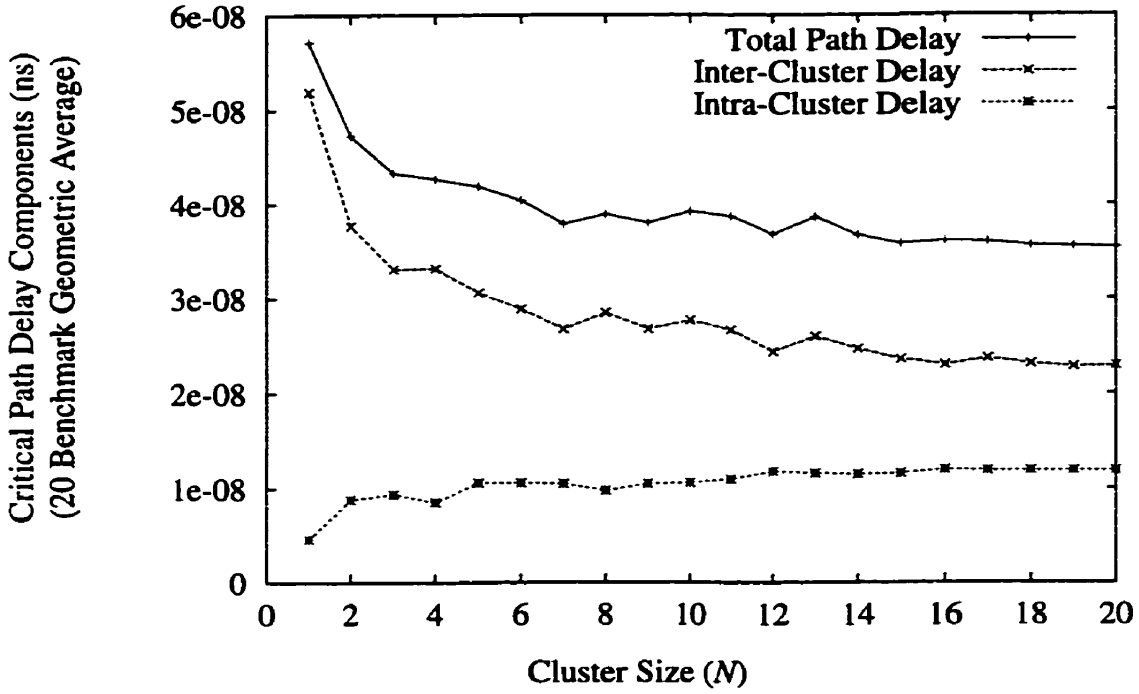


FIGURE 4.11 Breakdown of critical path delay into inter-cluster and intra-cluster components.

(Figure 4.10); compare this to the inter-cluster component of the critical path delay, which has been reduced by 39% over the same range (Figure 4.11). This means the circuit speedup visible in Figure 4.11 for larger cluster sizes is not only caused by a reduction in the number of inter-cluster connections on the critical path, but also by *inter-cluster connections on the critical path becoming faster.*

The improvement in inter-cluster delay with increased cluster size is caused primarily by a reduction in the "logical" manhattan distance spanned by connections in the FPGA, as illustrated in Figure 4.12. By sizing the routing pass transistors and buffers[1] (as discussed in Section 4.2.4) to compensate for the increased physical length of routing wire segments associated with larger clusters, the delay of each routing segment has remained roughly constant. Since the total number

1. Changes in delay and area due to different size routing buffers and pass transistors are accounted for in the timing and area models used in this research.

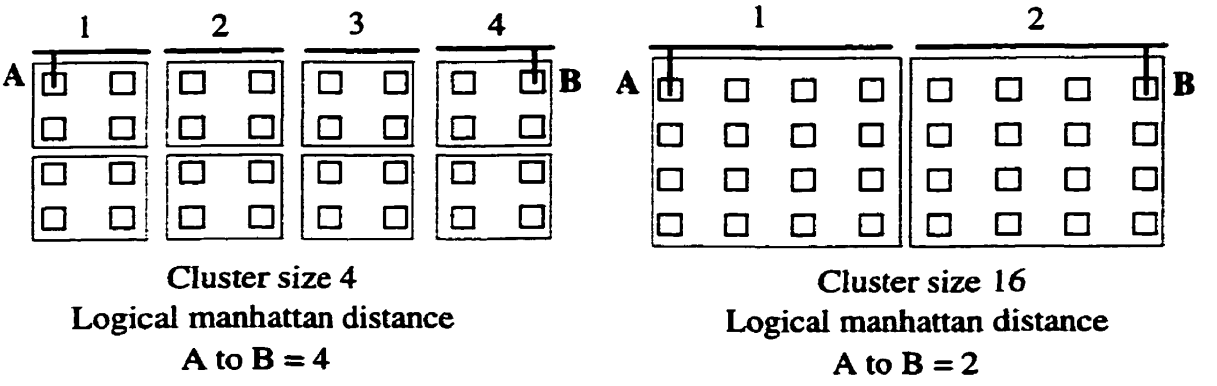


FIGURE 4.12 Decrease in logical manhattan distance as cluster size increases.

of routing segments on the critical path has decreased due to the reduction of the "logical" manhattan distance, the result is a greater improvement in circuit delay than the reduction in the number of inter-cluster nets on the critical path would indicate.

4.6 *Effect of Cluster Size on Compile Time*

In this section we demonstrate that cluster-based FPGA architectures can significantly improve compile time. Figure 4.13 shows how the average CPU time (on a 300 MHz UltraSparc workstation) required to implement circuits varies with cluster size. The solid line in Figure 4.13 shows the total (packing, placement, and routing) compile time, while the three dashed lines show the individual components of this compile time. The routing time is the time taken for low-stress routings (minimum width + 20%).

As larger logic clusters are employed in an FPGA the time to compile circuits is dramatically reduced. This occurs because as larger clusters are employed, fewer of these clusters are required to implement each circuit. Since the size of a placement problem is proportional to the number of logic clusters that a circuit is mapped to, this dramatically reduces placement time. In Figure 4.13, for example, one can see that the placement time is reduced by a factor of 8.8 times

as the cluster size increases from 1 to 20. Larger logic clusters also reduce the routing time. This is the result of more connections using the local cluster routing, with the effect that the router has fewer inter-cluster connections to route. For example, using a size 20 logic cluster reduces routing time by 2.7 times vs. using a size 1 cluster. Building an FPGA with a size 20 logic cluster reduces the total CPU time required for placement and routing by 7 times vs. a size 1 logic cluster.

4.7 Summary

Using the area-delay product evaluation metric, we have demonstrated that logic clusters containing between 4 and 10 BLEs all achieve good performance, so any cluster in this range is a reasonable choice. Compared to FPGAs using a single BLE logic block, logic clusters in this size range achieve significant area and speed improvements. For example, an FPGA employing a size 7 logic

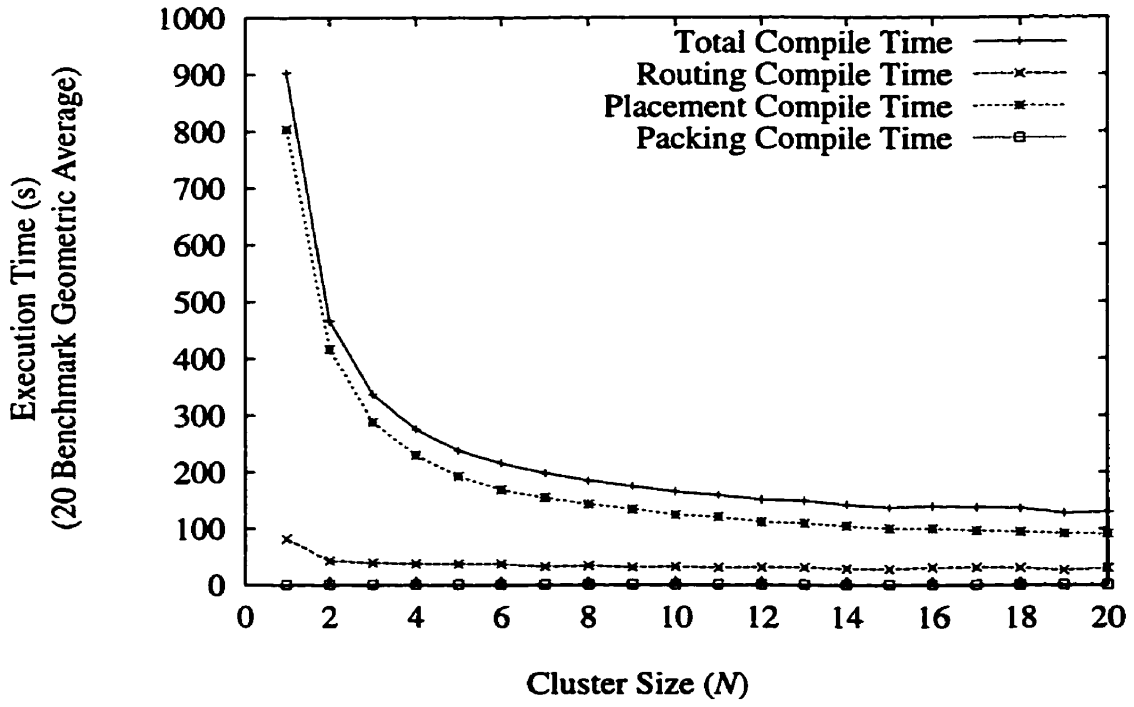


FIGURE 4.13 Variation of circuit compile time with logic cluster size.

cluster requires 19% less area, achieves 21% higher speed, and has an area-delay product 33% lower than an FPGA using a single BLE logic block. Additionally, large logic clusters significantly reduce design compile time. Size 7 and 20 logic clusters reduce placement and routing time by 4.6 times and 7 times compared to a single BLE logic block, respectively.

CHAPTER 5 *Timing-Driven Placement*

In this chapter we first give a brief introduction to timing-driven placement. After this we describe our new timing-driven placement algorithm, and how we have tuned various parameters within the algorithm. Then we give results comparing the speed and area of circuits placed with the new timing-driven algorithm to circuits placed with an existing non-timing-driven algorithm.

5.1 Introduction

Recall that placement is the process by which a netlist of circuit blocks (which are either I/Os or logic clusters) are mapped onto physical locations in an FPGA. The location of the circuit blocks can significantly affect the performance of the FPGA. A timing-driven placement algorithm attempts to map circuit blocks that are on the critical path into physical locations that are close together so as to minimize the amount of interconnect that critical signals must traverse.

The VPR place and route tool [Betz99, Betz98b] incorporates a timing-driven router, but does not consider timing during placement (VPR's placement tool VPlace is described in Section 2.2.3.2). A timing-driven router can only produce routings that are as good as the placement on which the routing is performed, so to extract more speed out of an FPGA it is essential that timing-driven placement algorithms be used. Previous timing-driven placement algorithms (described in Section

2.2.3.3) have done a good job in reducing circuit delay, but they are very computationally intensive. To be useful, a timing-driven placement algorithm must produce high quality placements in reasonable amounts of time.

5.2 *Timing-Driven Placement: T-VPlace*

We have developed a new placement tool called T-VPlace which is an extension to VPlace (described in Section 2.2.3.2) and is integrated into VPR. T-VPlace is both wireability-driven (minimizing wiring requirements) *and* timing-driven. It is essential to consider both the goal of minimizing wiring and reducing critical path delay because a timing-driven only approach will lead to circuits that require an unacceptable amount of routing resources. T-VPlace simultaneously considers critical path delay and wireability and finds a reasonable compromise between the two. T-VPlace is simulated annealing-based and it uses the same annealing schedule as VPlace (as discussed in Section 2.2.3.1). In Figure 5.1 we show the pseudo-code for the T-VPlace algorithm. The following sections describe T-VPlace in detail.

5.2.1 Delay Modeling and Cost Function

To maximize speed, T-VPlace must model the delay of each connection and the available slack for each connection. In this section we will discuss the computation of these two components as well as how they are combined into our new cost function.

For a placement algorithm to minimize the delay of the resulting circuits in a reasonable amount of time, accurate delay modeling is required. To model delay we compute a *delay lookup matrix* which contains the delay between a source and a sink located at (x_{source}, y_{source}) and (x_{sink}, y_{sink}), depending only on $\Delta x = |x_{source} - x_{sink}|$ and $\Delta y = |y_{source} - y_{sink}|$. In this model, the delay between the two locations depends only on the relative distance between the two locations. The delay lookup matrix is described more thoroughly in Section 5.2.1.1.

```
S = RandomPlacement ();
T = InitialTemperature ();
R_limit = InitialR_limit ();
Criticality_Exponent = ComputeNewExponent();

ComputeDelayMatrix();

while (ExitCriterion () == False) {        /* "Outer loop" */

    TimingAnalyze();          /*Perform a timing-analysis and update each connections criticality*/
    Previous_Cost_linear congestion = Cost_linear congestion(S);   /*wirelength minimization normalization term*
    Previous_Timing_Cost = Timing_Cost(S);              /*delay minimization normalization term*/

    while (InnerLoopCriterion () == False) {    /* "Inner loop" */

        S_new = GenerateViaMove (S, R_limit);
        ΔTiming_Cost = Timing_Cost(S_new) - Timing_Cost(S);
        ΔCost_linear congestion = Cost_linear congestion(S_new) - Cost_linear congestion(S);
        ΔC = λ·(ΔTiming_Cost/Prev_Timing_Cost) +
                          (1-λ)·(ΔCost_linear congestion/Previous_Cost_linear congestion); /*new cost fcn*/
        if (ΔC < 0) {
            S = S_new /*Move is good, accept*/
        }
        else {
            r = random (0,1);
            if (r < e^(-ΔC/T)) {
                S = S_new; /*Move is bad, accept any way*/
            }
        }
    }          /* End "inner loop" */

    T = UpdateTemp ();
    R_limit = UpdateR_limit ();
    Criticality_Exponent = ComputeNewExponent();

}   /* End "outer loop" */
```

FIGURE 5.1 Pseudo-code T-VPlace.

It is also essential that timing analysis be done in an efficient manner. To make sure that the computation time spent performing timing analysis does not significantly degrade the placement compile time, we periodically perform a timing analysis after a certain number of simulated-annealing swaps are completed. This means that the slack values (obtained by this "infrequent" timing-analysis) used in the placer may be based on delay values that do not precisely reflect the

connection delays of the current placement. We have experimentally determined how often a timing analysis must be done to get the best results, and we discuss these experiments in Section 5.2.2.

Finally, we need a new cost function that takes into account both the slack of each connection, and the delay on each connection. By reducing delay on connections with little slack while increasing delay on connections with lots of slack, we are able to reduce the critical path. We have developed a cost function based on slack and delay that is described in Section 5.2.1.2.

5.2.1.1 Delay Lookup Matrix

To allow an efficient assessment of the delay between blocks that are Δx and Δy distance apart in a tile-based FPGA[1] we compute a delay lookup matrix indexed by Δx and Δy. To compute a given (Δx, Δy) entry in the matrix, we employ the VPR router to determine the delay between two blocks that are (Δx, Δy) distance apart. To do this, a source block is placed at a location (x_{source}, y_{source}) in the FPGA, and a sink block is placed at ($x_{source}+\Delta x$, $y_{source}+\Delta y$). Then VPR's timing-driven router is used to perform a routing between the two blocks, and the delay is recorded in the delay lookup matrix at location (Δx, Δy). This process is then repeated for every possible Δx and Δy value in the FPGA.

Since we use the timing driven router to compute the delay between blocks, we are able to take advantage of architectural features in the FPGA, i.e. if two blocks are on opposite sides of the FPGA and there is a long line crossing the FPGA, the timing-driven router will recognize this and the delay lookup matrix will reflect the smallest possible delay (the one using the long line) between the two locations. The reason that we use the smallest possible delay between two blocks

1. A tile-based FPGA is one in which the FPGA structure is homogenous (i.e. every x,y location in the FPGA is physically constructed with identical tiles). Since most FPGA architectures are tile-based and because the architectures we use are tile based, we obtain accurate delay estimates by exploiting this uniformity and only computing delays based on Δx and Δy values.

to compute the values in the delay lookup matrix is because we assume that after placement, the router will be smart enough to use the fastest resource to connect two locations on the critical path.

5.2.1.2 Cost Function

To properly balance the trade-off between wirelength minimization and critical path minimization, we have developed a new cost function that we call the *normalized-trade-off-Δcost* function. Before we discuss this new cost function we need to introduce some definitions that are used in our cost function.

We first introduce a new term called *Timing_Cost*. This is the portion of the cost function that will be responsible for minimizing the critical path delay. Timing_Cost is based on the *Criticality* of each connection, the *Delay* of each connection, and a user defined *Criticality_Exponent*. Where the Delay for each connection is obtained from the delay lookup matrix, the Criticality_Exponent is defined below, and Criticality is defined as follows

$$Criticality(i) = 1 - \frac{Slack(i)}{Dmax} \quad \textbf{(5.1)}$$

where D_{max} is the maximum arrival time of all sinks in the circuit, and Slack is the amount of delay that can be added to a connection without increasing the critical path delay (D_{max} and slack were described in detail in Section 2.2.1).

In our new cost equation, to control the relative importance of connections with different criticalities, we compute a power of the Criticality of each connection depending on a variable called Criticality_Exponent (i.e. $Criticality^{Criticality\_Exponent}$). The purpose of including an exponent on the Criticality in our new cost function is to heavily weight connections that are critical, while giving less weight to connections that are non-critical.

The Criticality_Exponent that we use can be *either* constant or "adaptive". An adaptive Criticality_Exponent is an exponent that gradually increases as the annealing temperature decreases. The reasoning behind this is that during the initial stages of the anneal, the current critical path is likely to significantly change, while at later stages of the annealing process the placer has a better idea of where the critical path lies. It is intuitively beneficial at these latter stages to heavily weight the critical path more than at the initial stages. To do this we have developed an equation that slowly increases the Criticality_Exponent starting at a user defined initial value called *Init_Exp*, up to its final user defined value called *Final_Exp*. This involves making use of an existing variable called R_{limit} [Betz98b, Betz99] (described in Section 2.2.3.1) that we can use to derive the desired behavior for the Criticality_Exponent. Remember that R_{limit} determines the range that the placer will consider for swapping blocks. Initially R_{limit} spans the entire chip (it is the maximum value of either the x or y dimension of the FPGA), and at the end of the anneal it considers only adjacent blocks (it is one). Since this variable is a good indicator of how far along the anneal is, and because it is changing gradually, we can use it as a guide to gradually increase our Criticality_Exponent with the following equation

$$Criticality\_Exponent = \left[1 - \frac{Curr\_R_{limit} - Final\_R_{limit}}{Init\_R_{limit} - Final\_R_{limit}}\right] \cdot [Final\_Exp - Init\_Exp] + Init\_\text{Exp} \quad \textbf{(5.2)}$$

where Curr_R_{limit} is the current value of R_{limit}. Final_R_{limit} is always one, and Initial_R_{limit} spans the entire chip.

We now define the Timing_Cost of a connection, *i*, as follows

$$Timing\_Cost(i) = Delay(i) \cdot Criticality(i)^{Criticality\_Exponent} \quad \textbf{(5.3)}$$

And the total Timing_Cost for a circuit is the sum of the Timing_Cost of all of its connections as follows

$$Timing\_Cost = \sum_{\forall i \subset circuit} Timing\_Cost(i) \quad (5.4)$$

We are now ready to discuss our normalized-trade-off-Δcost function. Our normalized-trade-off-Δcost function depends on the *change* in Timing_Cost and $Cost_{linear\ congestion}$ (given in Section 2.2.3.2, Equation 2.5). It uses a trade-off variable called λ to determine how much weight to give each component. To normalize the weight of these two components we use two normalization[1] variables called Previous_Timing_Cost and the $Previous\_Cost_{linear\ congestion}$ that are updated at every temperature. The effect of these two normalization components is to make the function weight the two components only with the λ variable, independent of their actual values. This is convenient because it automatically adjusts the weights of the two components so that the algorithm is always allocating λ importance to changes in the Timing_Cost, and (1-λ) importance to changes in the $Cost_{linear\ congestion}$. If λ is 1 then we have an algorithm that focuses only on timing, but ignores wirelength minimization. If λ is 0, then we have the original VPlace algorithm that focuses only on minimizing wirelength. We now present the normalized-trade-off-Δcost function

$$\Delta C = \lambda \cdot \frac{\Delta Timing\_Cost}{Previous\_Timing\_Cost} + (1-\lambda) \cdot \frac{\Delta Cost_{linear\ congestion}}{Previous\_Cost_{linear\ congestion}} \quad (5.5)$$

We use this cost function in our algorithm without modifying the annealing schedule from VPlace (described in Section 2.2.3.1). Since the annealing schedule is "adaptive"[2], it performs well with this new cost function. When λ is 0 this new cost function attempts to minimize only wirelength like the original VPlace algorithm, however the results are slightly (about 3%) worse for the post-

1. For example, if we have a λ value of 0.7, we want every move to be 70% due to changes in Timing_Cost, and 30% due to changes in $Cost_{linear\ congestion}$. If we did not normalize, and we had Timing_Cost values that were orders of magnitude less than $Cost_{linear\ congestion}$ then the cost function would only be affected by changes in the $Cost_{linear\ congestion}$ even though we desired this to only account for 30% of the change in total cost. Another benefit of this normalized approach is that as the temperature changes, we are constantly re-normalizing the weights of the two components. Compare this to other approaches that only normalize the components once at the beginning of the algorithm [Swar95], which means that if the two components change at different rates, this normalization will become increasingly inaccurate, and will inadvertently allocate more weight to one of the component than was desired.

2. For a full description of the adaptive annealing schedule, see [Betz98b, Betz99]

place-and-route channel width. It is likely that we could have fine-tuned the schedule for this new normalized-trade-off-Δcost function, but we did not feel that this degradation was significant enough to warrant the extra effort.

5.2.2 Algorithm Tuning

In our algorithm there are parameters that must be tuned to get the best performance. We must find the best value for λ, the best Criticality_Exponent (and whether it should be constant or adaptive), and determine how often we must re-timing analyze the circuit during placement. To find the best values for these parameters, we performed experiments on the 20 largest MCNC circuits, using size one clusters, and the same architecture as described in Section 4.3.

By using the delays from the delay lookup matrix annotated onto connections in the circuits, we are able to obtain critical path delay estimates from the placement algorithm without performing a routing. These estimates allow us to fairly compare different algorithm parameters in a reasonable amount of computation time. We will later show in Section 5.2.3 that these placement estimates are a good tool (have good fidelity with respect to the final routed delay) for comparing values for different parameters.

The first parameter that we discuss is the re-timing analysis interval. For this experiment we set the value of λ to 1 (fully timing-driven) and the Criticality_Exponent to 1. We then varied how often we re-timing analyze the circuit and update the connection Criticalities and Slacks. The sweep went from once at the beginning of execution all the way up to re-analyzing timing within the inner loop of the placement algorithm. We present two tables showing the effect of this re-analysis interval. The first results shown in Table 5.1 are for timing analysis in the outer loop of the placement algorithm. This first column in this table shows the number of temperature changes between each timing-analysis (which we call the re-timing-analysis interval), the second column shows the placement estimated critical path, and the third column shows the $Cost_{linear\ congestion}$.

TABLE 5.1 Effect of re-timing-analysis in the outer loop

Re-Timing-Analysis Interval	Placement Estimated Critical Path (ns) (20 Circuit Geometric Average)	$Cost_{linear\ congestion}$
1	39.3	529.6
2	39.5	531.1
4	40.1	530.5
8	40.5	531.0
16	39.5	530.3
32	41.4	534.5
64	41.3	528.3
128	43.0	522.9
Never	43.0	522.9

Table 5.2 shows the effect of re-timing-analyzing the circuit in the inner loop of the placement algorithm. The first column shows how many re-timing-analysis are being performed in the inner loop of the annealer at each temperature, the second column shows the placement estimated critical path, and the third column shows the $Cost_{linear\ congestion}$.

TABLE 5.2 Effect of re-timing-analysis in the inner loop

Number of Re-Timing-Analysis in the Inner Loop at Each Temperature	Placement Estimated Critical Path (ns) (20 Circuit Geometric Average)	$Cost_{linear\ congestion}$
1	39.3	529.6
10	39.2	528.8
50	40.1	525.6
100	39.7	530.9

These results indicate that performing a timing analysis once per temperature is sufficient to obtain the best placement results. Surprisingly, timing-analyzing more than this causes a very small degradation in the performance (with respect to delay) of the placement algorithm, however, this is likely due to random effects of the simulated annealing placement algorithm. It appears that this re-analysis interval does not affect the bounding box cost of the placement. For the remainder of our experiments, we set T-VPack to re-timing-analyze each circuit once per temperature change.

The next parameter that we will discuss is the Criticality_Exponent. We have performed two sets of experiments to determine the best value for the Criticality_Exponent, in the first experiment we have set λ to 0.5, and have used both constant Criticality_Exponent and adaptive Criticality_Exponent values. We then performed the same experiments with λ set to 1. Again, all of the results presented are the placement estimated critical paths and $Cost_{linear\ congestion}$.

TABLE 5.3 Effect of Criticality_Exponent with a λ value of 0.5.

Criticality_Exponent	Adaptive Criticality_Exponent (Init_Exp = 1)		Constant Criticality_Exponent	
	Placement Estimated Critical Path (ns) (20 Circuit Geometric Average)	$Cost_{linear\ congestion}$	Placement Estimated Critical Path (ns) (20 Circuit Geometric Average)	$Cost_{linear\ congestion}$
1	38.9	342.0	38.9	342.0
2	37.1	342.3	37.1	343.4
3	35.7	343.4	35.9	344.0
4	34.6	344.0	34.8	344.7
5	34.2	341.5	34.7	343.7
6	34.3	341.5	34.8	341.6
7	33.9	339.5	34.3	339.6
8	34.0	339.9	34.3	340.1
9	34.4	336.5	33.8	339.6
10	33.9	336.5	34.3	337.9
11	34.6	336.1	34.3	336.3

We first show the effect of the constant and adaptive Criticality_Exponents with $\lambda = 0.5$ in Table 5.3. These results show that increasing the criticality exponent up to about size 8 or 9 improves the placement estimated critical path, at which point no more gains are apparent. These experiments show that the adaptive Criticality_Exponent is slightly better than the constant Criticality_Exponent (in most cases the critical path is less, and in all cases the $Cost_{linear\ congestion}$ is less). These results also show that large exponents improve the $Cost_{linear\ congestion}$. This is not surprising since large exponents make very few connections have a high Timing_Cost, and all other connections have an insignificant Timing_Cost. As a result, the placement algorithm is able to focus on minimizing only area for nets that do not have a critical connection (which is likely the majority of nets).

TABLE 5.4 Effect of Criticality_Exponent with a λ value of 1

Criticality_Exponent	Adaptive Criticality_Exponent (Init_Exp = 1)		Constant Criticality_Exponent	
	Placement Estimated Critical Path (ns) (20 Circuit Geometric Average)	$Cost_{linear\ congestion}$	Placement Estimated Critical Path (ns) (20 Circuit Geometric Average)	$Cost_{linear\ congestion}$
1	39.3	529.6	39.3	529.6
2	36.3	545.4	36.4	540.9
3	35.3	561.1	36.1	567.4
4	35.9	581.9	37.6	593.3
5	36.1	606.9	36.5	623.8
6	40.1	651.5	40.2	681.0
7	40.3	693.1	43.8	717.3

The next experiment (Table 5.4) shows that when $\lambda = 1$, an exponent value of 2 or 3 is the best. It also shows that using the adaptive Criticality_Exponent is slightly better than using a constant Criticality_Exponent. Compared to the results that we displayed in Table 5.3 the critical path is worse, and the $Cost_{linear\ congestion}$ is much worse. It is surprising that the delay results for a λ value

of 1 are worse than a λ value of 0.5 since in $\lambda = 1$ case, the algorithm is only attempting to minimize delay, while in the $\lambda = 0.5$ case the algorithm is considering both delay and wirelength minimization. This result deserves more discussion.

When we set up the algorithm to only minimize delay (by setting λ=1), it attempts to minimize the current critical path at the cost of extending other non-critical paths. Since we are only re-timing-analyzing the circuit once per temperature, the algorithm has many moves between updates of the connection criticalities and slacks. This means that it is likely that the algorithm is able to significantly reduce critical paths during one iteration of the outer loop, but at the same time inadvertently make other paths very critical. This oscillation effect makes it difficult for the placement algorithm to converge onto the best placement solution.

By including a wirelength minimization term in the cost equation, we are able to reduce the oscillations of the placement. This is because, the wirelength term will not let the placer make moves that significantly increase the wirelength of the placement, even if the move would significantly reduce the current critical path. Effectively, the wirelength term acts as a damper on the delay minimization term in our cost function.

The above results show that using an adaptive Criticality_Exponent value of 8 with a trade-off of 0.5 provides the best results so far. Based on these results, we use an adaptive Criticality_Exponent set to 8, and perform a sweep of λ. The results of this sweep are shown in Table 5.5.

This table shows that an algorithm that is only wirelength driven produces the best $Cost_{linear\ congestion}$. It also shows that an algorithm with a λ of 0.9 produces circuits with the best placement estimated critical path delay. A λ of 1 is bad for both critical path, and delay for the reasons explained above. We feel that setting λ to 0.5 provides the best compromise between wirelength and critical path minimization, and so the remainder of our experiments use this value.

TABLE 5.5 Effect of λ with an adaptive Criticality_Exponent of 8

λ	Placement Estimated Critical Path (ns) (20 Circuit Geometric Average)	$Cost_{linear\ congestion}$
0	51.6	312.7
0.1	40.0	315.8
0.2	37.8	318.5
0.3	36.7	322.8
0.4	35.6	331.1
0.5	34.0	339.8
0.6	33.2	353.6
0.7	32.5	373.9
0.8	32.5	400.7
0.9	32.4	439.7
1	43.4	725.3

5.2.3 Verification of the Fidelity of the Placement Estimated Critical Path Delay

In the previous section we used placement estimated critical path delays to compare different parameters used in the placement algorithm. It is interesting to see how much correlation there is between this estimate and the actual post-place-and-route critical path delays. To study the correlation, we present a λ sweep graph with a Criticality_Exponent of 8 in Figure 5.2. This graph shows the infinite routing resource post-place-and-route delay, the low-stress post-place-and-route delay, and the placement estimated post-place-and-route delay. There is a excellent correlation between the placement estimated critical path and the infinite routing-resource critical path, additionally the low-stress results follow the same trend as the placement-estimated results. We therefore believe that our placement-estimated delay results are a valid indicator of the best values for the various parameters that we evaluated in the previous section.

5.2.4 Time Complexity

The complexity of our algorithm is essentially the same as VPlace. We perform a timing analysis once per temperature change which is an O(n) operation. At each temperature we execute the inner loop of the placer $O(n^{4/3})$ times (i.e. we perform $O(n^{4/3})$ swaps). In the inner loop we have an incremental-bounding-box-update operation that is worst case $O(k_{max})$, where K_{max} is the fanout of the largest net in the circuit. The average case complexity for this bounding box update is O(1) [Betz98b, Betz99]. Also in the inner loop is the computation of the Timing_Cost for each connection affected by a swap. This is also $O(k_{max})$. In the average case this is $O(k_{avg})$ where k_{avg} is the average fanout of all nets in the circuit. Since k_{avg} is typically quite small, the average complexity of this Timing_Cost computation is O(1) as well. The overall result is that our algorithm is worst case $O[(k_{max} \cdot n)^{4/3}]$, but on average it is $O(n^{4/3})$. Our algorithm takes about 2.5 times as long as VPlace to place the largest MCNC circuit (clma).

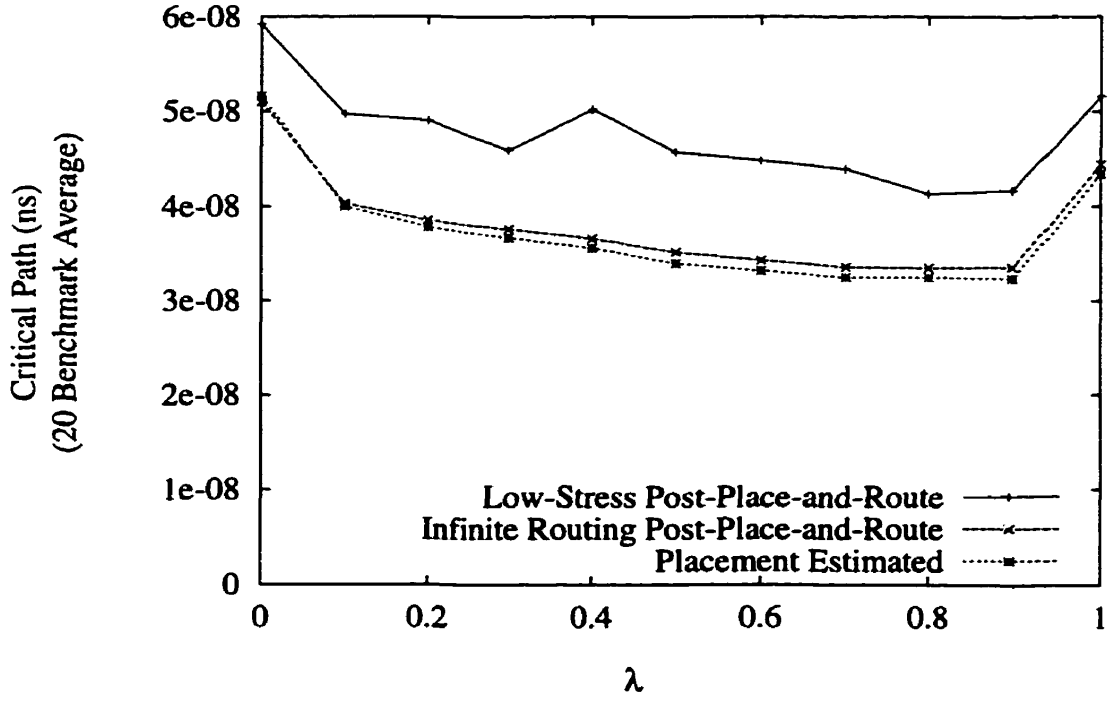


FIGURE 5.2 Graph showing fidelity of placement estimated critical path.

5.3 Results: VPlace vs. T-VPlace

In this section we compare the post-place-and-route results from VPlace and T-VPlace. We show the results for size 1,4, and 8 clusters. We also show the effect of using T-VPack with T-VPlace vs. using VPack with VPlace. Again, our results are the average of the 20 MCNC benchmark circuits, and we use the routing architecture described in Section 4.3. Additionally, all of the results that we present are based on an adaptive Criticality_Exponent of 8, and a re-timing-analysis interval of once per temperature change.

The first results that we present are for size 1 clusters. We show post-place-and-route VPlace results and T-VPlace results. Table 5.6 shows that for the infinite routing case, T-VPlace improves circuit speed by about 42% (a 30% decrease in delay) on average compared to VPlace. For the low stress routing case, T-VPlace improves circuit speed by 25% (a 20% reduction in delay) on average compared to VPlace. The cost of this speed gain is only a 5% increase in the minimum channel width. It is likely that the low-stress routing results do not show the same improvement in speed as the infinite routing results due to the fact that the placement algorithm has made it more difficult for the router to optimize the critical path(s). This is because T-VPlace produces circuits that have shorter critical paths than VPlace, but more of them[1]. The result is that the router has many more paths to shorten, making it more difficult in the low-stress routing case for the router to get close to the "lower bound" that the infinite routing results represent.

The next results that we present in Table 5.7 are for size 8 clusters. In this case, we show VPlace results for VPack and T-VPack, and T-VPlace results for T-VPack. This allows us to evaluate the combined effect of using T-VPack and T-VPlace (our contributions) vs. using VPack and VPlace.

This table shows that when we have infinite routing resources, T-VPack combined with T-VPlace speeds up circuit speed by 39% (a 28% reduction in delay) compared to using Vpack and VPlace. If we compare only the placement algorithms using the same packing algorithm, we see that T-VPlace improves circuit speed by 21% (an 18% reduction in delay) compared to using VPlace.

1. We show the critical path distributions in Appendix C.

TABLE 5.6 Post-place-and-route comparison of VPlace and T-VPlace (cluster size = 1).

Circuit	Post-Place-and-Route Minimum Channel Width (W_{min})			Post-Place-and-Route Critical Path (ns) $W = \infty$			Post-Place-and-Route Critical Path (ns) $W = W_{min} + 20\%$		
	VPlace	T-VPlace ($\lambda = 0$)	T-VPlace ($\lambda = 0.5$)	VPlace	T-VPlace ($\lambda = 0$)	T-VPlace ($\lambda = 0.5$)	VPlace	T-VPlace ($\lambda = 0$)	T-VPlace ($\lambda = 0.5$)
alu4	14	14	14	40.3	40.4	29.8	42.4	41.2	33.4
apex2	15	17	16	46.9	46.3	32.3	47.7	46.5	48.8
apex4	17	16	18	40.9	44.8	28.2	42.0	46.8	31.7
bigkey	13	13	10	36.0	35.2	21.6	36.7	35.4	25.2
clma	16	16	17	90.2	91.1	72.3	116.0	166.0	130.0
des	11	12	11	40.5	48.9	30.2	50.4	57.4	43.7
diffeq	11	11	12	35.2	37.5	30.8	38.9	41.0	34.9
dsip	12	12	12	27.9	27.2	21.7	28.3	28.8	22.9
elliptic	14	16	15	70.6	76.1	46.1	79.5	79.6	58.1
ex1010	14	15	15	85.0	77.5	52.9	96.2	78.6	70.5
ex5p	17	17	19	39.6	40.4	28.1	42.7	42.7	43.5
frisc	16	17	18	70.8	73.2	59.6	76.8	79.6	61.6
misex3	14	15	15	39.0	40.2	26.6	39.3	75.0	34.3
pdc	22	21	24	81.7	74.5	49.9	122.0	114.0	73.0
s298	11	12	12	74.8	72.0	53.6	116.0	78.7	77.8
s38417	11	11	12	61.7	71.0	33.7	70.0	74.6	37.2
s38584.1	11	11	11	45.3	44.1	31.8	49.7	44.3	36.4
seq	16	16	16	45.7	41.0	28.1	46.4	43.7	39.5
spla	18	18	20	58.4	67.4	39.7	74.8	100.0	69.4
tseng	9	10	11	33.7	33.1	28.3	39.8	38.4	33.1
Geom. Av.	13.78	14.22	14.50	50.1	51.0	35.2	57.1	59.2	45.7
%diff w.r.t VPlace	—	+3.2%	+5.2%	—	+1.8%	-29.7%	—	+1.04%	-20.0%

TABLE 5.7 Post-place-and-route comparison of VPlace and T-VPlace (cluster size = 8).

Circuit	Post-Place-and-Route Minimum Channel Width (W_{min})				Post-Place-and-Route Critical Path (ns) $W = \infty$				Post-Place-and-Route Critical Path (ns) $W = W_{min} + 20\%$			
	VPack with VPlace	T-VPack with VPlace	T-VPack with T-VPlace ($\lambda = 0$)	T-VPack with T-VPlace ($\lambda = 0.5$)	VPack with VPlace	T-VPack with VPlace	T-VPack with T-VPlace ($\lambda = 0$)	T-VPack with T-VPlace ($\lambda = 0.5$)	VPack with VPlace	T-VPack with VPlace	T-VPack with T-VPlace ($\lambda = 0$)	T-VPack with T-VPlace ($\lambda = 0.5$)
alu4	55	39	38	38	28.1	25.1	24.8	22.6	30.7	27.9	32.7	29.4
apex2	58	55	55	50	37.5	32.7	31.8	26.6	37.9	34.5	35.0	30.8
apex4	53	52	52	53	28.5	25.5	26.7	22.3	34.5	32.9	34.2	27.6
bigkey	41	27	27	28	17.4	16.6	17.8	12.8	18.7	17.5	18.4	16.2
clma	75	64	68	68	82.9	64.7	66.7	48.5	84.3	77.1	67.5	58.3
des	36	29	29	29	28.8	27.4	26.7	22.3	29.9	29.4	29.9	25.3
diffeq	37	33	32	33	38.5	26.6	28.1	26.6	41.3	31.6	28.7	28.3
dsip	41	23	24	24	19.2	20.2	18.3	12.3	23.1	22.2	19.3	15.4
elliptic	57	49	50	50	50.2	40.1	40.5	34.6	60.5	49.1	51.2	46.8
ex1010	61	58	59	59	45.4	42.7	43.8	35.1	53.0	56.3	55.0	42.1
ex5p	55	53	51	50	27.6	28.1	26.7	23.4	31.9	30.9	31.2	31.0
frisc	57	58	56	56	75.0	56.4	58.4	48.3	80.2	65.3	63.6	54.7
misex3	49	43	46	44	25.5	25.3	25.2	23.6	29.1	30.6	28.1	27.4
pdc	82	76	75	78	56.7	52.7	46.9	35.3	57.9	81.8	58.3	50.2
s298	48	28	28	31	49.9	49.7	47.2	44.4	58.0	63.3	80.6	64.0
s38417	47	42	44	45	51.2	41.6	40.6	30.5	59.7	45.0	46.1	40.8
s38584.1	43	44	44	45	39.6	29.3	28.7	26.0	40.2	30.3	30.6	27.2
seq	57	47	48	50	30.2	27.2	26.6	23.2	31.5	37.0	30.1	26.1
spla	76	59	61	68	47.0	41.0	36.0	30.7	48.3	47.0	45.7	35.3
tseng	39	33	34	33	37.1	29.0	29.3	28.8	38.2	33.8	33.5	31.0
Geom. Av.	51.9	43.4	43.9	44.3	37.7	33.0	32.5	27.2	41.2	38.9	38.0	33.1
% diff w.r.t Vpack with VPlace	—	-16.4%	-15.4%	-14.6%	—	-12.5%	-13.8%	-27.9%	—	-5.6%	-7.8%	-19.7%

Also shown are low-stress delay results which show that T-VPack combined with T-VPlace speeds up circuit execution by 25% (a 20% reduction in delay) compared to using Vpack and VPlace. If we compare only the placement algorithms using the same packing algorithm, we see that T-VPlace improves circuit speed by 16% (a 14% reduction in delay) compared to using VPlace.

The results shown in Table 5.8 are for a Xilinx 4000x-like architecture presented in [Swar98b]. The logic block used in this experiment is a logic cluster consisting of 4 BLEs with 10 inputs. The routing contains 25% length 1 wires, 12.5% length 2 wires, 37.5% length 4 wires, and 25% "one-quarter longs", whose length is one-fourth of the chip. The length 1 and 2 wires connect with pass transistors, while the longer wires connect with tri-state buffers. As well, there are pass-transistor switches connecting length 4 wires to the length 1 and 2 wires, and connecting the one-quarter longs to the length 1 wires. In this table we show results for T-VPlace with λ=0 and λ=0.5. The λ=0 results are essentially the same as what the original VPlace would produce, so we have not generated results using VPlace on this particular architecture.

These results show that incorporating timing information into the cost function results in a speedup of 22% (an 18% reduction in delay) in the infinite routing case, and a speedup of 14% (a 12% reduction in delay) for the low-stress routing case. The cost of this is a 5% increase in the minimum channel width.

5.4 Summary

In this chapter we discussed our new timing-driven placement algorithm T-VPlace. In our discussions, we explained how various parameters within the algorithm were selected. We also discussed the complexity of our algorithm, which is the same as that of VPlace, and takes about 2.4 times as long as VPlace to place the largest circuit (clma). After this, we demonstrated the improvements resulting from our new algorithm compared to the existing non-timing-driven VPlace algorithm. We showed for an architecture with size 1 clusters, T-VPlace was able to speed

TABLE 5.8 Post-place-and-route comparison with Xilinx-like architecture (cluster size = 4).

Circuit	Post-Place-and-Route Minimum Channel Width (W_{min})		Post-Place-and-Route Critical Path (ns) W = ∞		Post-Place-and-Route Critical Path (ns) W = W_{min} + 20%	
	T-VPlace ($\lambda = 0$)	T-VPlace ($\lambda = 0.5$)	T-VPlace ($\lambda = 0$)	T-VPlace ($\lambda = 0.5$)	T-VPlace ($\lambda = 0$)	T-VPlace ($\lambda = 0.5$)
alu4	27	27	31.7	28.2	33.7	35.0
apex2	34	34	34.6	30.9	42.2	43.6
apex4	34	35	32.9	26.3	46.8	35.7
bigkey	17	19	20.9	15.3	27.2	23.8
clma	41	45	70.2	58.6	80.8	67.4
des	18	19	31.1	24.9	32.8	31.3
diffeq	23	23	35.5	32.1	36.6	32.8
dsip	16	20	20.6	13.7	29.1	19.3
elliptic	31	31	51.6	40.1	60.6	51.1
ex1010	36	37	54.9	45.7	64.4	62.4
ex5p	33	34	31.6	26.2	36.4	34.0
frisc	39	38	63.9	51.5	79.2	56.7
misex3	32	33	30.0	24.8	34.6	29.1
pdc	47	51	57.6	43.2	78.2	56.6
s298	22	25	66.6	48.4	78.3	78.8
s38417	28	29	42.7	38.0	48.5	42.7
s38584.1	27	29	29.5	29.3	34.5	34.6
seq	32	33	32.7	25.6	38.4	34.1
spla	43	44	51.0	38.0	63.0	62.5
tseng	18	19	31.2	29.7	34.6	31.4
Geom. Av.	28.5	30.0	38.5	31.5	45.9	40.4
%diff w.r.t. $\lambda = 0$	—	+5.2%	—	-18.2%	—	-12.0%

up the resulting circuits by 40% at a cost of only about a 5% increase in the minimum channel width. For size 8 clusters, T-VPlace was able to speed up the resulting circuits by 21% with only a 5% increase in the minimum channel width. Overall, it is clear that timing-driven placement can significantly improve performance without making large sacrifices in area.

CHAPTER 6 *Conclusions and Future Work*

6.1 Conclusions and Contributions

The goal of this thesis was to explore new architectures and CAD algorithms to reduce FPGA delay without sacrificing large amounts of area. This involved exploring cluster-based FPGA architectures, as well as developing new timing-driven packing and placement algorithms.

In Chapter 3 we described a new packing algorithm called T-VPack. This algorithm was designed to absorb critical path connections into logic clusters to take advantage of the fast speed offered by intra-cluster connections. T-VPack not only improved the delay of the resulting circuits, but also improved the area. Compared to VPack, T-VPack generated circuits that required 16% fewer tracks to route and were 13% faster.

In Chapter 4 we studied the effectiveness of using logic-cluster based FPGA architectures. Previous work [Betz98b, Betz99] had studied the effectiveness of logic clusters with respect to area-efficiency, however that work did not consider the effect of logic clusters on FPGA speed. We believe that our work is the first to study logic-cluster based architectures with respect to both speed and area. Using the area-delay product metric we showed that an FPGA architecture should be composed of logic clusters containing between 4 and 10 basic logic elements.

In order to reduce FPGA delay, it is important that placement algorithms consider delay. In Chapter 5 we developed a new timing-driven placement algorithm called T-VPlace and we ran many experiments to tune various parameters within the algorithm. Our algorithm significantly reduced the post-place-and-route critical path delay of the resulting circuits. For an FPGA with size 1 clusters, T-VPlace reduced the critical path by 30% (a speed increase of 42%) compared to using VPlace, while only sacrificing a small amount of routing area. Additionally, our algorithm was quite computationally efficient — it only took 2.4 times as long to place the largest MCNC circuit as VPlace.

6.2 *Future Work*

This thesis explored three approaches to improving FPGA performance, cluster-based logic blocks, timing-driven packing, and timing-driven placement.

In the future, it would be interesting to see how a cluster-based architecture with LUTs other than size 4 or a combination of different size LUTs would perform. For FPGAs composed of these new logic clusters, it would be interesting to see how many inputs are required for full utilization, and how flexible the routing interface should be.

Another area of interest would be to study an FPGA with nearest-neighbor interconnects. These connections would allow critical paths to be routed between clusters using fast nearest-neighbor connections rather than using the relatively slow inter-cluster routing. To study this issue, it would be necessary to develop new CAD tools capable of exploiting this enhancement.

Finally, our CAD flow has an arbitrary division between packing and placement. It would be interesting to see how much improvement could be obtained by removing this arbitrary division and allowing the placer to freely place BLEs anywhere within the FPGA. This would increase the CPU compile time, but it may be worth it if the gains are significant.

APPENDIX A *MCNC Benchmarks*

In this appendix we give a description of the 20 MCNC circuits that we use in our experiments.

TABLE A.1 MCNC benchmark circuits

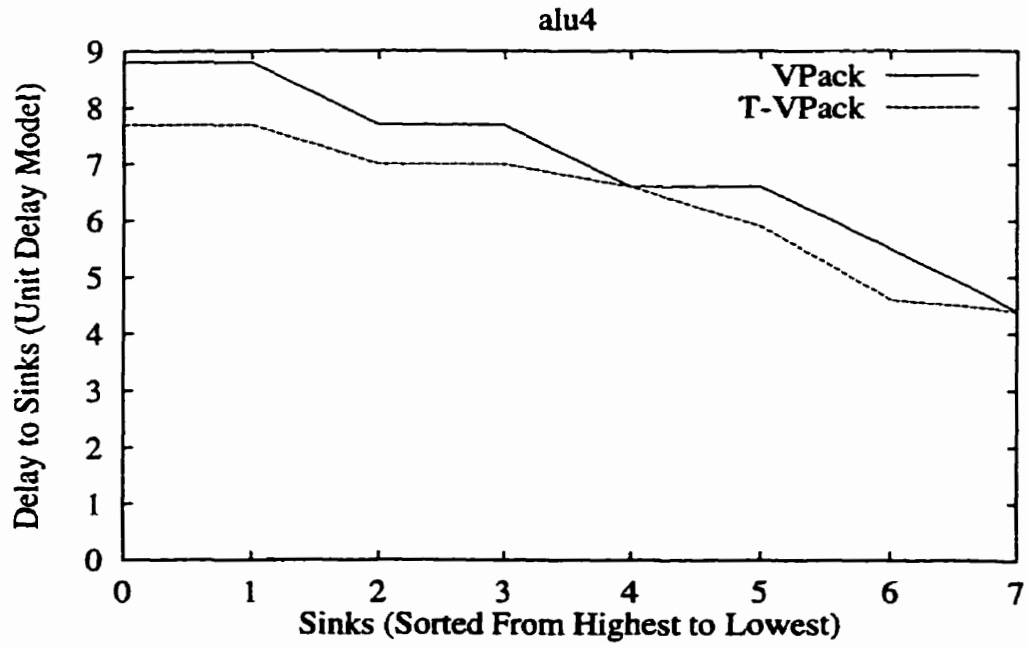
Circuit	Circuit Description					Circuit Description (after mapping to BLEs)		
	Number of LUTs	Number of Latches	Number of Nets	Number of Inputs	Number of Outputs	Number of BLEs	Number of Nets	Number of Point to Point Connections
alu4	1522	0	1536	14	8	1522	1536	5408
apex2	1878	0	1917	39	3	1878	1916	6692
apex4	1262	0	1271	9	19	1262	1271	4479
bigkey	1707	224	2194	263	197	1707	1936	6313
clma	8381	33	8797	383	82	8383	8445	30462
des	1591	0	1847	256	245	1591	1847	6110
diffeq	1494	377	1935	64	39	1497	1561	5296
dsip	1370	224	1823	229	197	1370	1599	5645
elliptic	3602	1122	4855	131	114	3604	3735	12634
ex1010	4598	0	4608	10	10	4598	4608	16078

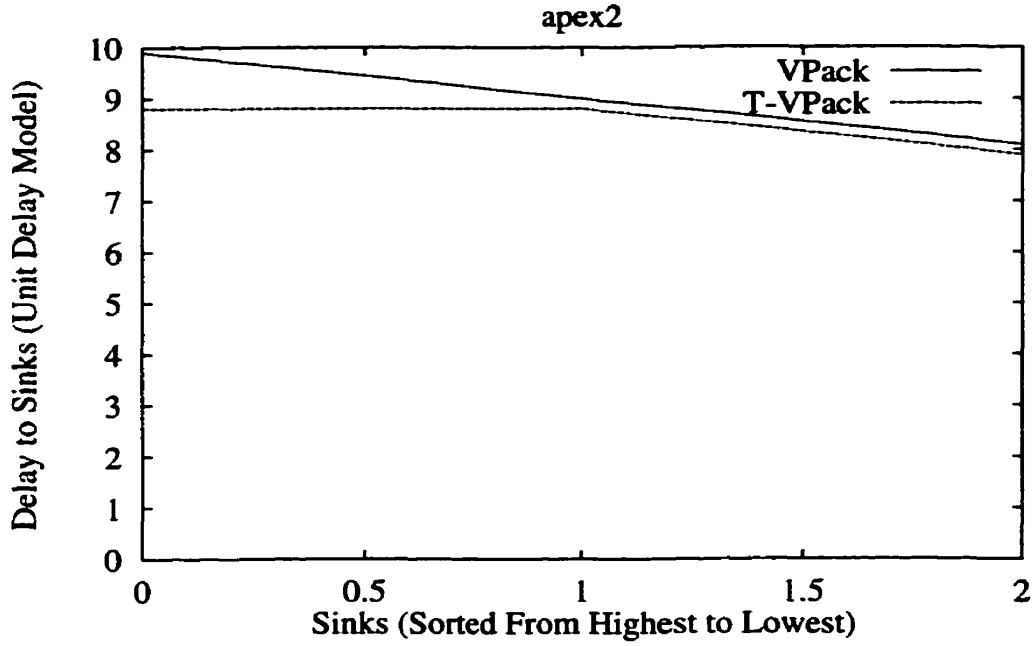
TABLE A.1 MCNC benchmark circuits

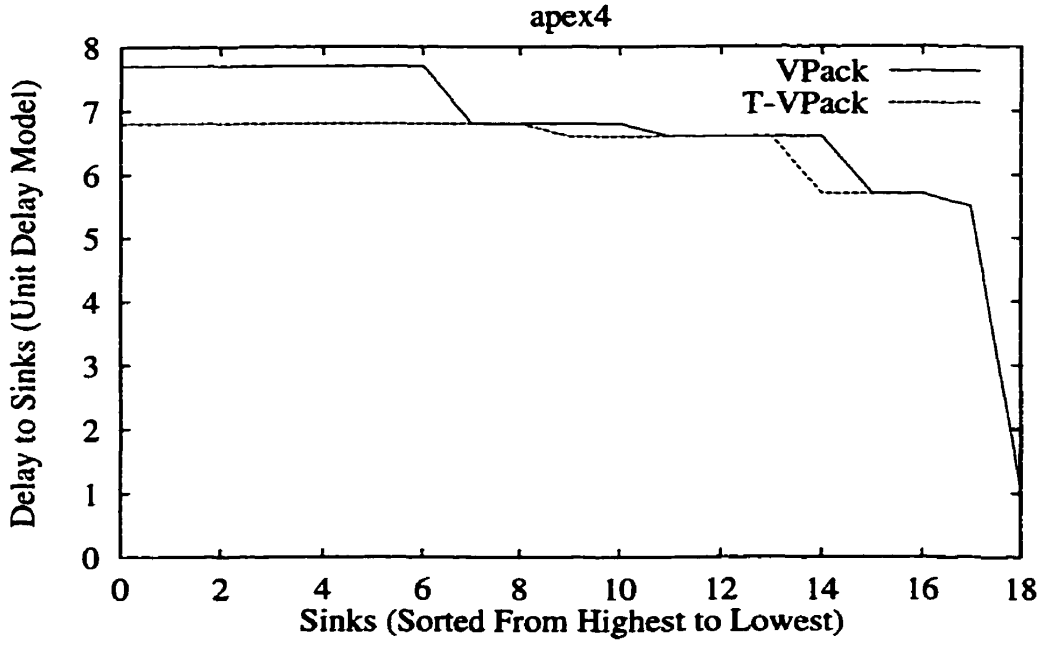
Circuit	Circuit Description					Circuit Description (after mapping to BLEs)		
	Number of LUTs	Number of Latches	Number of Nets	Number of Inputs	Number of Outputs	Number of BLEs	Number of Nets	Number of Point to Point Connections
ex5p	1064	0	1072	8	63	1064	1072	4002
frisc	3539	886	4445	20	116	3556	3576	12772
misex3	1397	0	1411	14	14	1397	1411	4968
pdc	4575	0	4591	16	40	4575	4591	17193
s298	1930	8	1942	4	6	1931	1935	6951
s38417	6096	1463	7588	29	106	6406	6435	21344
s38584.1	6281	1260	7580	39	304	6447	6485	20840
seq	1750	0	1791	41	35	1750	1791	6193
spla	3690	0	3706	16	46	3690	3706	13808
tseng	1046	385	1483	52	122	1047	1099	3760

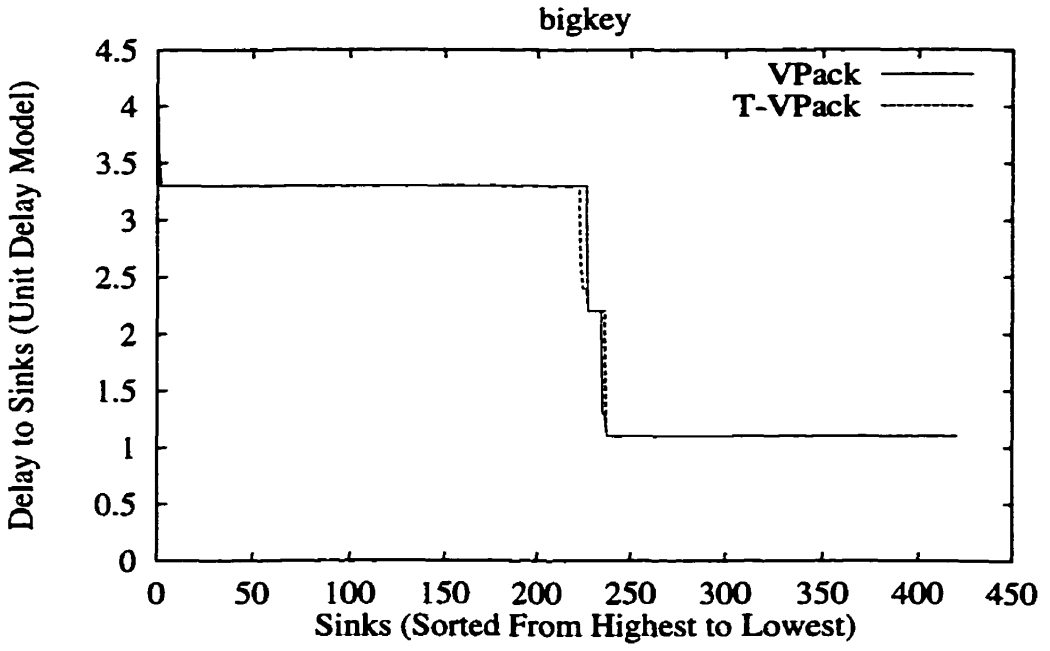
APPENDIX B *VPack and T-VPack Sink Delay Distributions: Size 8 Clusters*

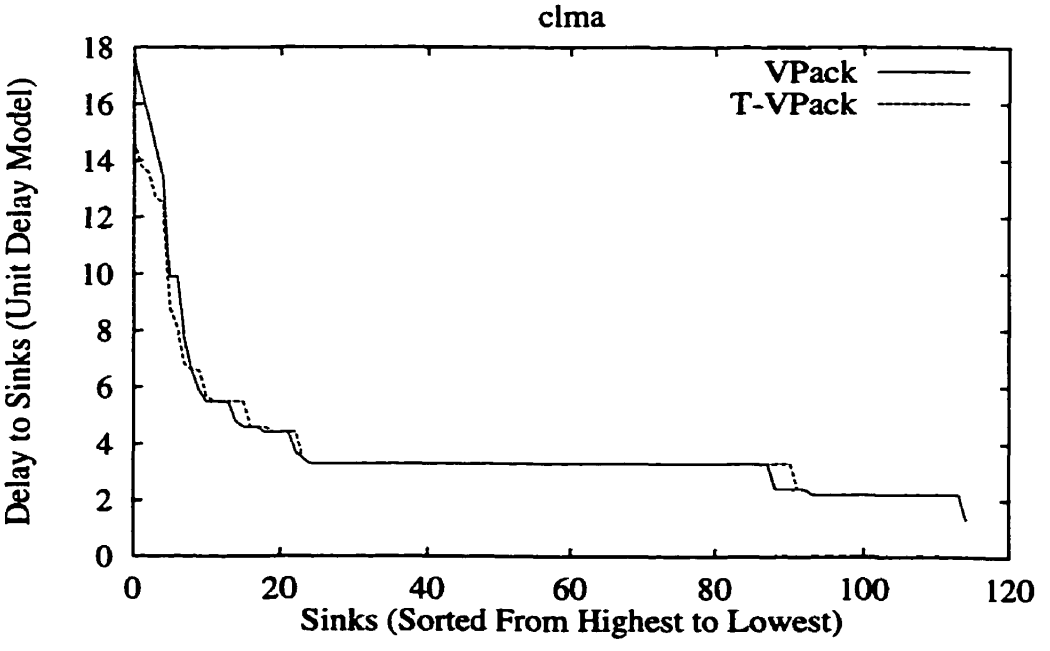
In this appendix we present pre-place-and-route sink-delay distributions for the 20 largest MCNC circuits as computed by VPack and T-VPack using the unit delay models that we described in Section 3.2.1. All of the results that we show in this appendix are for size 8 clusters. The T-VPack, results were obtained with an α of 0.75 and never re-timing analyzing.

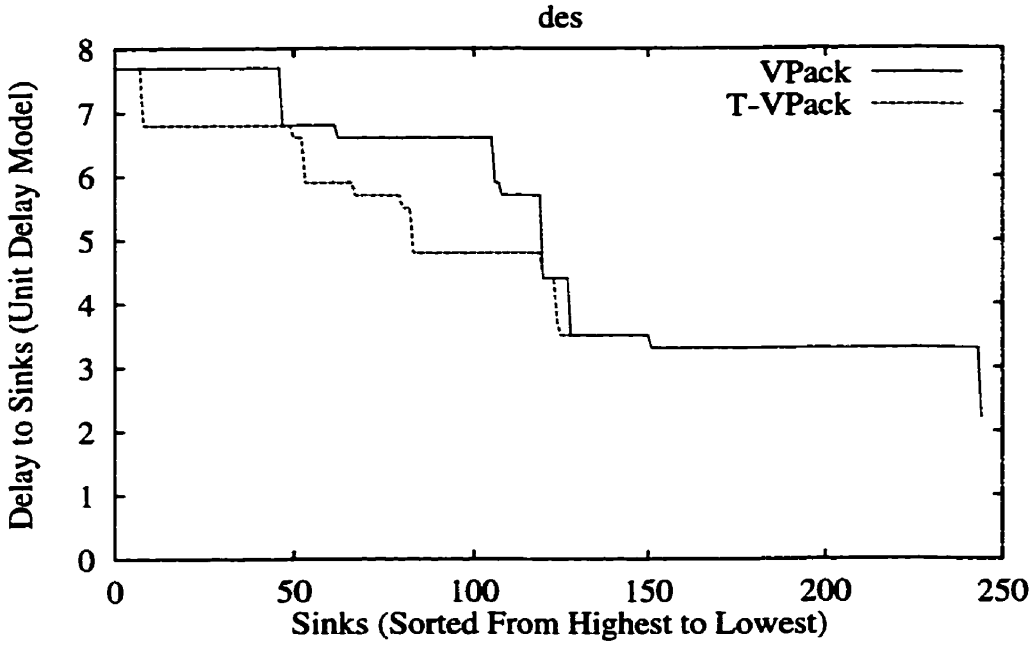


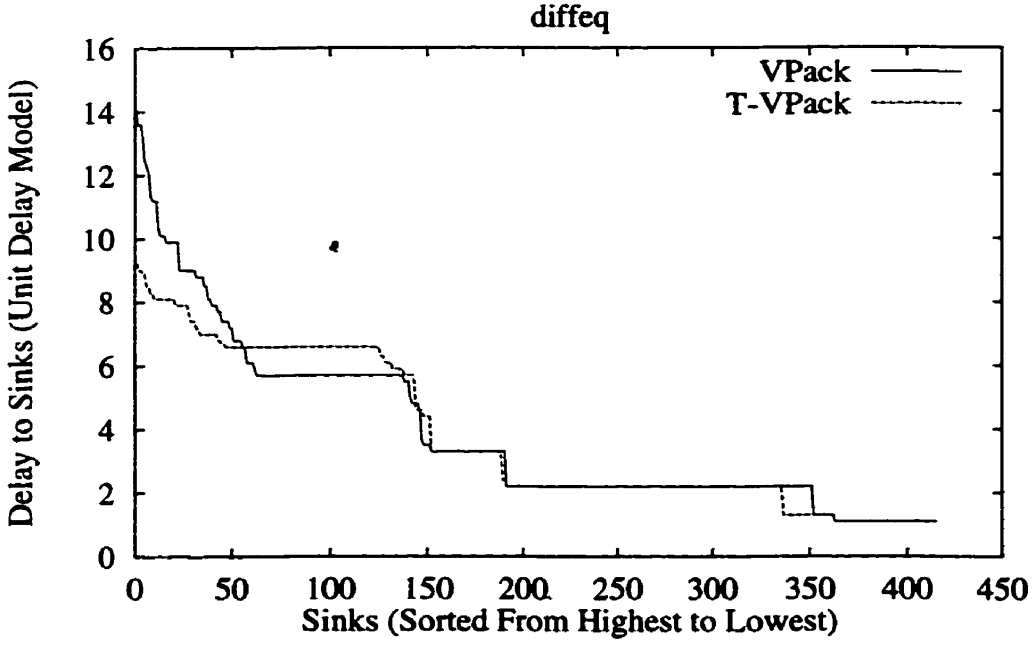

apex2
VPack
T-VPack
Delay to Sinks (Unit Delay Model)
Sinks (Sorted From Highest to Lowest)
0
1
2
3
4
5
6
7
8
9
10
0
0.5
1
1.5
2

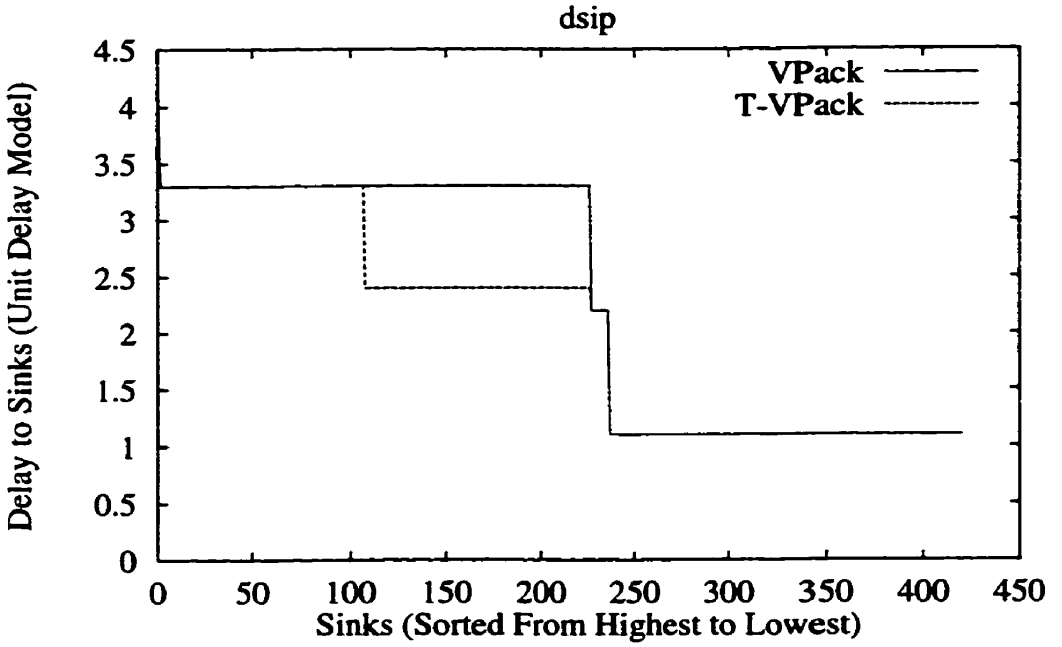

apex4
VPack
T-VPack
Delay to Sinks (Unit Delay Model)
Sinks (Sorted From Highest to Lowest)
0
1
2
3
4
5
6
7
8
0
2
4
6
8
10
12
14
16
18

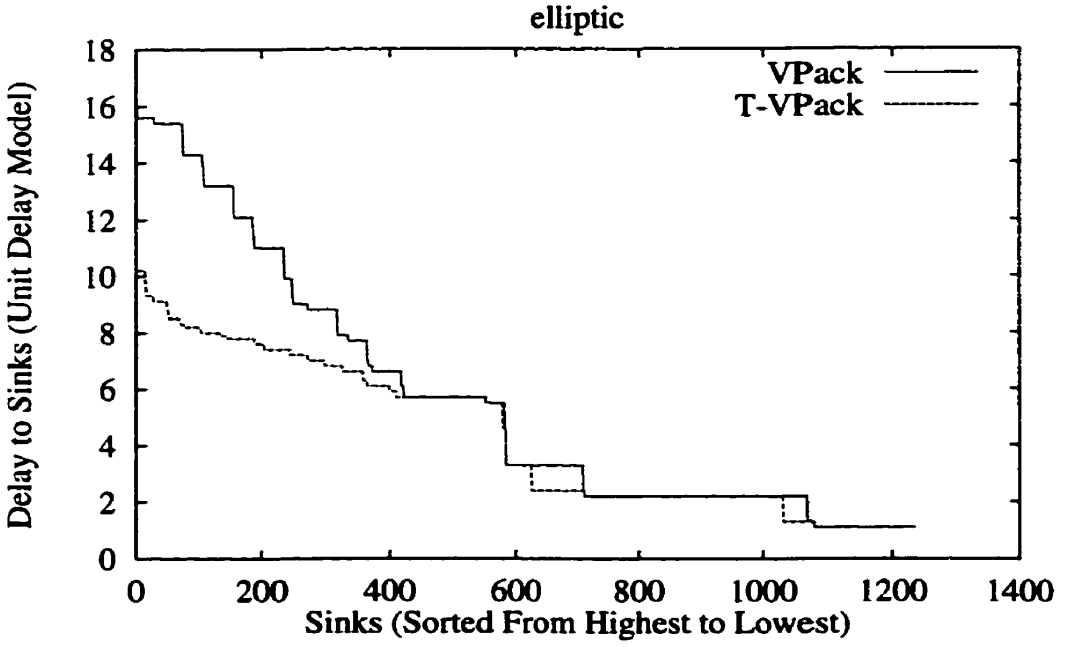

bigkey
VPack
T-VPack
Delay to Sinks (Unit Delay Model)
Sinks (Sorted From Highest to Lowest)
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
0
50
100
150
200
250
300
350
400
450

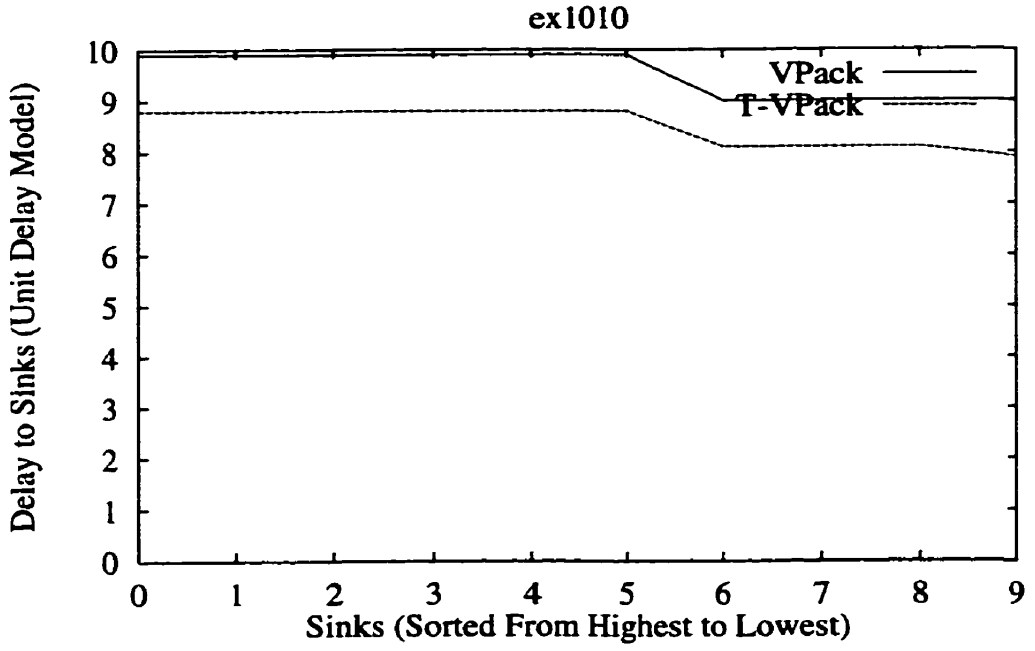

clma
VPack
T-VPack
Delay to Sinks (Unit Delay Model)
Sinks (Sorted From Highest to Lowest)
0
2
4
6
8
10
12
14
16
18
0
20
40
60
80
100
120

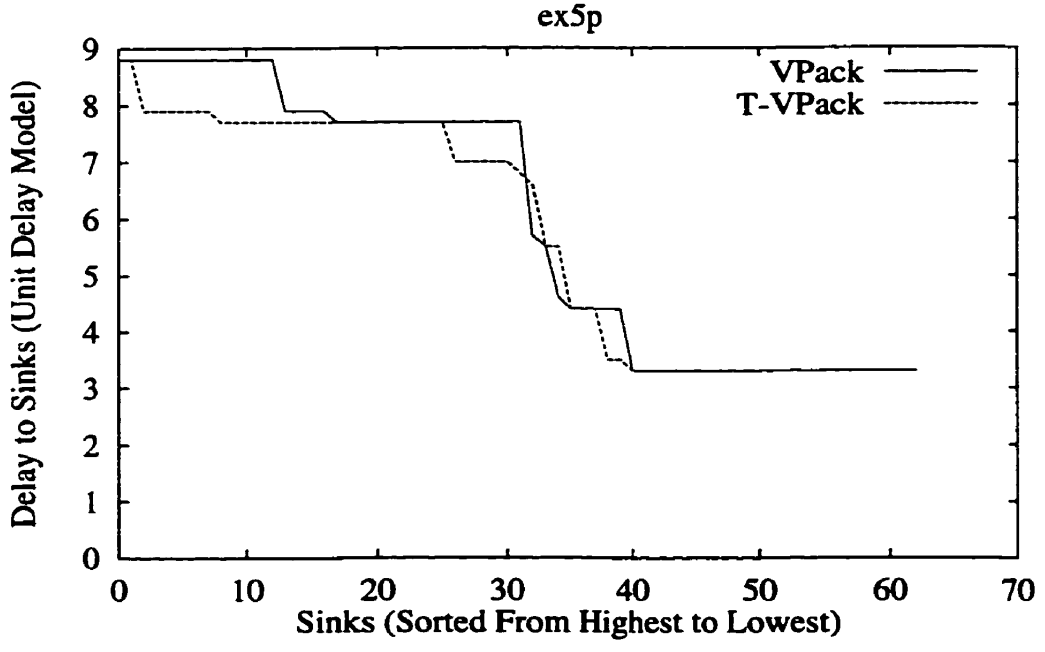

des
VPack
T-VPack
Delay to Sinks (Unit Delay Model)
Sinks (Sorted From Highest to Lowest)
0
1
2
3
4
5
6
7
8
0
50
100
150
200
250

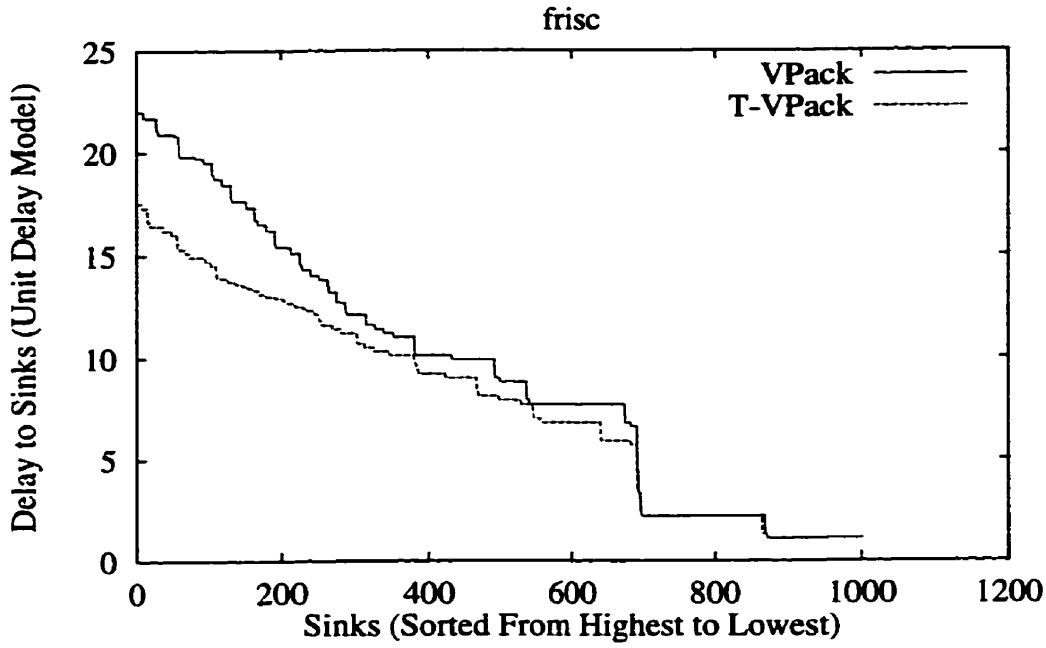

diffeq
VPack
T-VPack
Delay to Sinks (Unit Delay Model)
Sinks (Sorted From Highest to Lowest)
0
2
4
6
8
10
12
14
16
0
50
100
150
200
250
300
350
400
450

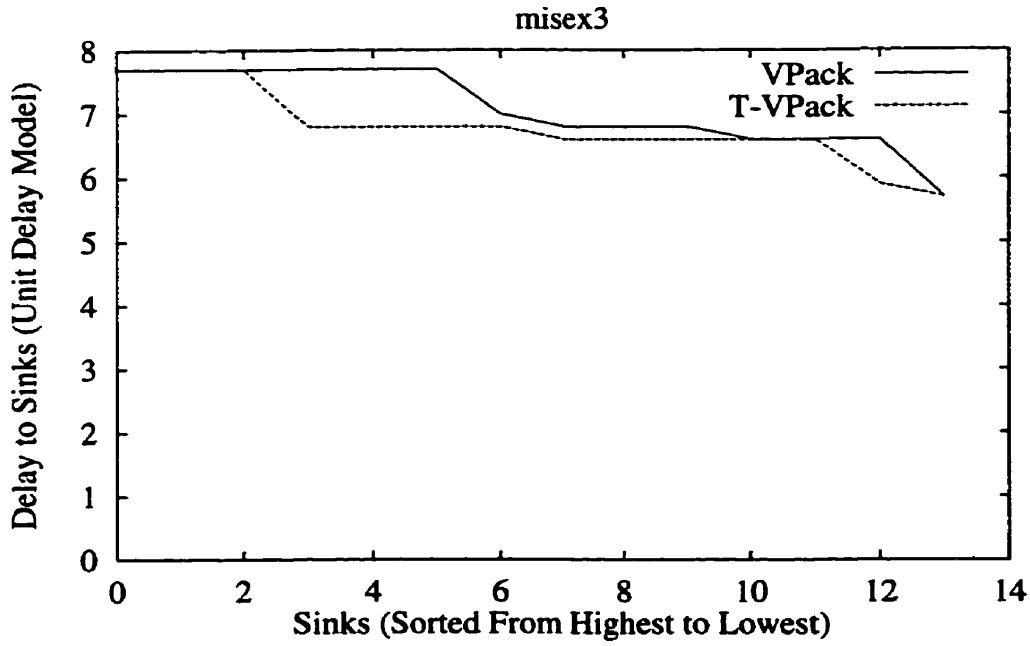

dsip
VPack
T-VPack
Delay to Sinks (Unit Delay Model)
Sinks (Sorted From Highest to Lowest)
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
0
50
100
150
200
250
300
350
400
450

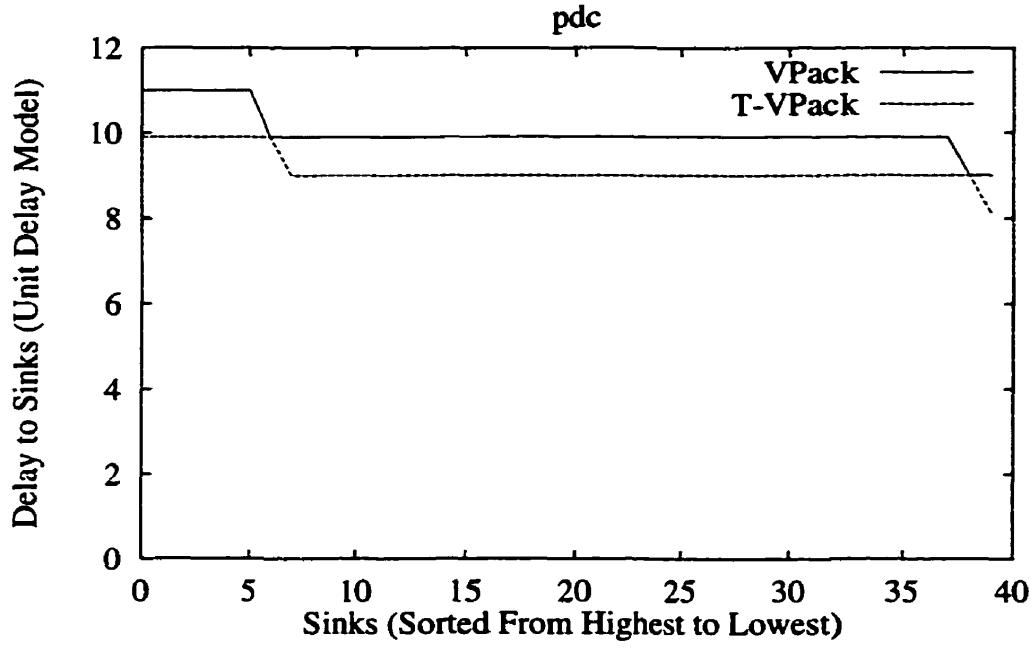

elliptic
VPack
T-VPack
Delay to Sinks (Unit Delay Model)
Sinks (Sorted From Highest to Lowest)
0
2
4
6
8
10
12
14
16
18
0
200
400
600
800
1000
1200
1400

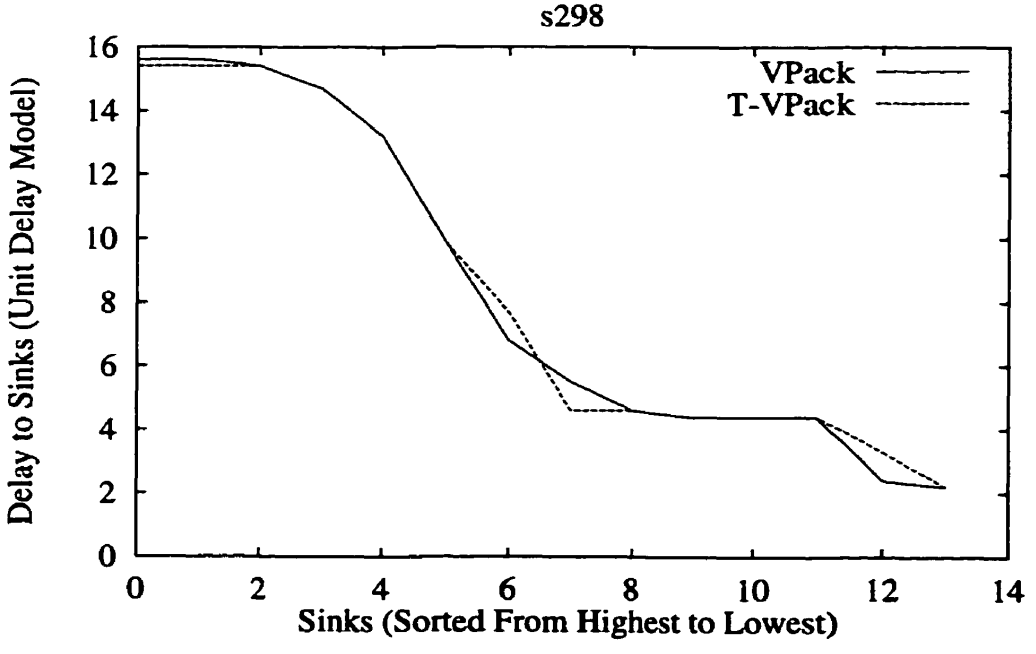

ex1010
VPack
T-VPack
Delay to Sinks (Unit Delay Model)
Sinks (Sorted From Highest to Lowest)
0
1
2
3
4
5
6
7
8
9
10

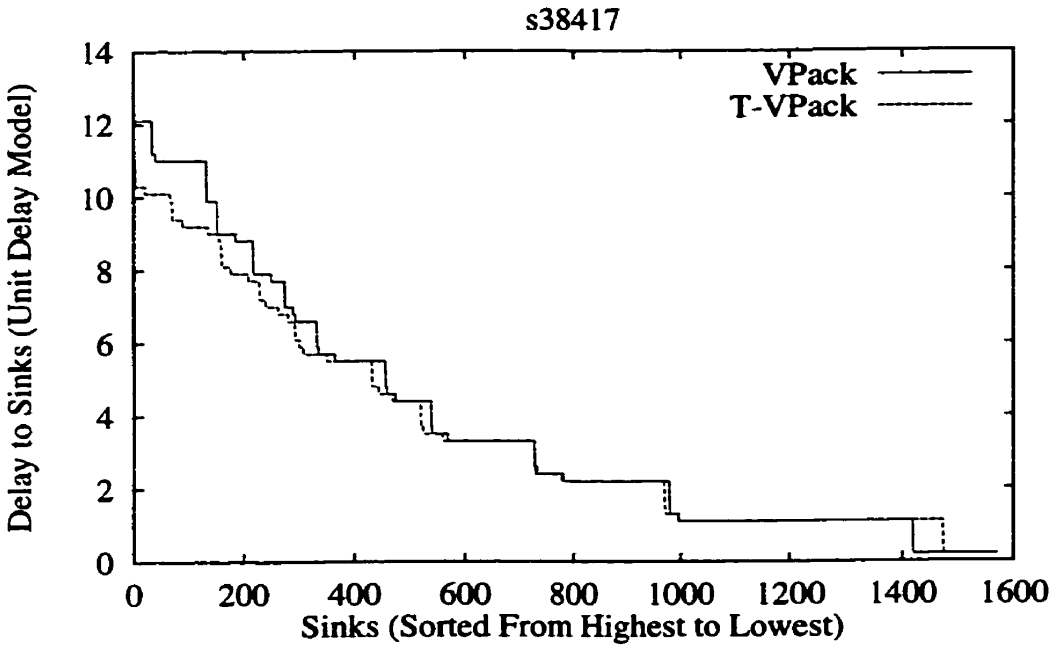

ex5p
VPack
T-VPack
Delay to Sinks (Unit Delay Model)
Sinks (Sorted From Highest to Lowest)
0
10
20
30
40
50
60
70
1
2
3
4
5
6
7
8
9

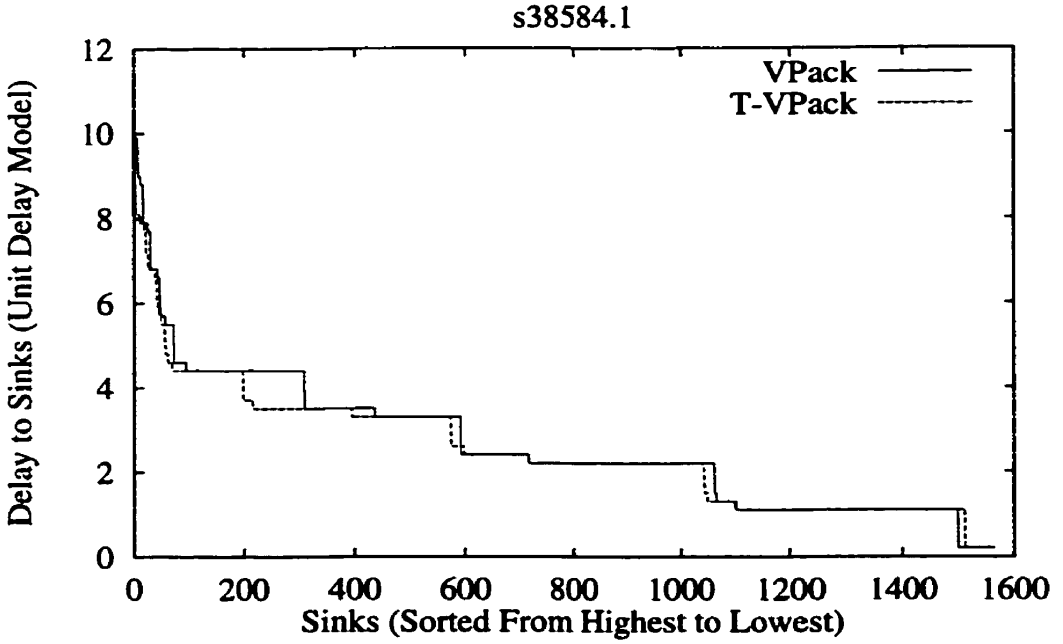

frisc
VPack
T-VPack
Delay to Sinks (Unit Delay Model)
Sinks (Sorted From Highest to Lowest)
0
5
10
15
20
25
0
200
400
600
800
1000
1200

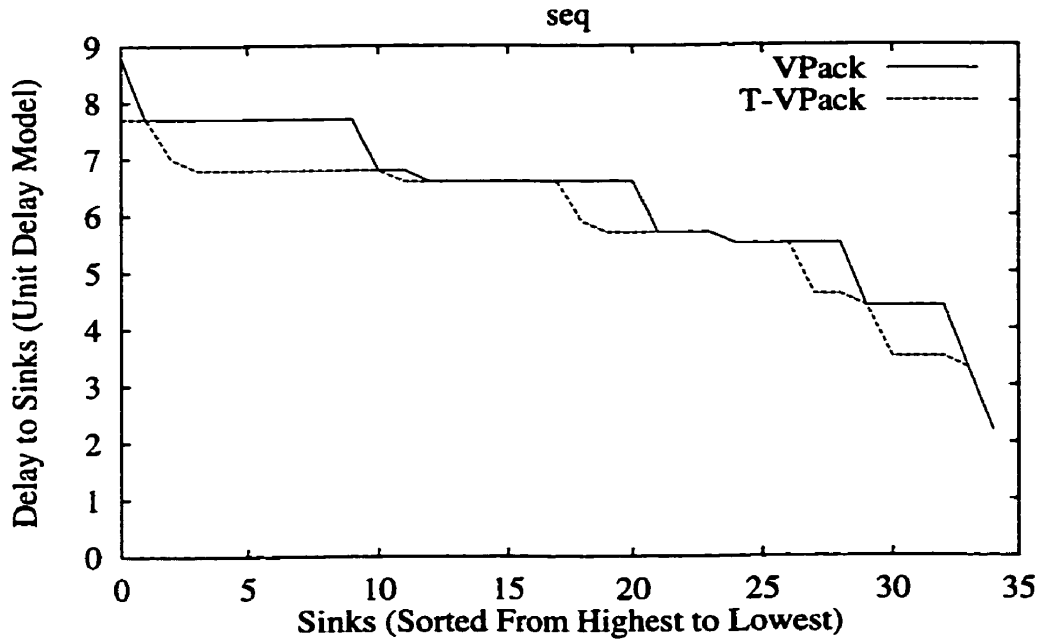

misex3
VPack
T-VPack
Delay to Sinks (Unit Delay Model)
Sinks (Sorted From Highest to Lowest)
0
1
2
3
4
5
6
7
8
0
2
4
6
8
10
12
14

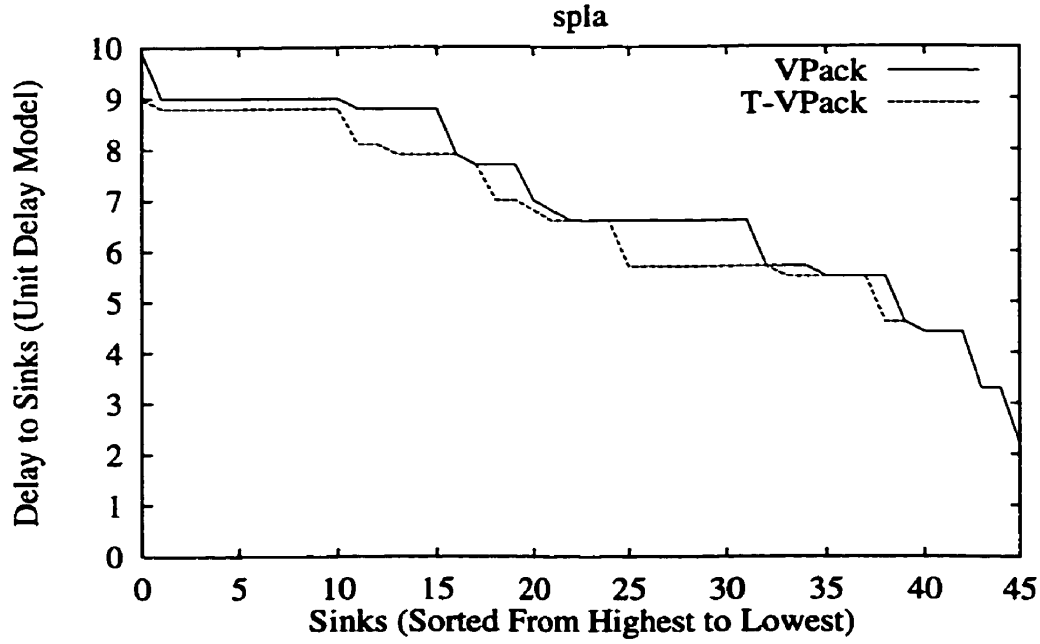

pdc
VPack
T-VPack
Delay to Sinks (Unit Delay Model)
Sinks (Sorted From Highest to Lowest)
0
2
4
6
8
10
12
0
5
10
15
20
25
30
35
40

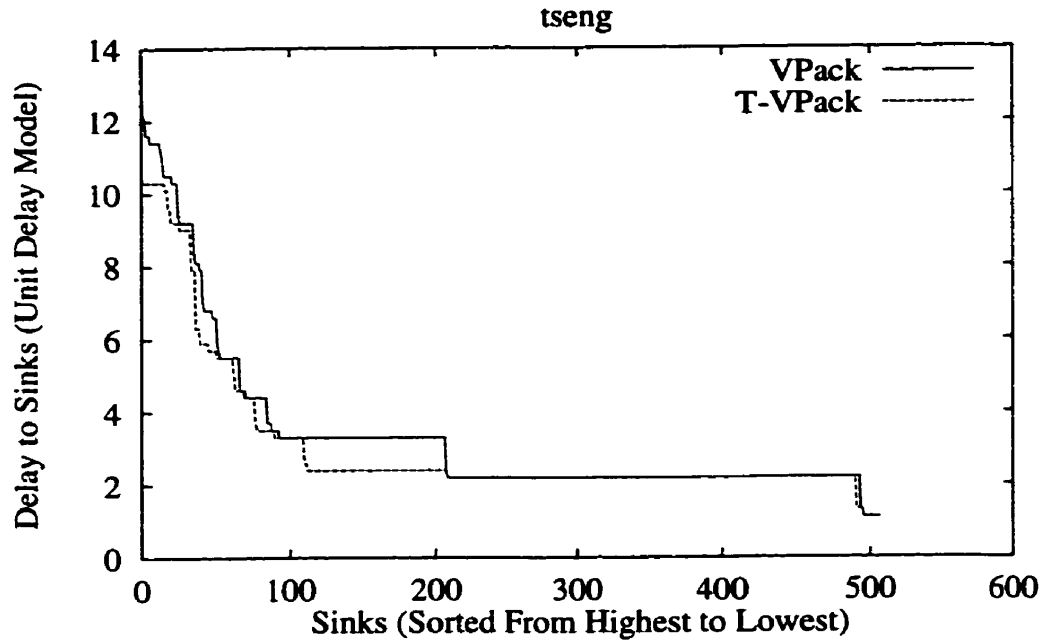

s298
VPack
T-VPack
Delay to Sinks (Unit Delay Model)
Sinks (Sorted From Highest to Lowest)
0
2
4
6
8
10
12
14
16
0
2
4
6
8
10
12
14


s38417
VPack
T-VPack
Delay to Sinks (Unit Delay Model)
Sinks (Sorted From Highest to Lowest)
0
2
4
6
8
10
12
14
0
200
400
600
800
1000
1200
1400
1600


s38584.1
VPack
T-VPack
Delay to Sinks (Unit Delay Model)
Sinks (Sorted From Highest to Lowest)
0
2
4
6
8
10
12
0
200
400
600
800
1000
1200
1400
1600


seq
VPack
T-VPack
Delay to Sinks (Unit Delay Model)
Sinks (Sorted From Highest to Lowest)
0
1
2
3
4
5
6
7
8
9
0
5
10
15
20
25
30
35


spla
VPack
T-VPack
Delay to Sinks (Unit Delay Model)
Sinks (Sorted From Highest to Lowest)
0
1
2
3
4
5
6
7
8
9
10
0
5
10
15
20
25
30
35
40
45

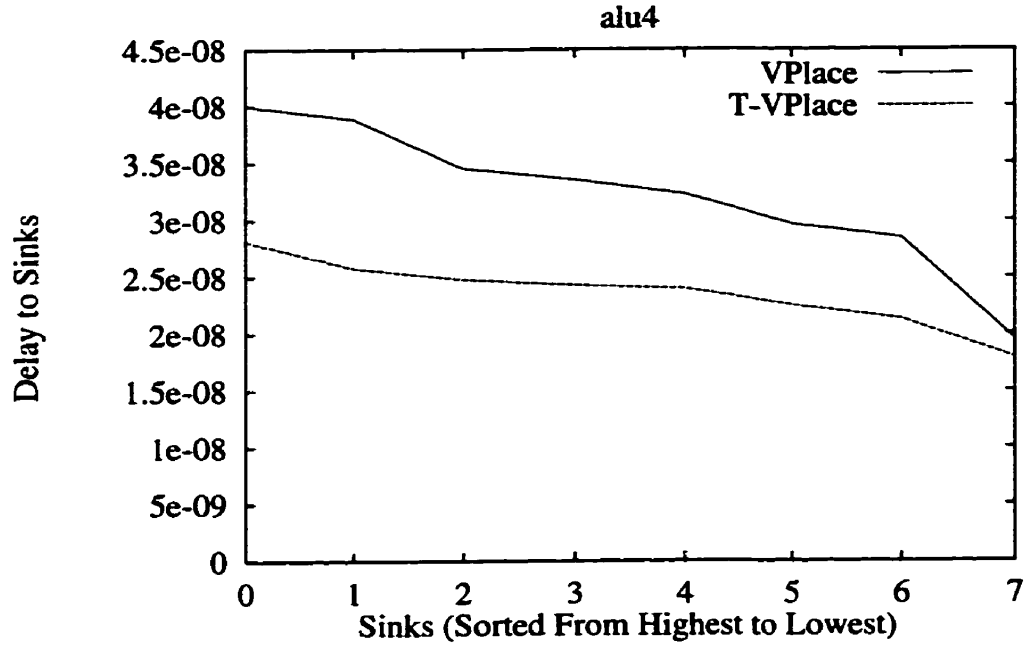

tseng
VPack
T-VPack
Delay to Sinks (Unit Delay Model)
Sinks (Sorted From Highest to Lowest)
0
2
4
6
8
10
12
14
0
100
200
300
400
500
600

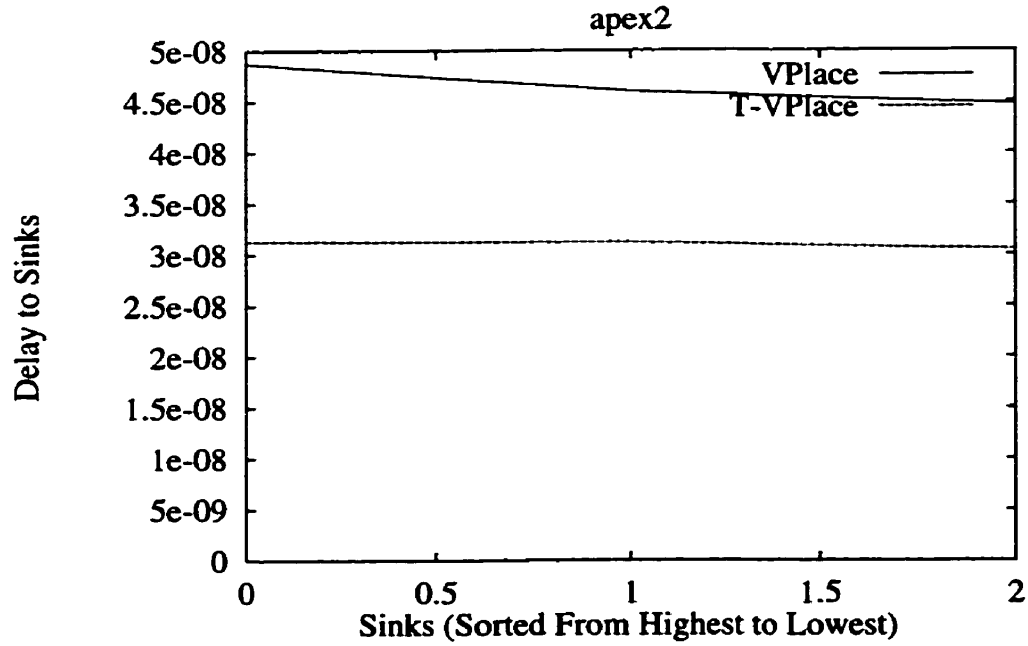
APPENDIX C *Sink Delay Distributions for the 20 MCNC Benchmark Circuits*

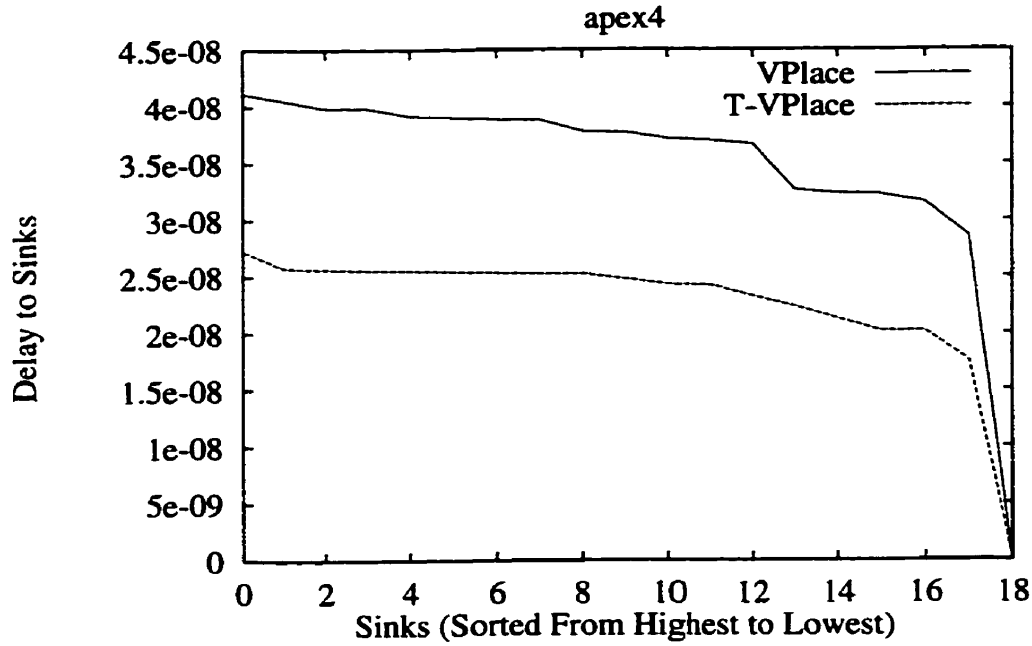
In this appendix we present sink delay distributions for the 20 largest MCNC benchmark circuits. Section C.1 presents the post-placement estimated sink delay distributions for the 20 largest MCNC circuits implemented in an architecture with size 1 clusters. Then Section C.2 present low-stress post-place-and-route results for the same circuits in the same architecture. After this, Sections C.3 and C.4 present the placement-estimated and low-stress sink delay distributions for the circuits implemented in an architecture with size 8 clusters.

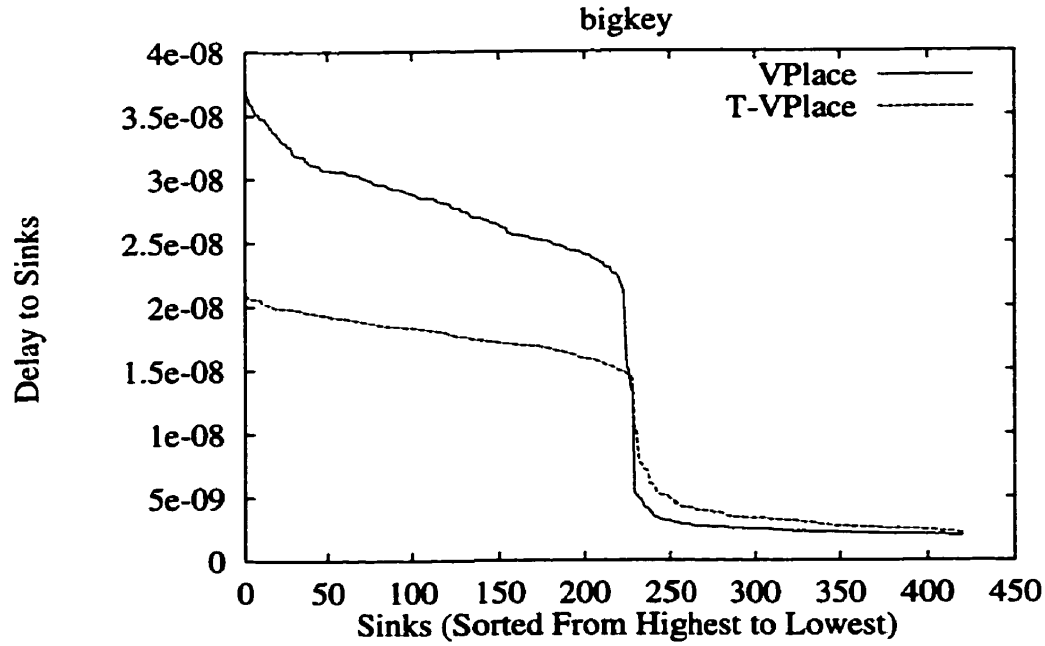
C.1 Placement Estimated Sink Delay Distributions: Size 1 Clusters

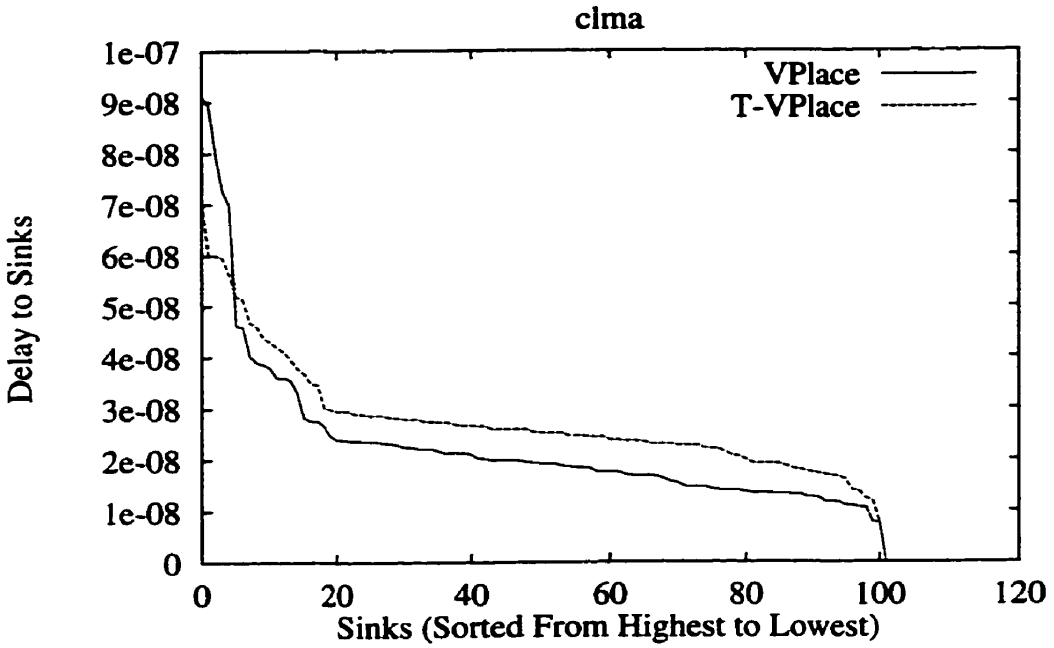
In this section we present the post-placement sink delay distributions for the 20 largest MCNC benchmark circuits using size 1 clusters. The delays that we present are placement-estimated delays as we discussed in Section 5.2.2. For T-VPlace, we set the adaptive Criticality_Exponent to 8, and λ to 0.5.

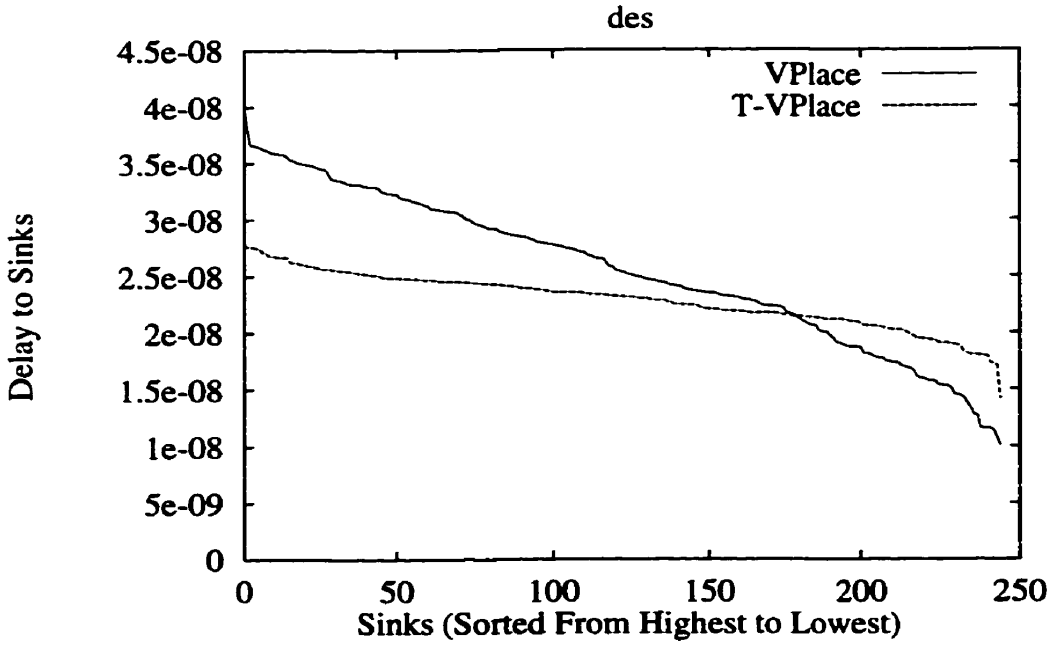

alu4
VPlace
T-VPlace
Delay to Sinks
4.5e-08
4e-08
3.5e-08
3e-08
2.5e-08
2e-08
1.5e-08
1e-08
5e-09
0
0
1
2
3
4
5
6
7
Sinks (Sorted From Highest to Lowest)

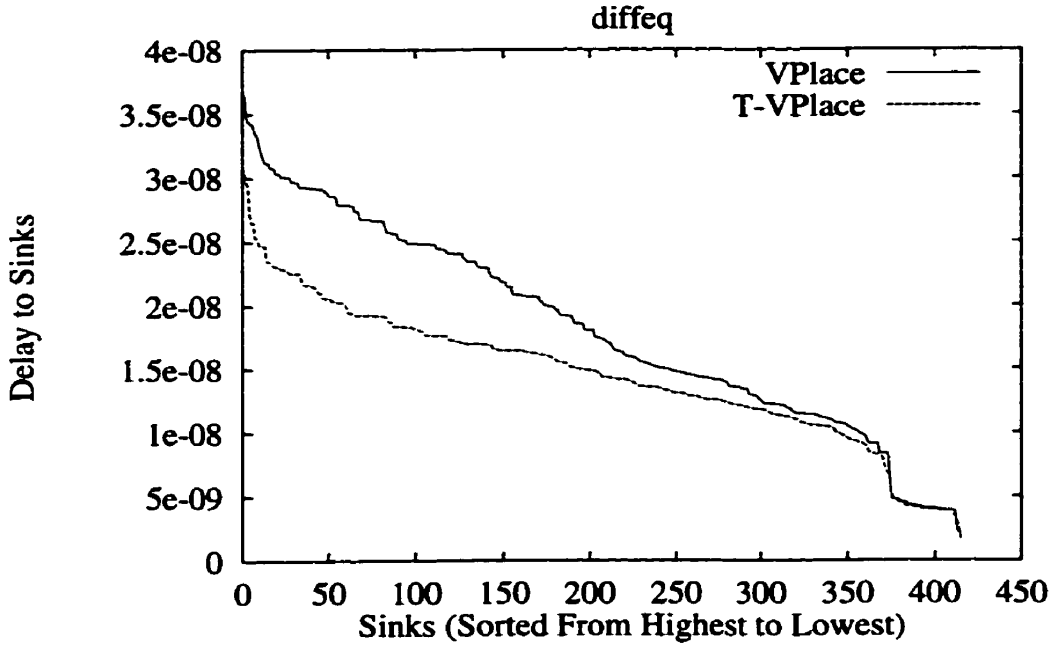

apex2
VPlace
T-VPlace
Delay to Sinks
5e-08
4.5e-08
4e-08
3.5e-08
3e-08
2.5e-08
2e-08
1.5e-08
1e-08
5e-09
0
0
0.5
1
1.5
2
Sinks (Sorted From Highest to Lowest)

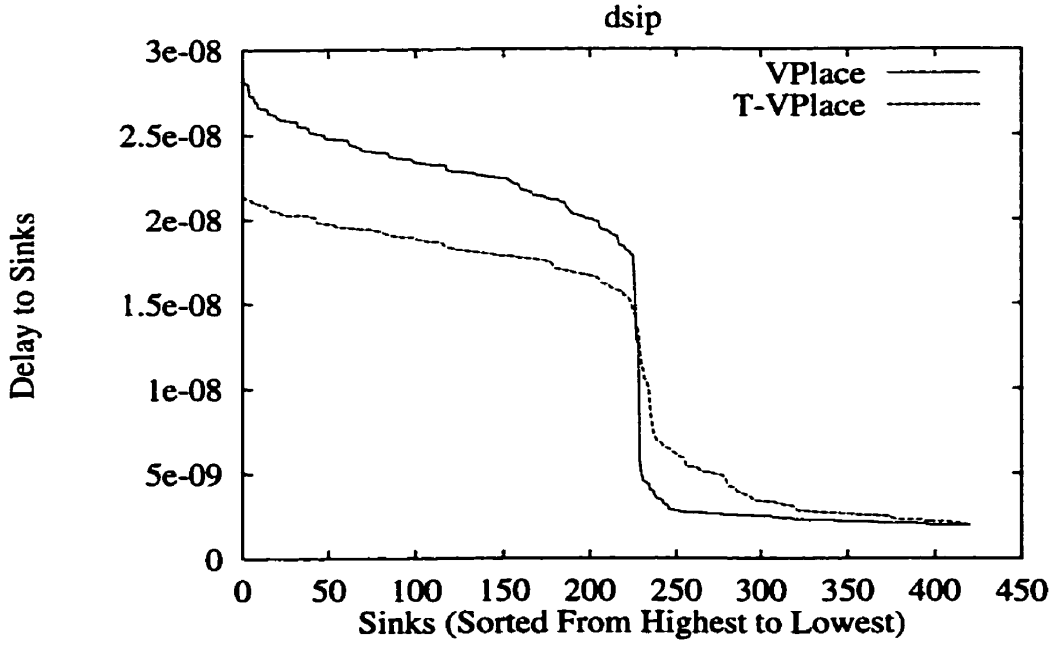

apex4
VPlace
T-VPlace
Delay to Sinks
4.5e-08
4e-08
3.5e-08
3e-08
2.5e-08
2e-08
1.5e-08
1e-08
5e-09
0
0 2 4 6 8 10 12 14 16 18
Sinks (Sorted From Highest to Lowest)

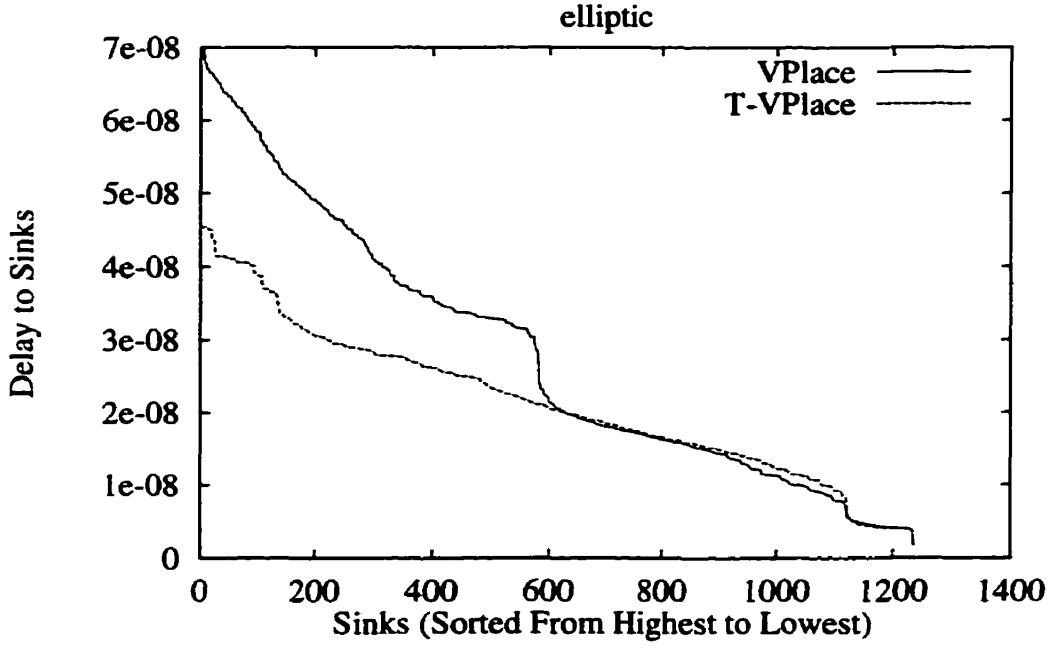

bigkey
VPlace
T-VPlace
Delay to Sinks
4e-08
3.5e-08
3e-08
2.5e-08
2e-08
1.5e-08
1e-08
5e-09
0
0 50 100 150 200 250 300 350 400 450
Sinks (Sorted From Highest to Lowest)

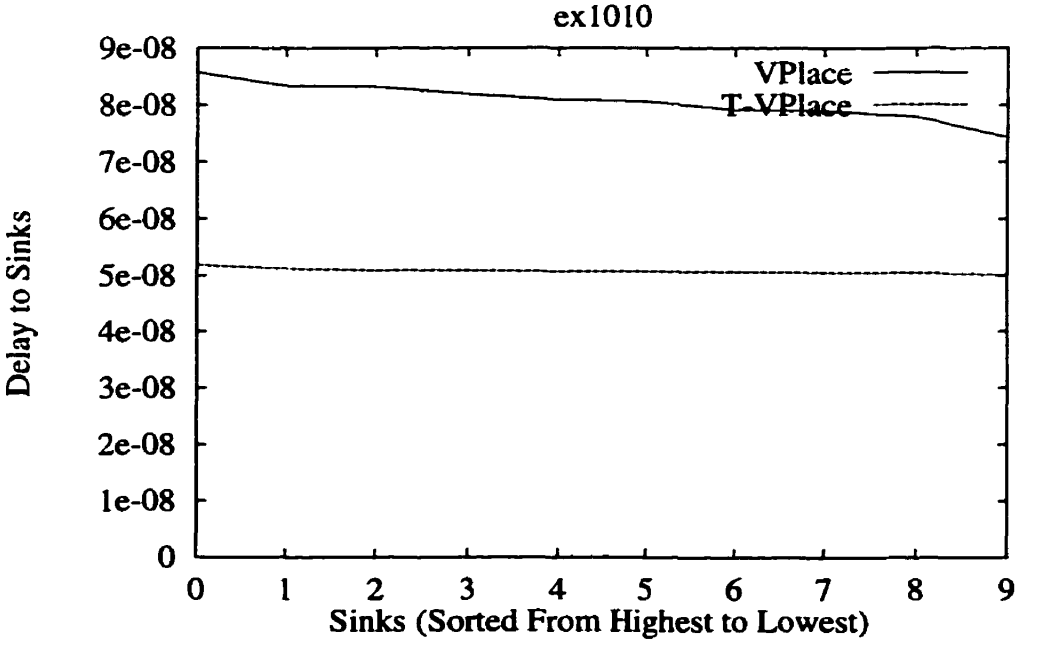

clma
VPlace
T-VPlace
Delay to Sinks
1e-07
9e-08
8e-08
7e-08
6e-08
5e-08
4e-08
3e-08
2e-08
1e-08
0
0
20
40
60
80
100
120
Sinks (Sorted From Highest to Lowest)

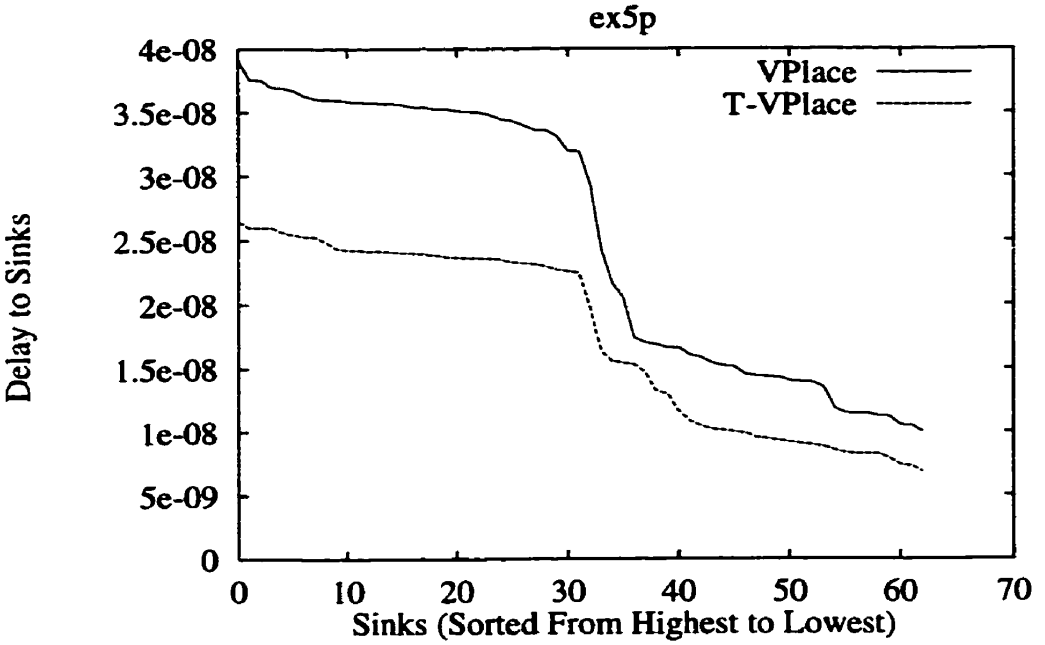

des
VPlace
T-VPlace
Delay to Sinks
4.5e-08
4e-08
3.5e-08
3e-08
2.5e-08
2e-08
1.5e-08
1e-08
5e-09
0
0
50
100
150
200
250
Sinks (Sorted From Highest to Lowest)

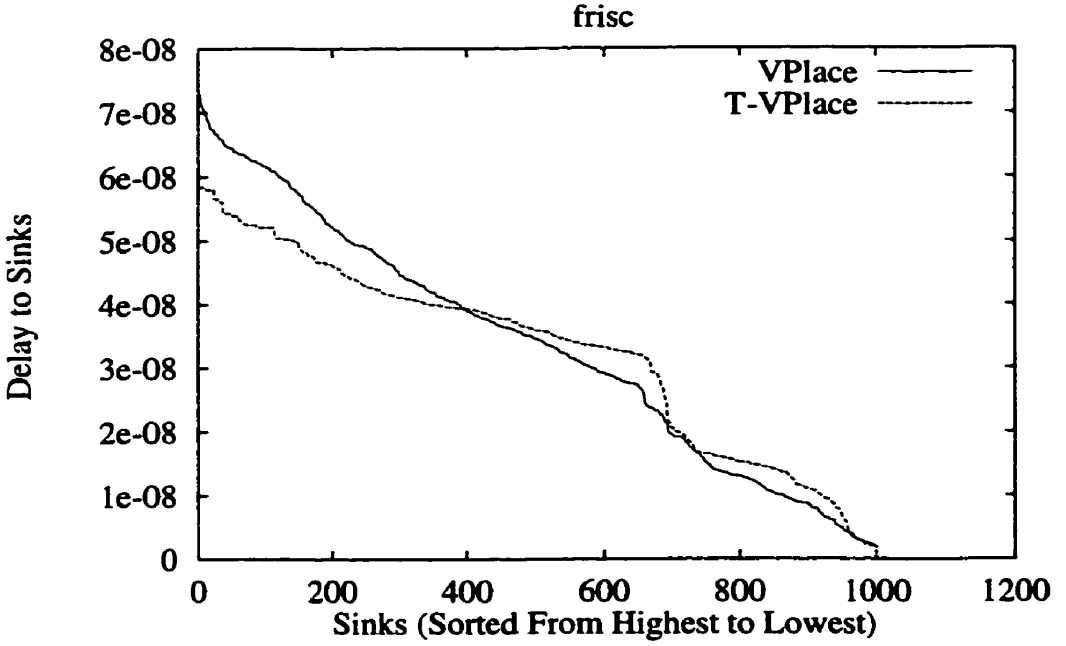

diffeq
VPlace
T-VPlace
Delay to Sinks
4e-08
3.5e-08
3e-08
2.5e-08
2e-08
1.5e-08
1e-08
5e-09
0
0
50
100
150
200
250
300
350
400
450
Sinks (Sorted From Highest to Lowest)

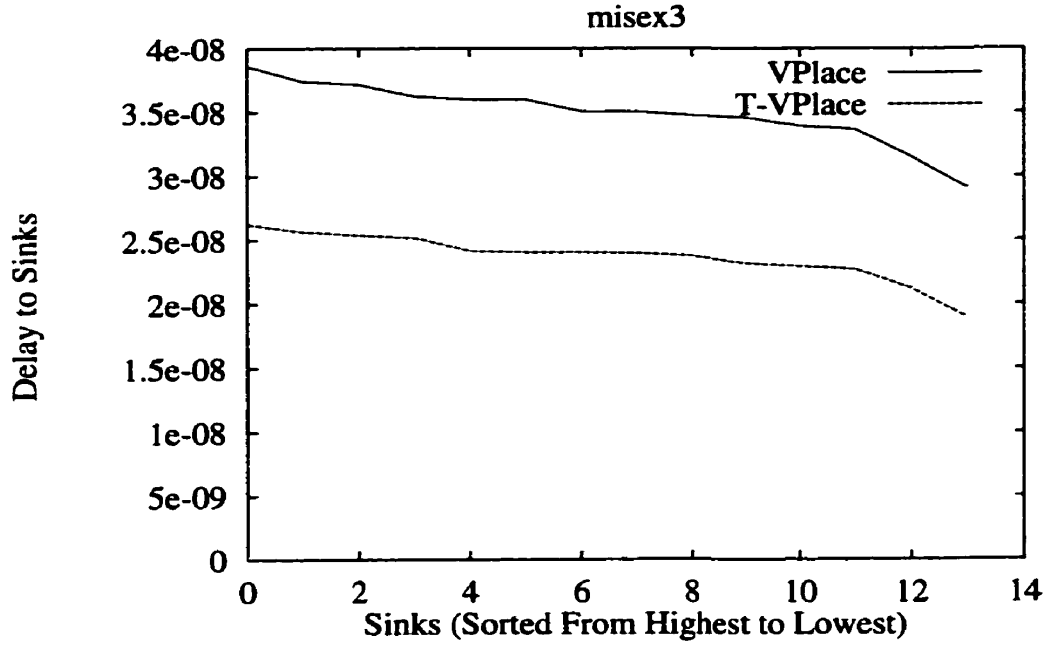

dsip
VPlace
T-VPlace
Delay to Sinks
3e-08
2.5e-08
2e-08
1.5e-08
1e-08
5e-09
0
0
50
100
150
200
250
300
350
400
450
Sinks (Sorted From Highest to Lowest)

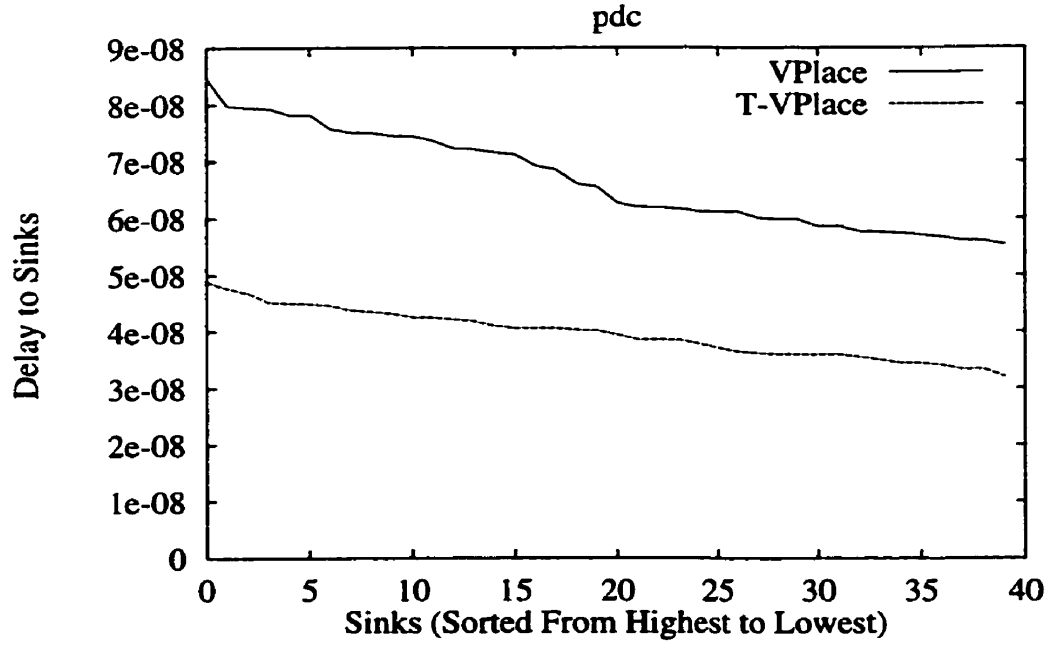

elliptic
VPlace
T-VPlace
Delay to Sinks
7e-08
6e-08
5e-08
4e-08
3e-08
2e-08
1e-08
0
0
200
400
600
800
1000
1200
1400
Sinks (Sorted From Highest to Lowest)

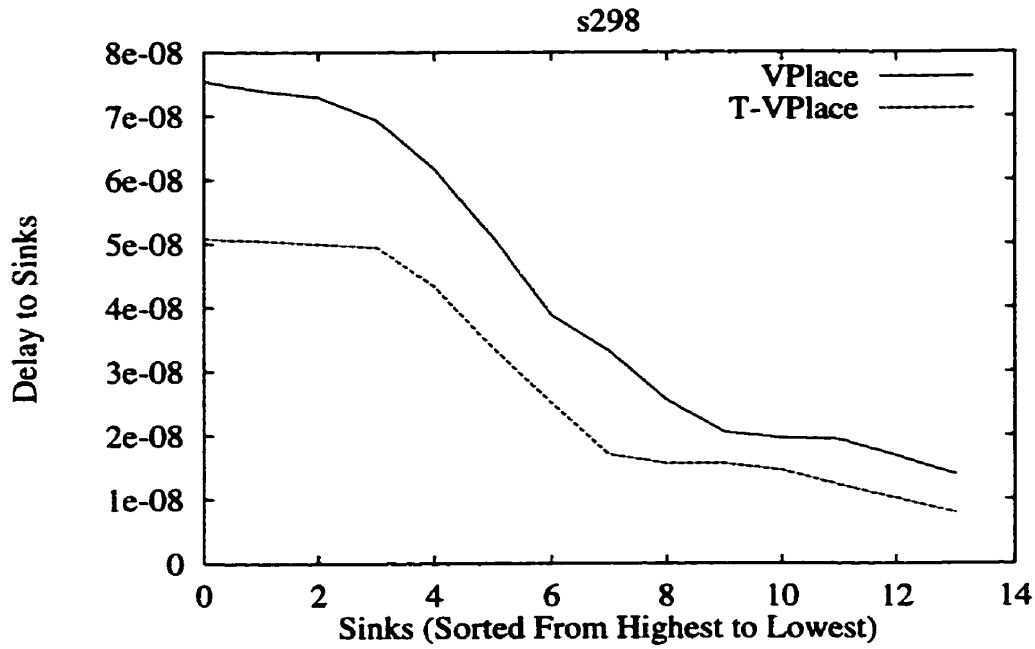

ex1010
VPlace
T-VPlace
Delay to Sinks
9e-08
8e-08
7e-08
6e-08
5e-08
4e-08
3e-08
2e-08
1e-08
0
0
1
2
3
4
5
6
7
8
9
Sinks (Sorted From Highest to Lowest)

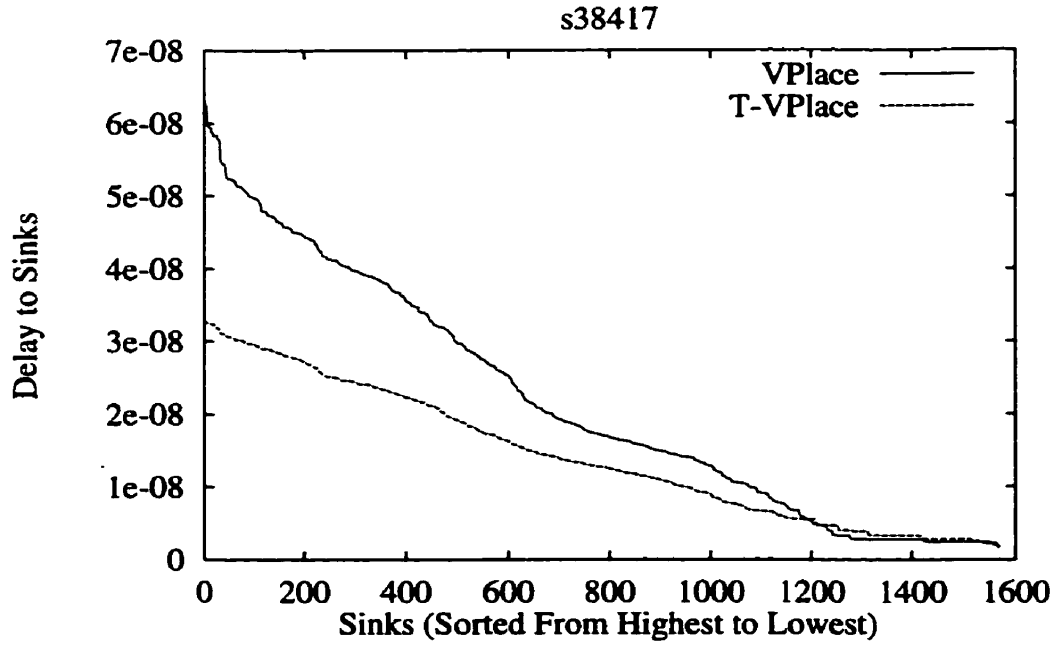

ex5p
VPlace
T-VPlace
Delay to Sinks
4e-08
3.5e-08
3e-08
2.5e-08
2e-08
1.5e-08
1e-08
5e-09
0
0
10
20
30
40
50
60
70
Sinks (Sorted From Highest to Lowest)

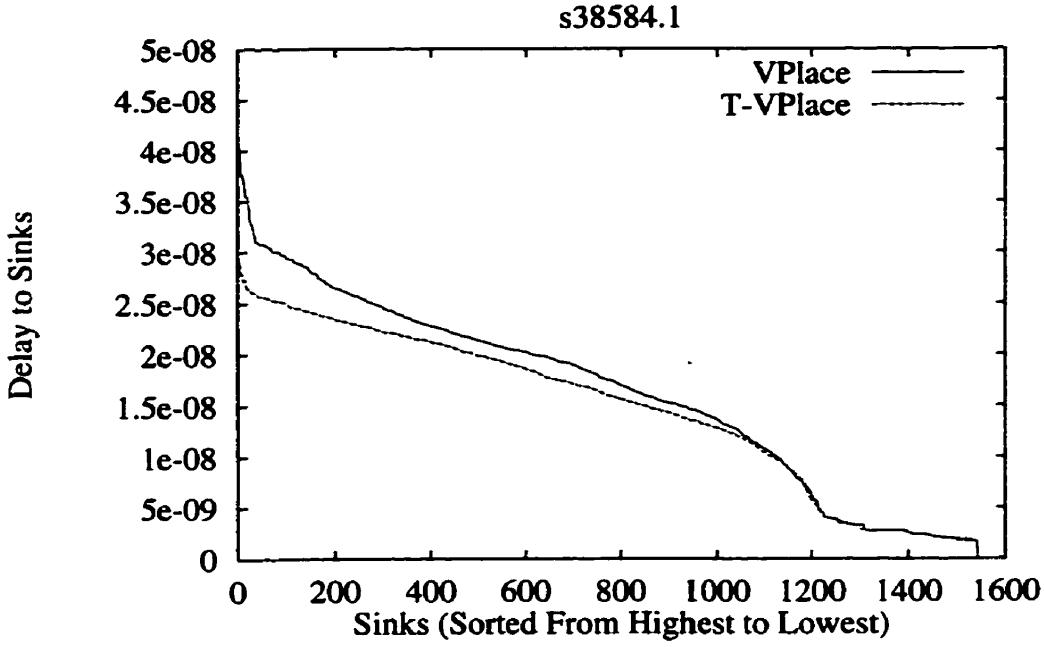

frisc
VPlace
T-VPlace
Delay to Sinks
8e-08
7e-08
6e-08
5e-08
4e-08
3e-08
2e-08
1e-08
0
0
200
400
600
800
1000
1200
Sinks (Sorted From Highest to Lowest)

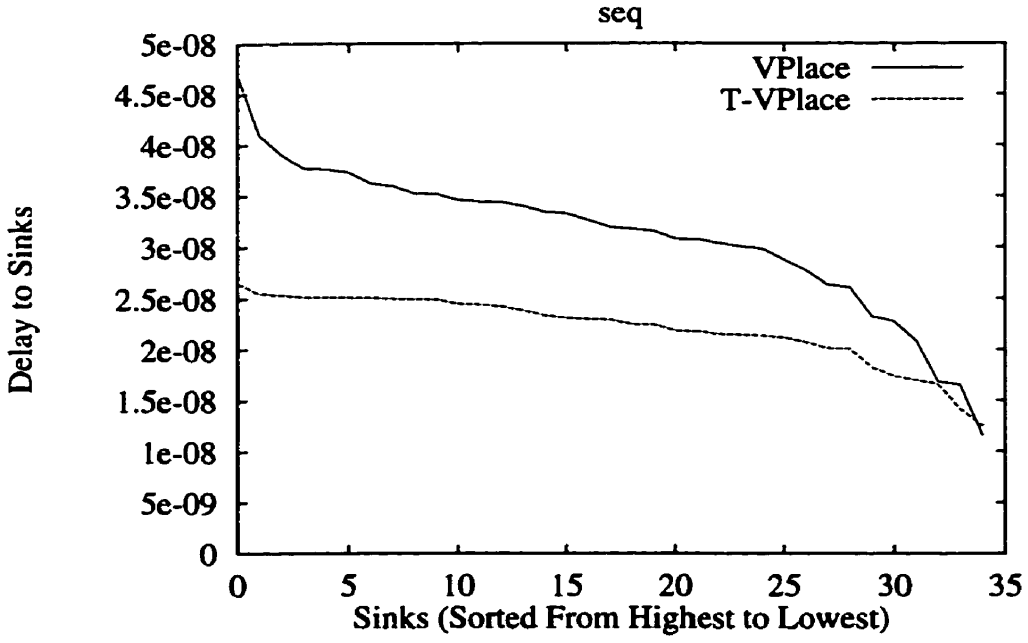

misex3
VPlace
T-VPlace
Delay to Sinks
4e-08
3.5e-08
3e-08
2.5e-08
2e-08
1.5e-08
1e-08
5e-09
0
0
2
4
6
8
10
12
14
Sinks (Sorted From Highest to Lowest)

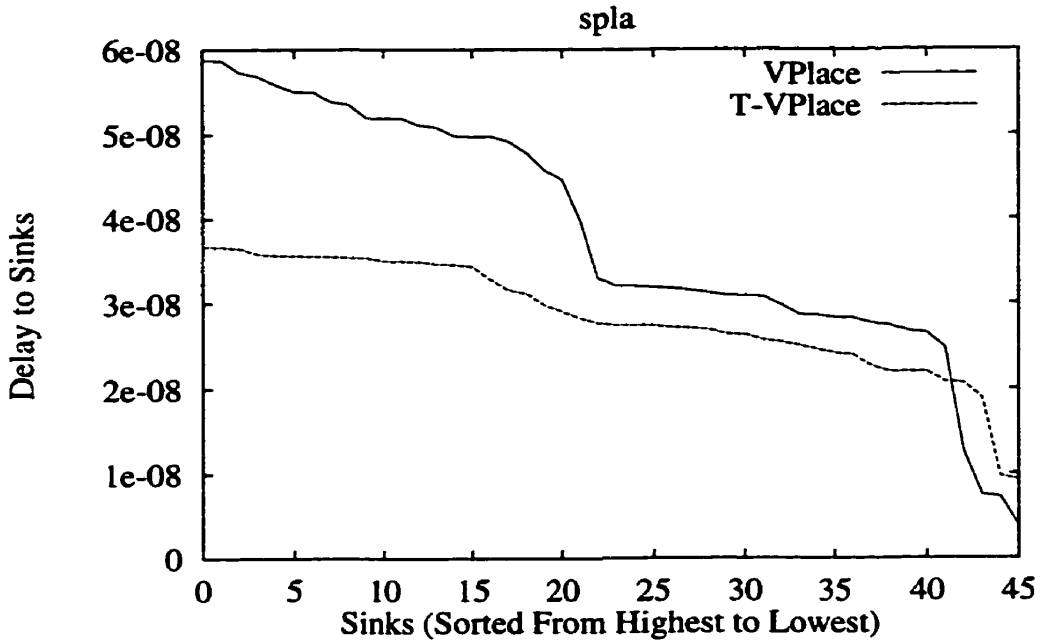

pdc
VPlace
T-VPlace
Delay to Sinks
9e-08
8e-08
7e-08
6e-08
5e-08
4e-08
3e-08
2e-08
1e-08
0
0
5
10
15
20
25
30
35
40
Sinks (Sorted From Highest to Lowest)

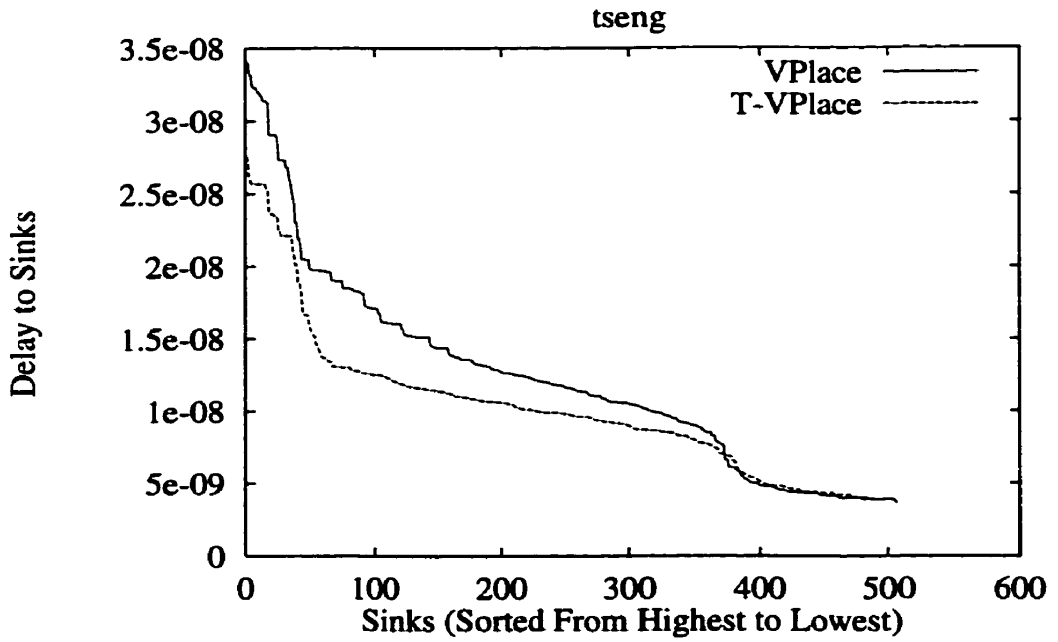

s298
VPlace
T-VPlace
Delay to Sinks
8e-08
7e-08
6e-08
5e-08
4e-08
3e-08
2e-08
1e-08
0
0
2
4
6
8
10
12
14
Sinks (Sorted From Highest to Lowest)


s38417
VPlace
T-VPlace
Delay to Sinks
7e-08
6e-08
5e-08
4e-08
3e-08
2e-08
1e-08
0
0
200
400
600
800
1000
1200
1400
1600
Sinks (Sorted From Highest to Lowest)


s38584.1
VPlace
T-VPlace
Delay to Sinks
5e-08
4.5e-08
4e-08
3.5e-08
3e-08
2.5e-08
2e-08
1.5e-08
1e-08
5e-09
0
0
200
400
600
800
1000
1200
1400
1600
Sinks (Sorted From Highest to Lowest)

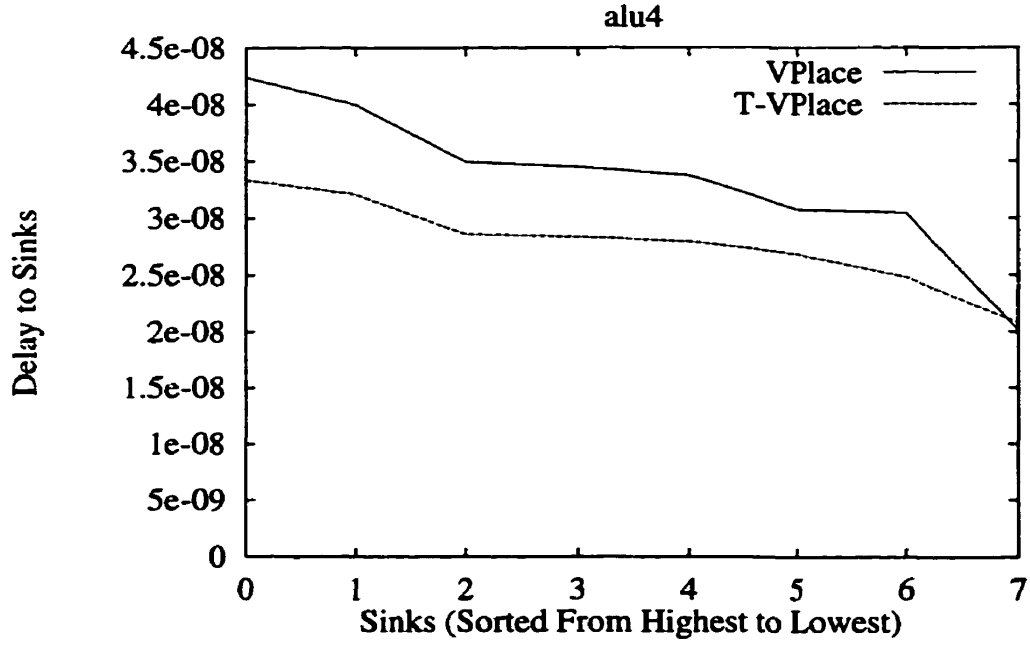

seq
VPlace
T-VPlace
Delay to Sinks
5e-08
4.5e-08
4e-08
3.5e-08
3e-08
2.5e-08
2e-08
1.5e-08
1e-08
5e-09
0
0
5
10
15
20
25
30
35
Sinks (Sorted From Highest to Lowest)

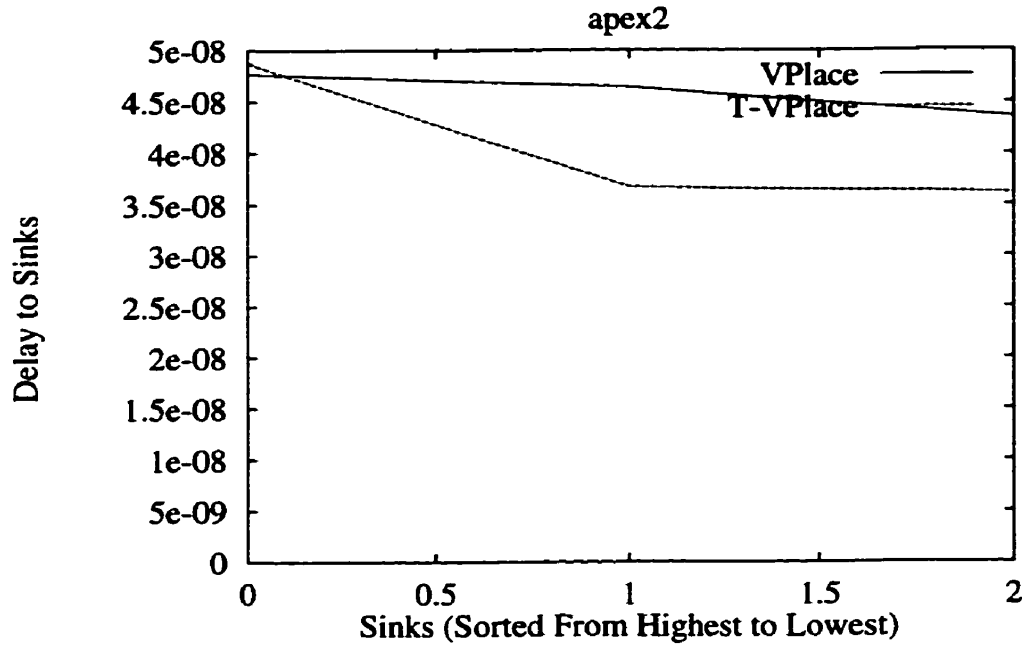

spla
VPlace
T-VPlace
Delay to Sinks
6e-08
5e-08
4e-08
3e-08
2e-08
1e-08
0
0
5
10
15
20
25
30
35
40
45
Sinks (Sorted From Highest to Lowest)

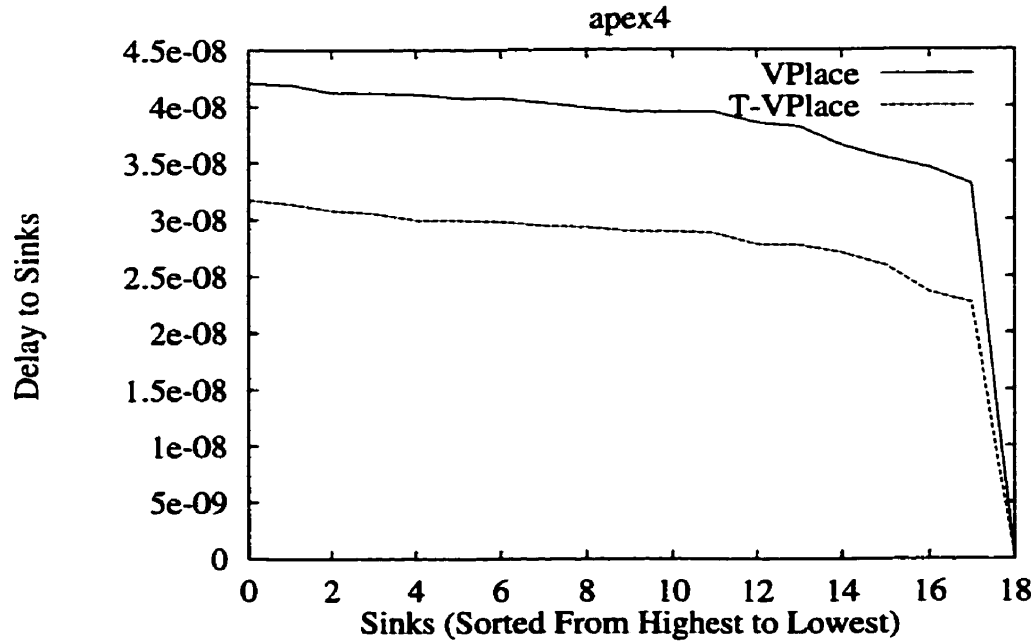

tseng
VPlace
T-VPlace
Delay to Sinks
3.5e-08
3e-08
2.5e-08
2e-08
1.5e-08
1e-08
5e-09
0
0
100
200
300
400
500
600
Sinks (Sorted From Highest to Lowest)

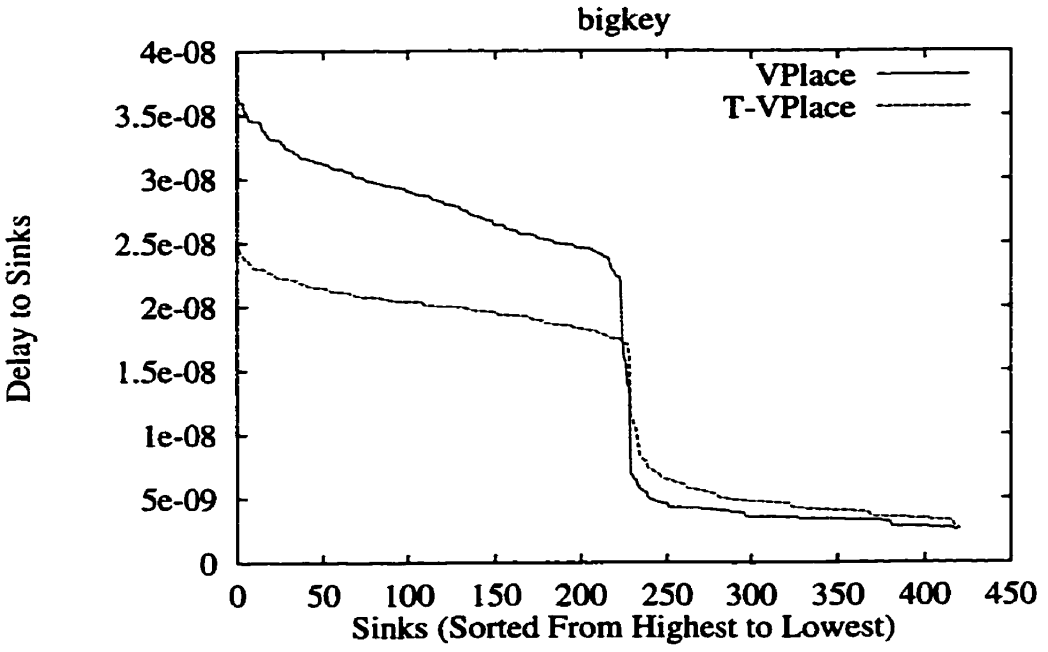
C.2 Low-Stress Sink Delay Distributions: Size 1 Clusters

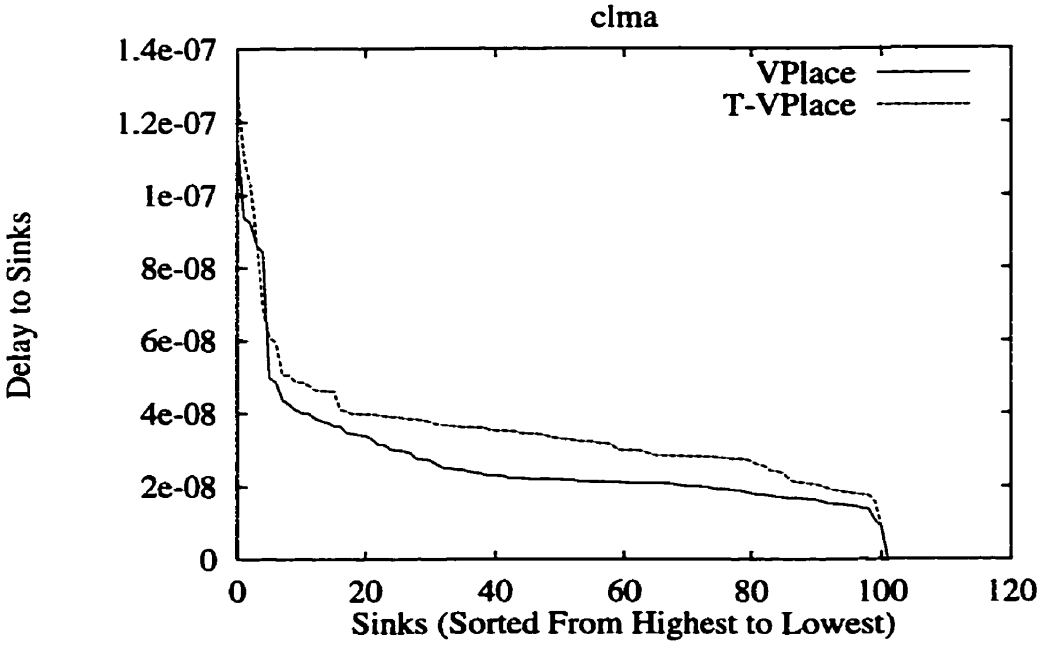
In this section we present the post place-and-route sink delay distributions for the 20 largest MCNC benchmark circuits using size 1 clusters. The delays that we present are low-stress which we defined in Section 3.1. For T-VPlace, we set the adaptive Criticality_Exponent to 8, and λ to 0.5.

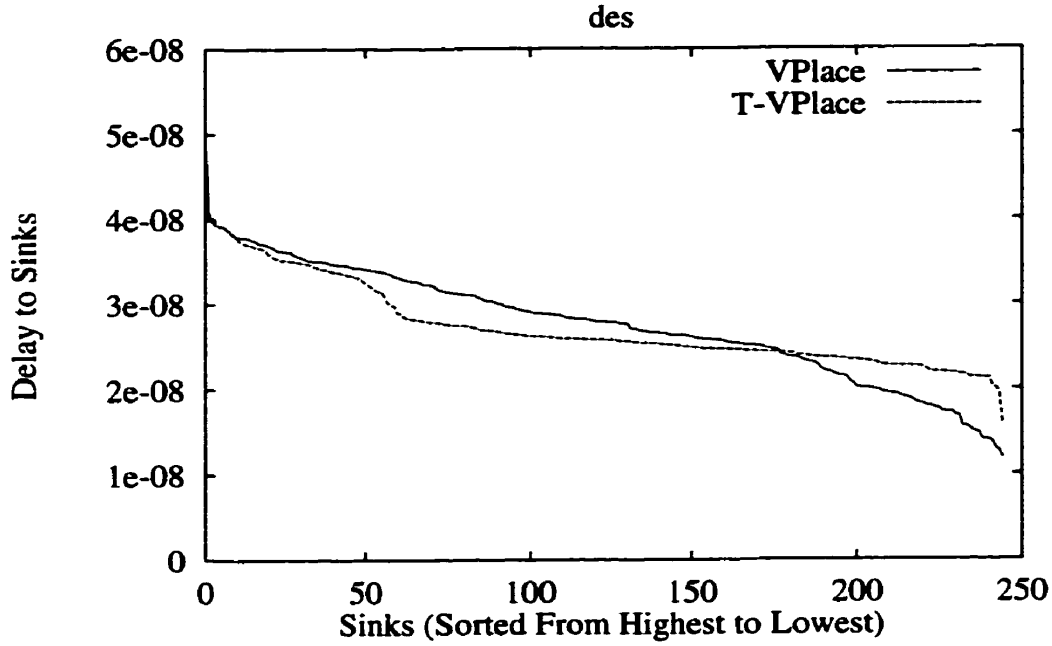


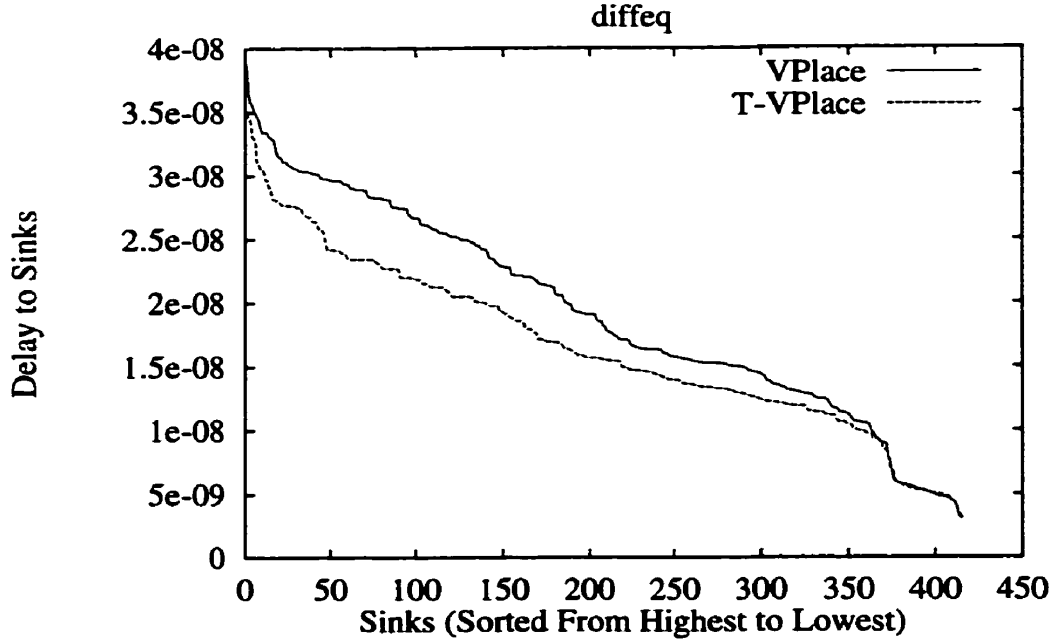

apex2
VPlace
T-VPlace
Delay to Sinks
5e-08
4.5e-08
4e-08
3.5e-08
3e-08
2.5e-08
2e-08
1.5e-08
1e-08
5e-09
0
0
0.5
1
1.5
2
Sinks (Sorted From Highest to Lowest)

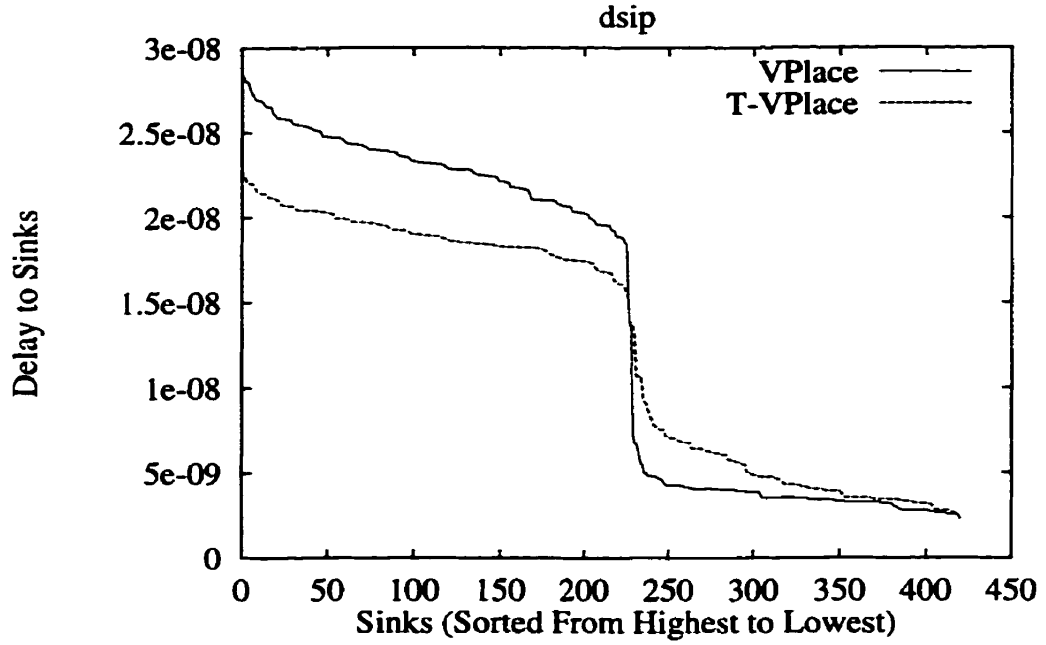

apex4
VPlace
T-VPlace
Delay to Sinks
4.5e-08
4e-08
3.5e-08
3e-08
2.5e-08
2e-08
1.5e-08
1e-08
5e-09
0
0
2
4
6
8
10
12
14
16
18
Sinks (Sorted From Highest to Lowest)

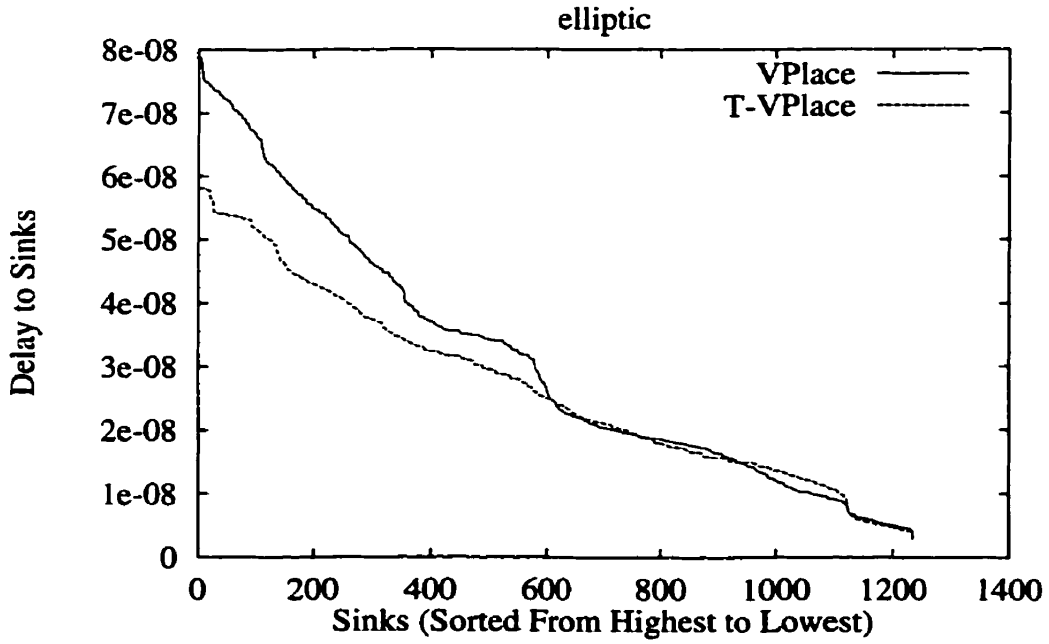

bigkey
VPlace
T-VPlace
Delay to Sinks
4e-08
3.5e-08
3e-08
2.5e-08
2e-08
1.5e-08
1e-08
5e-09
0
0
50
100
150
200
250
300
350
400
450
Sinks (Sorted From Highest to Lowest)

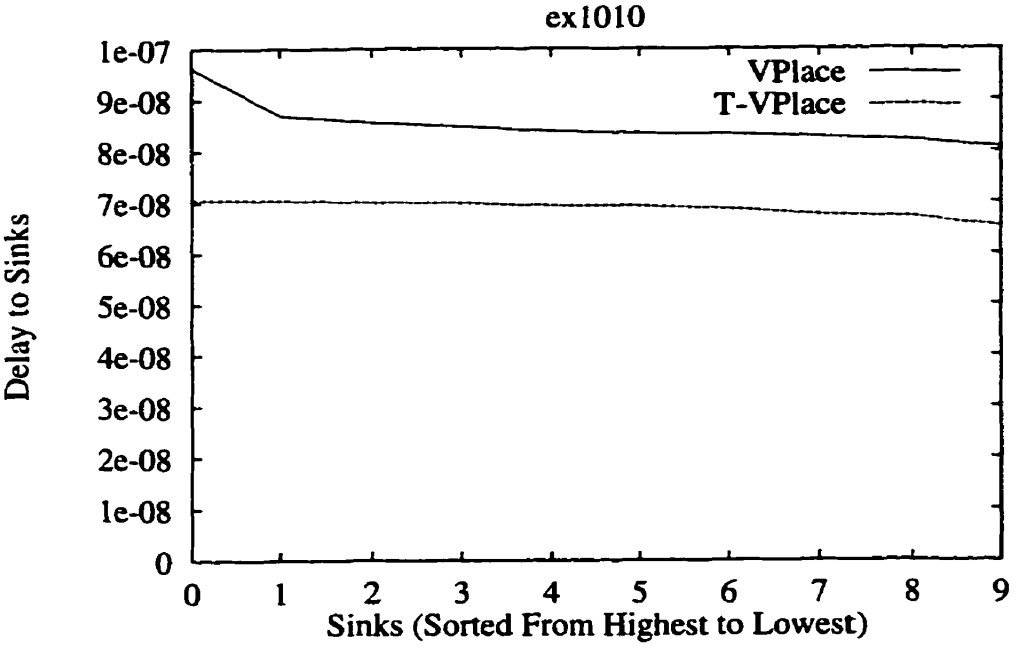

clma
VPlace
T-VPlace
Delay to Sinks
1.4e-07
1.2e-07
1e-07
8e-08
6e-08
4e-08
2e-08
0
0
20
40
60
80
100
120
Sinks (Sorted From Highest to Lowest)

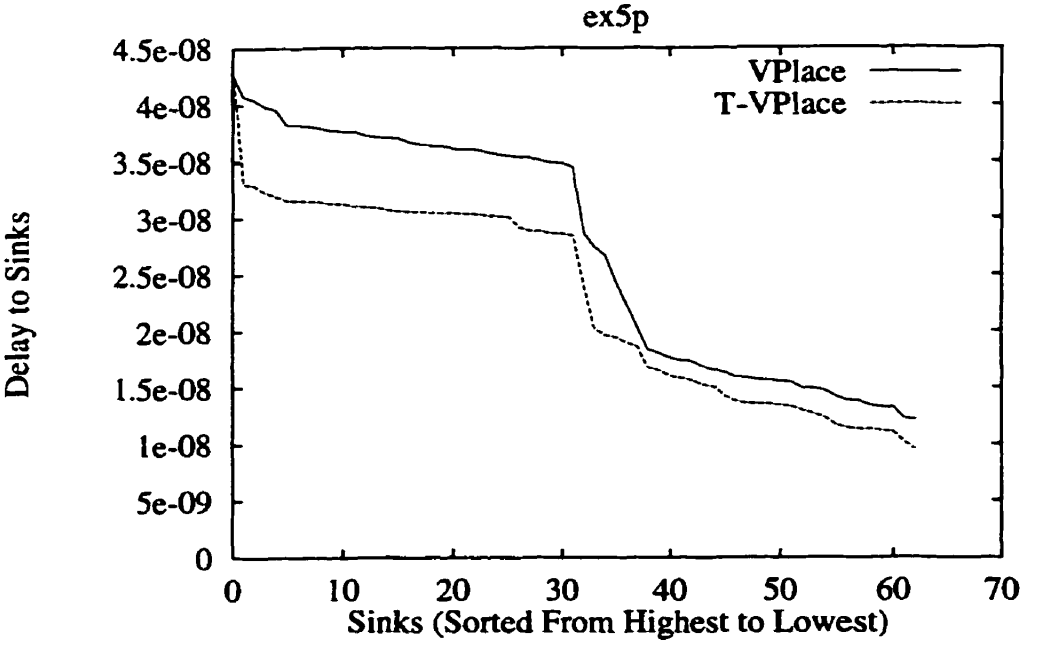

des
VPlace
T-VPlace
Delay to Sinks
6e-08
5e-08
4e-08
3e-08
2e-08
1e-08
0
0
50
100
150
200
250
Sinks (Sorted From Highest to Lowest)

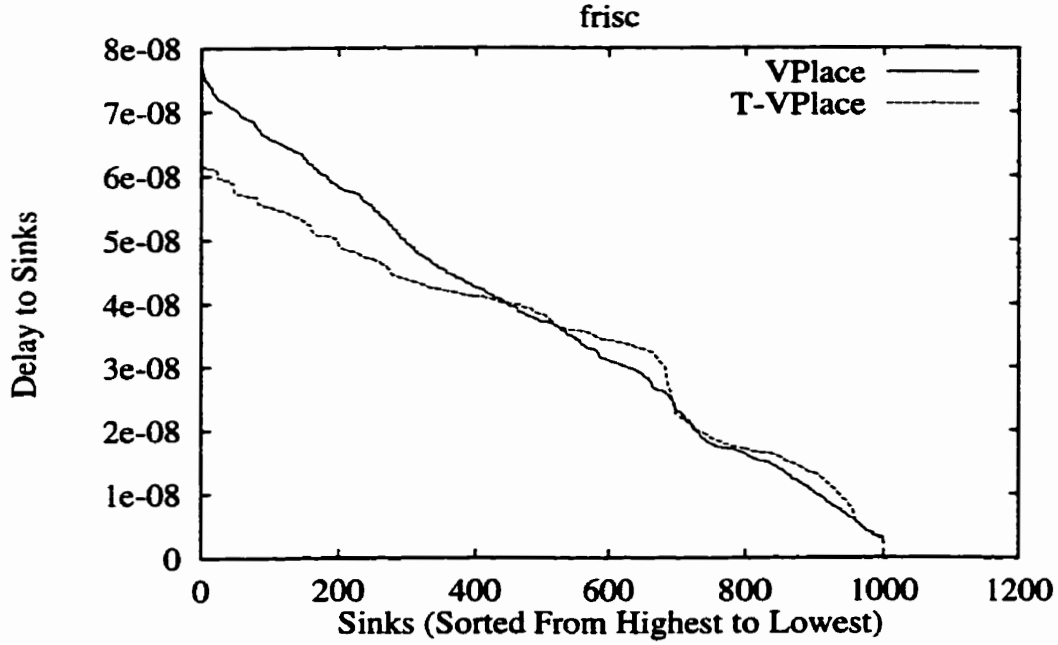

diffeq
VPlace
T-VPlace
Delay to Sinks
4e-08
3.5e-08
3e-08
2.5e-08
2e-08
1.5e-08
1e-08
5e-09
0
0
50
100
150
200
250
300
350
400
450
Sinks (Sorted From Highest to Lowest)

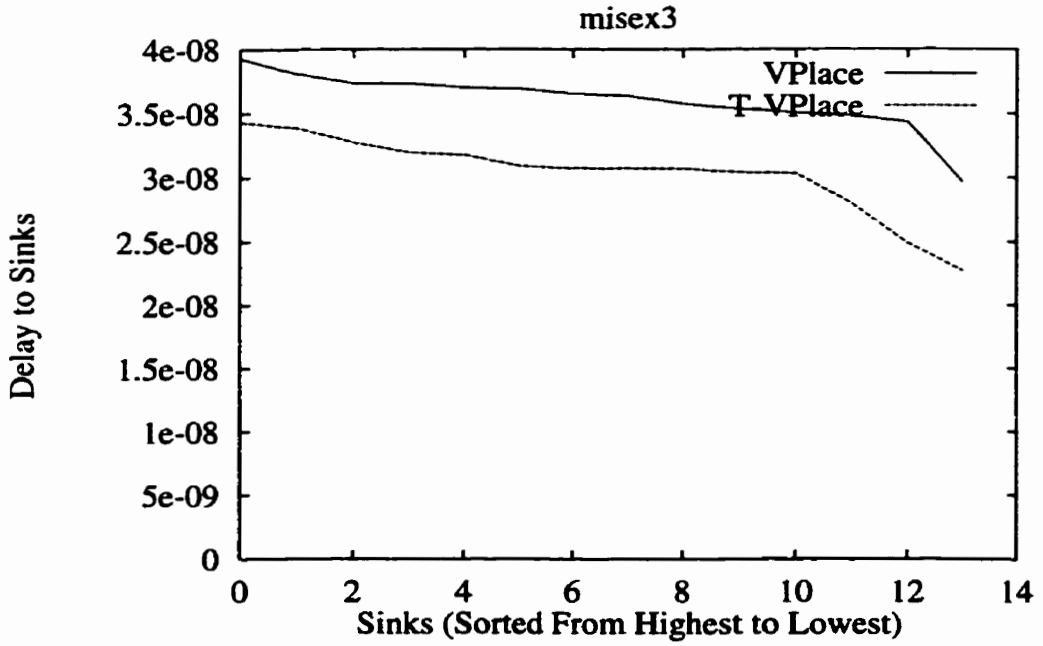

dsip
VPlace
T-VPlace
Delay to Sinks
3e-08
2.5e-08
2e-08
1.5e-08
1e-08
5e-09
0
0
50
100
150
200
250
300
350
400
450
Sinks (Sorted From Highest to Lowest)

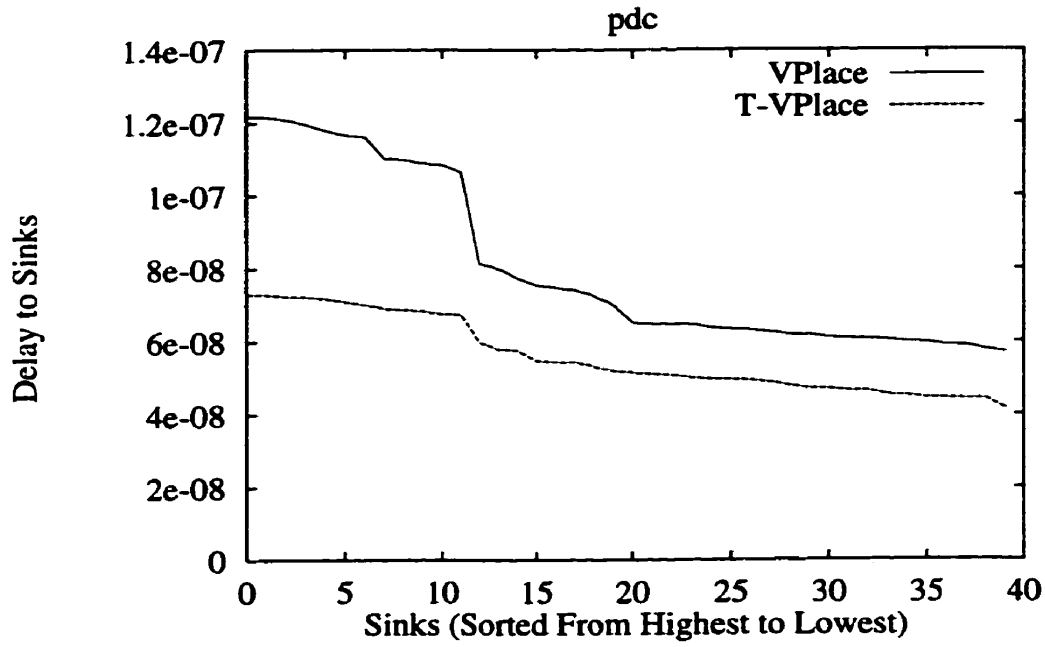

elliptic
VPlace
T-VPlace
Delay to Sinks
8e-08
7e-08
6e-08
5e-08
4e-08
3e-08
2e-08
1e-08
0
0
200
400
600
800
1000
1200
1400
Sinks (Sorted From Highest to Lowest)

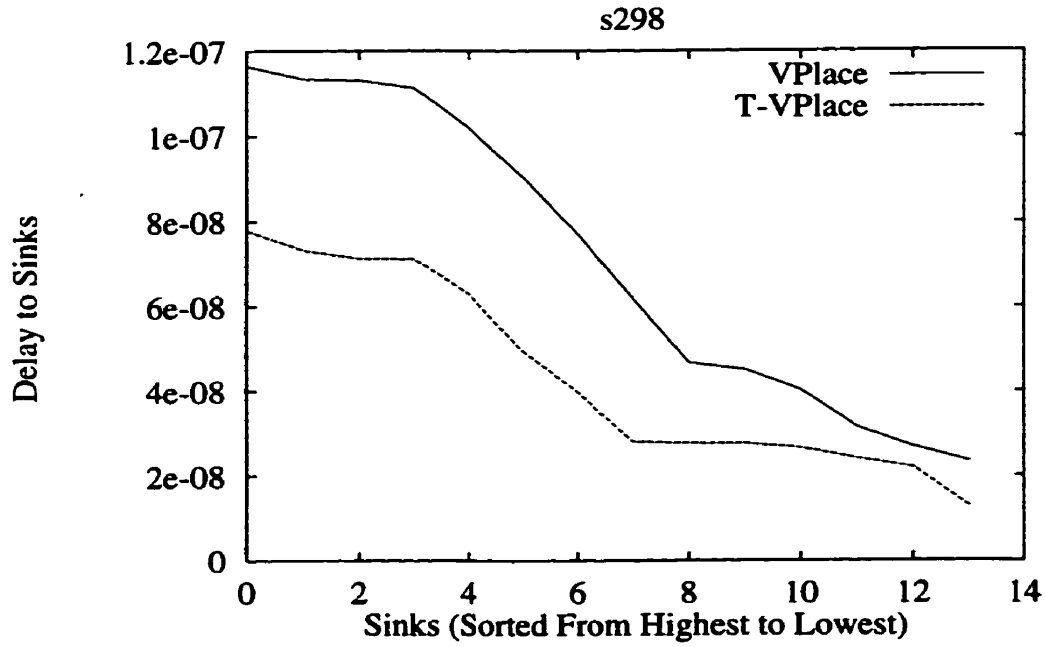

ex1010
VPlace
T-VPlace
Delay to Sinks
1e-07
9e-08
8e-08
7e-08
6e-08
5e-08
4e-08
3e-08
2e-08
1e-08
0
0 1 2 3 4 5 6 7 8 9
Sinks (Sorted From Highest to Lowest)

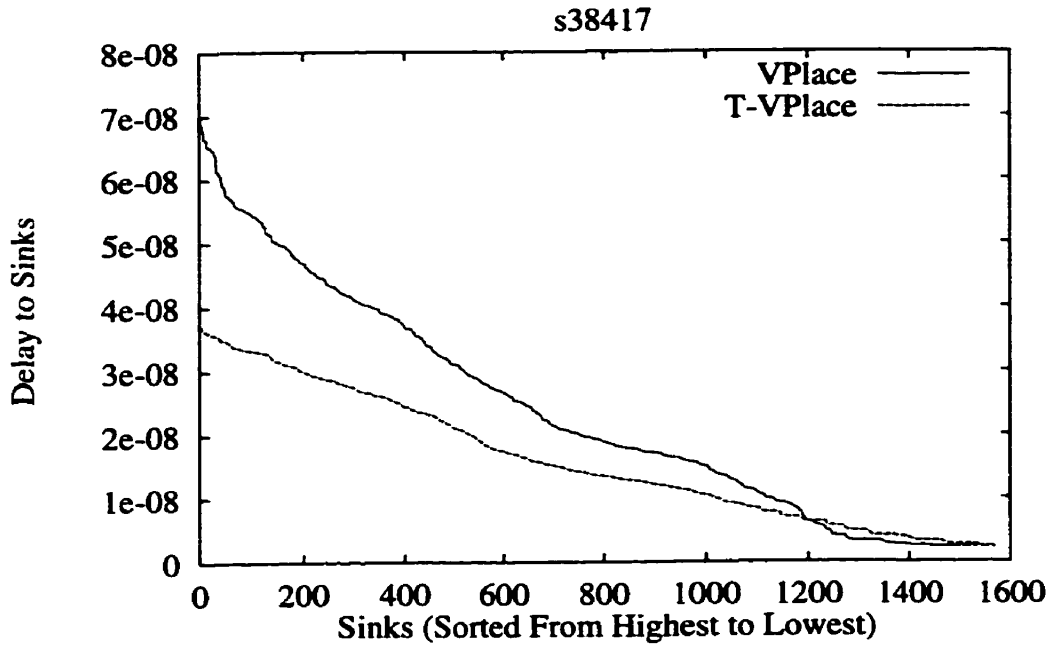

ex5p
VPlace
T-VPlace
Delay to Sinks
4.5e-08
4e-08
3.5e-08
3e-08
2.5e-08
2e-08
1.5e-08
1e-08
5e-09
0
0 10 20 30 40 50 60 70
Sinks (Sorted From Highest to Lowest)

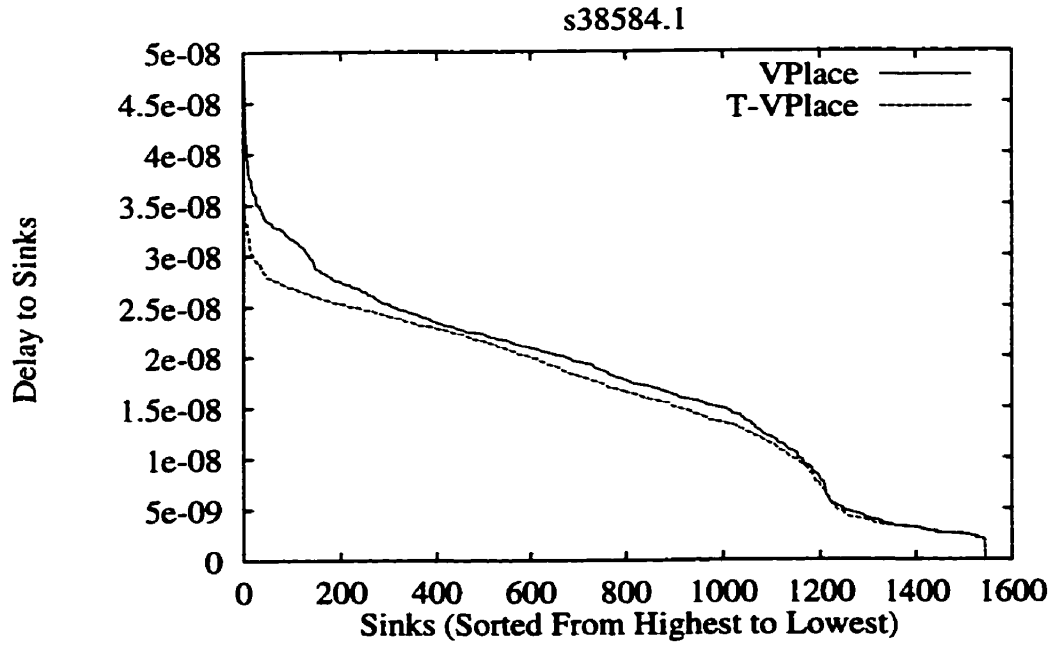

frisc
VPlace
T-VPlace
Delay to Sinks
8e-08
7e-08
6e-08
5e-08
4e-08
3e-08
2e-08
1e-08
0
0
200
400
600
800
1000
1200
Sinks (Sorted From Highest to Lowest)

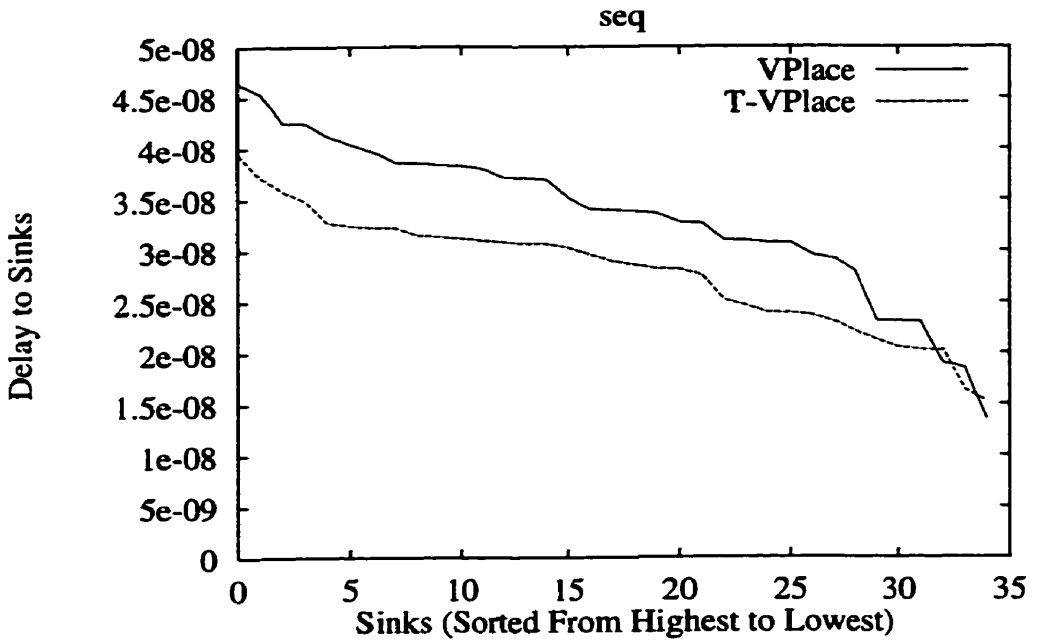

misex3
VPlace
T-VPlace
Delay to Sinks
4e-08
3.5e-08
3e-08
2.5e-08
2e-08
1.5e-08
1e-08
5e-09
0
0
2
4
6
8
10
12
14
Sinks (Sorted From Highest to Lowest)

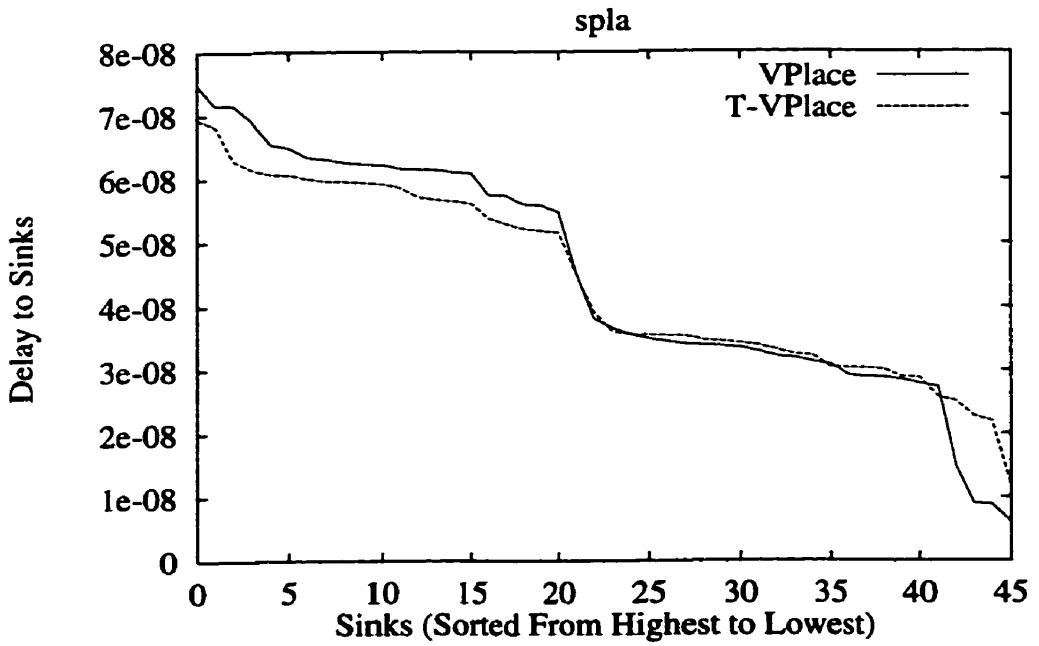

pdc
VPlace
T-VPlace
Delay to Sinks
1.4e-07
1.2e-07
1e-07
8e-08
6e-08
4e-08
2e-08
0
0
5
10
15
20
25
30
35
40
Sinks (Sorted From Highest to Lowest)

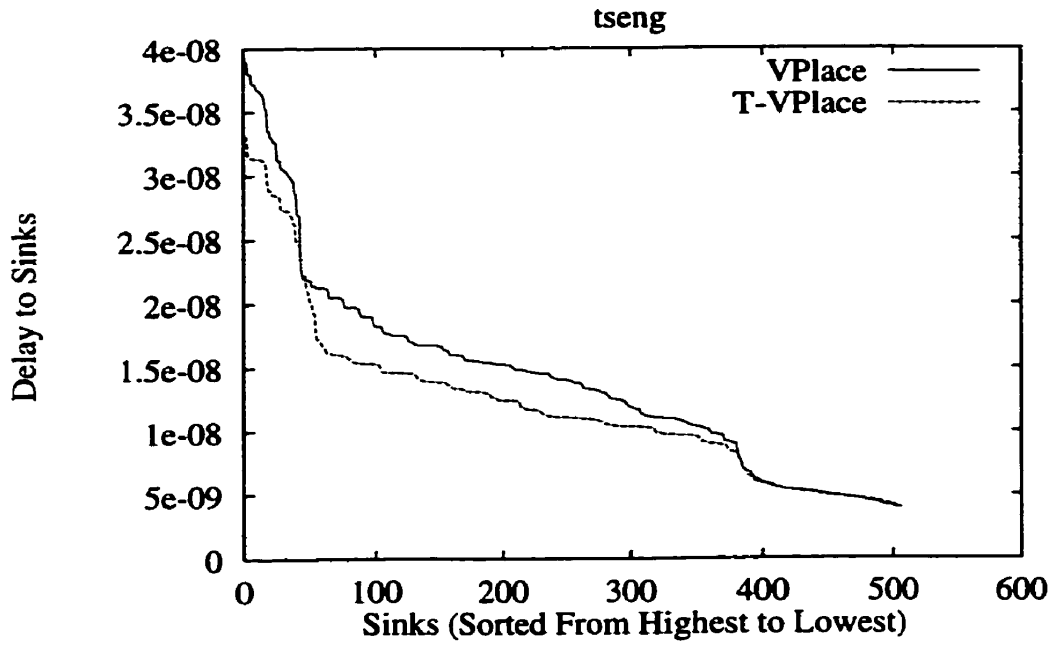

s298
VPlace
T-VPlace
Delay to Sinks
1.2e-07
1e-07
8e-08
6e-08
4e-08
2e-08
0
0
2
4
6
8
10
12
14
Sinks (Sorted From Highest to Lowest)


s38417
VPlace
T-VPlace
Delay to Sinks
8e-08
7e-08
6e-08
5e-08
4e-08
3e-08
2e-08
1e-08
0
0
200
400
600
800
1000
1200
1400
1600
Sinks (Sorted From Highest to Lowest)


s38584.1
VPlace
T-VPlace
Delay to Sinks
5e-08
4.5e-08
4e-08
3.5e-08
3e-08
2.5e-08
2e-08
1.5e-08
1e-08
5e-09
0
0
200
400
600
800
1000
1200
1400
1600
Sinks (Sorted From Highest to Lowest)

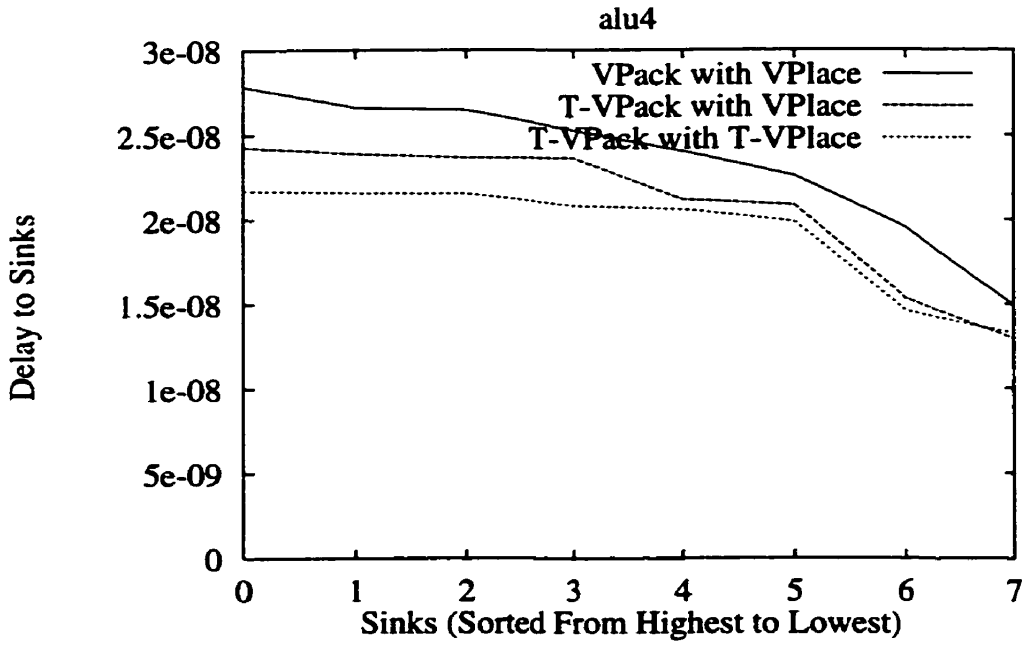

seq
VPlace
T-VPlace
Delay to Sinks
5e-08
4.5e-08
4e-08
3.5e-08
3e-08
2.5e-08
2e-08
1.5e-08
1e-08
5e-09
0
0
5
10
15
20
25
30
35
Sinks (Sorted From Highest to Lowest)

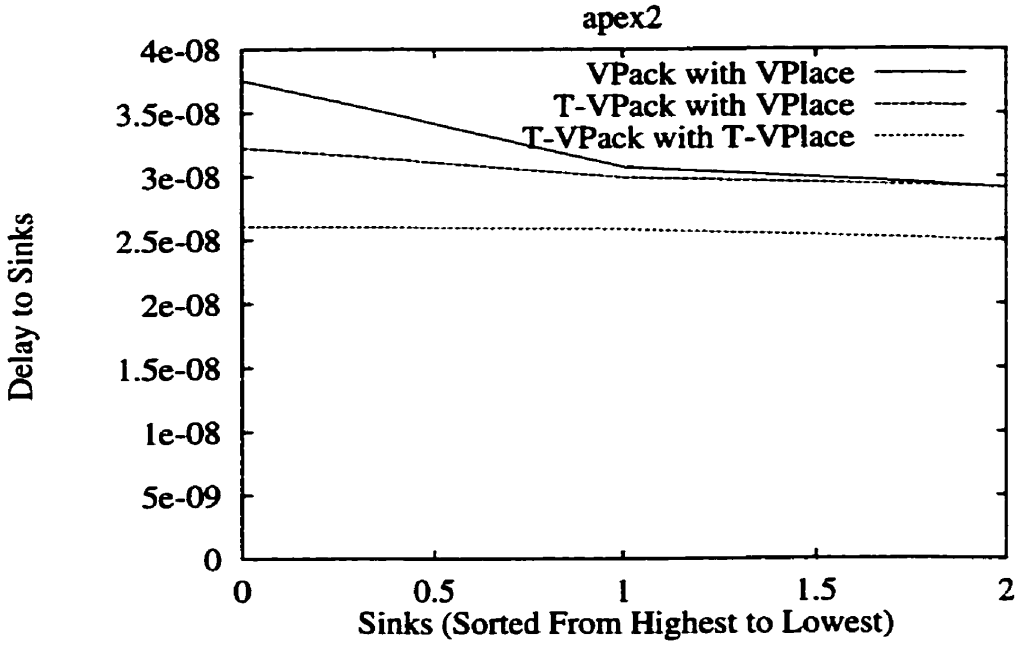

spla
VPlace
T-VPlace
Delay to Sinks
8e-08
7e-08
6e-08
5e-08
4e-08
3e-08
2e-08
1e-08
0
0
5
10
15
20
25
30
35
40
45
Sinks (Sorted From Highest to Lowest)

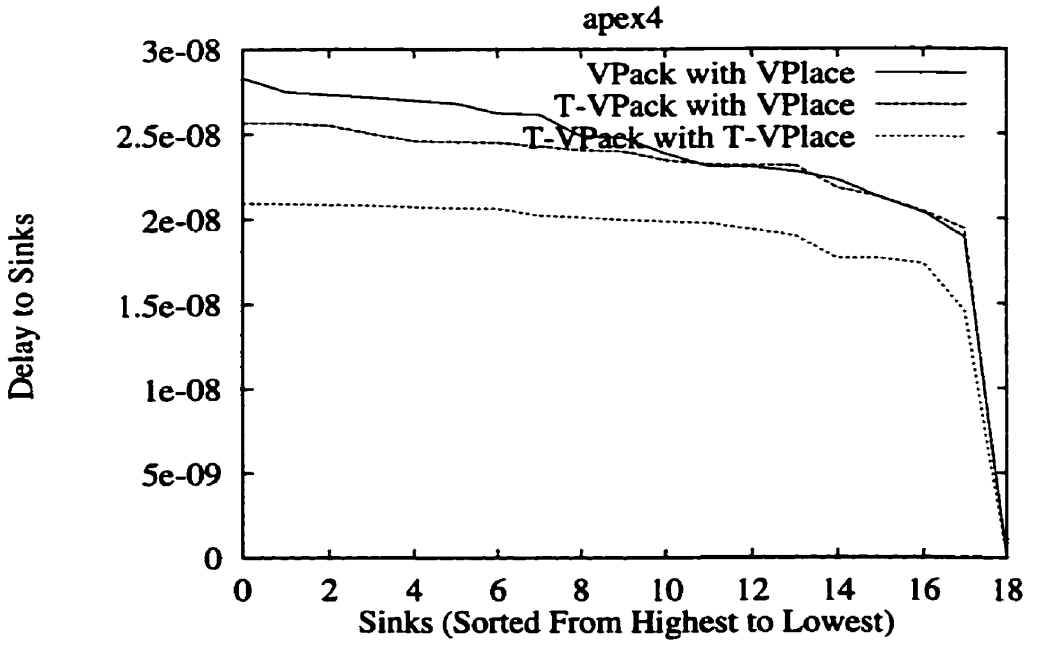

tseng
VPlace
T-VPlace
Delay to Sinks
4e-08
3.5e-08
3e-08
2.5e-08
2e-08
1.5e-08
1e-08
5e-09
0
0
100
200
300
400
500
600
Sinks (Sorted From Highest to Lowest)

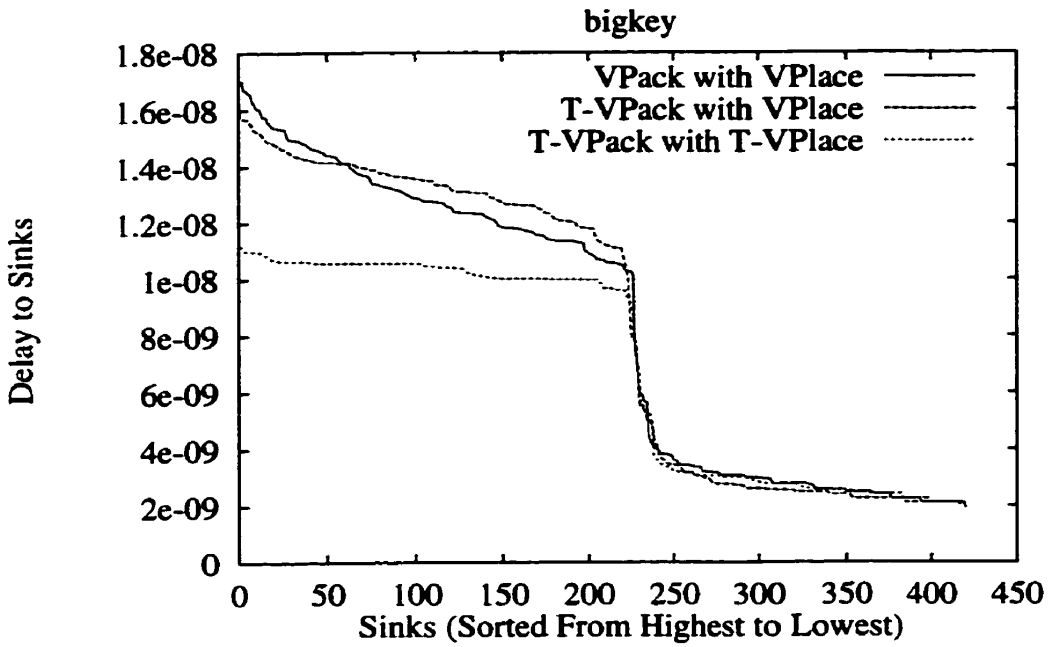
C.3 Placement Estimated Sink Delay Distributions: Size 8 Clusters

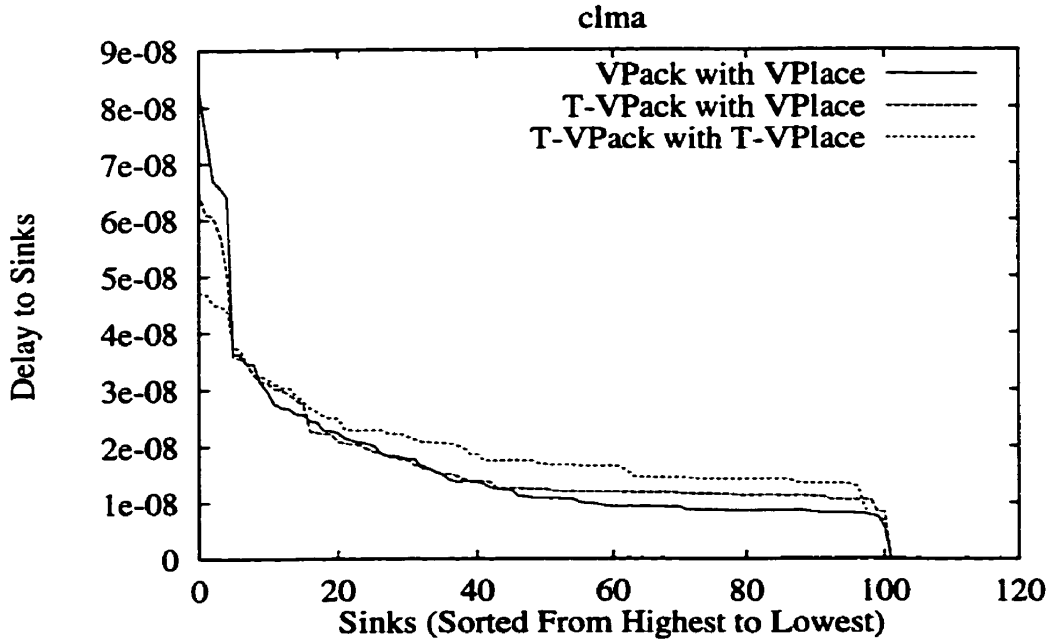
In this section we present the post-placement estimated sink delay distributions for the 20 largest MCNC benchmark circuits using size 8 clusters. The delays that we present are placement-estimated delays as we discussed in Section 5.2.2. For T-VPlace, we set the adaptive Criticality_Exponent to 8, and λ to 0.5.

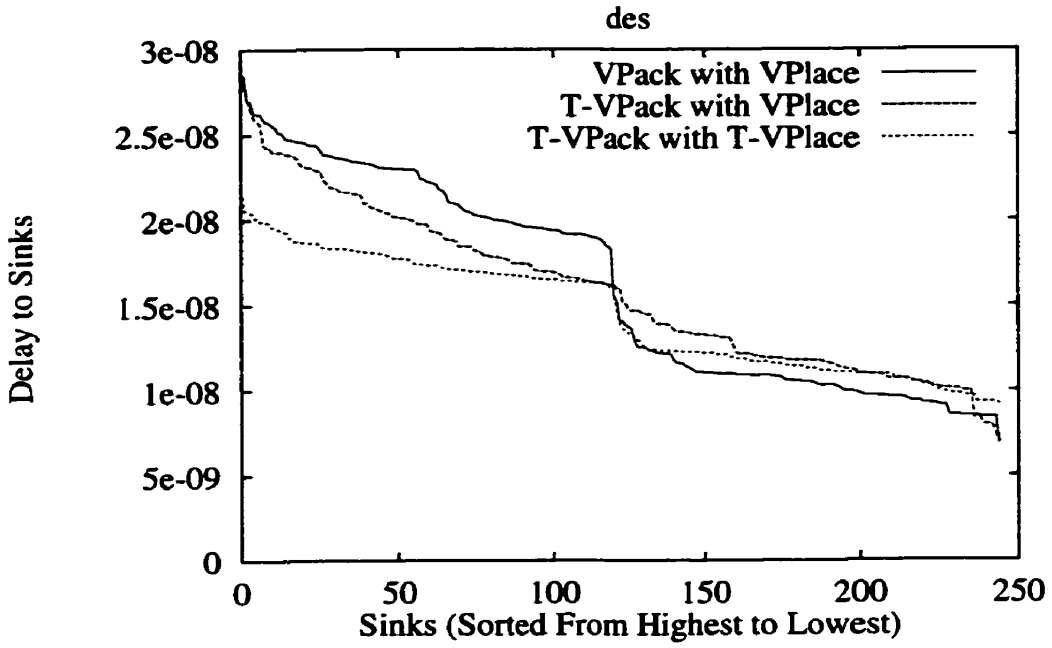


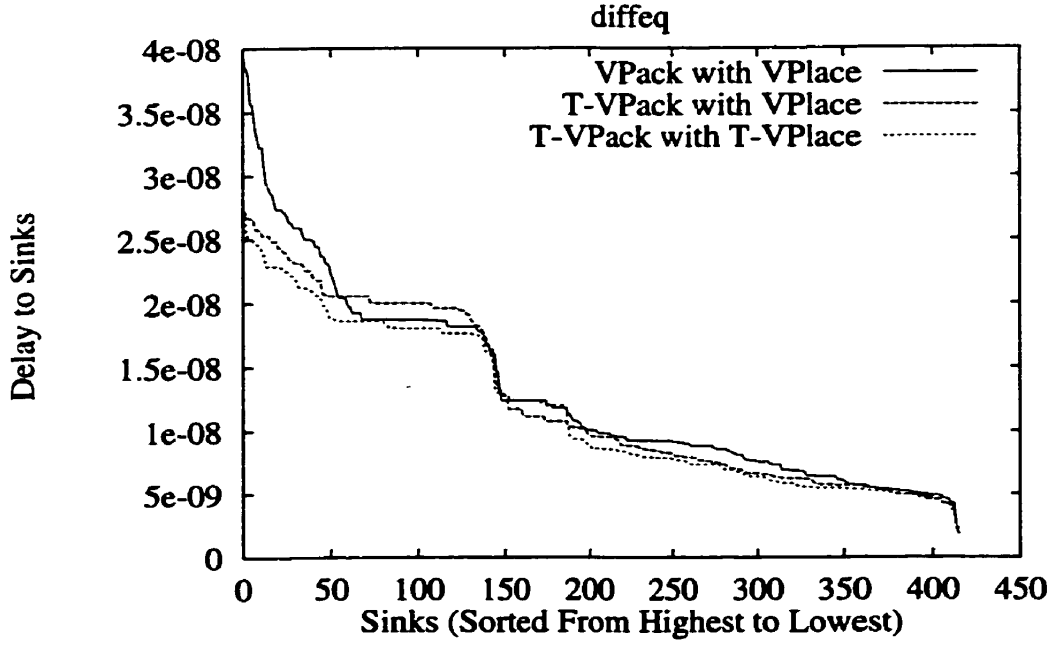

apex2
VPack with VPlace
T-VPack with VPlace
T-VPack with T-VPlace
Delay to Sinks
4e-08
3.5e-08
3e-08
2.5e-08
2e-08
1.5e-08
1e-08
5e-09
0
0
0.5
1
1.5
2
Sinks (Sorted From Highest to Lowest)

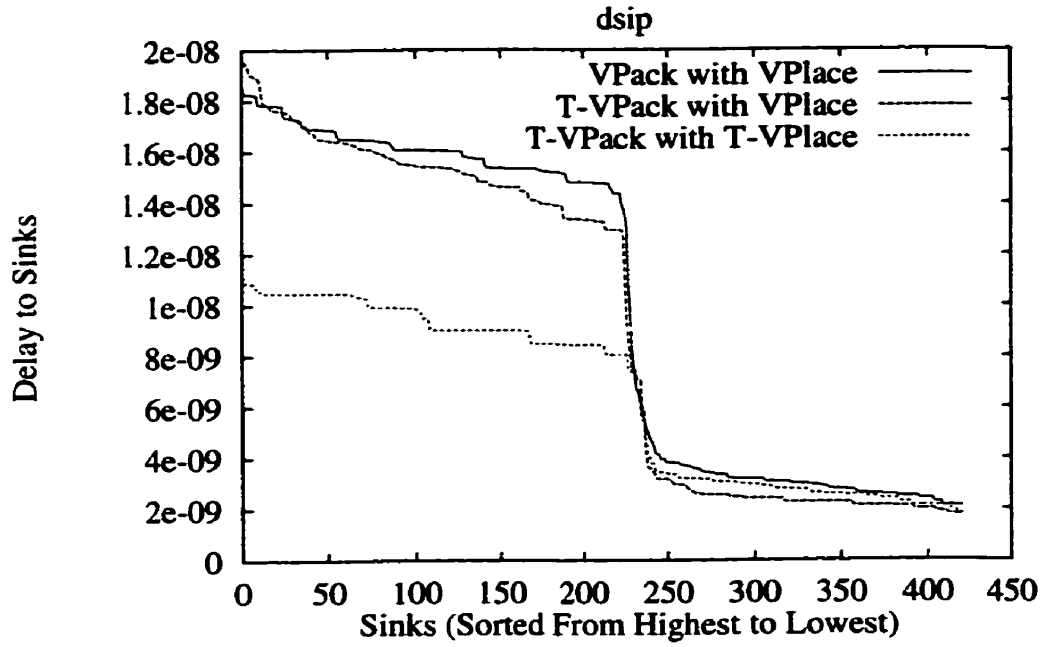

apex4
VPack with VPlace
T-VPack with VPlace
T-VPack with T-VPlace
Delay to Sinks
3e-08
2.5e-08
2e-08
1.5e-08
1e-08
5e-09
0
0
2
4
6
8
10
12
14
16
18
Sinks (Sorted From Highest to Lowest)

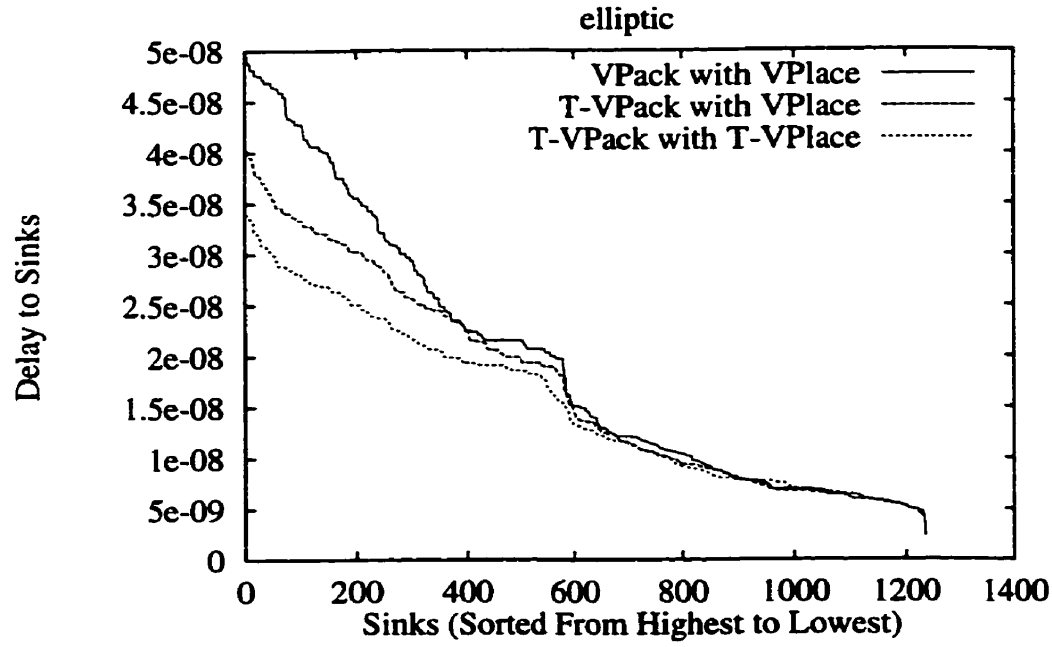

bigkey
VPack with VPlace
T-VPack with VPlace
T-VPack with T-VPlace
Delay to Sinks
1.8e-08
1.6e-08
1.4e-08
1.2e-08
1e-08
8e-09
6e-09
4e-09
2e-09
0
0
50
100
150
200
250
300
350
400
450
Sinks (Sorted From Highest to Lowest)

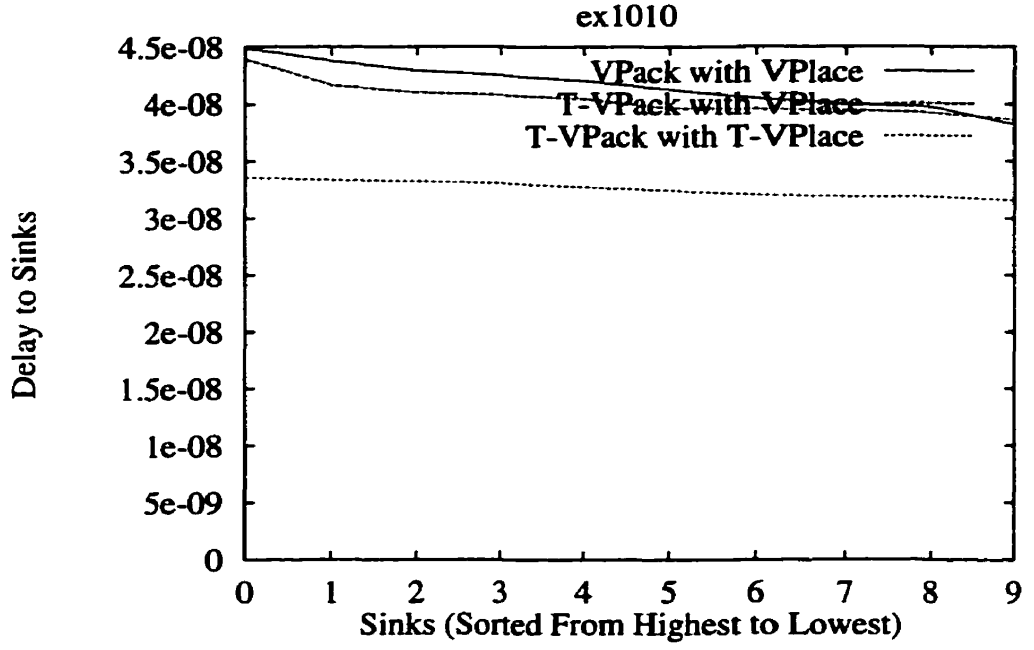

clma
VPack with VPlace
T-VPack with VPlace
T-VPack with T-VPlace
Delay to Sinks
9e-08
8e-08
7e-08
6e-08
5e-08
4e-08
3e-08
2e-08
1e-08
0
0
20
40
60
80
100
120
Sinks (Sorted From Highest to Lowest)

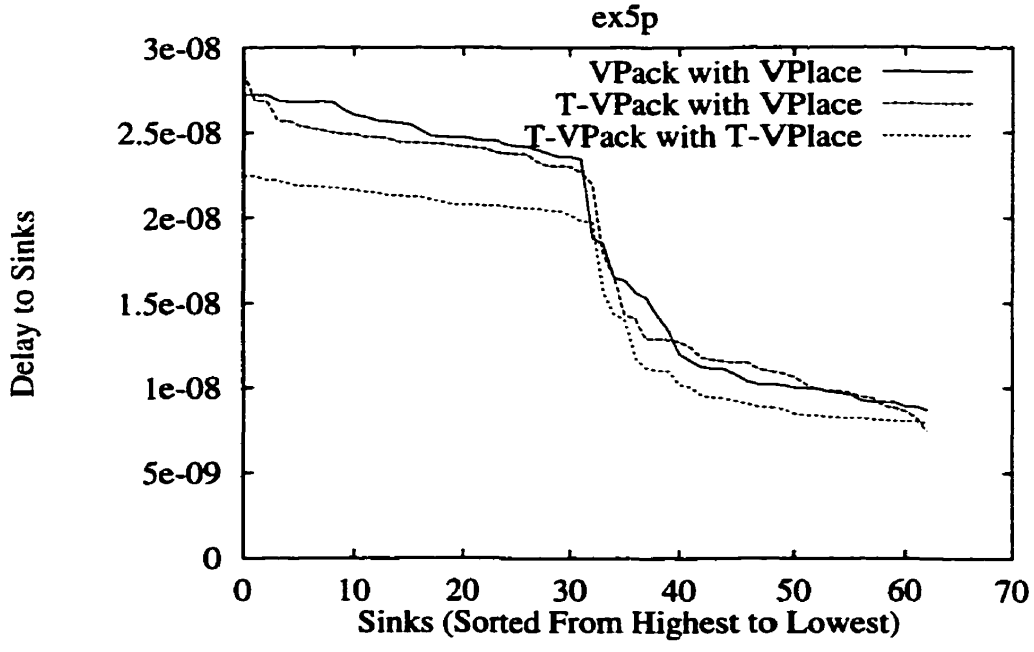

des
VPack with VPlace
T-VPack with VPlace
T-VPack with T-VPlace
Delay to Sinks
3e-08
2.5e-08
2e-08
1.5e-08
1e-08
5e-09
0
0
50
100
150
200
250
Sinks (Sorted From Highest to Lowest)

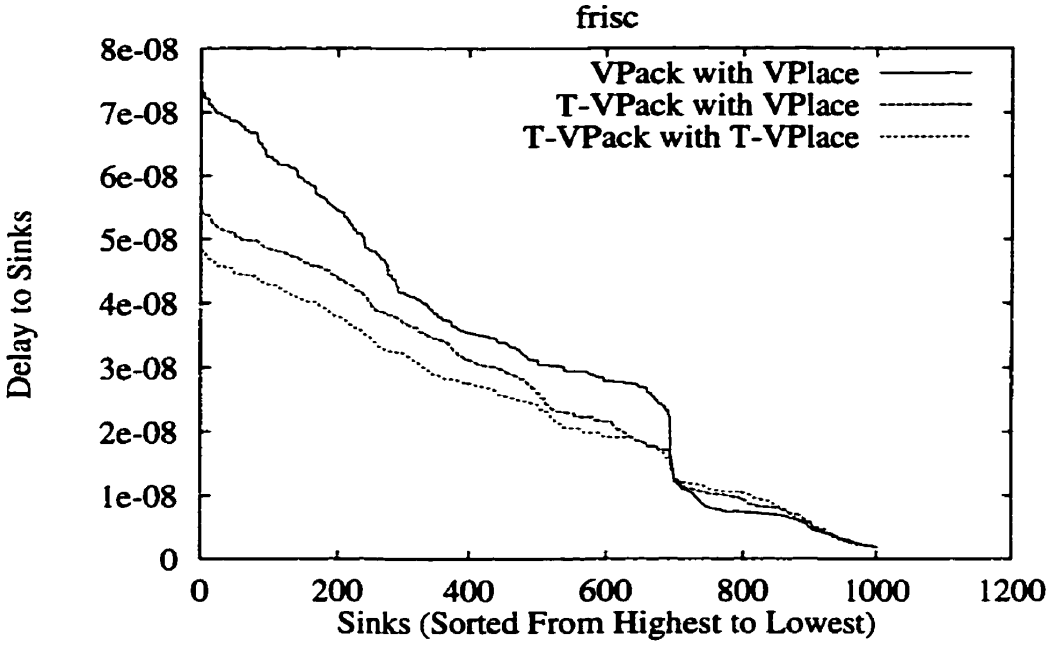

diffeq
VPack with VPlace
T-VPack with VPlace
T-VPack with T-VPlace
Delay to Sinks
4e-08
3.5e-08
3e-08
2.5e-08
2e-08
1.5e-08
1e-08
5e-09
0
0
50
100
150
200
250
300
350
400
450
Sinks (Sorted From Highest to Lowest)

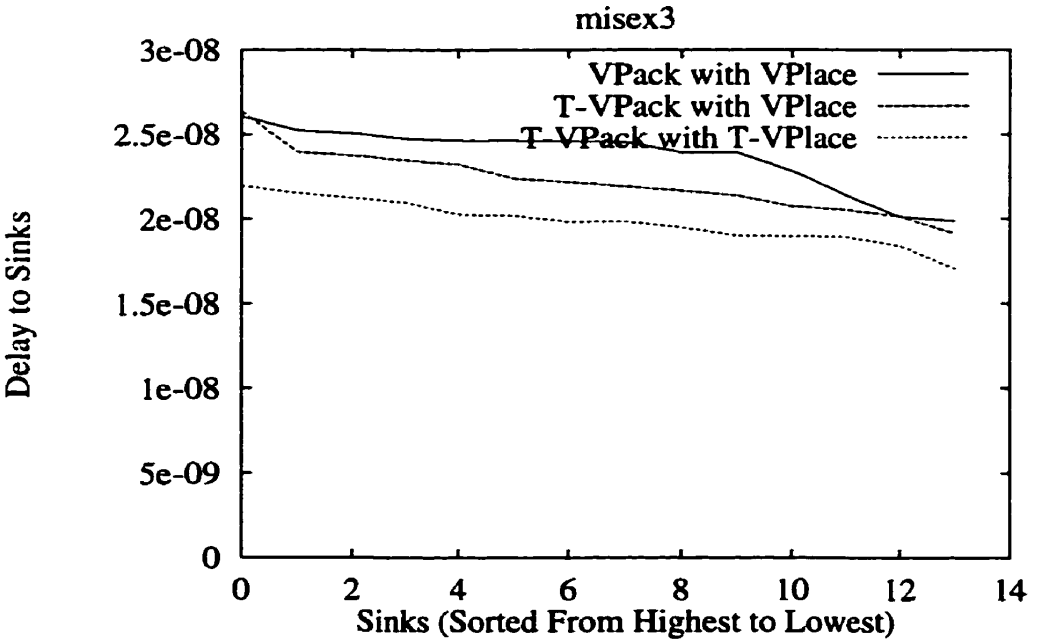

dsip
VPack with VPlace
T-VPack with VPlace
T-VPack with T-VPlace
Delay to Sinks
2e-08
1.8e-08
1.6e-08
1.4e-08
1.2e-08
1e-08
8e-09
6e-09
4e-09
2e-09
0
0
50
100
150
200
250
300
350
400
450
Sinks (Sorted From Highest to Lowest)

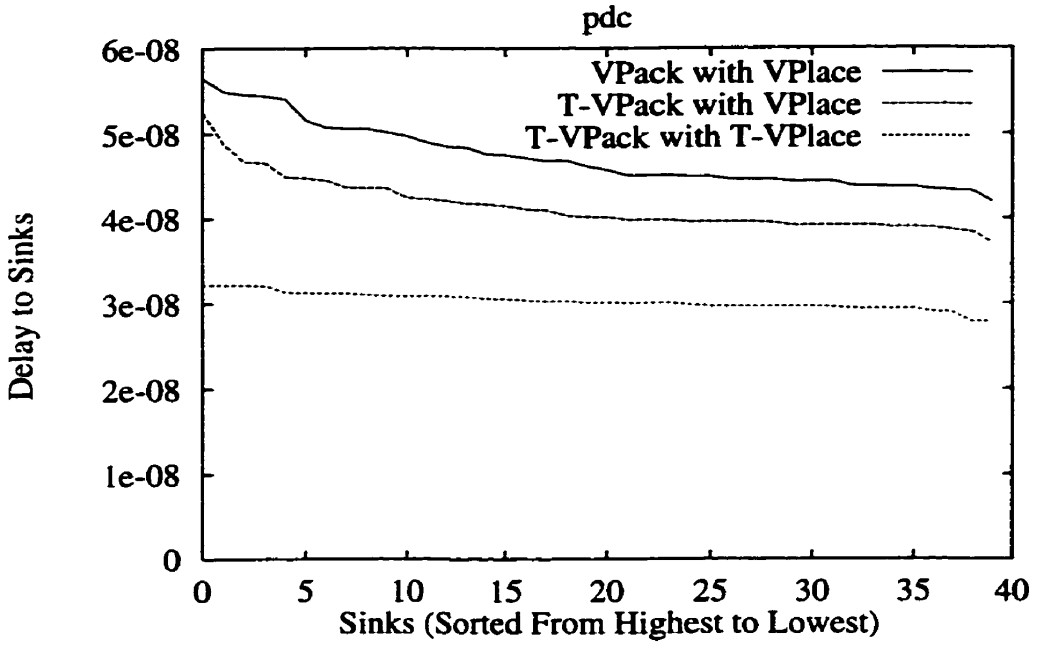

elliptic
VPack with VPlace
T-VPack with VPlace
T-VPack with T-VPlace
Delay to Sinks
5e-08
4.5e-08
4e-08
3.5e-08
3e-08
2.5e-08
2e-08
1.5e-08
1e-08
5e-09
0
0
200
400
600
800
1000
1200
1400
Sinks (Sorted From Highest to Lowest)

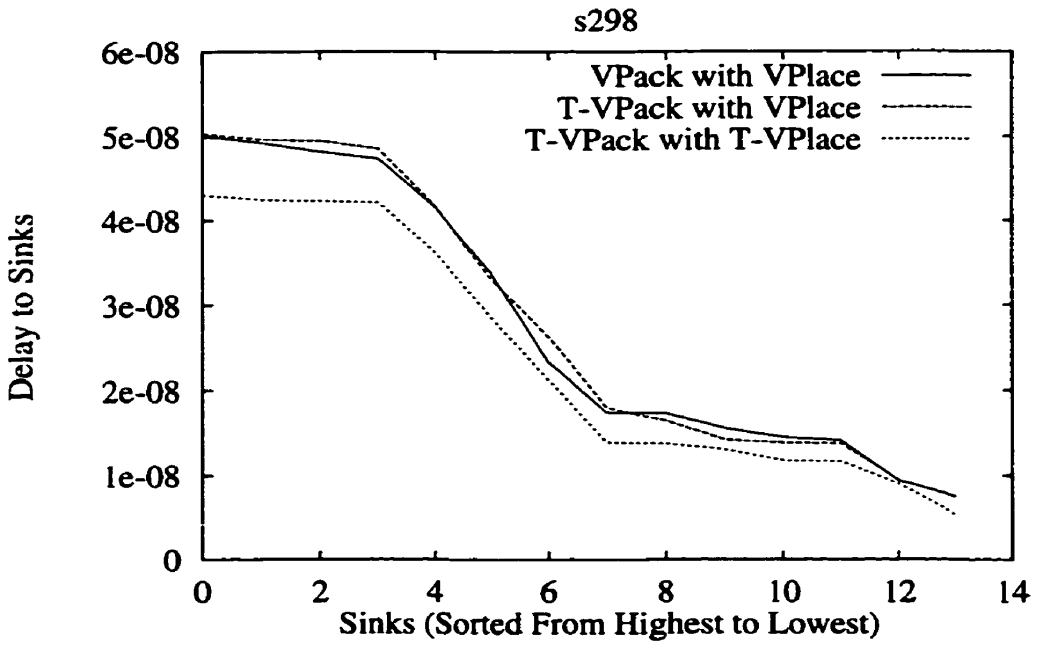

ex1010
VPack with VPlace
T-VPack with VPlace
T-VPack with T-VPlace
Delay to Sinks
4.5e-08
4e-08
3.5e-08
3e-08
2.5e-08
2e-08
1.5e-08
1e-08
5e-09
0
0 1 2 3 4 5 6 7 8 9
Sinks (Sorted From Highest to Lowest)

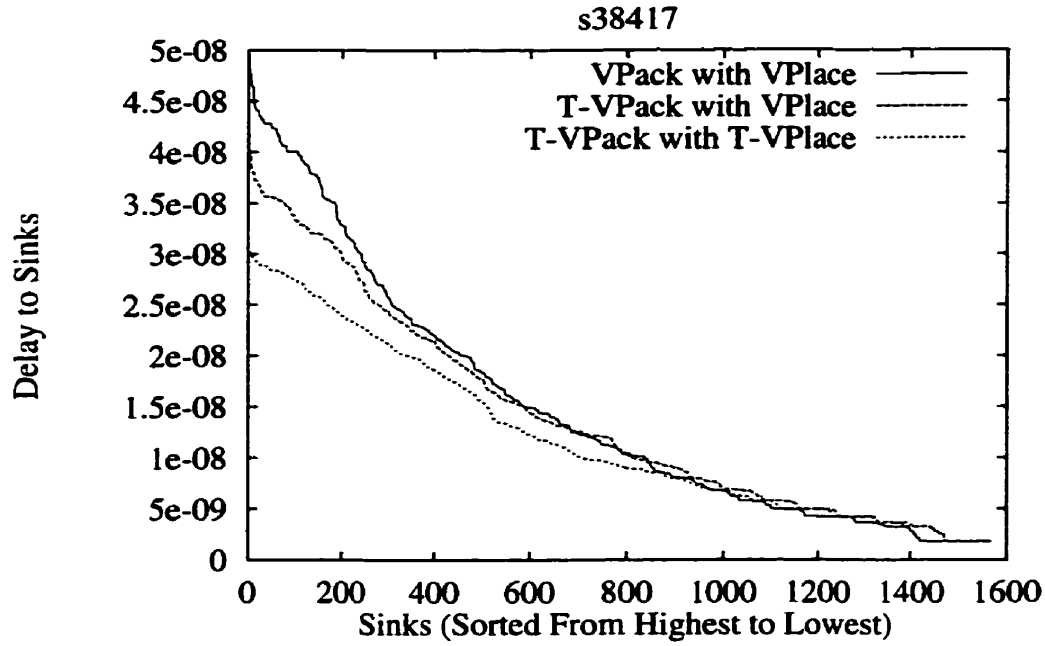

ex5p
VPack with VPlace
T-VPack with VPlace
T-VPack with T-VPlace
Delay to Sinks
3e-08
2.5e-08
2e-08
1.5e-08
1e-08
5e-09
0
0 10 20 30 40 50 60 70
Sinks (Sorted From Highest to Lowest)

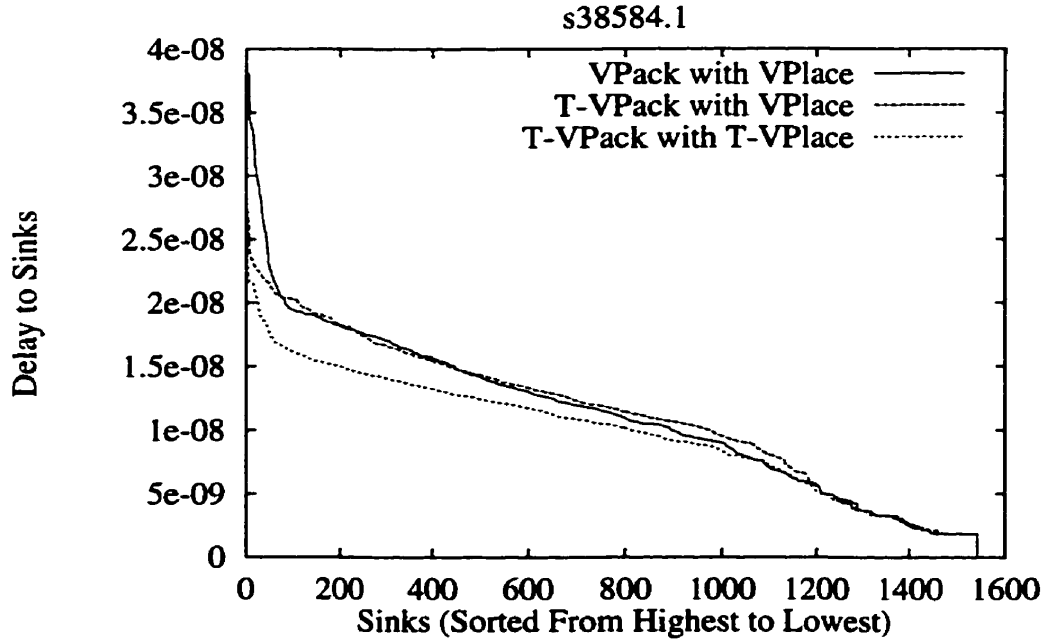

frisc
VPack with VPlace
T-VPack with VPlace
T-VPack with T-VPlace
Delay to Sinks
8e-08
7e-08
6e-08
5e-08
4e-08
3e-08
2e-08
1e-08
0
0
200
400
600
800
1000
1200
Sinks (Sorted From Highest to Lowest)

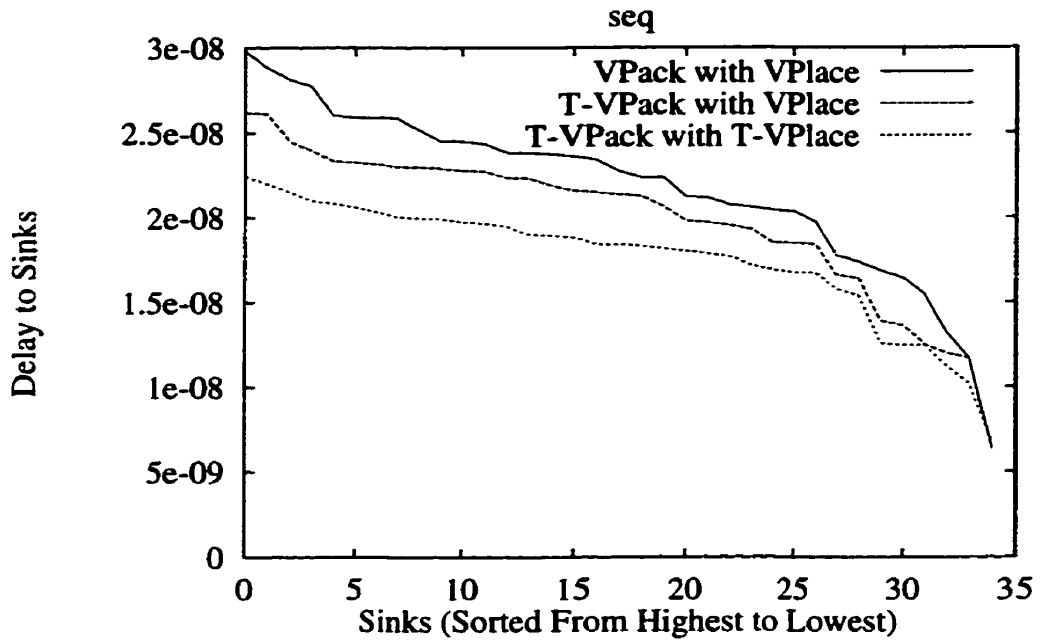

misex3
VPack with VPlace
T-VPack with VPlace
T-VPack with T-VPlace
Delay to Sinks
3e-08
2.5e-08
2e-08
1.5e-08
1e-08
5e-09
0
0
2
4
6
8
10
12
14
Sinks (Sorted From Highest to Lowest)

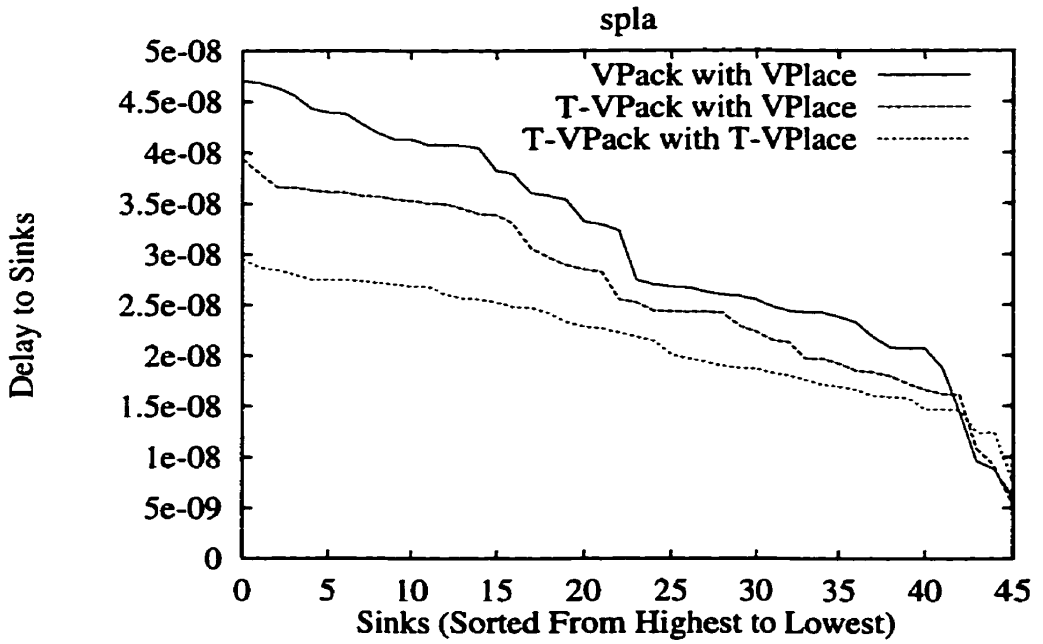

pdc
VPack with VPlace
T-VPack with VPlace
T-VPack with T-VPlace
Delay to Sinks
6e-08
5e-08
4e-08
3e-08
2e-08
1e-08
0
0
5
10
15
20
25
30
35
40
Sinks (Sorted From Highest to Lowest)

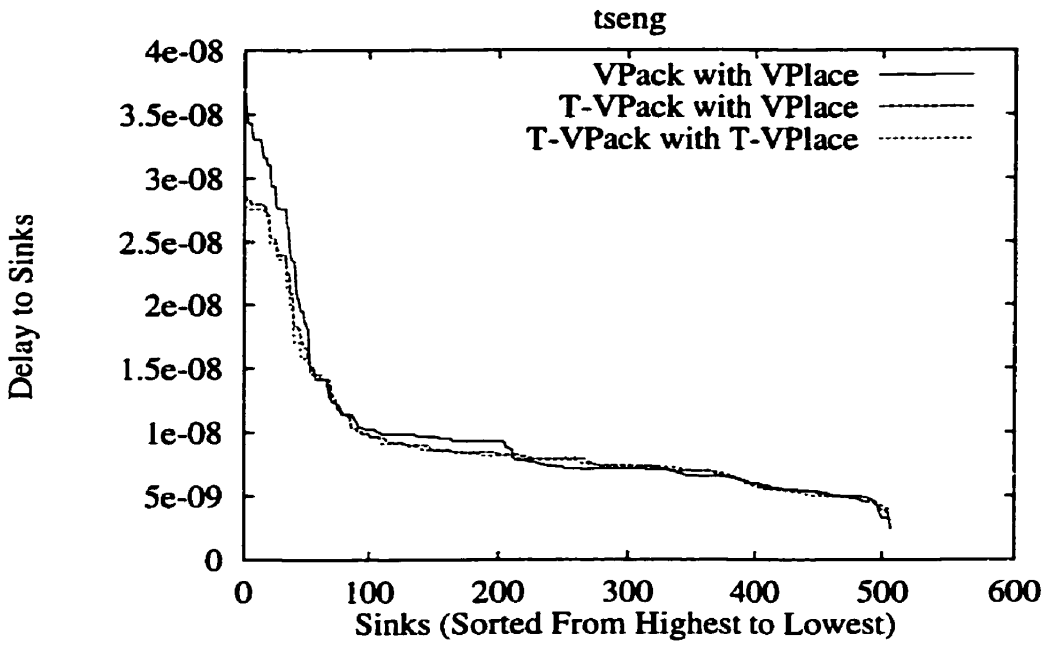

s298
VPack with VPlace
T-VPack with VPlace
T-VPack with T-VPlace
Delay to Sinks
6e-08
5e-08
4e-08
3e-08
2e-08
1e-08
0
0
2
4
6
8
10
12
14
Sinks (Sorted From Highest to Lowest)


s38417
VPack with VPlace
T-VPack with VPlace
T-VPack with T-VPlace
Delay to Sinks
5e-08
4.5e-08
4e-08
3.5e-08
3e-08
2.5e-08
2e-08
1.5e-08
1e-08
5e-09
0
0
200
400
600
800
1000
1200
1400
1600
Sinks (Sorted From Highest to Lowest)


s38584.1
VPack with VPlace
T-VPack with VPlace
T-VPack with T-VPlace
Delay to Sinks
4e-08
3.5e-08
3e-08
2.5e-08
2e-08
1.5e-08
1e-08
5e-09
0
0
200
400
600
800
1000
1200
1400
1600
Sinks (Sorted From Highest to Lowest)

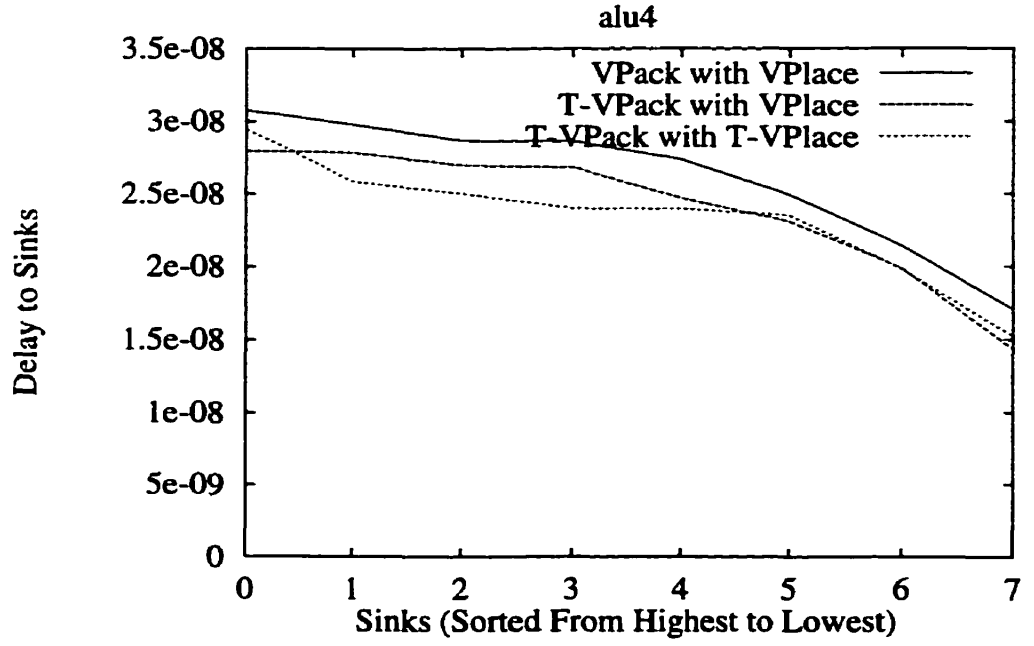

seq
VPack with VPlace
T-VPack with VPlace
T-VPack with T-VPlace
Delay to Sinks
3e-08
2.5e-08
2e-08
1.5e-08
1e-08
5e-09
0
0
5
10
15
20
25
30
35
Sinks (Sorted From Highest to Lowest)

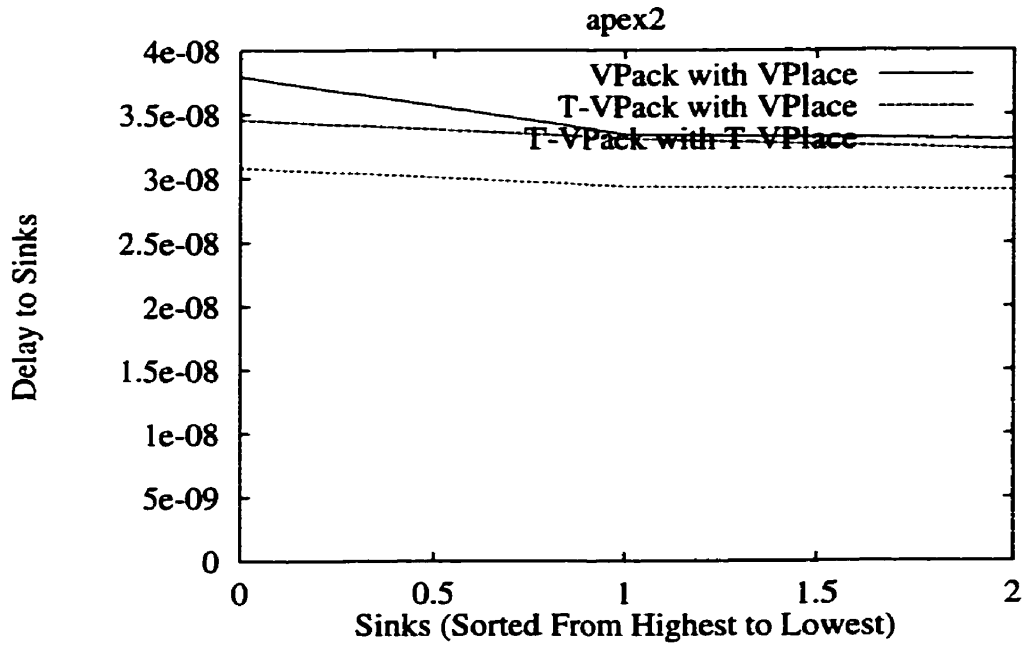

spla
VPack with VPlace
T-VPack with VPlace
T-VPack with T-VPlace
Delay to Sinks
5e-08
4.5e-08
4e-08
3.5e-08
3e-08
2.5e-08
2e-08
1.5e-08
1e-08
5e-09
0
0
5
10
15
20
25
30
35
40
45
Sinks (Sorted From Highest to Lowest)

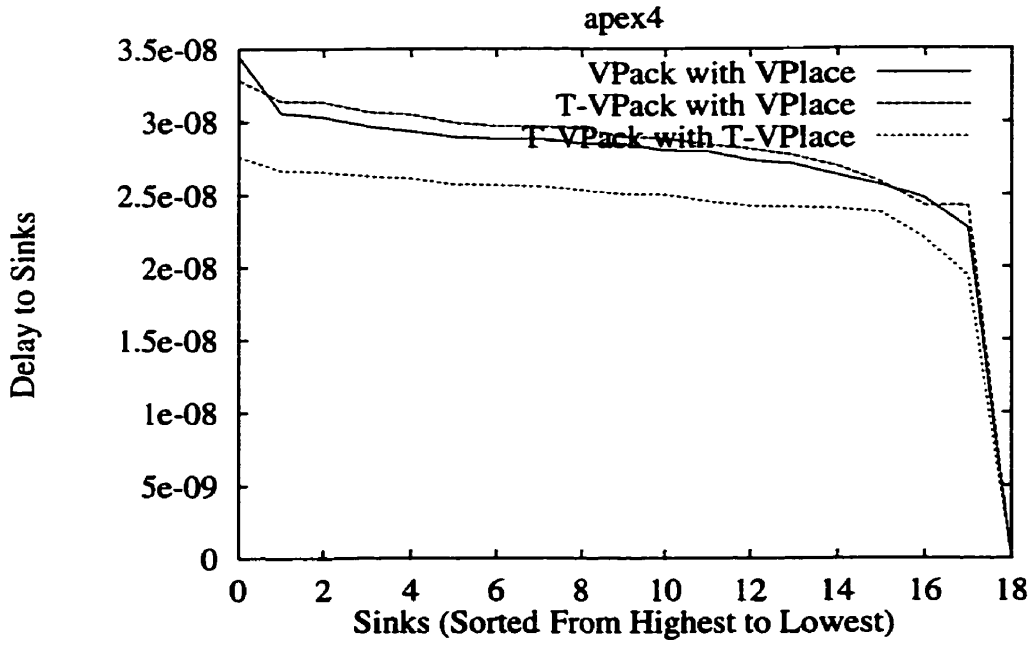

tseng
VPack with VPlace
T-VPack with VPlace
T-VPack with T-VPlace
Delay to Sinks
4e-08
3.5e-08
3e-08
2.5e-08
2e-08
1.5e-08
1e-08
5e-09
0
0
100
200
300
400
500
600
Sinks (Sorted From Highest to Lowest)

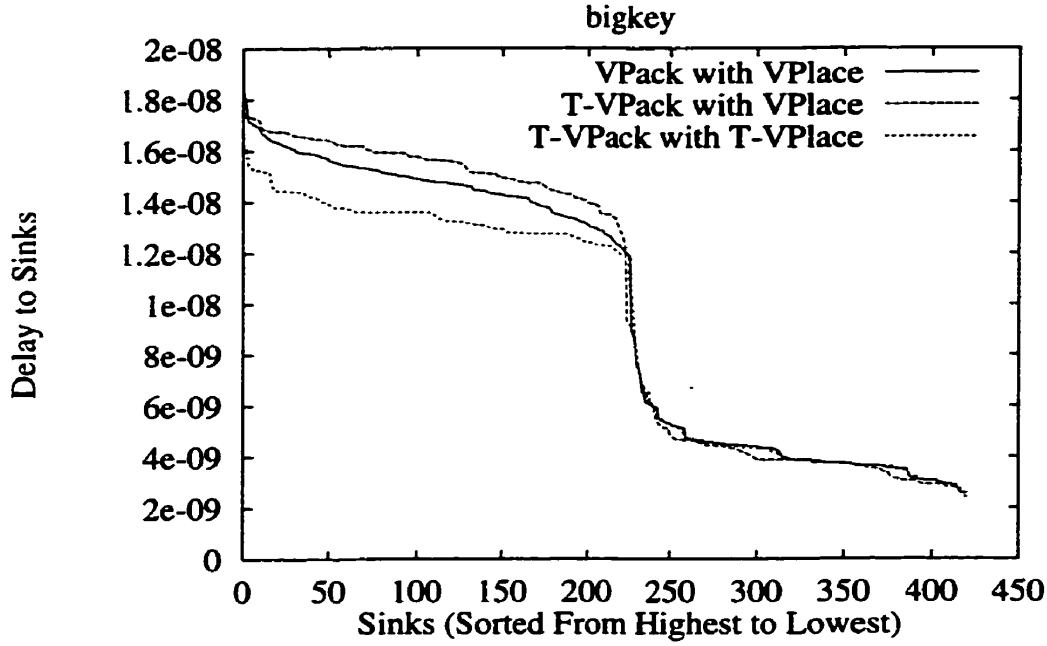
C.4 Low-Stress Sink Delay Distributions: Size 8 Clusters

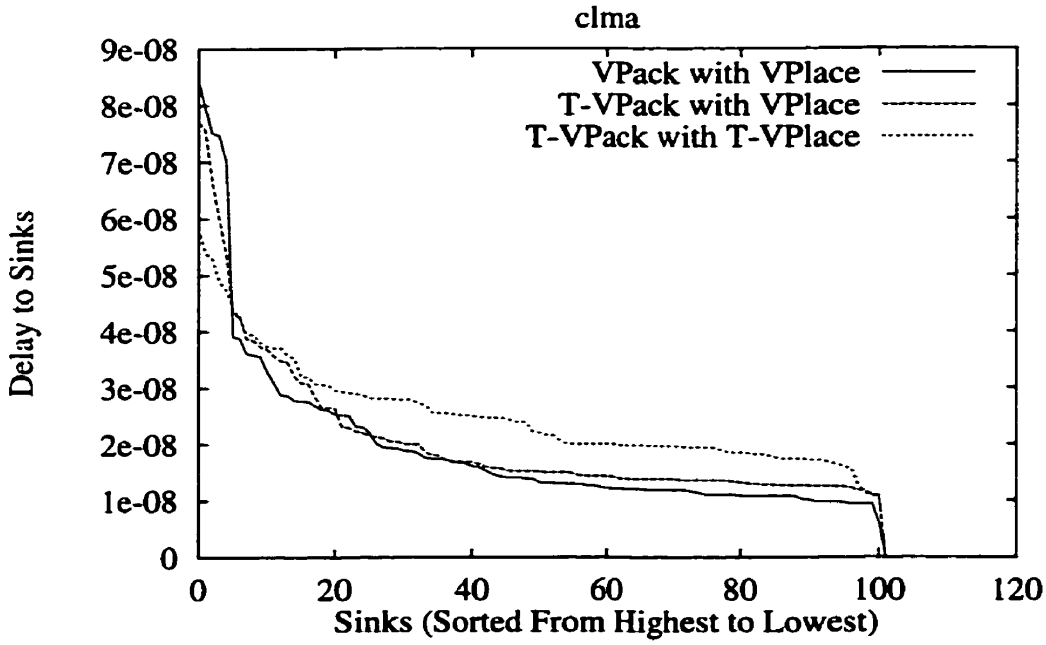
In this section we present the post place-and-route sink delay distributions for the 20 largest MCNC benchmark circuits using size 8 clusters. The delays that we present are the low-stress which we defined in Section 3.1. For T-VPlace, we set the adaptive Criticality_Exponent to 8, and λ to 0.5.

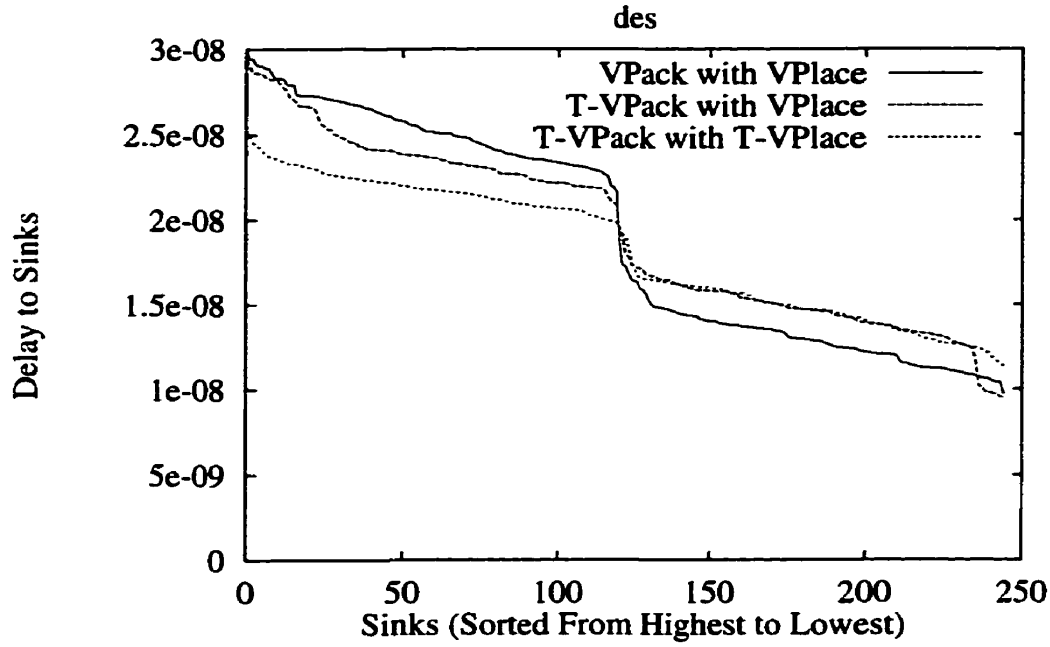


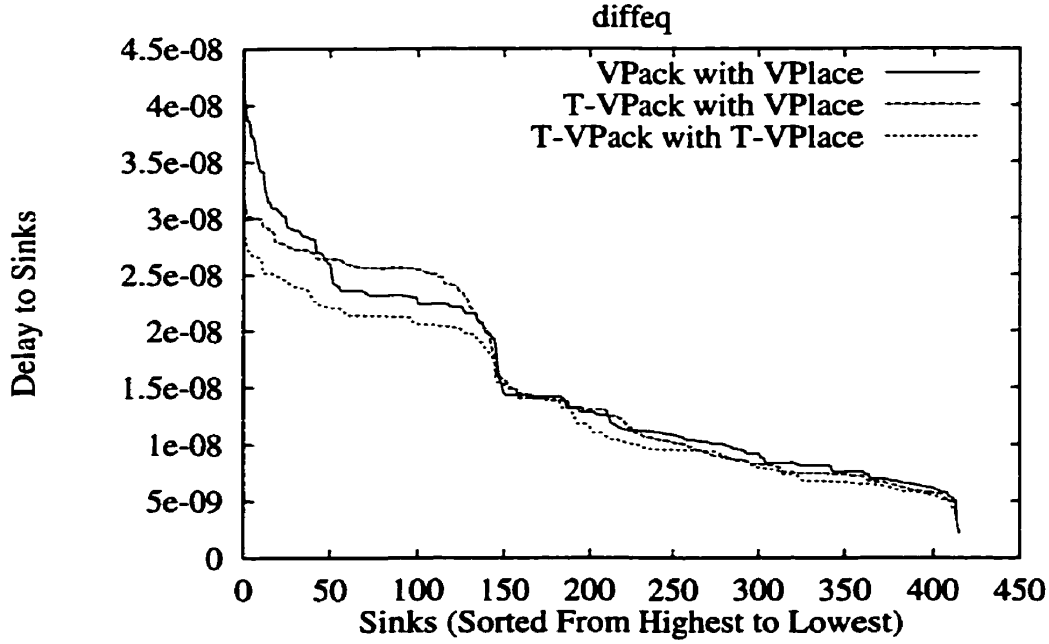

apex2
VPack with VPlace
T-VPack with VPlace
T-VPack with T-VPlace
Delay to Sinks
4e-08
3.5e-08
3e-08
2.5e-08
2e-08
1.5e-08
1e-08
5e-09
0
0
0.5
1
1.5
2
Sinks (Sorted From Highest to Lowest)

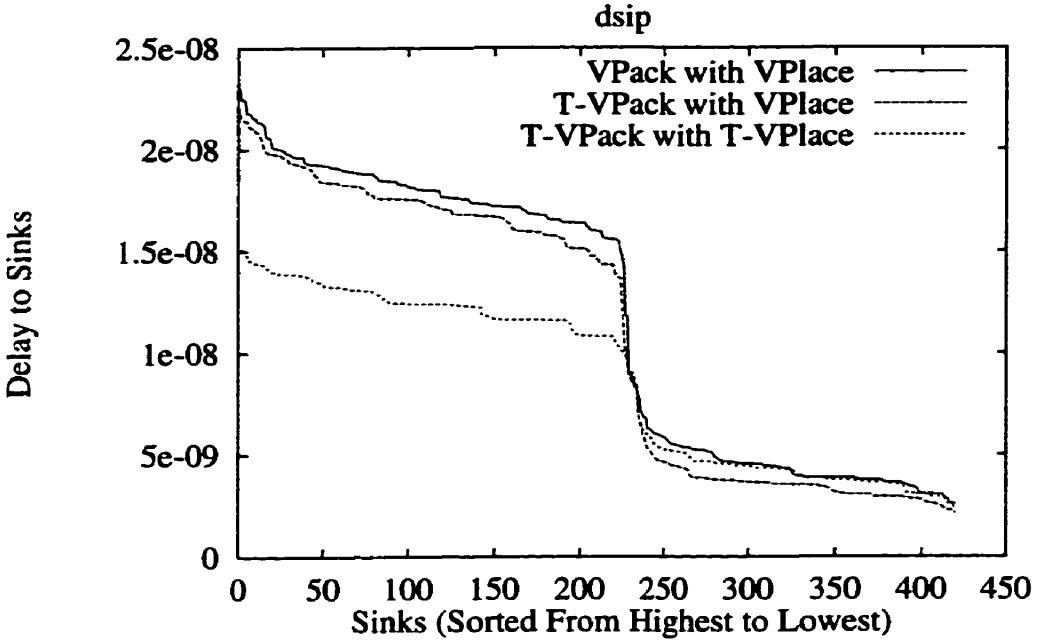

apex4
VPack with VPlace
T-VPack with VPlace
T-VPack with T-VPlace
Delay to Sinks
3.5e-08
3e-08
2.5e-08
2e-08
1.5e-08
1e-08
5e-09
0
0
2
4
6
8
10
12
14
16
18
Sinks (Sorted From Highest to Lowest)

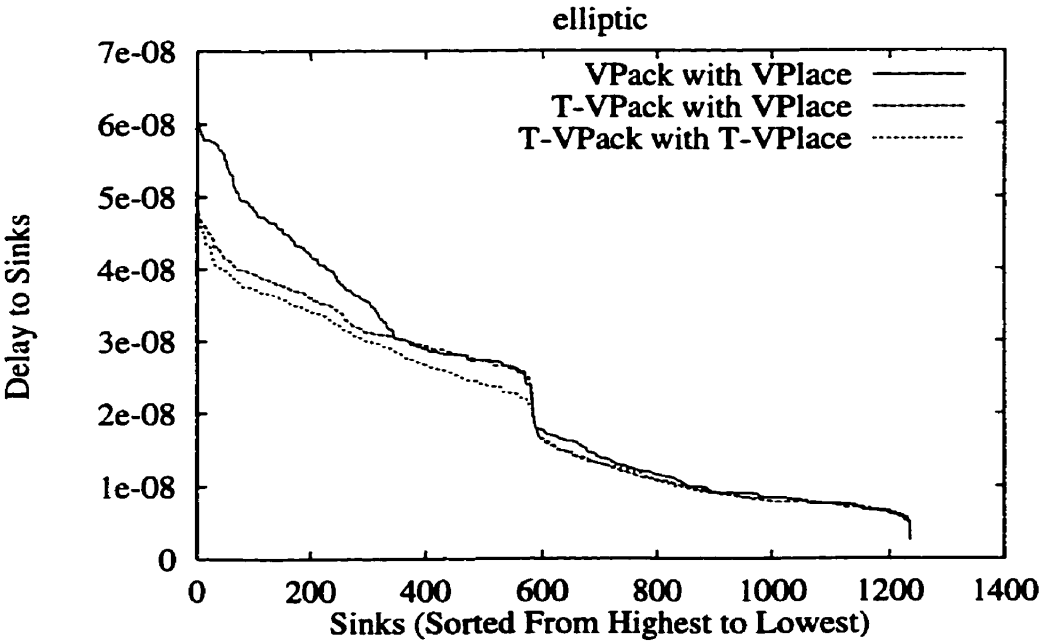

bigkey
VPack with VPlace
T-VPack with VPlace
T-VPack with T-VPlace
Delay to Sinks
2e-08
1.8e-08
1.6e-08
1.4e-08
1.2e-08
1e-08
8e-09
6e-09
4e-09
2e-09
0
0
50
100
150
200
250
300
350
400
450
Sinks (Sorted From Highest to Lowest)

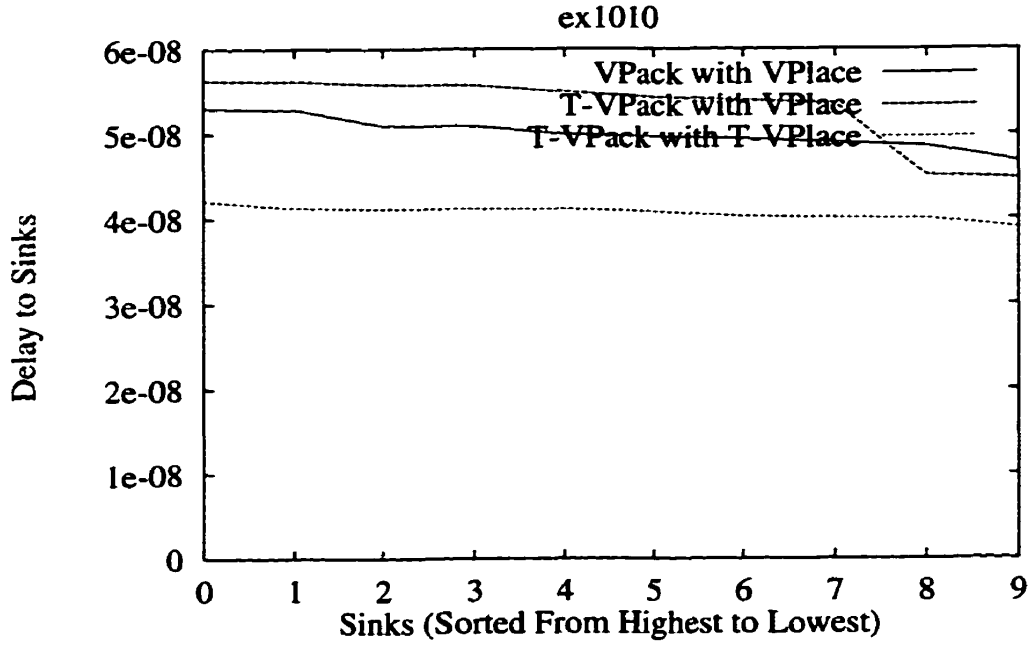

clma
VPack with VPlace
T-VPack with VPlace
T-VPack with T-VPlace
Delay to Sinks
9e-08
8e-08
7e-08
6e-08
5e-08
4e-08
3e-08
2e-08
1e-08
0
0
20
40
60
80
100
120
Sinks (Sorted From Highest to Lowest)

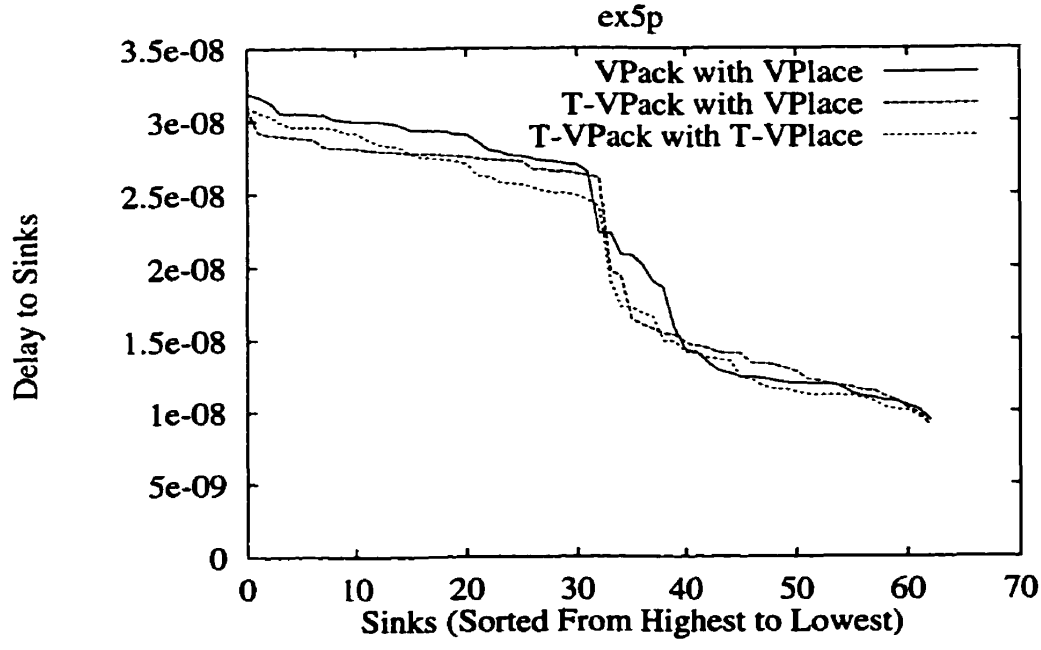

des
VPack with VPlace
T-VPack with VPlace
T-VPack with T-VPlace
Delay to Sinks
3e-08
2.5e-08
2e-08
1.5e-08
1e-08
5e-09
0
0
50
100
150
200
250
Sinks (Sorted From Highest to Lowest)

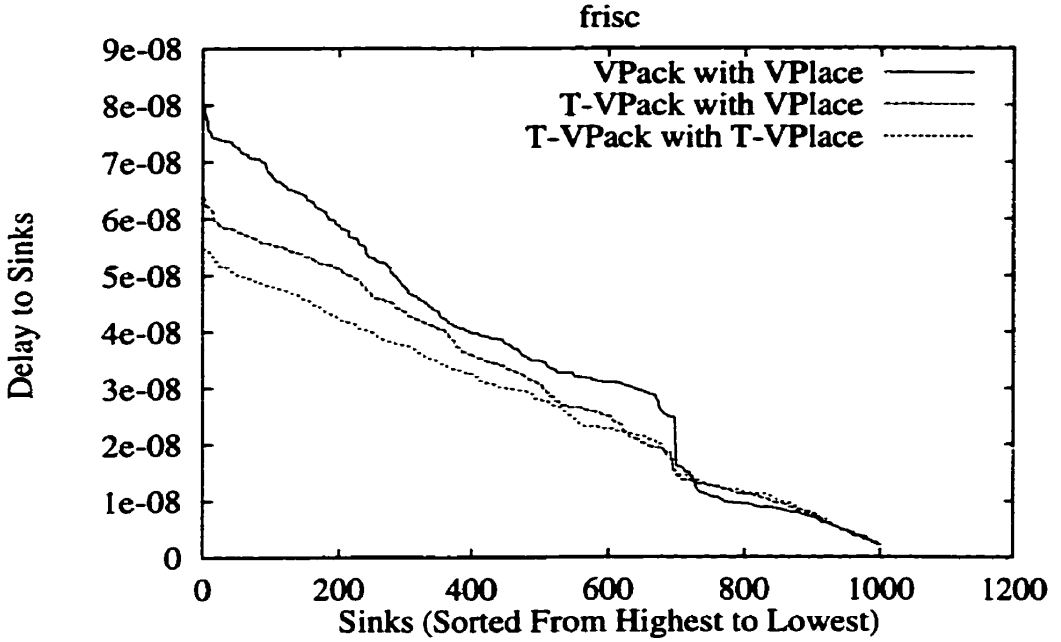

diffeq
VPack with VPlace
T-VPack with VPlace
T-VPack with T-VPlace
Delay to Sinks
4.5e-08
4e-08
3.5e-08
3e-08
2.5e-08
2e-08
1.5e-08
1e-08
5e-09
0
0
50
100
150
200
250
300
350
400
450
Sinks (Sorted From Highest to Lowest)

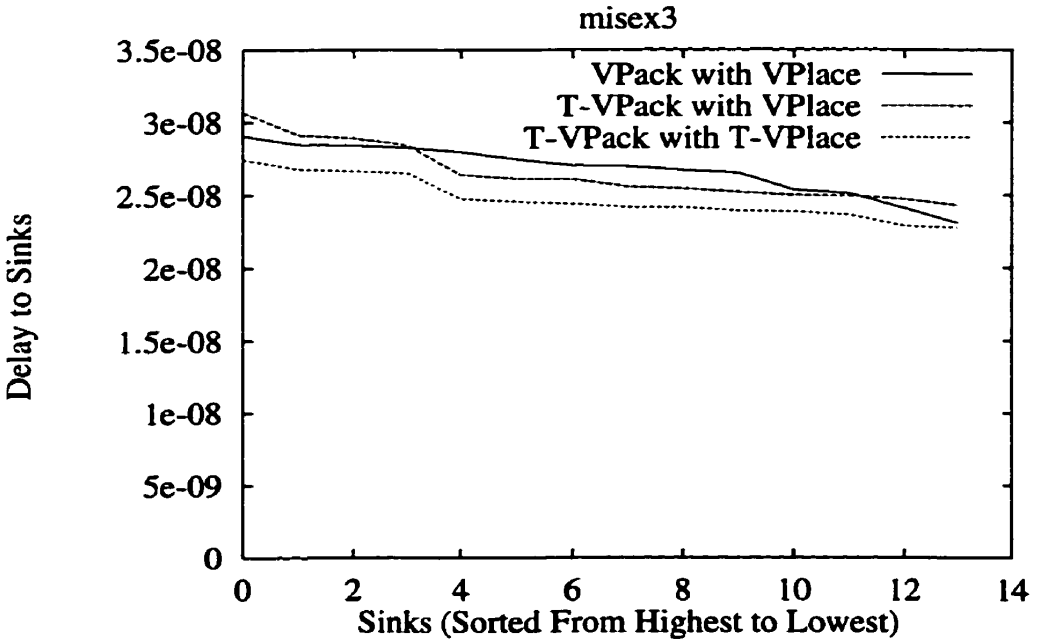

dsip
VPack with VPlace
T-VPack with VPlace
T-VPack with T-VPlace
Delay to Sinks
2.5e-08
2e-08
1.5e-08
1e-08
5e-09
0
0
50
100
150
200
250
300
350
400
450
Sinks (Sorted From Highest to Lowest)

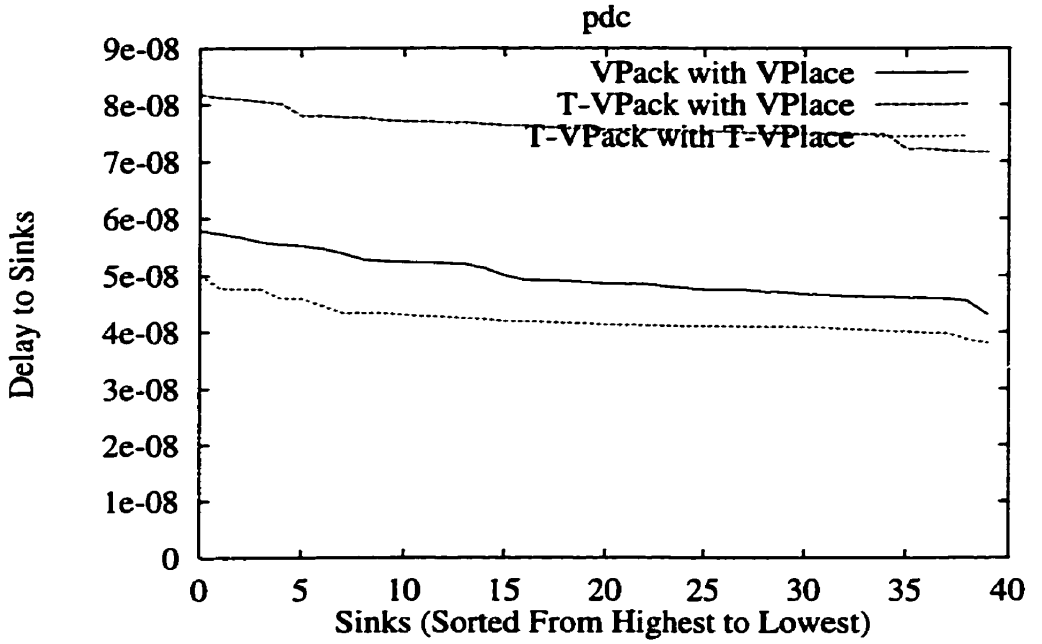

elliptic
VPack with VPlace
T-VPack with VPlace
T-VPack with T-VPlace
Delay to Sinks
7e-08
6e-08
5e-08
4e-08
3e-08
2e-08
1e-08
0
0
200
400
600
800
1000
1200
1400
Sinks (Sorted From Highest to Lowest)

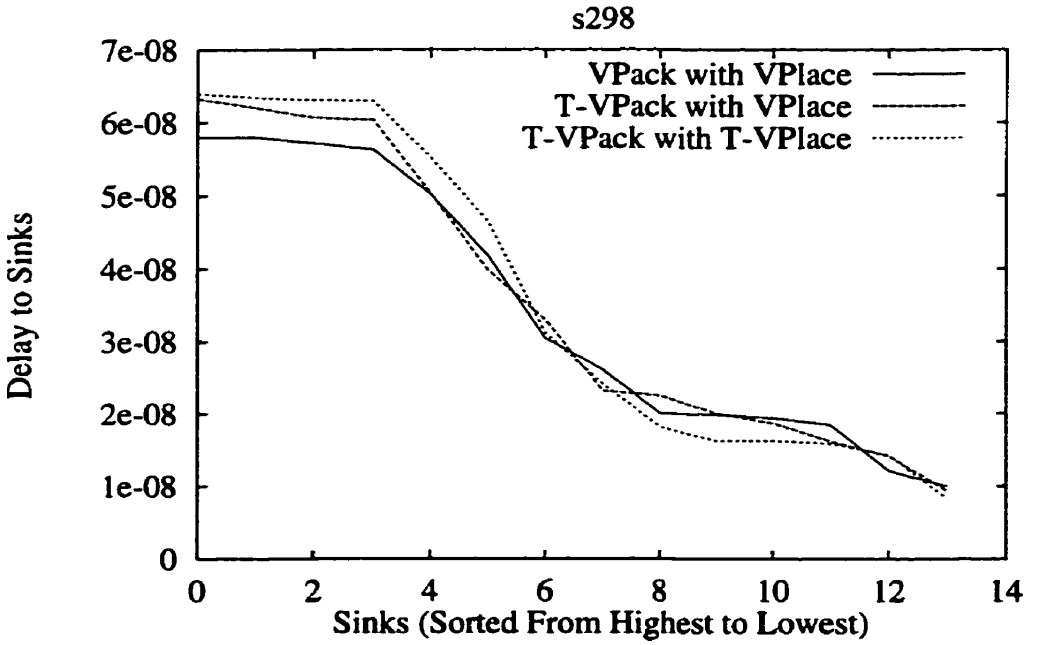

ex1010
VPack with VPlace
T-VPack with VPlace
T-VPack with T-VPlace
Delay to Sinks
6e-08
5e-08
4e-08
3e-08
2e-08
1e-08
0
0 1 2 3 4 5 6 7 8 9
Sinks (Sorted From Highest to Lowest)

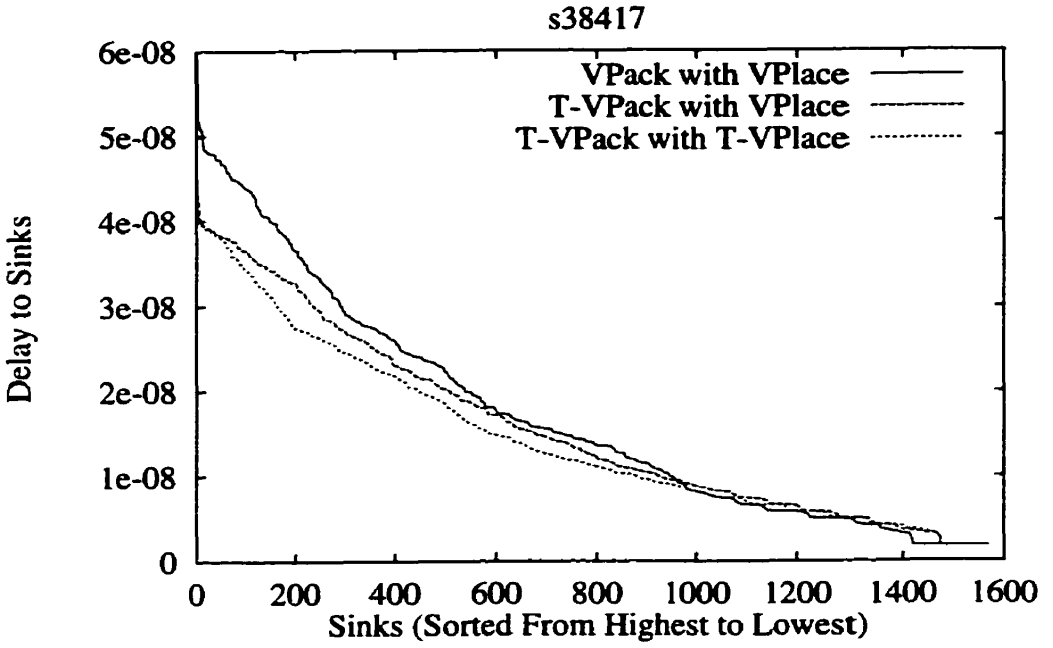

ex5p
VPack with VPlace
T-VPack with VPlace
T-VPack with T-VPlace
Delay to Sinks
3.5e-08
3e-08
2.5e-08
2e-08
1.5e-08
1e-08
5e-09
0
0 10 20 30 40 50 60 70
Sinks (Sorted From Highest to Lowest)

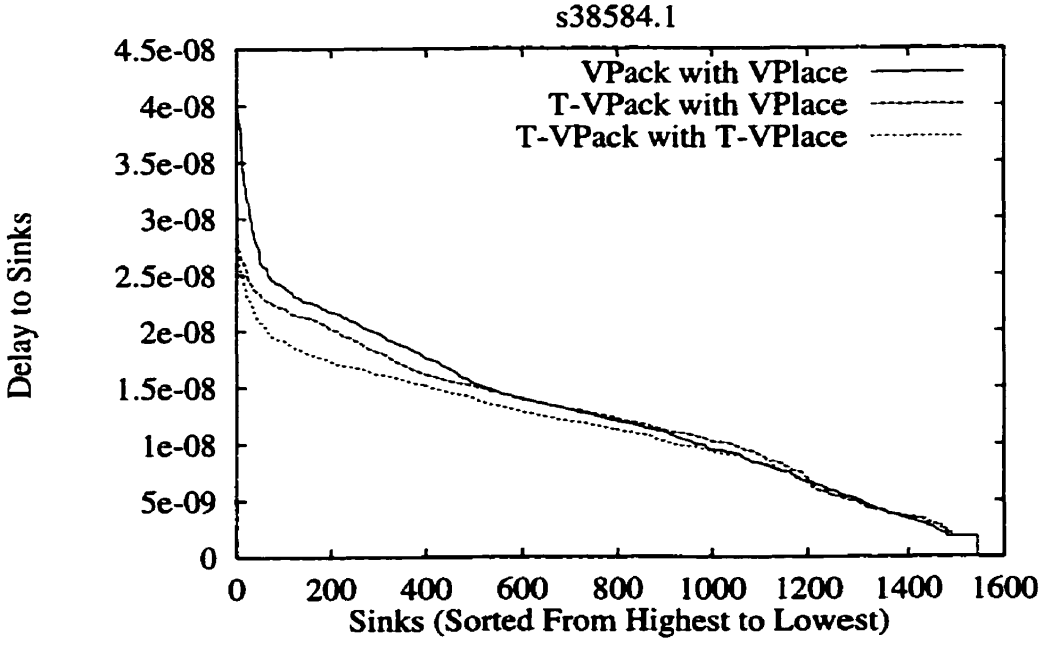

frisc
VPack with VPlace
T-VPack with VPlace
T-VPack with T-VPlace
Delay to Sinks
9e-08
8e-08
7e-08
6e-08
5e-08
4e-08
3e-08
2e-08
1e-08
0
0
200
400
600
800
1000
1200
Sinks (Sorted From Highest to Lowest)

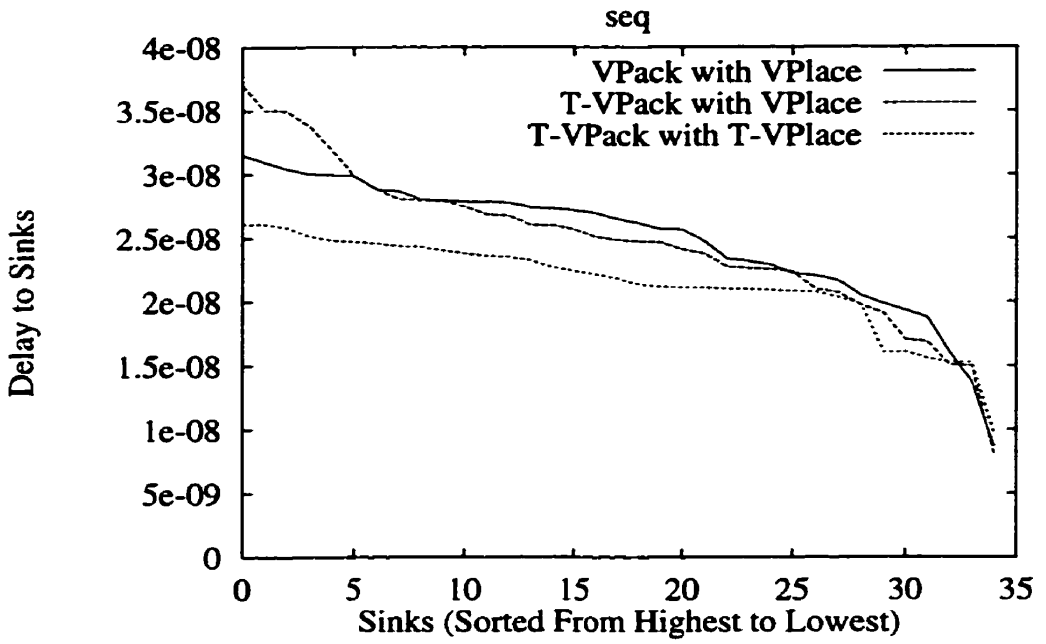

misex3
VPack with VPlace
T-VPack with VPlace
T-VPack with T-VPlace
Delay to Sinks
3.5e-08
3e-08
2.5e-08
2e-08
1.5e-08
1e-08
5e-09
0
0
2
4
6
8
10
12
14
Sinks (Sorted From Highest to Lowest)

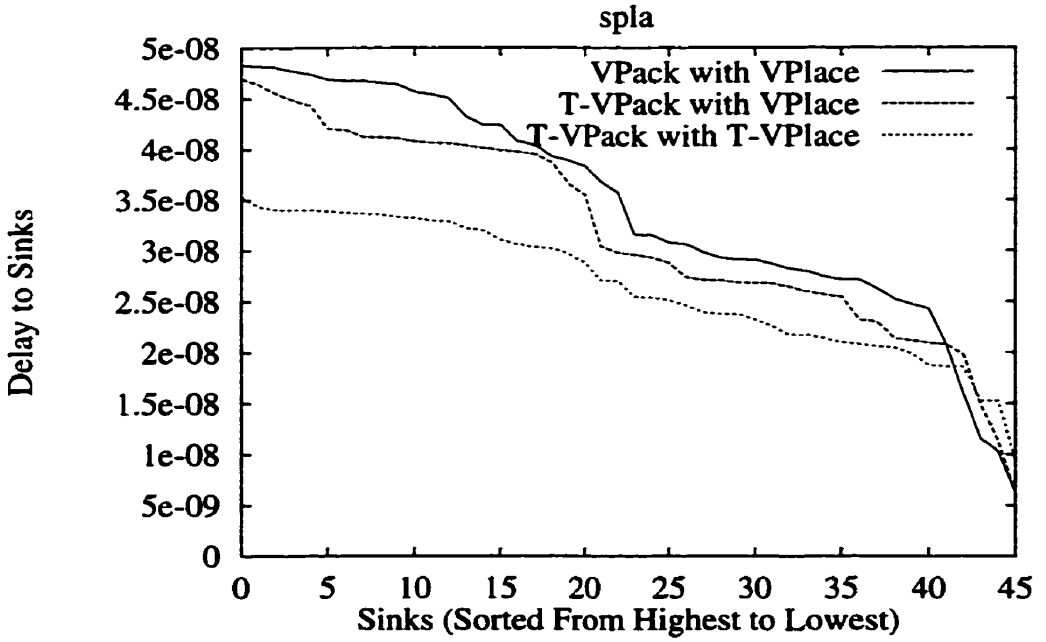

pdc
VPack with VPlace
T-VPack with VPlace
T-VPack with T-VPlace
Delay to Sinks
9e-08
8e-08
7e-08
6e-08
5e-08
4e-08
3e-08
2e-08
1e-08
0
0
5
10
15
20
25
30
35
40
Sinks (Sorted From Highest to Lowest)

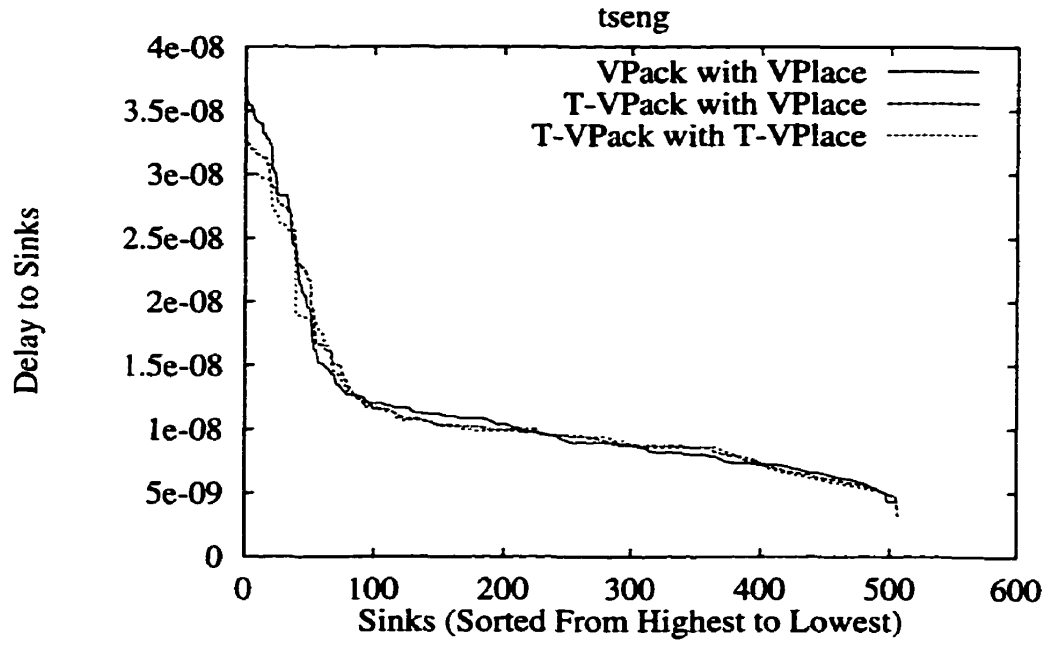

s298
VPack with VPlace
T-VPack with VPlace
T-VPack with T-VPlace
Delay to Sinks
7e-08
6e-08
5e-08
4e-08
3e-08
2e-08
1e-08
0
0
2
4
6
8
10
12
14
Sinks (Sorted From Highest to Lowest)


s38417
VPack with VPlace
T-VPack with VPlace
T-VPack with T-VPlace
Delay to Sinks
6e-08
5e-08
4e-08
3e-08
2e-08
1e-08
0
0
200
400
600
800
1000
1200
1400
1600
Sinks (Sorted From Highest to Lowest)


s38584.1
VPack with VPlace
T-VPack with VPlace
T-VPack with T-VPlace
Delay to Sinks
4.5e-08
4e-08
3.5e-08
3e-08
2.5e-08
2e-08
1.5e-08
1e-08
5e-09
0
0
200
400
600
800
1000
1200
1400
1600
Sinks (Sorted From Highest to Lowest)


seq
VPack with VPlace
T-VPack with VPlace
T-VPack with T-VPlace
Delay to Sinks
4e-08
3.5e-08
3e-08
2.5e-08
2e-08
1.5e-08
1e-08
5e-09
0
0
5
10
15
20
25
30
35
Sinks (Sorted From Highest to Lowest)


spla
VPack with VPlace
T-VPack with VPlace
T-VPack with T-VPlace
Delay to Sinks
5e-08
4.5e-08
4e-08
3.5e-08
3e-08
2.5e-08
2e-08
1.5e-08
1e-08
5e-09
0
0
5
10
15
20
25
30
35
40
45
Sinks (Sorted From Highest to Lowest)


tseng
VPack with VPlace
T-VPack with VPlace
T-VPack with T-VPlace
Delay to Sinks
4e-08
3.5e-08
3e-08
2.5e-08
2e-08
1.5e-08
1e-08
5e-09
0
0
100
200
300
400
500
600
Sinks (Sorted From Highest to Lowest)

References

[Acte99] S. Kaptanoglu et. al., "A new high density and very low cost reprogrammable FPGA Architecture", *FPGA*, 1999, pp. 3 - 12.

[Alte98] Altera Inc., *Data Book*, 1998.

[Alte95] Altera Inc., "MAX+PLUS II Getting Started," 1995

[Betz97a] V. Betz and J. Rose, "Cluster-Based Logic Blocks for FPGAs: Area-Efficiency vs. Input Sharing and Size," *IEEE Custom Integrated Circuits Conference*, Santa Clara, CA, 1997, pp. 551-554.

[Betz97b] V. Betz and J. Rose, "VPR: A New Packing, Placement and Routing Tool for FPGA Research," *Int'l Workshop on FPL*, 1997, pp. 213-222.

[Betz98a] V.Betz and J. Rose, "How Much Logic Should Go in an FPGA Logic Block?," *IEEE Design and Test Magazine*, Spring 1998, pp. 10-15.

[Betz98b] V. Betz, "Architecture and CAD for Speed and Area Optimization of FPGAs," *Ph. D. Dissertation, University of Toronto*, 1998.

[Betz98c] V. Betz, "VPR and VPack User's Manual (Version 4.17)," May 5, 1998. (*Available for download from http://www.eecg.toronto.edu/~vaughn/vpr/vpr.html).*

[Betz99] V. Betz, J. Rose, A. Marquardt, *Architecture and CAD for Deep-Submicron FPGAs*, Kluwer Academic Publishers, 1999.

[Boes93] K. Boese, A. Kahng, B. McCoy and G. Robins, "Fidelity and Near-Optimality of Elmore-Based Routing Constructions," *ICCAD*, 1993, pp. 81 - 84.

[Brow92] S. Brown, R. Francis, J. Rose, and Z. Vranesic, *Field-Programmable Gate Arrays*, Kluwer Academic Publishers, 1992.

[Brow96] S. Brown and J. Rose, "FPGA and CPLD Architectures: A Tutorial," *IEEE Design & Test of Computers*, Summer 1996, pp. 42-57.

[Chen94] C. Cheng, "RISA: Accurate and Efficient Placement Routability Modeling," *ICCAD*, 1994, pp. 690 - 695.

[Cong94] J. Cong and Y. Ding, "Flowmap: An Optimal Technology Mapping Algorithm for Delay Optimization in Lookup-Table Based FPGA Designs," *IEEE Trans. on CAD*, Jan. 1994, pp 1-12.

[Cong96] J. Cong, J. Peck and Y. Ding, "RASP: A General Logic Synthesis System for SRAM-based FPGAs," *ACM Symp. on FPGAs*, 1996, pp. 137 - 143

[Elmo48] W. C. Elmore, "The Transient Response of Damped Linear Networks with Particular Regard to Wideband Amplifiers," *J. Applied Physics*, Vol. 19, January 1948, pp. 55-63.

[Fran92] J. Frankle, "Iterative and Adaptive Slack Allocation for Performance-Driven Layout and FPGA Routing," DAC, 1992, pp. 536 - 542.

[Gall98] D. Galloway, "Implementation of Grayscale Conversion for Video Image Processing on the Transmogrifier-2a," *Personal Communication.*

[Hame98] I. Hamer, "Implementation of DES on the Transmogrifier-2a," *Personal Communication.*

[Haug87] P. Hauge, R. Nair, and E. Yoffa, "Circuit Placement for Predictable Performance," *ICCAD*, 1987, pp. 88-91.

[Hitc83] R. Hitchcock, G. Smith and D. Cheng, "Timing Analysis of Computer-Hardware," *IBM Journal of Research and Development*, Jan. 1983, pp. 100 - 105.

[Kehl93] M. Khellah, S. Brown and Z. Vranesic, "Modelling Routing Delays in SRAM-based FPGAs," *Proc. Canadian Conf. on VLSI*, 1993, pp. 1042 - 1056.

[Kirk83] S. Kirkpatrick, C. Gelatt and M. Vecchi, "Optimization by Simulated Annealing," *Science*, May 13, 1983, pp. 671 - 680.

[Leve98] P. Leventis, "Using edif2blif Version 1.0," June 30, 1998. (*Available for download from http://www.eecg.toronto.edu/~leventi/edif2blif/edif2blif.html*).

[Luce98] Lucent Technologies, *FPGA Data Book*, 1998

[Marq99] A. Marquardt, "Using Cluster-Based Logic Blocks and Timing-Driven Packing to Improve FPGA Speed and Density", *FPGA*, 1999, pp 37-46.

[Meta92] Meta-Software, *Hspice User's Manual*, 1992.

[Nag95] S. Nag and R. Rutenbar, "Performance-Driven Simultaneous Place and Route for Row-Based FPGAs", ICCAD 1995, pp. 332 - 338.

[Okam96] T. Okamoto and J. Cong, "Buffered Steiner Tree Construction with Wire Sizing for Interconnect Layout Optimization," *ICCAD*, 1996, pp. 44 - 49.

[Padi98] K. Padalia, "Implementation of Grayscale Conversion for Video Image Processing on the Transmogrifier-2a," *Personal Communication.*

[Rama94] S. Raman, C. Liu, and L. Jones, "A Delay Driven FPGA Placement Algorithm," ACM *Proceedings Euro-DAC with Euro-VHDL*, 1994, pp. 277 - 282.

[Ries95] B. Riess and G. Ettelt, "SPEED: Fast and Efficient Timing Driven Placement," *IEEE International Symposium on Circuits and Systems*, 1995

[Rose90] J. Rose, R. J. Francis, D. Lewis and P. Chow, "Architecture of Programmable Gate Arrays: The Effect of Logic Block Functionality on Area Efficiency," *IEEE Journal of Solid State Circuits*, Oct. 1990, pp. 1217 - 1225.

[Rose91] J. Rose and S. Brown. "Flexibility of Interconnection Structures for Field-Programmable Gate Arrays," *JSSC*, March 1991, pp. 277 - 282.

[Rose93] J. Rose, A. El Gamal and A. Sangiovanni-Vincentelli, "Architecture of Field-Programmable Gate Arrays," *Proceedings IEEE,* vol. 81, no. 7, July 1993, pp. 1013 - 1029.

[Sauc93] G. Saucier, D. Brasen, J. Hiol, "Partitioning with Cone Structures," *ICCAD* 1993, pp. 236 - 239.

[Sent92] E. M. Sentovich et al, "SIS: A System for Sequential Circuit Analysis," *Tech. Report No. UCB/ERL M92/41,* University of California, Berkeley, 1992.

[Swar95] W. Swartz and C. Sechen, "Timing Driven Placement for Large Standard Cell Circuits," *DAC,* 1995, pp. 211 - 215.

[Swar98a] J. Swartz, V. Betz and J. Rose, "A Fast Routability-Driven Router for FPGAs," *FPGA,* 1998, pp. 140 - 149.

[Swar98b] J. Swartz, "A high-Speed Timing-Aware Router for FPGAs," *M.A.Sc. Thesis, University of Toronto,* 1998.

[Vant98] Vantis Corporation, "VF1 Field Programmable Gate Array," *Preliminary Data Sheet,* 1998.

[Vant99] O. Agrawal et. al. , "An Innovative, Segmented High Performance FPGA Family with Variable-Grain-Architecture and Wide-gating Functions," *FPGA,* 1999, pp. 17 - 26.

[West93] N. Weste and K Eshraghian, *Principles of CMOS VLSI Design; A System Perspective; Second Edition, Addison Wesley,* 1993.

[Xili94] Xilinx Inc., *The Programmable Logic Data Book,* 1994.

[Xili97] Xilinx Inc., "XC5200 Series of FPGAs", *Data Book,* 1997.

[Xili98] Xilinx Inc., "Virtex 2.5 V Field Programmable Gate Arrays", *Advance Product Data Sheet,* 1998.

[Yang91] S. Yang, "Logic Synthesis and Optimization Benchmarks, Version 3.0," *Tech. Report,* Microelectronics Center of North Carolina, 1991.

[Ye98] A. Ye, "Procedural Texture Mapping on FPGAs", *M.A.Sc. Thesis, University of Toronto,* 1998

[Yous90] H. Youssef and E. Shragowitz, "Timing Constraints for Correct Performance," *ICCAD*, 1990, pp. 24 - 27