ISSN (Online) 2394-2320

International Journal of Engineering Research in Computer Science and Engineering (IJERCSE)
Vol 3, Issue 6, June 2016

Scratchpad

[1] Vishwesh J [2] Neethi M V
[1]Assistant Professor, Computer Science Department,
GSSS Institute of Engineering & Technology for Women, Mysuru, Karnataka, India -570016
[2]Student, M.Tech, CCT branch, University of Mysore, Mysore

***Abstract* - Scratchpad is a high-speed internal memory used for temporary storage of calculations data and other work in progress. Scratchpad refers to a special high speed memory used to hold small items of data for rapid retrieval and these are employed for simplification of caching logic. In this paper the scratchpad algorithm is designed to hold a value temporarily and for faster results LUT is considered. The whole algorithm is designed by considering LUT as a case study**

Each time loop continues and the execution of the internal operations are concurrently stored in scratchpad, in this paper if the sum value is greater than 9 than the value which is stored in 10^{th} position is sent to the variable C, all the values which are stored in C is again stored in Temp before the value in C is overwritten.

***Index Terms*—Scratchpad, LUT**

I. INTRODUCTION

Scratchpad is a temporary memory which has high speed for data retrieval. It is a very fast intermediate storage which often supplements main core memory. Scratch-pad memories are small but very fast and tightly coupled memories, typically ranging from 4kB to 1MB in size. Unlike caches, transfer of data to and from these scratch-pad memories is under explicit software control.

It is well known that processor speeds are increasing at a significantly faster rate than external memory chips. This has led to increasing costs for memory accesses relative to processor clock rate, a phenomenon sometimes known as the memory wall. Since most applications exhibit temporal locality in data access patterns, memory access overheads can be significantly reduced by storing an application's working set in a smaller but much faster on-chip memory. Many architectures structure this on-chip memory as a cache, where movement of data between on- and off-chip memories is under hardware control. Caches perform very well under most situations, conveniently hiding the non-uniformity of the memory system from software, while decreasing average memory access latency. However, caches suffer from a number of penalties compared with non-associative memories, such as increased silicon area and energy requirements, increased access latency, complex coherency mechanisms for multi-core systems, and lack of real-time guarantees due to potential cache misses. These factors have led a number of processor architectures to provide explicit access to on-chip memory, in the form of scratch-pad memory.

They can be found in embedded and real-time processor architectures where they have been found to not only reduce power and silicon area requirements compared to an equivalent cache, but also potentially increase run-time performance. Mutli-core processors, especially heterogeneous architectures have also begun to incorporate scratch-pad memories in their designs.

A lookup table (LUT) is an array that replaces runtime computation with a simpler array indexing operation. The savings in terms of processing time can be significant, since retrieving a value from memory is often faster. The tables may be pre-calculated and stored in static program storage, calculated as part of a program's initialization phase, or even stored in hardware in application-specific platforms. Lookup tables are also used extensively to validate input values by matching against a list of valid (or invalid) items in an array.

Scratchpad is a memory array with decoding and the column circuitry logic. Considering LUT this scratchpad is designed to hold the values which we obtain in 10^{th} position during the calculations. The assumption here is that scratchpad memory occupies one unit of memory to store the value with respect to LUT. The scratchpad is used to store the values which exceeds 9(LUT considered is of size 9*9) to store the temporary values during calculations.

ISSN (Online) 2394-2320
International Journal of Engineering Research in Computer Science and Engineering (IJERCSE)
Vol 3, Issue 6, June 2016

II. LOOKUP TABLE CASE STUDY (LUT)

In computer science, a lookup table is an array that replaces runtime computation with a simpler array indexing operation. The savings in terms of processing time can be significant. The tables may be pre calculated and stored in static program storage, calculated (or "pre-fetched") as part of a program's initialization phase (memorization), or even stored in hardware in application-specific platforms. Lookup tables are also used extensively to validate input values by matching against a list of valid (or invalid) items in an array and, in some programming languages, may include pointer functions (or offsets to labels) to process the matching input.

Before the advent of computers, lookup tables of values were used by people to speed up hand calculations of complex functions, such as in trigonometry, logarithms, and statistical density functions

Early in the history of computers, input/output operations were particularly slow – even in comparison to processor speeds of the time. It made sense to reduce expensive read operations by a form of manual caching by creating either static lookup tables (embedded in the program) or dynamic pre-fetched arrays to contain only the most commonly occurring data items. Despite the introduction of system wide caching that now automates this process, application level lookup tables can still improve performance for data items that rarely, if ever, change.

This is known as a linear search or brute-force search, each element being checked for equality in turn and the associated value, if any, used as a result of the search. This is often the slowest search method unless frequently occurring values occur early in the list. For a one dimensional array or linked list, the lookup is usually to determine whether or not there is a match with an 'input' data value.

Storage caches (including disk caches for files or processor caches for either code or data) work also like a lookup table. The table is built with very fast memory instead of being stored on slower external memory, and maintains two pieces of data for a sub range of bits composing an external memory (or disk) address (notably the lowest bits of any possible external address).

The LUT which we have considered in this paper is represented as shown below.

Lookup Table (LUT)

	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
1	1	2	3	4	5	6	7	8	9
2		4	6	8	10	12	14	16	18
3			9	12	15	18	21	24	27
4				`16	20	24	28	32	36
5					25	30	35	40	45
6						36	42	48	54
7							49	56	63
8								64	72
9									81

Table 1 Lookup Table

LUT from the memory address to read or write, then the other piece contains the cached value for this address one piece (the tag) contains the value of the remaining bits of the address; if these bits match with. The other piece maintains the data associated to that address.

A single (fast) lookup is performed to read the tag in the lookup table at the index specified by the lowest bits of the desired external storage address, and to determine if the memory address is hit by the cache. When a hit is found, no access to external memory is needed (except for write operations, where the cached value may need to be updated asynchronously to the slower memory after some time, or if the position in the cache must be replaced to cache another address).

In this table best case is used to find out minimum numbers i.e. <10 numbers. it used take at least 1-9 numbers. These case only one times the execution takes place .because it checks only one times the conditions and comes out of the loop.

In worst case it maximum 9x9 i.e. up to last values. They used to check 45 position still the required value is found.

III. ALGORITHM

In this paper LUT is referred as case study to perform the calculations to store the values in temporary buff .The algorithm for scratchpad based on the concept of LUT is designed as follows

ISSN (Online) 2394-2320
International Journal of Engineering Research in Computer Science and Engineering (IJERCSE)
Vol 3, Issue 6, June 2016

```
1. Algorithm: design scratchpad (LUT case study)
2. Sum ()
3. {
4. Int sum;
5. Int c=0;
6. While (sum>=10)
7. Sum=sum-10;
8. C=c+1;
9.}
10. Temp[i] =c;
```

Listing 1 Algorithm for Scratchpad

The algorithm works as follows, each time it checks the conditions based on the LUT the positions to be identified & get the values if the values exceeds the condition it perform specified calculations until loop ends. In this algorithm we have initialized sum to 0,and here C stands for carry during calculation if carry occurs i.e., the value in the 10^{th} position is stored in the variable C and it is also initialized to 0, Here it first checks for the condition whether the sum which is obtained is greater than 9, if it is greater than it goes into the loop than checks for the condition if it is satisfied than it enters the loop than executes the statement sum=sum-10,after each execution of this statement the value of c is incremented this executes until the condition is dissatisfied and finally the value which is stored in C is sent to TEMP which holds all the values of C. In this way the scratchpad is designed.

The loop will be executed n+1 times because in best-case it executed only one time when conditions fails that will not executed. In every calculations (add, mul, sub, div) the conditions the loop executed as number of times until get the result. When the re∑∑quires sum>10 according to LUT table then it goes to be next stage of operations i.e. sum=sum-10, this condition continues the execution until the last value will be found. It may get the carry and it adds up with the next one. It continues sane execution until the final result. This algorithm calculates the sum &internal operations are stored in temporary buff called scratchpad**.**

The algorithm flow can be explained by taking an example

1. Taking 2 arrays for addition

A[i] =838
B[i] =445

2. The array of numbers are added and stores the result in sum array

Sum[i] =A[i] +B[i]
Sum[i] =8+5=13
Sum[i] =13

If the sum not exceeds >10 keep that as sum. 3. Each iteration the for loop check the condition i.e.

While
(sum[i]>=10)
While (13>=10)
yes Then
Subtract the s um[i] =sum[i]-
10 Temp[i] =13-10=3
Return 1 for next iteration
Then sum[i] =3 loop
continues

4. Next c=c+1 for next position. Loop continues execution until n-1 iterations.

IV.ANALYSIS OF ALGORITHM

The analysis of algorithm is the determination of the amount of resources (such as time and storage) necessary to execute them. Most algorithms are designed to work with inputs of arbitrary length. Usually, the efficiency or running time of an algorithm is stated as a function relating the input length to the number of steps (time complexity) or storage locations (space complexity).Algorithm analysis is an important part of a broader computational complexity theory, which provides theoretical estimates for the resources needed by any algorithm which solves a given computational problem. Exact (not asymptotic) measures of efficiency can sometimes be computed but they usually require certain assumptions concerning the particular implementation of the algorithm, called model of computation.

The algorithm of the scratchpad is analysed based on the time efficiency, based on the efficiency we can analyse whether the algorithm is efficient or not if its takes minimum time than the algorithm is efficient. In order to analyse the algorithm usually two algorithms are considered and those two are compared with time and space efficiencies which take the minimum time and space it is considered as efficient.

ISSN (Online) 2394-2320
International Journal of Engineering Research in Computer Science and Engineering (IJERCSE)
Vol 3, Issue 6, June 2016

The time constraint of the scratchpad algorithm is found in the following way.

Algorithm	Time	Frequency
While(sum>=10)	1	n+1
Sum=sum-10;	2	n
C=c+1;	2	n

Table 2 Time Constraint

Tα1*(n+1) +2n+2n
Tα5n+1

Time complexity of the entire algorithm is given by O(max{n})=O(n)

The time which we have computed is called as detailed computing time.

a. Analysis of Best-case and Worst-case Efficiencies

According to the prior analysis the efficiency of the algorithm with respect to time is calculated it can be represented by using graph by taking various values for n.

The time efficiency obtained is 5n+1 this has to be analysed through various values to know the best case worst case and the average case of the algorithm designed it is based on the concept of LUT.

The best case of the algorithm is n=1 for T αΩ (5n+1) its uses minimum time,

The worst case of the algorithm is for the value of n=16 i.e., T αO (5n+1) which requires more time hence it uses more resources and

Average case of the designed algorithm is when n=6 i.e., T α Θ (5n+1).the efficiency of all the cases is plotted using the graph.

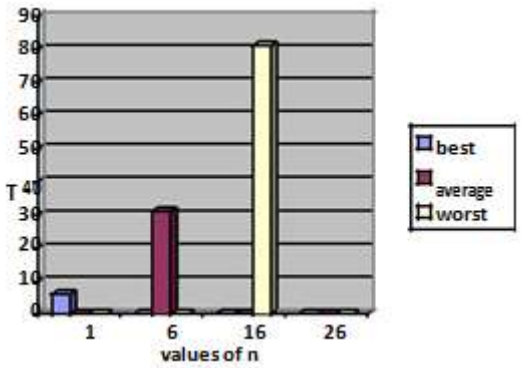

Figure 1 Graph of best average and worst case of prior values

The best case of the algorithm is for the value n=1

$$f(n) \alpha \Omega(1)$$

And the worst-case is

$$f(n) \alpha O(n)$$

According to the posterior analysis the best and worst cases are measured by

If the first comparison itself of the program gives output saving all other comparisons time then it will be the best case

$$f(n) \alpha \Omega(1)$$

Worst-case of the algorithm gives an upper bound on the resources required by the algorithm. The worst-case is given by

$$f(n) = 1.\ i=0 \sum n-1$$

After solving this we obtain f (n) α O (n). It is represented by graph as shown below

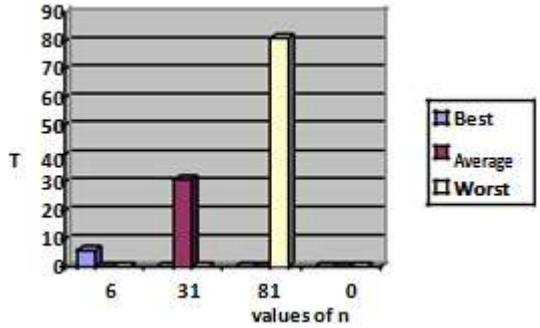

Figure 2 Efficiency graph for posterior result

ISSN (Online) 2394-2320
International Journal of Engineering Research in Computer Science and Engineering (IJERCSE)
Vol 3, Issue 6, June 2016

For both prior and posterior analysis the results are same

i.e.,

$$f(n) \alpha O(n).$$

V.CONCLUSION

In this paper we have proposed the scratchpad to hold the values temporarily here we have built the scratchpad algorithm by considering LUT case study. The LUT which we have considered is of size 9*9 and whenever the value more than 9 occurs than the scratchpad is used to hold the value which is in 10^{th} position. This scratchpad holds the temporary value for different calculations like addition, subtraction, multiplication and division operations and thus time constraint can be minimized.

REFERENCES

[1] Joe Sventek, Poler Dickman, Roes Mcllory. "Efficient Dynamic Heap Allocation of Scratchpad Memory in Embeded systems." University of Glasgow,UK

[2] Anany Levitin."Introduction to the Design and Analysis of Algorithms," University of Villanova, 2003.

[3] Angel Dominguez, Rajeev Barua, Sumesh Udaykumaran. "Heap Data Allocation to Scratchpad Memory in Embeded Systems."