Parallel Prefix addition

The parallel prefix adder presented next, performs the addition of two binary numbers in time of complexity *O(log n)* and linear cost *O(n).*

Lets notice the following fact: If the carry bits c_n, ..., c_0 are known, performing the addition is simplified; to get the final result $<s^n>$ would suffice to perform $s_i = XOR(a_i, b_i, c_i)$ for $i \in [0, \ldots, n-1]$ [1]. And vice-versa, if the result is known, the carry bits can be easily obtained by applying the XOR function: $c_i = XOR(a_i, b_i, s_i)$.

Conclusion: calculating the result bits is almost as complex as calculating the carry bits.

We consider now the problem of computing the carry bits. The input consists of $a, b \in \{0, 1\}^n$, $c_{IN} \in \{0, 1\}$. The output consists of the carry bits $c_n, \ldots, c_0 \in \{0, 1\}$

Carry calculation

Define $\sigma_{-1}, \ldots, \sigma_n \in \{0, 1, 2\}$ by:

$$\sigma_i = \begin{cases} a_i + b_i & n \geq i \geq 0 \\ 2 & i = -1, c_0 = 1 \\ 0 & i = -1, c_0 = 0 \end{cases}$$

Sometimes the values 0, 1 and 2 are named ‘kill’, ‘propagate’ and ‘generate’ respectively.
We’ll use regular expression notation. In regular expressions the symbol ‘*’ indicates that the symbol preceding it can appear 0 or more times in the expression. For example:

$1^*2 \equiv \{2, 12, 112, 1112, \ldots, 111\ldots12, \ldots\}$

Claim: For every $i \in [-1, \ldots, n-1]$, $c_{i+1} = 1$ if and only if there exists $j \leq i$ such that

$(\sigma_i, \sigma_{i-1}, \ldots, \sigma_j) \in (1^*2)$.

First direction is simple, for every *i*, given $j \leq i$ such that $(\sigma_i, \sigma_{i-1}, \ldots, \sigma_j) \in (1^*2)$, it can be seen that the carry generated at σ_j is propagated through, up to σ_i.
The second direction will be proved by induction, given that $c_{i+1} = 1$ we’ll find $j \leq i$ such that

$(\sigma_i, \sigma_{i-1}, \ldots, \sigma_j) \in (1^*2)$

Proof 1 (by induction on *i*):

Base: *i=-1.*
By definition $c_0 = 1 \Leftrightarrow \sigma_{-1} = 2$.

1. For *i=n* it’s just $s_n=c_n$

Step: $c_{i+1} = 1 \Rightarrow a_i + b_i + c_i \geq 2$

We consider 3 cases:

(1) $\sigma_i = 2$: In this case both input bits are 1, and carry is generated.

We choose $j = i$

(2) $\sigma_i = 1$:In this case one of the input bits is 1.

If $c_{i+1} = 1$ the induction assumption for *i* leads the proof for *i+1*:

If only one of the input bits is one, and carry is generated, at bit *i*, then the carry at bit *i-1*, has to be 1. Therefore exist $j<i$ that complies with the claim, according to the induction assumption

$$c_{i+1} = 1 \text{ and } \sigma_i = 1$$
$$\Rightarrow c_i = 1$$
$$\Rightarrow \exists j < i-1 \ (\sigma_{i-1}, \ldots, \sigma_j) \in (1^* 2)$$
$$\Rightarrow (\sigma_i, \sigma_{i-1}, \ldots, \sigma_j) \in (1^* 2)$$

(3) $\sigma_i = 0 \Rightarrow c_{i+1} = 0$

Parallel Prefix Calculation

Now we want to find out if there exists *j* such that $(\sigma_i, \ldots, \sigma_j) \in (1^* 2)$. We'll build an automata (aka state machine - SM or finite state machine - FSM) consisting of only two states (which means only one bit is needed to implement it). The input of the FSM, at step *i*, is σ_i. The initial state is S_0. State S_1 indicates, at step *i*, that there exists $j \leq i$ such that $(\sigma_i, \ldots, \sigma_j) \in (1^* 2)$; state S_0 indicates there is not such $j \leq i$.

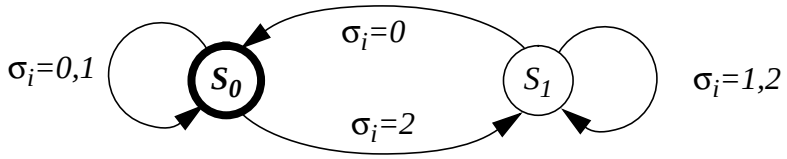

Figure 1. Carry Calculation State Machine

Lets prove, by induction on *i*, that the FSM works correctly, i.e, that after reading a given sequence of inputs $\sigma_{-1}, \ldots, \sigma_i$ the FSM is in state S_k if and only if $c_{i+1} = k$.

It is simple to show the first direction, after reading the sequence $\sigma_{-1}, \ldots, \sigma_i$, if $c_{i+1} = 1$ then the FSM is in state $S(i) = S_1$:

Lets consider the two cases:

(1) Assume $c_{i+1} = 1$. According to proof 1, exists *j* such that $(\sigma_i, \ldots, \sigma_j) \in (1^* 2)$.

From Figure 1 can be seen that when σ_j (=2) is read the FSM goes to state S_1 (regardless of its previous state), any further σ (=1) that is read, does not change the state,

therefore $S(i) = S_1$

(2) Assume $c_{i+1} = 0$. Using the opposite argument than before, there exists no *j* such that $(\sigma_i, \ldots, \sigma_j) \in (1^*2)$ (otherwise, according to proof 1, $c_{i+1} = 1$).
The first element of every sequence is σ_{-1}, which, by definition can be either 0 or 2. Then for every *j*:

$$(\sigma_i, \ldots, \sigma_j) \notin (1^*2)$$
$$\Rightarrow (\sigma_i, \ldots, \sigma_j) \in (1^*0)$$

From Figure 1 can be seen that when σ_j (=0) is read the FSM goes to state S_0 (regardless of its previous state), any further σ (=1) that is read does not change the state, therefore $S(i) = S_0$.

Given the FSM in Figure 1 we will build a network that receives, as inputs, σ_i for all *i*, and calculates, in parallel, the sequence of states $S^{n-1}, \ldots, S^0$ that the FSM will go through for those inputs.

Parallel Prefix Function

Lets define the $\bullet$ function, and its truth table:

Table 1: '$\bullet$' truth table

a* • *b		***b***		
		0	*1*	*2*
a	*0*	***0***	***0***	***0***
	1	***0***	***1***	***2***
	2	***2***	***2***	***2***

The $\bullet$ operator is an associative operator, i.e. $a \bullet b \bullet c = (a \bullet b) \bullet c = a \bullet (b \bullet c)$, but it is not commutative, i.e. $a \bullet b \neq b \bullet a$. Non commutativity can be immediately seen from the fact that the table above is not a symmetric matrix. Table 2 proves associativity[1].

1. The symbol 'ϕ' in the table represents "don't care"

Table 2: Associativity of '•' function

a	b	c	$(a \bullet b)$	$(a \bullet b) \bullet c$	$(b \bullet c)$	$a \bullet (b \bullet c)$
0	ϕ	ϕ	0	**0**	ϕ	**0**
2	ϕ	ϕ	2	**2**	ϕ	**2**
1	0	ϕ	0	**0**	0	**0**
1	2	ϕ	2	**2**	2	**2**
1	1	0	1	**0**	0	**0**
1	1	2	1	**2**	0	**2**
1	1	1	1	**1**	1	**1**

Lets define the following notation: $\prod_{k=-1}^{i} \sigma_k \equiv \sigma_i \bullet \sigma_{i-1} \bullet \ldots \bullet \sigma_0 \bullet \sigma_{-1}$

Claim: $c_{i+1} = 1 \Leftrightarrow \prod_{k=-1}^{i} \sigma_k = 2$

We already proved that $\exists j \le i \quad (\sigma_i, \ldots, \sigma_j) \in (1^* 2) \quad \Leftrightarrow \quad c_{i+1} = 1$

Direction 1: Given that $c_{i+1} = 1$ lets prove that for *k* from *-1 to i*, the multiplication of all σ_k, has to have result 2.

$$c_{i+1} = 1 \Rightarrow \prod_{k=-1}^{i} \sigma_k = 2$$

Applying the truth table of •: If $c_{i+1} = 1$ then exist $j \le i$ such that $(\sigma_i, \ldots, \sigma_j) \in (1^* 2)$ and by opening the regular expression and multiplying its elements, using $1 \bullet 2 = 2$, $1 \bullet 1 = 1$ and associativity, we see that $1 \bullet 1 \bullet \ldots \bullet 1 \bullet 2 = 2$, therefore $\prod_{k=-1}^{i} \sigma_k$ can be divided in two: up to element *j* and from element j down to *-1:*

$$\Rightarrow \prod_{k=-1}^{i} \sigma_k = \left(\prod_{k=j}^{i} \sigma_k \right) \bullet \left(\prod_{k=-1}^{j-1} \sigma_k \right) = (1 \bullet \ldots \bullet 2) \bullet (?) = 2 \bullet ? = 2$$

Direction 2: Given that the multiplication of all σ_k, for *k* from *-1 to i*, has the result 2, lets prove that $c_{i+1} = 1$

$$\prod_{j=-1}^{i} \sigma_j = 2 \Rightarrow c_{i+1} = 1$$

Lets look at the given prefix:

$$\prod_{j=-1}^{i} \sigma_j = 2 \quad \equiv \quad \sigma_i \bullet \ldots \bullet \sigma_{-1} = 2 \qquad \textbf{[EQ - 1]}$$

We'll show that the above equation is true if $(\sigma_i, \ldots, \sigma_{-1}) \in (1^* 2.^*)$.

Afterwards, by proof 1: $\exists j \le i \quad (\sigma_i, \ldots, \sigma_j) \in (1^* 2) \quad \Leftrightarrow \quad c_{i+1} = 1$, we can conclude that $c_{i+1} = 1$

Proof 2 (by induction on *i*) of: $(\sigma_i, \ldots, \sigma_{-1}) \in (1^* 2.^*)$

Base, *k=-1*:
According to **EQ-1:** $\sigma_{-1} = 2 \in (1^* 2.^*)$

Step, lets assume it is true for $k \le i-1$ and prove it for $k = i$ for all three values of σ_i:

$$(\sigma_i, \ldots, \sigma_{-1}) = (\sigma_i, \sigma_{i-1}, \ldots, \sigma_{-1}) = (\sigma_i, 1^* 2.^*)$$

$$= \begin{cases} (21^* 2.^*) = 2 \in (1^* 2.^*) & \sigma_i = 2 \\ (11^* 2.^*) \in (1^* 2.^*) & \sigma_i = 1 \\ \text{contradiction with EQ-1} & \sigma_i = 0 \end{cases}$$

The Problem of Parallel Prefix Computation

Inputs: $a_{n-1} \ldots a_0$
$\{a_i\} \in \Sigma$
$\bullet : \Sigma \times \Sigma \to \Sigma$

Outputs: The set of all prefixes of $a_{n-1} \bullet a_{n-2} \bullet \ldots \bullet a_1 \bullet a_0$, namely:

$$\text{For every } i\text{: } 0 \le i \le n \qquad \left\{ \prod_{j=0}^{i} a_j \right\}$$

We transformed this way, the problem of calculating the carry bits into the problem of calculating all bits prefixes.

(Fast) Algorithm for Parallel Prefix Calculation (PPC)

Stage 1: Reduction**.** Transform a problem of size *n* into smaller problems of size *n/2*

Stage 2. Combine the outputs of the smaller problems with the original inputs and get the final output

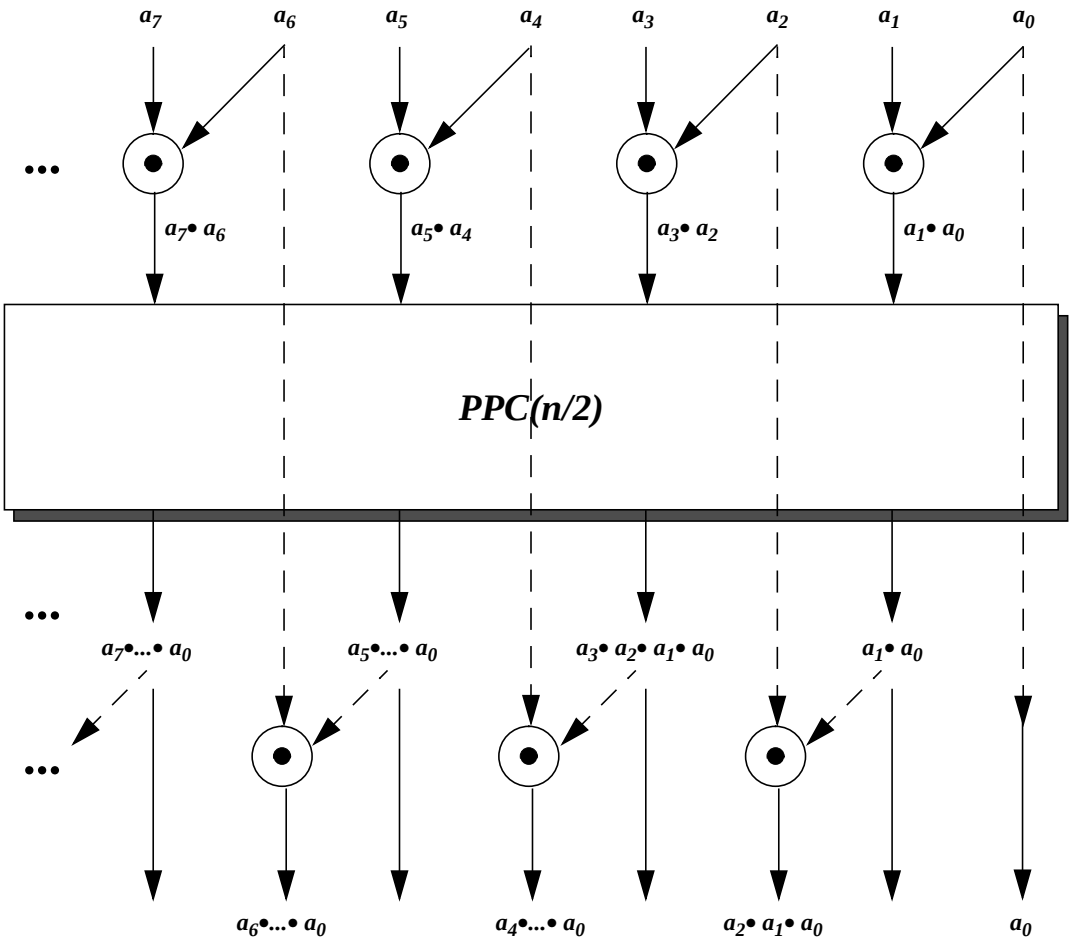

Figure 2. PPC network

From Figure 2 it can be seen that the outputs of the *PPC(n/2)* network are the odd prefixes, and that, by combining them with the even inputs, we can obtain the even prefixes.

Time and Cost complexity

Assuming that $T(\ \bullet\ ) = O(1)$ and $C(\ \bullet\ ) = O(1)$

$$\boldsymbol{T(n)} = T\left(\frac{n}{2}\right) + 2 \cdot T(\ \bullet\ ) = 2 \cdot \log(n) \cdot T(\ \bullet\ )$$

$$= \boldsymbol{O(\log n)}$$

$$\boldsymbol{C(n)} = C\left(\frac{n}{2}\right) + (n-1) \cdot C(\ \bullet\ )$$

$$= \left((n-1) + \left(\frac{n}{2} - 2\right) + \left(\frac{n}{4} - 2\right) + \ldots\right) \cdot C(\ \bullet\ )$$

$$\leq 2 \cdot n \cdot C(\ \bullet\ ) = \boldsymbol{O(n)}$$
