International Research Journal of Computer Science (IRJCS)
Issue 10, Volume 2 (October 2015)
ISSN: 2393-9842
www.irjcs.com

Speeding up Multiplication Operation by using Parallel Processing Computers

IMAD MATTI BAKKO
Lecturer in Computer Science Dep. Alma'mon University College,
Baghdad, Iraq

Abstract— The conventional method for multiplication in digital number systems requires approximately c×n×n operations to multiply an n-digit number by an n-digit number, where c is a constant. This paper produces an approach to reduce the problem of multiply two 2n-bit numbers to three distinct multiplications of n-bit numbers; this approach will be faster than the traditional method on computers. The main advantages of this approach is speeding up the multiplication of the two 2n-bit numbers by assigning the three distinct multiplications to three distinct processors in parallel processing computer. The theory and the details are explained with many examples in this paper.

Keywords— Digit numbers multiplication, Parallel processing, Speed up metrics, Algorithm, program.

1. INTRODUCTION

The conventional method for multiplication in positional systems requires approximately c×n×m operations to multiply an n-digit number by an m-digit number, where c is a constant.

Let us assume for convenience that m = n , therefore , for multiplying two n-digit numbers it require an execution time proportional to n^2 as n increases.
Let us assume that we are working with integers expressed in binary notation.

If we have two 2n bit numbers:
$u = (u_{2n-1} \dots u_n , u_{n-1} .. u_0)_2$ and $v = (v_{2n-1} \dots v_{n-1} \dots v_0)$, [2]
We can write
$u = 2^n U_1 + U_0$, $v = 2^n V_1 + V_0$ …….. (1) [2]
Where $U_1 = (u_{2n-1} \dots u_n)_2$ is the "most significant half" of the number u [2]
And $U_0 = (u_{n-1} \dots u_0)_2$ is the "least significant half" of the number u [2]

Similarly
$V_1 = (v_{2n-1} \dots v_n)_2$ and $V_0 = (v_{n-1} \dots v_0)_2$. [2]
Now we have [2]
$uv = (2^{2n} + 2^n)U_1V_1 + 2^n(U_1 – U_0) (V_0 – V_1) + (2^n + 1)U_0V_0$ …. (2)
This formula reduces the problem of multiplying 2n-bit numbers to three multiplications, namely
U_1V_1 , $(U_1 – U_0) (V_0 – V_1)$ and U_0V_0
Plus some simple shifting and adding operations [2].
We can prove the equality in formula (2) above :

The left hand side :
$uv = (2^nU_1 + U_0) (2^n V_1 + V_0)$ from formula (1)
$= 2^{2n} U_1V_1 + 2^n U_1V_0 + 2^n U_0V_1 + U_0V_0$

The right hand side :
$= 2^{2n} U_1V_1 + 2^n U_1V_0 – 2^n U_1V_1 – 2^n U_0V_0 + 2^n U_0V_1 + 2^n U_0V_0 + U_0V_0$
$= 2^{2n} U_1V_1 + 2^n U_1V_0 + 2n U_0V_1 + U_0V_0$
Which is equal to the left hand side.
Example (1):
$2^3 \ 2^2 \ 2^1 \ 2^0$ $2^3 \ 2^2 \ 2^1 \ 2^0$
Let $u = (1 \ 1 \ 1 \ 1)_2$ and $v = (1 \ 0 \ 1 \ 0)_2$
Be two 2n = 4 bit binary numbers. According to formula (1) , we can find
$2^3 \ 2^2$
$U_1 = (u_{2n-1} \dots u_n) = (1 \ 1)_2$ is the "most significant half" of the number u.
$2^1 \ 2^0$
$U_0 = (u_{n-1} ..... u_0) = (1 \ 1)_2$ is the "least significant half" of the number u.
Similarly
$2^3 \ 2^2$
$V_1 = (v_{2n-1} \dots v_n) = (1 \ 0)_2$ is the "most significant half" of the number v.
$2^1 \ 2^0$

$V_0 = (v_{n-1} \ldots V_0) = (1 \quad 0)_2$ is the "least significant half" of the number v.
Now, from formula (1) , we can find
$u = 2^n U_1 + U_0 = 2^2 (1 \quad 1)_2 + (1 \quad 1)_2 = 4(3) + 3 = (15)_{10}$
$v = 2^n V_1 + V_0 = 2^2 (1 \quad 0)_2 + (1 \quad 0)_2 = 4(2) + 2 = (10)_{10}$
Using traditional method for multiplication , we find
$uv = (15)_{10} (10)_{10} = (150)_{10}$ left hand side of formula (2).

Now, from right hand side of formula (2), we find
$= (2^{2n} + 2^n)U_1 V_1 + 2^n (U_1 – U_0) (V_0 – V_1) + (2^n + 1)U_0 V_0$
$= (2^4 + 2^2) (1 \ 1)_2 (1 \ 0)_2 + 2^2 [(1 \ 1)_2 – (1 \ 1)_2] [(1 \ 0)_2 – (1 \ 0)_2]$
$+ (2^2 + 1) (1 \ 1)_2 (1 \ 0)_2$
$= (16 + 4) (3) (2) + 4 [3 – 3] [2 – 2] + (4 + 1) (3) (2)$
$= 120 + 0 + 30 = (150)_{10}$
Hence R.H.S = L.H.S.

From the above example, we conclude that the time required for execution using formula (2), is approximately proportional to $3 (n^2) \approx 3(2^2) = (12)_{10}$,

Since the problem of multiplication is reduced to three multiplications of n-bit numbers, namely
$U_1 V_1$, $(U1 – U_0) (V0 – V_1)$, $U_0 V_0$

While the time required for execution in conventional method for multiplication (positional number systems) is approximately proportional
To $(2n)^2 \approx (2x2)^2 = 4^2 = (16)_{10}$.

Example (2):
$2^3 \quad 2^2 \quad 2^1 \quad 2^0 \qquad\qquad 2^3 \quad 2^2 \quad 2^1 \quad 2^0$
Let $u = (1 \quad 1 \quad 1 \quad 0)_2$ and $v = (0 \quad 1 \quad 1 \quad 0)_2$

Be two 2n = 4 bit binary numbers. According to formula (1), we can find
$2^3 \quad 2^2$
$U_1 = (u2n_{-1} \ldots u_n) = (1 \quad 1)_2$ is the "most significant half" of the number u.
$2^1 \quad 2^0$
$U_0 = (u_{n-1} ..... u_0) = (1 \quad 0)_2$ is the "least significant half" of the number u.
Similarly
$2^3 \quad 2^2$
$V_1 = (v_{2n-1} \ldots v_n) = (0 \quad 1)_2$ is the "most significant half" of the number v.
$2^1 \quad 2^0$
$V_0 = (v_{n-1} \ldots V_0) = (1 \quad 0)_2$ is the "least significant half" of the number v.
Now , from formula (1) , we can find
$u = 2^n U_1 + U_0 = 2^2 (1 \quad 1)_2 + (1 \quad 0)_2 = 4(3) + 2 = (14)_{10}$
$v = 2^n V_1 + V_0 = 2^2 (0 \quad 1)_2 + (1 \quad 0)_2 = 4(1) + 2 = (6)_{10}$

Using traditional method for multiplication, we find
$uv = (14)_{10} (6)_{10} = (84)_{10}$ left hand side of formula (2).
Now, from right hand side of formula (2), we find
$= (2^{2n} + 2^n)U_1 V_1 + 2^n (U_1 – U_0) (V_0 – V_1) + (2^n + 1)U_0 V_0$
$= (2^4 + 2^2) (1 \ 1)_2 (0 \ 1)_2 + 2^2 [(1 \ 1)_2 – (1 \ 0)_2] [(1 \ 0)_2 – (0 \ 1)_2]$
$+ (2^2 + 1) (1 \ 0)_2 (1 \ 0)_2$
$= (16 + 4) (3) (1) + 4 (3 – 2) (2 – 1) + (4 + 1) (2) (2)$
$= 60 + 4 + 20 = (84)_{10}$
Hence R.H.S = L.H.S.
From the above example, we conclude that the time required for execution using formula (2), is approximately proportional to $3 (n^2) \approx 3x2^2 = (12)_{10}$,
Since the problem of multiplication is reduced to three multiplications of n-bit numbers, namely
$U_1 V_1$, $(U1 – U_0) (V0 – V_1)$, $U_0 V_0$
While the time required for execution in conventional method for multiplication (positional number systems) is approximately proportional
To $(2n)^2 \approx (2x2)^2 = 4^2 = (16)_{10}$.

2. THE ADVANTAGES OF MULTIPLICATION USING FORMULA (2)

- In addition to the reduction problem of multiplying two 2n-bit digit numbers to three multiplication as in formula (2) , the main advantages of this approaches is that we can speed up the multiplication of uv , by using parallel processing computer , that include three processors , say processor (1) , processor (2) , and processor (3) , and as follows :
- Assigning the multiplication of $U_1 V_1$ to processor (1) ,
- Assigning the multiplication of $(U_1 – U_0) (V_0 – V_1)$ to processor (2) , and
- Assigning the multiplication of U0 V_0 to processor (3). See Figure (1)

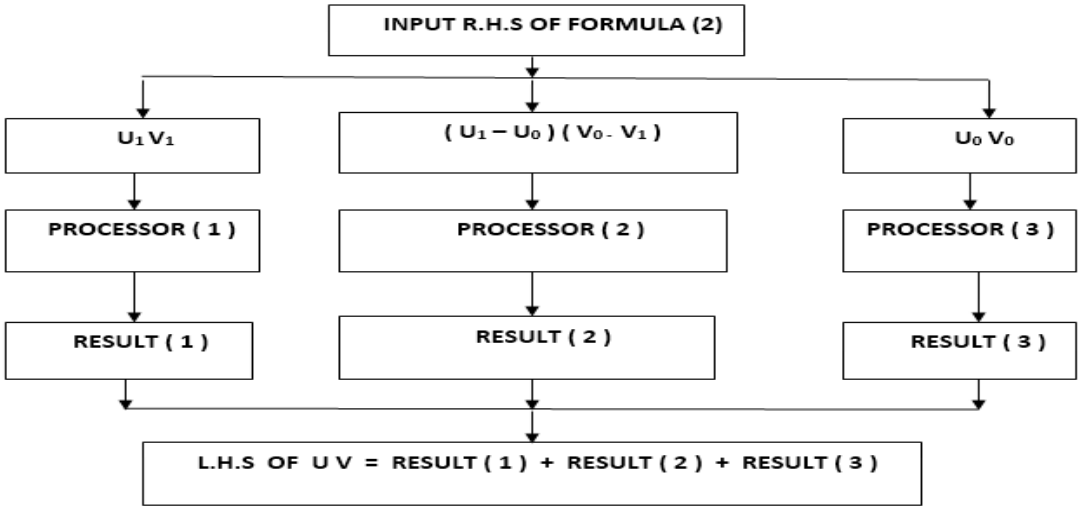

Figure (1) : parallel processing of formula (2)

3. ALGORITHM FOR MULTIPLYING TWO 4-BIT DIGIT NUMBERS (USING FORMULA 2)

a. Let u, v, uv , x1 , x2 , and x3 be 4-bit digit numbers.
b. Input u = $(u_3, u2, u_1, u_0)_2$.
c. Input v = $(v3, v_2, v_1, v_0)_2$.
d. Break u and v into two parts as follows:
   1. $U_1 = (u3\ u_2)_2$ is the most significant half .
   2. $U_0 = (u_1\ u_0)_2$ is the least significant half .
   3. $V_1 = (v_3\ v_2)_2$ is the most significant half .
   4. $V_0 = (v_1\ v_0)_2$ is the least significant half .

e. concurrently execute
   1. $x1 = (2^4 + 2^2)U_1 V_1$ on processor (1)
   2. $x2 = 2^2 (U_1 – U_0) (V_0 – V_1)$ on processor (2)
   3. $x3 = (2^2 + 1)U_0 V_0$ on processor (3)

f. uv = x1 + x2 + x3 on any idle processor (1) or (2) or (3)
g. output uv .

3.1 Testing elapsed time required in figure (1)

Using MatLab software, we introduce here a program to show and test the effect of the elapsed time required of the parallel processing of figure(1) in comparison with the conventional multiplication.

```
n=input ('enter n: ');              / number of bits representing u and v
u=input ('enter u: ');
v=input ('enter v: ');
u1=input ('enter u1: ');
u0=input ('enter u0: ');
v1=input ('enter v1: ');
v0=input ('enter v0: ');
tic
x1=(2^(2*n)+2^n)*u1*v1;  / first branch in figure(1)
toc
tic
x2=2*n*(u1-u0)*(v0-v1);  / second branch in figure(1)
toc
tic
x3=((2^n)+1)*u0*v0;  / third branch in figure(1)
toc
```

```
tic
x=x1+x2+x3;
toc
tic
uv=((2^(2*n))+2^n)*u1*v1+2^n*(u1-u0)*(v0-v1)+((2^n)+1)*u0*v0;
toc
```

3.1.1 Test data 1:

Suppose n=8, u=65535, v=43690
u_1=255, u_0=255, v_1=170, v_0=170
The results of the first running are:
X1=0.000011 sec
x2=0.000003 sec
x3=0.000006 sec
x=0.000003 sec
uv=0.000007 sec
The results of second running are:
x1=0.000011 sec
x2=0.000003 sec
x3=0.000005 sec
x=0.000003 sec
uv=0.000007 sec

3.1.2 Test data 2:

Suppose n=4, u=255, v=170
u_1=15, u_0=15, v1=10, v_0=10
The results of the first running are:
x1=0.000012 sec
x2=0.000003 sec
x3=0.000004 sec
x=0.000003 sec
uv=0.000006 sec
The results of second running are:
x1=0.000011 sec
x2=0.000003 sec
x3=0.000004 sec
x=0.000003 sec
uv=0.000006 sec

From the above results we can conclude that the elapsed time required for execution in uv (conventional multiplication method) is larger than the elapsed time required in each branch of figure (1) (parallel processing method).

Note: in Mat Lab software the results will be change in every new run.

4. EVALUATION OF SPEED UP AND EFFICIENCY OF MULTIPLICATION FOR FORMULA (2).

Speedup is defined by the following formula

$Sp = T_1 / T_p$ where

- p is the number of processors
- T_1 is the execution time of the sequential algorithm
- T_p is the execution time of the parallel algorithm with p processors

Efficiency is a performance metric defined as

$E_p = S_p / p = T_1 / pT_p$

It is a value , typically between zero and one.
Algorithms running on a single processor have an efficiency of 1.
While many difficult to parallelize algorithms have efficiency such as $1 / \ln_p$
That approaches zero as the number of processors increase [4] [5].

Now referring to the results of examples (1) and (2) , since the time required for execution using formula (2) , is approximately proportional to $3 (n^2) \approx 3x2^2 = (12)_{10}$, and the time required for execution in conventional method for multiplication (positional number systems) is approximately proportional

To $(2n)^2 \approx (2x2)^2 = 4^2 = (16)_{10}$,
hence

The speed up $S_p = T_1 / T_p = 12 / 16 = 0.750$
The efficiency $E_p = S_p / p = 0.750 / 3 = 0.250 < 1$

4.1 Amdahl' Law: speed up Metrics for formula (2)

Amdahl's law states that potential program speed up is defined the fraction of code(p) that can be parallelized
Speed up = 1 / (1 – p)

. if none of the code can be parallelized , p = 0
and the speed up = 1 (no speed up)
. if all the code is parallelized , p = 1 and the speed up is infinite
(in theory)
. if 50 % of the code can be parallelized , maximum speed up = 2 ,
Meaning the code will run twice as fast [7] .

Introducing the number of processors performing the parallel fraction of Work , the relationship can be modelled by
Speed up = 1 / [(P / N) + S]
Where

P = parallel fraction
N = number of processors
S = serial fraction [7].
Now referring to example (1), example (2) and figure (1):
P = 1 / 3 the parallel fraction (as illustrated in figure (1)).
N = 3 the number of processors.
S = 1 / 3 the serial fraction.
Hence, the potential program speedup:
Speedup = 1 / (1 – p) = 1 / [1 – (1 / 3)] = 1.5 (1st law) &
Speedup = 1 / [(P / N) + S] = 1 / [(1/3)/3 + (1/3)]
= 9 / 4 = 2.250 (2nd law)

4.2 The Actual Speedup

The speedup that can be achieved by parallel computer with N identical processors working concurrently on a single problem is at most N time faster than a single processor in a computer. In practice the speedup is much less, since some processors are idle at a given time. The actual speedup, ranging from a lower-bound (known a Minsky's conjecture) is $\log_2 n$ to upper-bound $n / \ln_n$ [2]. The parallel computer architecture model which is suitable for the multiplication problem is SIMD (Single Instruction stream, Multiple Data stream). These processors executes instructions at the same time [1] [3] [5] [6].

5. CONCLUSIONS

1. It's recommended to find a way to apply the Algorithm for multiplying two 4-bit digit numbers on computers with multiple processors, like using the algorithm as a shell attached to operating systems.
2. Formula (1) and (2) are concerning with integer numbers, it's recommended to find a formula that deals with fraction numbers.
3. Referring to point 2 above, it's recommended to write an algorithm that deals with fraction numbers .
4. Referring to the algorithm for multiplying 4-bit digit numbers mentioned earlier , we can extend the numbers as follows: $u = (u_7\ u_6\ u_5\ \dots\ u_2\ u_1\ u_0)_2$ and $v = (v_7\ v_6\ v_5\ \dots\ v_2\ v_1\ v_0)_2$ also we can extend for 16 bits numbers, 32 bits numbers, 64 bits numbers, and so on.
5. Formula (2) can be used for double-precision multiplication when we want a quadruple-precision result, and it will be a little faster than the traditional method on many computers.

REFERENCES

[1].M.Morris Mano, Charles R.Kime," Logical and Computer Design Fundamentals", fourth edition,Pearson Prentice Hall , 2008 .
[2].Knuth D., "The Art of Computer Programming", vol.2, Semi numerical Algorithms, Addison-Wesley, reading, mass, 1969.
[3].M. Morris Mano, "Computer System Architecture", 3rd edition, Prentice – Hall International inc. 2008.
[4].www.Google Chrome.com "speeding up computations by parallel computers" Speedup-Wikipedia, the free encyclopaedia.
[5].www.Google Chrome.com "speeding up computations parallel computers" [pdf] introduction to parallel computing. Frank Willmore , February , 2012.
[6].www.yahoo.com "Parallel computing" Wikipedia, the free encyclopaedia.
[7].www.google chrome.com speeding computations by parallel computers "Introduction to parallel computing".