A Note on the Addition of m k-Bit Numbers

Günter Hotz and Alexander Gamkrelidze

Department of Computer Science, University of the Saarland
66041 Saarbrücken, Germany
e- mail: hotz@cs.uni-sb.de sandro@cs.uni-sb.de

Abstract—**In this paper, we give a method to construct a parallel adder of m k-bit numbers based on the school method of addition. As a result, we get a parallel adder with depth $4 \cdot \log n + 12$ and hardware cost $k \cdot m$. We apply our method to develop a parallel multiplier with asymptotical depth $c \cdot \log n$ and hardware cost n^2.**

I. Introduction

The problem of addition of m k- bit keys has an old, rich history in computer science. Because of its importance in computer arithmetics and arithmetic units, fast and low- cost adders had been studied extensively. Mostly, it is studied in the context of VLSI design, where gate count and delay, as well as the limited fan- out are performance factors.

In this paper, we show that the parallelized school method is efficient in terms of hardware cost (number of gates) and time (maximal delay).

We represent the k- bit numbers $\zeta^i = (\xi_{i1}, ..., \xi_{ik})$, $i = 1, ..., m$ as a matrix (ξ_{ij}) and apply our ideas to it iteratively. In each big step, we reduce the number of addends from m_{i-1} to $m_i = \lceil \log m_{i-1} \rceil$ $(m_0 = \lceil \log m \rceil)$. Applying it iteratively, we get m, $\lceil \log m \rceil$, $\lceil \log \lceil \log m \rceil \rceil$ and so on, up to two elements that can be added by a conventional adder. Each big step we do by a recursive counting of bits.

The main idea can be traced back to the observation in the early sixties. In this observation, the cuts through the boolean net representing a function f define the generators of boolean algebras, and a sequence of parallel steps leads to a sequence of descending boolean algebras that converges to the boolean algebra $< f_1, ..., f_n >$ defined by $f = [f_1, ..., f_n]$. The sequence of descending algebras corresponds to an ascending sequence of incident automorphism groups that fix pointwise the elements of corresponded algebra.

In our calculations, we ignore the round offs, so the theoretical results do not match that in practice for $m < 32$. But, as we will show, this is not the case for larger number of elements.

In the last section, we will apply the ideas presented in the first sections to build an efficient multiplier circuit.

II. Basic Notations and Mathematical Background

Let $\zeta^i = (\xi_{i1}, ..., \xi_{ik})$, $i = 1, ..., m$ be the binary numbers to be added and $\xi^j = (\xi_{1j}, ..., \xi_{mj})$ the j- th column of the matrix (ξ_{ij}) of these numbers. Let $sum(\xi^j)$ be the binary sum of the elements of ξ^j. For the sake of simplicity, we write the binary sequences both as an encoded number or as a sequence, and do not use the square brackets [and].

We get then

$$ad(\zeta_1, ..., \zeta_m) = \sum_{i=1}^{k} sum(\xi^i) \cdot 2^{k-i}.$$

Further, let $i = l + j \cdot \lambda$, where l and λ are uniquely defined, if we set $\lambda = \lceil \log m \rceil$ and $0 \leq l < \lambda$.
Using these notations, we can represent the above sum as follows:

$$\begin{aligned} & ad(\zeta_1, ..., \zeta_m) = \\ & = \sum_{0 \leq l < \lambda} \sum_{0 \leq j \cdot l < k} sum(\xi^{l+j\cdot\lambda}) \cdot 2^{k-(l+j\cdot\lambda)} = \end{aligned}$$

$$= \sum_{0 \leq l < \lambda} \underbrace{\left(\sum_{0 \leq j \cdot l < k} sum(\xi^{l+j\cdot\lambda}) \cdot 2^{(k-l)-(l+j\cdot\lambda)} \right)}_{S^l} \cdot 2^l$$

Hence,

$$ad(\zeta_1, ..., \zeta_m) = \sum_{0 \leq l < \lambda} S^l \cdot 2^l.$$

We reduced the calculation of $ad(\zeta_1, ..., \zeta_m)$ to the calculation of the sum of λ elements, since S^l can be represented as a binary sequence

$$\sum(\xi^l), \sum(\xi^{l+\lambda}), \sum(\xi^{l+2\lambda}), ..., \sum(\xi^{l+j_l\cdot\lambda}).$$

Iterating this algorithm and reducing the number of addends up to two elements, we can calculate the total sum with a cheap adder, such as a Carry-Save adder.

III. Method of Construction

The reduction from m to two elements terminates in at most s steps, if the following inequality holds:

$$\left. 2^{2^{\cdot^{\cdot^{\cdot^{2}}}}} \right\} s\text{- times} \geq m$$

We denote, as usual, its inverse function by $\log^*(m)$. The maximal number of the steps in the recursion will be then $s = \lceil \log^*(m) \rceil$.

Each step of the reduction is divided in further reductions, as shown in fig. 1.

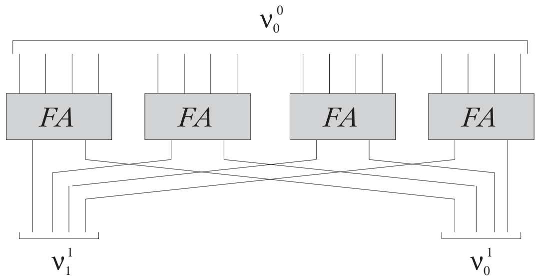

Fig. 1.

The sequence ν_0^1 contains the carry bits of the first step, ν_1^1 — the sums $mod2$. We iterate the algorithm by applying it to ν_0^1 and ν_1^1 respectively.

Ignoring the round offs of the divisions, we get:

$$|\nu_0^1| = |\nu_1^1| = \frac{m}{3}, \quad |\nu_0^2| = |\nu_2^2| = \frac{m}{3^2},$$

$$|\nu_1^2| = |\nu_0^1| + |\nu_1^1| = \frac{2}{3^2} \cdot m.$$

Continuing this scheme for l steps, we get the sequence

$$\nu_0^l, \nu_1^l, ..., \nu_l^l$$

with following features:

$$|\nu_0^{l+1}| = \frac{1}{3}|\nu_0^l|, \quad |\nu_{l+1}^{l+1}| = \frac{1}{3}|\nu_l^l|,$$

$$|\nu_i^{l+1}| = \frac{1}{3}(|\nu_{i-1}^l|+|\nu_i^l|) \text{ for } i = 1, ..., l-1.$$

It can be seen that $|\nu_i^l|$ has the following binomial representation:

$$|\nu_i^l| = \frac{1}{3^l} \cdot \binom{l}{i} \text{ for } i = 0, ..., l.$$

The iteration terminates if

$$m \cdot \frac{1}{3^t} \cdot \binom{t}{i} \leq 1 \text{ for } i = 0, ..., t$$

holds.

Further, we use the estimation that follows the Sterling formula

$$\binom{t}{i} \leq \left(\frac{e \cdot t}{i}\right)^i,$$

where e is the base of the natural logarithm.

We overestimate t, if we use following inequality:

$$3^t \geq m \cdot \left(\frac{e \cdot t}{i}\right)^i$$

or

$$t \cdot \log 3 \geq \log m + i(\log e + \log t - \log i).$$

The right side reaches its maximum if the following equation holds:

$$\log i = \log t - 1$$

That means, we can replace these inequality with

$$t \cdot \log 3 \geq \log m + \frac{1}{2} \cdot \log t \cdot (1 + \log e).$$

Then

$$t \geq \frac{1}{\log 3} \cdot \log m + \frac{1 + \log e}{2 \cdot \log 3} \cdot \log t$$

Let $m = 2^n$ and assume that $t = n$. We get

$$t \geq \frac{1}{\log 3} \cdot t + \frac{1 + \log e}{2 \cdot \log 3} \cdot \log t$$

and further

$$\frac{t}{\log t} \geq 2.1$$

for sufficiently large t.

IV. Time and Hardware Cost

The inequality in the previous section holds for $\forall t \geq 5$. Since each step of 2^n addends ($n \geq 5$) could be calculated in depth n, we get following estimation by adding the depths of $\log^*(m)$ steps of the whole circuit:

We can reduce the number of elements to five addends in depth

$$t_1 = \sum_{i=1}^{i_0} \log^i(m) \text{ with } \log^{i_0}(m) = 5.$$

These five elements could be reduced to two in three FA steps. Since $T(\text{FA}) = 4$,

the depth of the whole circuit will be $t = 4 \cdot (t_1 + 3)$.
Since

$$\frac{1}{\log m} \cdot \sum_{i=1}^{i_0} \log^i(m) \longrightarrow 1,$$

we get

Theorem 1: The summation of m binary numbers can be reduced to the summation of two binary numbers with the asymptotical depth $t = 4 \cdot \log m + 12$ and hardware cost $k \cdot m$.

We now calculate the hardware cost of the circuit (now and in further calculations, we take the cost of FA as unit).
Depending on iteration step i, it will be

$$C^i = \frac{1}{3} \cdot \left(\frac{2}{3}\right)^{i-1} \cdot m.$$

Through all these steps, the hardware cost will be $C_{1,m} < \frac{m}{3} \sum_i \left(\frac{2}{3}^i\right)$.

For k- bit keys, it will be $C_{k,m} < k \cdot m$ for one big step.

The complete hardware cost could be estimated as follows:

$$C^m < C_{k,m} + C_{k_1,m_1} + \cdots$$

where $m_1 = \lceil logm \rceil$, $m_{i+1} = \lceil \log m_i \rceil$, $k_0 = i$, $k_{i+1} = k_i + \lceil \log m_i \rceil$.

Hence,

$$C^m < C_{k,m} + C_{k_1,m_1} + \cdots =$$

$$= k \cdot m \cdot (1 + \frac{k_1}{k} \cdot \frac{m_1}{m} + \frac{k_2}{k} \cdot \frac{m_2}{m} + \cdots).$$

For $k \geq 32$, $C^m < k \cdot m \cdot (1 + 2 \cdot \frac{m_1}{m}) \longrightarrow k \cdot m$, $\lim m \to \infty$.

Following table shows the relations between m (number of addends) and the hardware cost of the circuit for some m.

m	32	64	128
$\frac{1}{k} \cdot C^m$	32,5	32,25	32,11

V. The Multiplier

Here we apply our idea to calculate the product of two n- bit numbers.

Let $(\xi_1, ..., \xi_n)$ and $(\zeta_1, ..., \zeta_n)$ be the numbers to be multiplyed.
We construct the matrix

$$\begin{pmatrix} 0 & \cdots & 0 & 0 & \eta_1^1 & \cdots & \eta_{n-1}^1 & \eta_n^1 \\ 0 & \cdots & 0 & \eta_1^2 & \eta_2^2 & \cdots & \eta_n^2 & 0 \\ . & & . & & . & & . & . \\ . & & . & & . & & . & . \\ . & & . & & . & & . & . \\ \eta_1^n & \cdots & \eta_{n-1}^n & \eta_n^n & 0 & \cdots & 0 & 0 \end{pmatrix}$$

where $\eta_i^j = \xi_i \cdot \zeta_j$.

Adding the elements
$\kappa_i = (\underbrace{0, ..., 0}_{n-i \text{ times}}, \eta_1^i, ..., \eta_n^i, \underbrace{0, ..., 0}_{i-1 \text{ times}})$, we get

the product $(\xi_1, ..., \xi_n) \cdot (\zeta_1, ..., \zeta_n)$.

So, we get

Theorem 2: The product of two n- bit numbers can be calculated in asymptotical depth $c \cdot \log n$ and hardware cost n^2.

This method is not asymptotically optimized, but it is slightly better than the direct application of the Wallace– Tree Method [Wall 64].

References

[SchStr 71] A. Schönhage, V. Strassen "Schnelle Multiplikation großer Zahlen" Computing 7, 1971 pp. 281 – 192 (in German)

[Wall 64] C. S. Wallace "A Suggestion for a fast multiplier"
IEEE Transactions on Computers 13, 1964 pp. 14 – 17

[Weg 96] I. Wegener "Effiziente Algorithmen für Grundlegende Funktionen"
B. G. Teubner Stuttgart 1996 (in German)