

A Segmented Parallel-Prefix VLSI Circuit with Small Delays for Small Segments (Full Version)

Bradley C. Kuszmaul
MIT CSAIL
bradley@mit.edu
MIT CSAIL, The Stata Center
32 Vassar St., 32-G766
Cambridge, MA 02139
339-223-0680

May 23, 2005

Abstract

Segmented parallel prefix circuits can be used to perform many interesting computations in VLSI, including carry-lookahead, and out-of-order execution of instructions, typically with $O(\log N)$ gate delay and $O(\sqrt{N})$ wire delay for an N -input circuit. Since VLSI circuits have bounded fan in and have a constant signal propagation speed, those bounds are tight for the general problem of computing a value across an entire chip. However, for many problems, the actual data dependencies for a particular computation may require data to cross only a small fraction of an entire chip. For example, when adding random numbers using carry lookahead, the chances are small that a carry will propagate more than $O(\log N)$ positions. This paper presents a VLSI circuit for segmented parallel prefix with gate delay $O(\log S)$ and wire delay $O(\sqrt{S})$ for segment size S , and total area $O(\log N)$. Thus, for example, for the problem of adding random numbers, in most cases, the addition would complete with only $O(\log \log N)$ gate delay and $O(\sqrt{\log N})$ wire delay. More generally, if the length of the longest chain of carry generates is k , then the circuit shown here has $O(\log k)$ gate delays, and $O(\sqrt{k})$ wire delay. By using this technique, segmented parallel prefix circuits spanning a large VLSI area may be able to run significantly faster in the average case than they do in the worst case.

Note: A one-page announcement of this result appears in SPAA 2005.

1 Introduction

One of the limitations to increasing clock speed for microprocessors is the time required for a VLSI circuit to perform an addition of two integers [11, 1]. Fast-carry-lookahead circuits implement addition of N -bit numbers in order $O(\log N)$ gate delays, and, with the proper VLSI layout, in area $O(N)$ and with $O(\sqrt{N})$ wire delay, assuming that signals propagate along wires at a constant velocity. That worst-case delay is incurred when the carry is propagated from the least significant bit all the way to the most significant bit, in which case the high-order bit of the answer depends on every bit of the input.

In the worst case data inputs, both the gate delay and wire delay performance bounds are optimal: Circuits have bounded fan-in, implying $\Omega(\log N)$ gate delay; and the circuit requires area at least $\Omega(N)$ area, implying that the distance traveled between the farthest input bit and the high-order output is $\Omega(\sqrt{N})$. This paper presents a VLSI circuit whose performance depends on the length, S , of the longest carry propagation in the actual numbers being added. In particular, the circuit incurs $O(\log S)$ gate delays, $O(\sqrt{S})$ wire delays, and area $O(N)$. For example, when adding random numbers, $S = O(\log N)$ with high probability, and the circuit can compute its answer in only $O(\log \log N)$ gate delays and $O(\sqrt{\log N})$ wire delay.

Circuits similar to fast-carry-lookahead circuits can be used to implement many other functions needed by microprocessors [6], and can even be used to implement the entire functionality of out-of-order execution for a microprocessor [8, 7]. The generalization of fast-carry-lookahead used for this wide range of problems is called *segmented prefix*, and parallel circuits for computing a segmented prefix computation are well known [3, Exercise 30-1]. The rest of this paper focuses on the general problem of implementing segmented parallel prefix circuits whose runtime depends on the “segment size”, rather than on the size of the entire circuit.

A segmented prefix computation is defined in terms of a binary, associative operator \otimes . The computation takes an input sequence $x = \langle x_0, x_1, \dots, x_{n-1} \rangle$ of values from the domain of \otimes and a *segmentation sequence* $s = \langle s_0, s_1, \dots, s_{n-1} \rangle$ of values from the domain $\{0, 1\}$ with $s_0 = 1$. The prefix computation produces as output a sequence $y = \langle y_0, y_1, \dots, y_n \rangle$ where

$$y_i = x_{k_i} \otimes x_{k_i+1} \otimes \dots \otimes x_i$$

where

$$k_i = \max\{j : j \leq i \text{ and } s_j = 1\}.$$

For example, if we the operator \otimes is addition then we might see the following computation:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$x =$	12,	4,	7,	3,	2,	1,	5,	4,	4,	1,	1,	1,	1,	1,	1,	1)
$s =$	1,	0,	0,	0,	1,	0,	1,	1,	0,	1,	0,	1,	0,	1,	0,	1)
$y =$	12,	12+4,	12+4+7,	12+4+7+3,	2, 2+1,	5,	4, 4+4,	1, 1+1,	1, 1+1,	1, 1+1,	1, 1+1,	1, 1+1,	1, 1+1,	1, 1+1,	1, 1+1,	1)
$=$	12,	16,	23,	26,	2,	3,	5,	4,	8,	1,	2,	1,	2,	1,	2,	1)

with vertical bars drawn to help show where the segment boundaries are.

The input is partitioned into *segments* by the segmentation sequence. A segment is an interval of integers, and each segment’s beginning is defined by the corresponding bit in the segmentation sequence. In our example, the segments are 0–3, 4–5, 6, 7–8, 9–10, 11–12, 13–14, and 15.

One of the simplest segmented prefix circuits is shown in Figure 1. The circuit includes a sequence of multiplexers (MUXs), each controlled by the corresponding segment bit. If $s_i = 1$ then the i th MUX chooses the input that computes x_i , otherwise it chooses the input which is $y_{i-1} \otimes x_i$. This circuit can compute y in time $O(n)$ operator delays, where an operator delay is the time it takes to compute $a \otimes b$. For many interesting segmented prefix problems, such as carry lookahead, the operator delay is $O(1)$ gate delays, and so the overall delay of the circuit in Figure 1 is $O(n)$ gate delays.

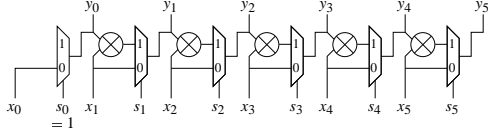


Figure 1: A segmented prefix circuit with $O(N)$ gate delays.

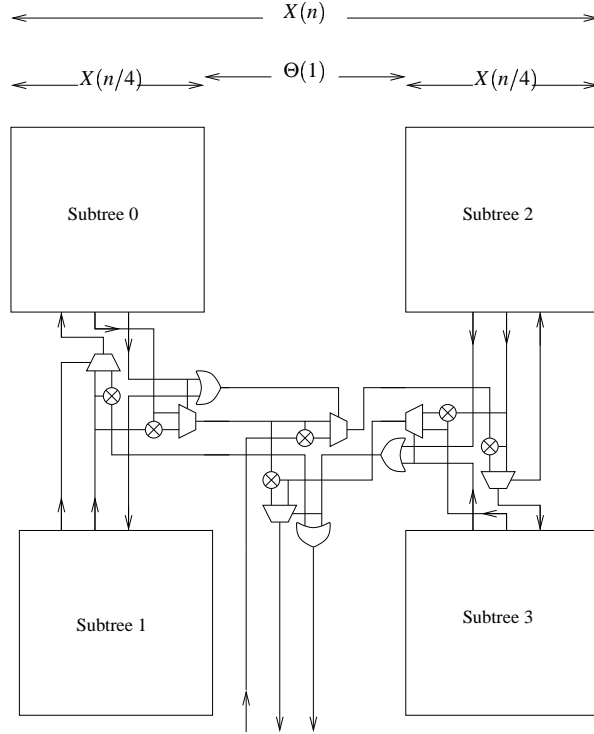


Figure 2: The general case for a tree-structured segmented prefix circuit. A circuit of n inputs is composed from four circuits each with $n/4$ inputs, and the four subcircuits are connected together in a so-called “H-tree” layout. The side-length recurrence derived from the lengths shown at the top of the figure.

In fact, a closer analysis of the linear-time circuit suggests that the circuit can sometimes run faster than time $O(N)$. For example, if all the segment bits are 1, then the outputs are all stable in time $O(1)$. Generally, if the longest segment is of size $O(S)$, then this circuit runs in time $O(S)$ rather than time $O(N)$. One of the ideas of this paper to perform this kind of analysis for tree-structured circuits like fast carry lookahead, reducing their runtime from, $O(\log N)$ to $O(\log S)$.

Figure 2 shows a tree-structured circuit and layout for computing a segmented prefix with only $O(\log N)$ operator delays and $O(\sqrt{N})$ wire delay. (For a complete tutorial on how this parallel circuit implements segmented prefix see [3].) In this recursive construction each subtree has two outputs and one input. At a leaf, the outputs are x_i and s_i , and the input is y_i .

As can be seen in Figure 2, the side-length $X(n)$ of a segmented parallel prefix circuit with, when laid out in VLSI, can be expressed as a recurrence relation:

$$X(n) = \begin{cases} O(1) & \text{if } n = 1, \\ O(1) + 2X(n/4) & \text{otherwise.} \end{cases}$$

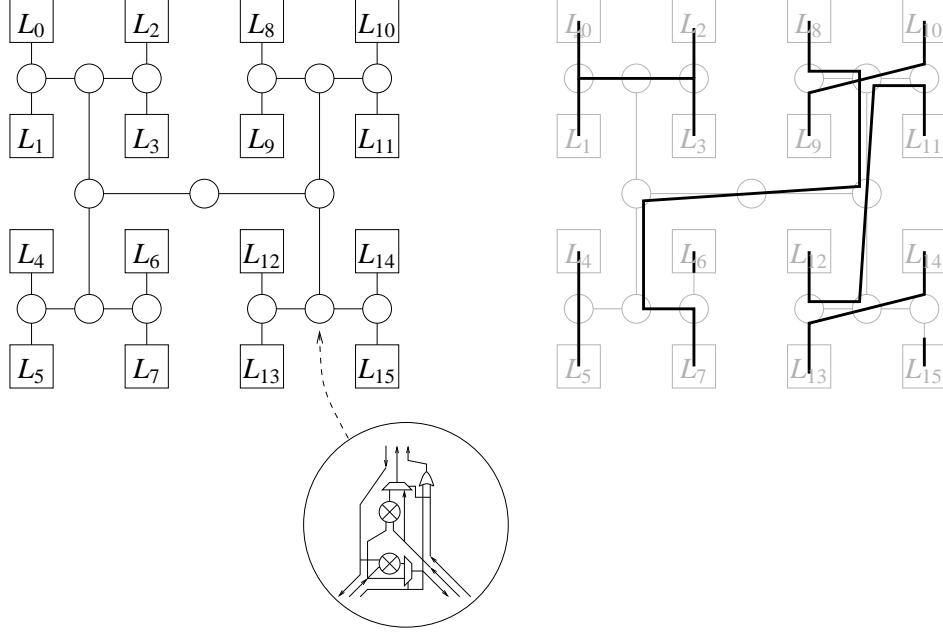


Figure 3: A 16-node H-tree layout of a segmented parallel prefix tree, and the paths used for the example inputs at the beginning of this paper.

This recurrence has solution $X(n) = O(\sqrt{n})$. Thus the area of the circuit is $O(n)$, which is optimal, since the circuit contains $O(n)$ operator circuits. H-tree layouts were analyzed in [9], and have been used for many years, for example in the layout of clock distribution trees on printed circuit boards.

To compute the wire delay, we need to calculate the longest path through the circuitry from the inputs to the outputs. We assume that signals propagate along wires at a constant speed. In many CMOS circuit technologies, that is typically near 5% of the speed of light. It turns out that the longest path essentially is the one from input x_0 to output y_{N-1} , and the signal must travel monotonically to the right and downward for a total distance of $O(\sqrt{N})$. This $O(\sqrt{N})$ bound is tight for the worst case, since the output x_N may depend on all the inputs, and the inputs must be distributed across a layout that has size at least $\Omega(\sqrt{N})$ on a side, which means that some signal must travel at least distance $\Omega(\sqrt{N})$.

What performance bounds, depending on S rather than N , can we hope for with this layout? Consider Figure 3, which shows an example of a 16-input segmented parallel prefix tree. Each of the circles contains two MUXs, two \otimes operators, and an OR gate. Each of the squares contains a leaf. Each of the wires connecting the tree together contains a value going toward the root (the sum of the subsegment ending in the subtree), a value going away from the root (the sum of the subsegment ending in the previous subtree), and a boolean value going toward the root (indicating whether there is a segment boundary in the subtree.) The right side of Figure 3 shows the paths followed for the particular inputs given in the example at the beginning of the paper. For example, the value from Leaf 4 needs to travel only a short distance to get to Leaf 5, but, the value from Leaf 7 must travel all the way to the root to get to Leaf 8.

Clearly some of the segments can run quickly. For example, not only is the path shorter for the segment containing Leaves 4 and 5, but the circuit is faster too. The output y_5 is simply $y_5 = x_4 + x_5$. the circuit implements that summation using only short wires (the wires traveled to compute y_5 are shown as thick lines in Figure 3, connecting Leaf 4 to Leaf 5) and a few gates. Similarly to compute output y_3 the signals need only travel to the root of the upper left subtree, instead of to the root of the entire tree. Leaves 7 and 8

suffer the full $\log N$ gate delay and \sqrt{N} wire delay to compute y_8 , however.

Thus, in general, if a segment happens to lie within a subtree, then to compute that segment’s outputs the signals will not leave the subtree, and hence it may be possible to compute the values in time substantially faster than $O(\sqrt{N})$. Unfortunately, if the alignment of the segment does not match that of the tree structure of the circuit, the signals must traverse long wires even though the segment is small.

This paper shows how to construct a segmented parallel prefix circuit with a “locality” property. Specifically, no matter how the segments are aligned, if a segment is of length S , then to compute the last output of that segment the signals must travel only distance $O(\sqrt{S})$, and the data need only pass through $O(\log S)$ of the MUXs, gates, and \otimes operators.

In fact, some outputs are ready even sooner. For example, the earlier outputs of a segment need to travel even less distance. More precisely, to compute y_i , given that the segment containing y_i starts at element k_i , the delay to compute y_i is $O(\log(i - k_i))$ gate delays and $O(\sqrt{i - k_i})$ wire delay. This locality property is the best one can hope for, since y_i depends on values from $i - k_i$ different inputs, and there is no way to lay out that many inputs into an area of diameter less than $\Omega(\sqrt{i - k_i})$.

This paper is organized as follows. Section 2 shows how to lay out a tree so that the distance between nodes is small enough to have a chance: If the geometric distance is too large, the wire delay will be too large. Then Section 3 shows how to reduce the number of gate and operator delays to $O(\log S)$, where S is the length of the longest segment, instead of $\log N$, where N is the size of the entire circuit. Introducing the necessary new gates and wires adds some additional area and wire length, but it turns out not to increase the area beyond $O(N)$ and the wire delay beyond $O(\sqrt{k})$. Section 4 shows the proof that the circuit works this fast, and Section 5 concludes with a brief discussion of related and future work.

2 Layout

This section explains how to lay out the tree to achieve compact segments. A segment, $i-j$, is *compact* if there it has a bounding box of size $O(\sqrt{j-i})$. Compact segments are a necessary condition to achieve the desired result that the wire delay for each segment is only $O(\sqrt{j-i})$. After addressing the layout Section 3 shows how to modify the circuit to compute the right value, and analyzes the impact on the layout.

Figure 4 shows how to lay out a 64-leaf parallel-prefix tree so that every segment is “compact”. That is the geometric distance between node L_i and L_j is $O(\sqrt{|i-j|})$. The layout for a 64-leaf tree is shown in Figure 4.

The layout is constructed as follows. For each tree size, we construct two different layouts, which we call the *inner layout*, and the *outer layout* for the tree. Both layouts fit in a square, and have the property that the lowest-numbered leaf is at one corner of the square, and the highest-numbered leaf is in an adjacent corner. Both layouts have their external wires, which connect the subtree to another subtree, routed to the center of an edge. The inner layout routes the wires to the center of the edge between the input and the output. The outer layout routes the wires to the center of an edge that is adjacent to either the input, or adjacent to the output, but not both. Figure 5 shows the two layouts, with a graphical notation to show where the lowest-numbered leaf, the highest-numbered leaf, and the external connections are placed.

The construction is performed recursively. Figure 6 shows how to build an outer layout and an inner layout containing 4^{D+1} leaves, given layouts with 4^D leaves. Note that when creating an outer layout, extra space is required between the sublayouts to route the external wire. That extra space does not change the recurrence for the side length, however, and the side length remains $O(\sqrt{N})$ for an N -leaf layout.

The proof that segments are compact in this layout is a simple variation of the one shown in the appendix.

It turns out to be easy to arrange that node L_0 is also adjacent to node L_{N-1} , by using four inner layouts at the outermost level of the recursion. This allows an implementation of *cyclic* segmented parallel prefix (in

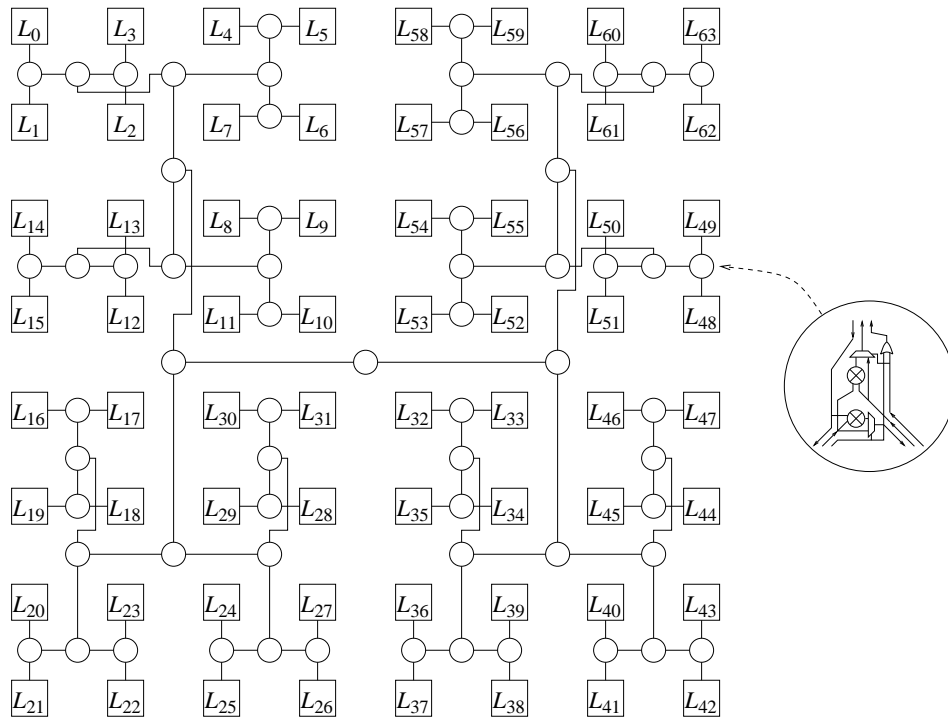


Figure 4: Layout of a 64-leaf tree with compact segments. To build a parallel prefix circuit each internal node of the tree is replaced with the circuit shown in the inset.

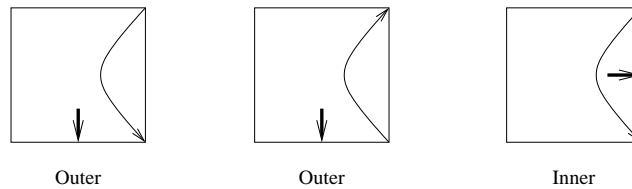


Figure 5: Inner and outer tree layouts. The thin curved arrow shows where the lowest-numbered and highest-numbered leaves are: The lowest numbered leaf is at the base of the thin arrow, and the highest-numbered leaf is at the arrowhead. The thick arrow shows where the external connections are routed out of the layout.

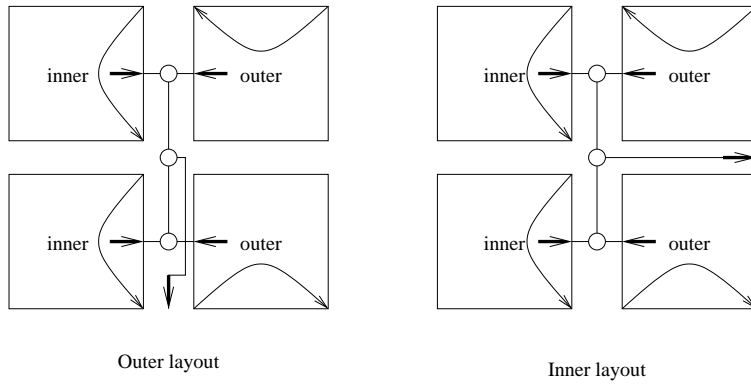


Figure 6: The recursive construction of inner and outer layouts.

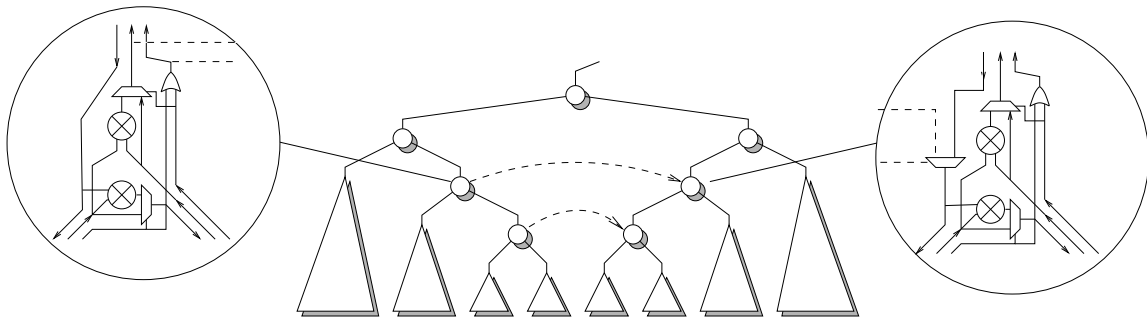


Figure 7: Shortcuts (shown as dashed lines) are added wherever two subtrees are adjacent to each other in the left-to-right ordering, but they do not share a parent. The shortcut signals come from the left and go to the right. Entire subtrees are represented as triangles. The inset on the left shows where the shortcut signals come from in a tree node. The inset on the right shows what is done with the shortcut signals—An extra MUX is added to the node to select the shortcut signal if the segment bit is 1. The idea is that if there is a segment boundary on the tree to the left, then the shortcut carries that information to the tree on the right, where it is selected by the MUX. Since the signal did not travel all the way up the tree, it arrives much more quickly.

which segments are allowed to wrap around from node L_{N-1} back to L_0 [5]) to run in time that is a function of the segment size.

3 Shortcut logic

Now that we have a layout that makes each segment physically compact, so it is conceivable that the circuit could compute each segment’s results quickly, this section shows how to modify the logic to take “shortcuts”.

The basic idea is illustrated in Figure 7, which shows a collection of subtrees ordered from left-to-right. At each depth, if a subtree’s successor is its sibling, nothing needs to be done. But if the subtree’s successor at the same depth is a distant cousin, then the shortcut signal is inserted.

Shortcuts are added everywhere that two subtrees “want” to be adjacent, but do not have the same parent. Two subtrees “want” to be adjacent if the last leaf of the first subtree is node L_i , and the first leaf of the second subtree is node L_{i+1} , for some i .

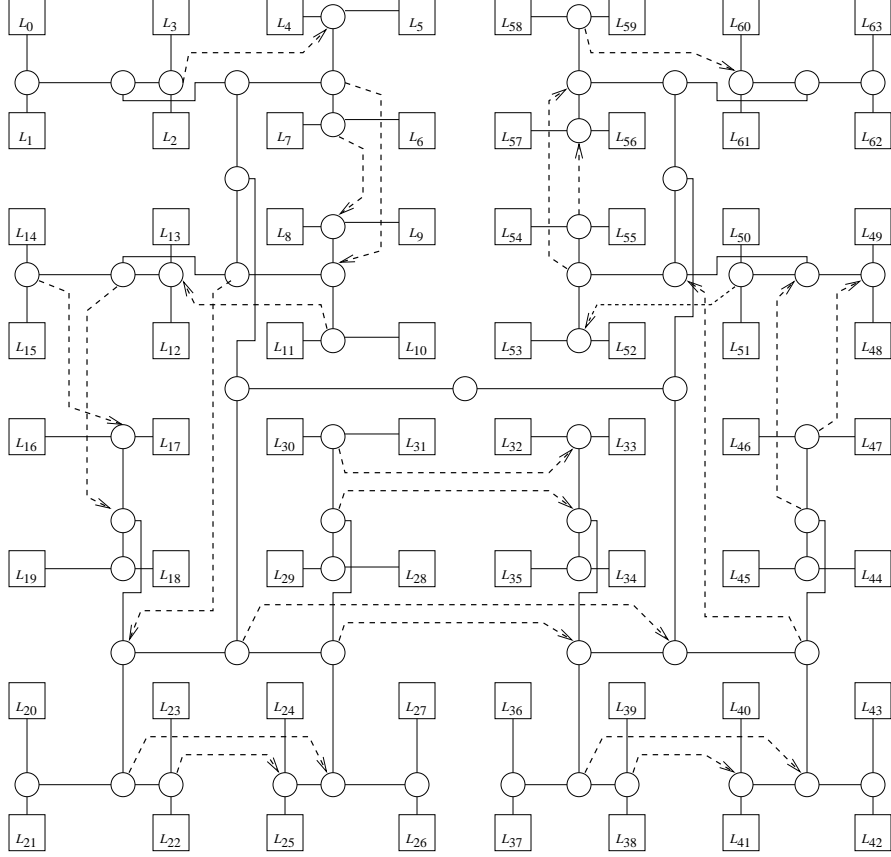


Figure 8: Local layout 64 with shortcuts. The shortcut wires are shown as dashed arrows.

Figure 8 shows the complete layout, including all the shortcuts, for a tree with 64 leaves. The root of the subtree containing leaves L_8-L_{11} wants to be adjacent to Subtree $L_{12}-L_{15}$, and in fact their parent is the same, so no shortcut is needed. Subtree L_12-L_{15} wants to be adjacent to Subtree L_16-L_{19} , but their least common ancestor is two more levels up the tree.

When we add the shortcut, we are adding gates and wires to the VLSI circuit, which may increase the size of the circuit. The gates have no asymptotic impact, since they only increase the size of the nodes shown as circles by a constant factor, but the wires could potentially add up to a lot of area. In Figure 8, extra space has been added to the layout to make room for the shortcut wires. Compare Figure 4, which is drawn at the same scale without the shortcuts.

How much extra space do the shortcut wires take? At each level of the tree, containing k leaves, we insert up to $O(\log k)$ new wires crossing horizontally, and up to $O(\log k)$ new wires running vertically. To make space, for each of those new wires, we insert a channel by “spreading” the circuit, using the technique of [9]. For example, in Figure 8, to make space for the shortcut from the parent of $L_{20}-L_{23}$, in the bottom left corner, extra space has been added across the whole row. The fact that the same extra space can be used for the shortcut wires from $L_{36}-L_{39}$ means that the side length recurrence is now

$$Y(n) = \begin{cases} O(1) & \text{if } n = 1, \\ O(\log n) + 2Y(n/4) & \text{otherwise.} \end{cases}$$

which also has solution $Y(n) = O(\sqrt{n})$. Thus the extra wires do not asymptotically increase the circuit size,

or the length of any wire.

4 The performance theorem

The circuit, as laid out with shortcuts has the following optimality property.

Theorem 1 *The distance that a signal must travel to communicate from Leaf L_i to Leaf L_j is $O(\sqrt{|i-j|})$ (which is an improvement over the previous best bound of $O(\sqrt{N})$) and that communication is performed by traveling through only $O(\log|i-j|)$ gates and \otimes operators.*

The proof of Theorem 1 is appended to the paper.

If the \otimes operator is a constant number of gates, then total gate delay is $O(\log|i-j|)$. Even for the addition problem shown in the beginning, where \otimes is the addition operator, the circuit can be made to run with only $O(\log|i-j|)$ gate delays by using carry-save addition. Of course, if the operator is more complex, such as floating point multiplication, then the delay is not measured in gate delays, but is instead measured in operator delays.

Analogous results apply to 3-dimensional technologies. For example, in a 3-dimensional packaging technology, the same idea allows a layout in which the distance for a signal to travel from L_i to L_j is only $O(|i-j|^{1/3})$.

5 Conclusion

This paper shows how to construct a segmented parallel prefix circuit in which the signals are stable sooner for some data inputs than for others. To actually take advantage of the improved speed requires a suitable timing scheme. If the circuit is globally clocked, then the clock speed must be limited to $\Omega(\sqrt{n})$ so that the worst case dependencies can be satisfied. One approach is to use a self-timed approach. Cheng et al [2] describe carry-lookahead adder circuitry with $O(\log N)$ worst case gate delay using self-timed circuitry, and $O(\log \log N)$ average case gate delay for addition of random numbers. In contrast to [2], which focuses only on gate delays, this work focuses on the VLSI performance, which includes gate delays, wire delay, and chip area. Also, unlike [2], which addresses only fast carry lookahead, this work shows how to speed up any segmented parallel prefix circuit, or indeed any parallel prefix circuit in which there is a “zero” value that causes outputs to the right not to depend on outputs to the left.

Greenberg [4] showed how to implement fat-tree routing network with shortcuts that are similar to the shortcuts used here. One could imagine building a system using Greenberg’s router and the parallel prefix tree presented here.

The locality property achieved here for parallel prefix VLSI circuits is similar to the “finger” property for search trees described by Sleator and Tarjan[12].

In the past, parallel prefix circuits have been important for parallel computers (see, for example, [10]), but we came across this problem while investigating the scaling of superscalar processors [7]. It turns out that almost everything done in a superscalar processor can be done asymptotically optimally using segmented parallel prefix circuits [8, 6]. Since the current VLSI roadmaps suggest t We believe that fast parallel prefix circuits will become even more important to microprocessors as VLSI technology scales up further.

Acknowledgments

I had a fun discussion with Andy Glew, now at AMD, and Dana S. Henry, which motivated my solution to this problem.

References

- [1] D. Brooks and M. Martonosi. Dynamically exploiting narrow width operands to improve processor power and performance. In *The Fifth International Symposium on High Performance Computer Architecture*, pp. 13–22, Orlando, Florida, Jan. 1999.
- [2] F.-C. Cheng, S. H. Unger, and M. Theobald. Self-timed carry-lookahead adders. *IEEE Transactions on Computers*, 49(7):659–672, July 2000.
- [3] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms*. The MIT Press, Cambridge, MA, 1990.
- [4] R. I. Greenberg. The fat-pyramid and universal parallel computation independent of wire delay. *IEEE Transactions on Computers*, 43(12):1358–1364, 1994.
- [5] D. S. Henry and B. C. Kuszmaul. Cyclic segmented parallel prefix. Ultrascalar Memo 1, Yale University, 51 Prospect Street, New Haven, CT 06525, Nov. 1998. <http://bradley.csail.mit.edu/~bradley/papers/usmemo1.ps.gz>.
- [6] D. S. Henry, B. C. Kuszmaul, G. H. Loh, and R. Sami. Circuits for wide-window superscalar processors. In *Proceedings of the 27th Annual International Symposium on Computer Architecture*, pp. 236–247, Vancouver, British Columbia, June 12–14, 2000.
- [7] D. S. Henry, B. C. Kuszmaul, and V. Viswanath. The Ultrascalar processor—an asymptotically scalable superscalar microarchitecture. In *The Twentieth Anniversary Conference on Advanced Research in VLSI (ARVLSI'99)*, pp. 256–273, Atlanta, GA, 21–24 Mar. 1999. <http://bradley.csail.mit.edu/~bradley/papers/usmemo3.pdf>.
- [8] B. C. Kuszmaul, D. S. Henry, and G. H. Loh. A comparison of asymptotically scalable superscalar processors. *Theory of Computer Systems (TOCS)*, 35:129–150, 2002.
- [9] C. E. Leiserson. *Area-Efficient VLSI Computation*. The MIT Press, 1982. ACM Doctoral Dissertation Award 1982.
- [10] C. E. Leiserson, Z. S. Abuhamdeh, D. C. Douglas, C. R. Feynman, M. N. Ganmukhi, J. V. Hill, W. D. Hillis, B. C. Kuszmaul, M. A. St. Pierre, D. S. Wells, M. C. Wong, S.-W. Yang, and R. Zak. The network architecture of the Connection Machine CM-5. *Journal of Parallel and Distributed Computing*, 33(2):145–158, 1996. Portions of this work were previously reported in 4th Annual Symposium on Parallel Algorithms and Architectures (SPAA '92), pp. 272–285, San Diego, CA, June 1992., <http://bradley.csail.mit.edu/~bradley/papers/jpdc96.ps.Z>.
- [11] T. Liu and S.-L. Lu. Performance improvement with circuit-level speculation. In *Proceedings of the 33rd Annual ACM/IEEE International Symposium on Microarchitecture*, pp. 348–355, Monterey, California, 2000.
- [12] D. D. Sleator and R. E. Tarjan. Self-adjusting binary search trees. *Journal of the ACM*, 32(3):652–686, July 1985.

A APPENDIX: Proof of Theorem 1

The reviewers can skip reading these proofs if they desire.

Proof: Define a k -subtree as a subtree of height k . That is, it includes, for some a , the leaves numbered $a2^k$ through $(a+1)2^k - 1$. In this case we call Leaf $L(a2^k)$ the first leaf of the k -subtree, and we call Leaf $L((a+1)2^k - 1)$ the last leaf of the k -subtree.

To finish the proof we need a few lemmas.

Lemma 2 *The distance from any leaf to the root of its k -subtree is exactly $\sqrt{2^k}$. (From here on out, the distance is measured in units in which one is the size of a leaf tile or the width of a wire, which we assume are the same distance. It is easy to modify the statement of the lemmas and their proofs to handle the situation in which a leaf's side-length is not the same as a wire width.)*

Proof: Write recurrence relation for the wire length and solve it, using for example, techniques from [3].

Lemma 3 *If Leaf L_i is the first leaf in a k -subtree, and Leaf L_j is in the same k -subtree, then the distance that a signal must travel to communicate between Leaves L_i and L_j is at most $2\sqrt{2(j-i)}$.*

Proof: We can assume without loss of generality that k is the smallest integer such that Leaves L_i and L_j are in the same k -subtree. (The signal to communicate between Leaves L_i and L_j need not leave the k -subtree, so no larger value k is relevant.) This implies that $2^k < 2(j-i)$, since if $2^k \geq 2(j-i)$ then Leaves L_i and L_j would both be in the same $(k-1)$ -subtree, which would be a contradiction. To communicate between L_i and L_j thus requires traveling to the root of the k -subtree containing them, and then back down, which is a distance of $2\sqrt{2^k}$, which is less than $2\sqrt{2(j-i)}$. Q.E.D.

Lemma 4 *If Leaf L_j is the last leaf in its k -subtree, and Leaf L_i in the same k -subtree, then the distance that a signal must travel to communicate between Leaves L_i and L_j is at most $2\sqrt{2(j-i)}$.*

Proof: Similar to Lemma 3.

Lemma 5 *If $0 \leq a \leq 1$ then $\sqrt{a} + \sqrt{1-a} \leq \sqrt{2}$.*

Proof: The maximum value of $\sqrt{a} + \sqrt{1-a}$ can be calculated by setting its derivative to zero.

$$\frac{d(\sqrt{a} + \sqrt{1-a})}{da} = \frac{1}{2\sqrt{a}} - \frac{1}{2\sqrt{1-a}},$$

which is zero exactly when $a = 1/2$. We thus have $\sqrt{a} + \sqrt{1-a} \leq \sqrt{1/2} + \sqrt{1-1/2} = 2\sqrt{1/2} = \sqrt{2}$. Q.E.D.

Lemma 6 *If $i \leq l \leq j$ then $\sqrt{l-i} + \sqrt{j-l} \leq \sqrt{2(j-i)}$.*

Proof: Let $a = (l-i)/(j-i)$. Then

$$\begin{aligned} \sqrt{l-i} + \sqrt{j-l} &= \sqrt{a(j-i)} + \sqrt{(1-a)(j-i)} \\ &= (\sqrt{a} + \sqrt{1-a})\sqrt{j-i} \\ &\leq \sqrt{2}\sqrt{j-i}. \end{aligned}$$

Q.E.D.

Lemma 7 *The distance through the network between any two leaves L_i and L_j ($i \leq j$) is at most $1 + 4\sqrt{j-i}$.*

Proof: Choose k so that $2^{k-1} < j-i \leq 2^k$. If L_i and L_j are in the same k -subtree then their distance in the network is at most $2\sqrt{2^k}$ which is less than or equal to $2\sqrt{2(j-i)}$ which is less than $4\sqrt{j-i}$.

If L_i and L_j are not in the same k -subtree, then let Leaf L_l be the last leaf in the k -subtree containing L_i . Note that Leaf $L(l+1)$ is the first leaf in the k -subtree containing Leaf L_j . By the previous lemmas we know that that the distance traveled from L_i to L_l is at most $2\sqrt{2(l-i)}$, and the distance from $L(l+1)$ to L_j is at most $2\sqrt{2(j-l-1)}$. The distance between Leaves L_l and $L(l+1)$ is one, since we added the shortcut between such leaves. Thus the total distance is

$$\begin{aligned}
 D &\leq 2\sqrt{2(l-i)} + 1 + 2\sqrt{2(j-l-1)} \\
 &\leq 1 + 2\sqrt{2(l-i)} + 2\sqrt{2(j-l)} \\
 &= 1 + 2\sqrt{2}(\sqrt{l-i} + \sqrt{j-l}) \\
 &\leq 1 + 2\sqrt{2}\sqrt{2(j-i)} \\
 &= 1 + 4\sqrt{j-i}.
 \end{aligned}$$

Q.E.D.

Now to finish proving Theorem 1 we simply note that $1 + 4\sqrt{j-i}$ is $O(\sqrt{|j-i|})$. Q.E.D.