Performance Improvement of Multi-Core Architecture Using Whetstone Application in Linux

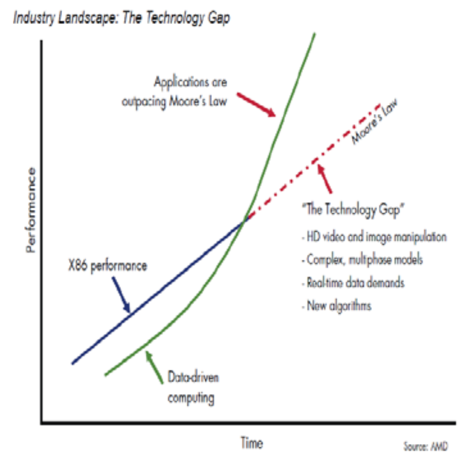
P. Bala Subramanyam Raju **P.Govindarajulu**

Research Scholar, Professor, Department of Computer Science, SVU College of CM&CS S.VUniversity, Tirupathi

Abstract
The most important characteristic of computers is the speed of the central processing unit. Application [1] Performance on modern processors has become increasingly dictated by the use of on-chip structures speed and software's like Operating system services, Compilers etc. The need of more performance was increasing day by day. The [21] computer system needs to satisfy an end-user application in terms of performance. It is useful as an initial filter to compare systems using 'standard' benchmark programs. This paper uses whetstone synthetic benchmark application to measure the speed and performance at which a computer performs floating point operations on multicore processor and improves the multi-core processor performance by modifying the existing algorithm to run on multicores by dividing the program as parent and child process to obtain the maximum performance on Linux operating systems.
Keywords:
Chip Multiprocessors (CMPs), MWIPS stands for Million Whetstones Instructions per Second. MIPS, Million Instructions per Second, MOPS Millions of Operations per Second.

1. Introduction

Application [2] demands have outpaced the conventional processor's ability to deliver. Since from processor invention it followed "Moore's Law" [3] in the last few years, however, the situation has changed. High performance computing (HPC) applications demanding more than processor alone can deliver, creating technology gap between demand and performance.
Many traditional applications have increased their demand on processor by implementing more and more complex algorithms. A number of new applications have also arisen and become widespread as their performance thresholds were met. E.g. Medical imaging, including ultrasound, computer aided tomography(CAT) scanning, and magnetic scanning resonance imaging(MRI).Even performance demand increases have begun exceeding Moore's Law processors have begun faltering. In this situation performance of a processor is the most important criteria. So this paper tries to achieve more performance by modifying the existing application.
The rest of the paper is organized as follows Section 2 provides brief introduction about whetstone application, Section 3 gives the whetstone algorithm Section 4 presents the modified whetstone algorithm, Section 5 presents the hardware and software environment details, Section 6 gives experimental Results, Section 7 concludes the paper with future work.



2. Whetstone

The Fortran [8][9][10][11] Whetstone programs were the first general purpose benchmarks that set industry standards of computer system performance. Whetstone programs also addressed the question of the efficiency of different programming language. The first Whetstone benchmark, known as HJC11 (later ALPR12), was written in Algol60 and completed in November 1972. The FORTRAN codes (HJC12 and HJC12D) were published in April 1973 as FOPR12 and FOPR13. The first results published were for IBM and ICL mainframes in 1973. The speed rating was calculated in terms of Kilo Whetstone Instructions per Second or KWIPS. Later, Millions or MWIPS was used.
It contains several modules that are meant to represent a mix of operations typically performed in scientific applications. A wide variety of C functions including sin, cos, sqrt, exp, and log are used as well as integer and floating-point math operations, array accesses, conditional branches, and procedure calls. The primary aim of this

Manuscript received October 5, 2013
Manuscript revised October 20, 2013

benchmark is to measure the performance of both integer and floating-point arithmetic.
The benchmark is very simple, comprising some 150 statements with eight active loops, three of which execute via procedure calls. Three loops carry out floating point calculations, two functions, one assignments, one fixed point arithmetic and one branching statements. The dominant loop, usually accounting for 30% to 50% of the time, carries out floating point calculations via procedure calls.
The tests only reference a small amount of data which will fit in the L1 cache of any CPU. Hence, L2 cache and memory speed should have no influence on performance ratings. Speeds are invariably proportional to CPU MHz on a given type of processor. The code was designed to be non-optimisable and optimizing compilers did not have a significant impact until the introduction of in-lining of subroutine instructions.

2.1 Pros of whetstone application

- The Whetstone benchmark was the first intentionally written to measure computer speed and performance and was designed to simulate floating point numerical applications.
- It is written in high-level language making it portable across different machines.
- It contains a large percentage of floating point data and instructions; A high percentage of execution time (approximately 50%) is spent in mathematical library functions;
- The majority of its variables are global and the test will not show up the advantages of architectures such as RISC where the large number of processor registers enhance the handling of local variables;
- Whetstone contains a number of very tight loops and the use of even fairly small instruction caches will enhance performance considerably;

3. Whetstone Algorithm

The[22] Whetstone benchmark main loop executes in a few milliseconds on an average modern machine, so its designers decided to provide a calibration procedure that will first execute 1 pass, then 5, then 25 passes, etc... until the calibration takes more than 2 seconds, and then guess a number of passes xtra that will result in an approximate running time of 100 seconds. It will then execute xtra passes of each one of the 8 sections of the main loop, measure the running time for each (for a total running time very near to 100 seconds) and calculate a rating in MWIPS, the Whetstone metric.

The main loop consists of 8 sections each containing a mix of various instructions representative of some type of computational task. Each section is itself a very short, very small loop, and has its own timing calculation.

Section 1 performs array elements operations

```
initialize i:=0
repeat the following steps until i<n1*n1mult
begin
        e1[0] := (e1[0] + e1[1] + e1[2] - e1[3]) * t;
        e1[1] := (e1[0] + e1[1] - e1[2] + e1[3]) * t;
        e1[2] := (e1[0] - e1[1] + e1[2] + e1[3]) * t;
        e1[3] := (-e1[0] + e1[1] + e1[2] + e1[3]) * t;
        i:=i+1;
end
 t := 1.0 - t;
 t :=  t0;
calculate time consumed
print the result using pout function
```

Section 2 performs passing array elements as arguments,

```
initialize ix:=0
repeat the following steps until ix<xtra
begin
initialize i:=0
   repeat the following steps until i<n2
  begin
              call function pa(e1,t,t2)
              i:=i+1;
  end
ix:=ix+1;
t: = 1.0 - t;
end
t :=  t0;
calculate time consumed
print the result using pout function
```

Section 3 performs conditional jump operations,

```
initialize j := 1,ix:=0
repeat the following steps until ix<xtra
begin
initialize i:=0;
          repeat the following steps until i<n3; i++)
          begin
                   if( j=1)     j:=2;
                   else         j:=3;
                   if(j>2)      j:=0;
                   else         j:=1;
                   if(j<1)      j:=1;
                   else         j:=0;
                    i:=i+1
```

```
        end
         ix=ix+1
end
calculate time consumed
print the result using pout function
```

Section 4 performs Integer Arithmetic operations

```
initialize j: = 1, k := 2, l := 3, ix:=0
 repeat the following steps until  ix<xtra; ix++)
 begin
        initialize i:=0;
        repeat the following steps until i<n4
           begin
             j:= j *(k-j)*(l-k);
             k:= l * k - (l-j) * k;
             l:= (l-k) * (k+j);
             e1[l-2]:= j + k + l;

             e1[k-2]:= j * k * l;
            i:=i+1
        end
        ix:=ix+1
 end
 calculate time consumed
 x := e1[0]+e1[1];
 print the result using pout function
```

Section 5 does Trigonometric functions

```
 initialize x:= 0.5,y:= 0.5,ix:=0;
 repeat the following steps until ix<xtra
 begin
     initialize i:=1;
     repeat the following steps until i<n5
           begin
             x:= t*atan(t2*sin(x)*cos(x)/
             (cos(x+y)+cos(x-y)-1.0));

             y:= t*atan(t2*sin(y)*cos(y)/
                (cos(x+y)+cos(x-y)-1.0));
            i:=i+1
           end
        t = 1.0 - t;
       ix:=ix+1
    end
t = t0;
calculate time consumed print the result using pout
function
```

Section 6 does procedure calls,

```
initialize x: = 1.0,y: = 1.0,z: = 1.0,ix=0
repeat the following steps until ix<xtra
begin
 initialize i:=0
  repeat the following steps until i<n6
     begin
            call function  p3(&x,&y,&z,t,t1,t2);
        i:=i+1
     end
 ix:=ix+1
end
calculate time consumed
print the result using pout function
```

Section 7 does Array Reference,

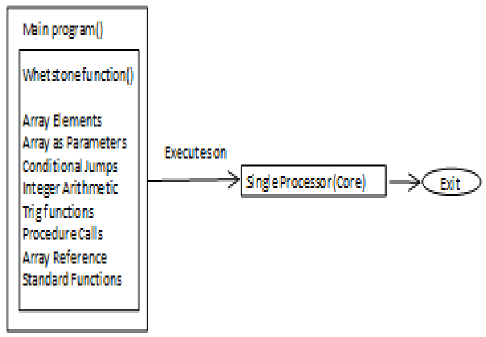
```
initialize j: = 0;k:= 1,l:= 2,e1[0]:= 1.0,
     e1[1]:= 2.0,e1[2]:= 3.0,ix:=0
repeat the following steps until ix<xtra
begin
        initialize i:=0
        repeat the following steps until i<n7
        begin ;i++)
          call function     po(e1,j,k,l);
                  i:=i+1
        end
        ix:=ix+1
end

calculate time consumed print the result using pout
function
```

Section 8 performs Standard functions.

```
initialize x:= 0.75,ix:=0
repeat the following steps until ; ix<xtra; ix++)
begin
        initialize i:=0;
        repeat the following steps until i<n8
        begin
            x = sqrt(exp(log(x)/t1));
            i:=i+1
        end
        ix:=ix+1
end
calculate time consumed print the result using pout
function
```

the below figure shows program execution on single core



4. Modified whetstone Algorithm

The whetstone program has been divided into two programs one is main program which calls whetstone program to perform calibration and mathematical operations test. The whetstone program consist of eight modules to perform the mathematical functions such array passing, array references, trigonometric functions etc. The whetstone program create a child process on successful creation it replaces its parent copy with the child assigned module, similarly creates another seven child processes for remaining seven modules and made the child process to execute on different cores based on the work load. These functions are created as child process using fork () method, and then they are assigned to run on different cores using taskset () method, the abstract algorithm is given below.

Algorithm Mainprogram()
begin
//Initialize variables
//Call whetstone function to perform calibration test
Initialize calibrate=0;
Whetstone(calibrate)
Display calibration results
//call whetstone function to perform mathematical opeations
Initialize calibrate=1;
Whetstone(calibrate)
//display mathematical operation results
End

Algorithm whetstone (calibrate)
begin
initialize variables

create new child process 1
assign processor 1 for execution of new process
call section6()

create new child process 2
assign processor 1 for execution of new process
call section8()
create new child process 3
assign processor 1 for execution of new process
call section5()

create new child process 4
assign processor 1 for execution of new process
call section7()

create new child process 5
assign processor 1 for execution of new process
call section4()

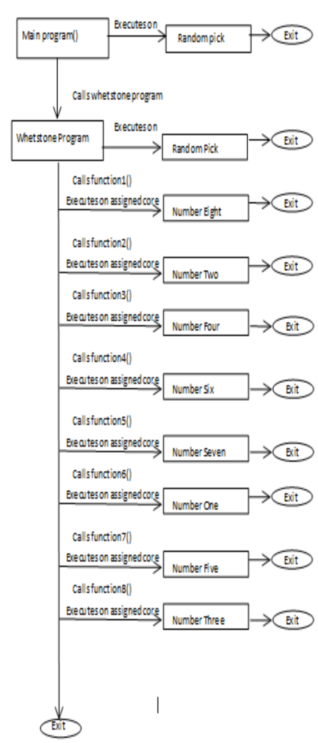
create new child process 6
assign processor 1 for execution of new process
call section3()

create new child process 7
assign processor 1 for execution of new process
call section2()

create new child process 8
assign processor 1 for execution of new process
call section1()
end

//end of whetstone function

the below figure shows modified program execution on multi- core



5. Experimental Test Bed

5.1 Software

- LINUX KERNEL 3.5.0-17-generic
- LINUX MINT OS
- GCC COMPILER

5.2 Hardware

Processor	Intel® Core™ i7-2670QM[17]
No of Cores	4
No of Threads	8
Clock Speed	2.2 GHz
Max Turbo Frequency	3.1 GHz
Intel® Smart Cache	6 MB
RAM:	4 GB

6. Experimental Results

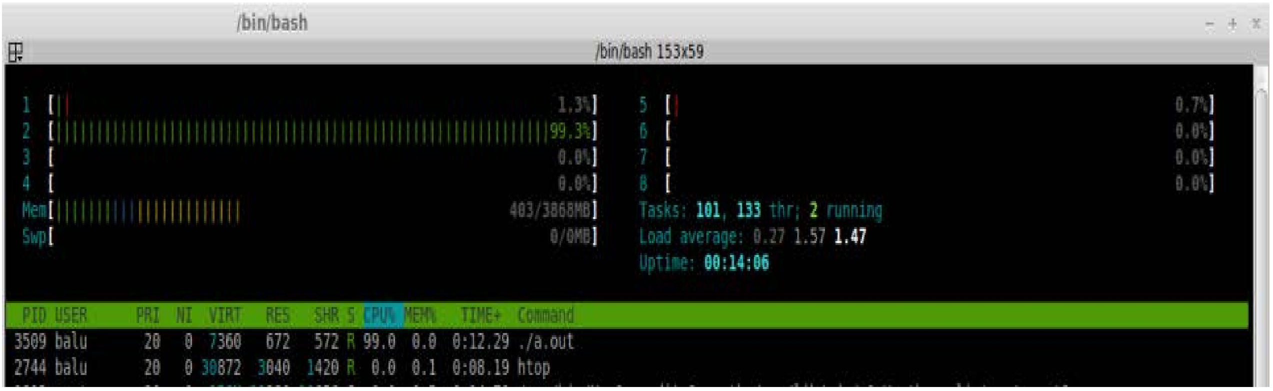
The whetstone benchmark application program has been run without and with modification to the program obtained from internet to compare the performance obtained by modifying the program while running it on 4/8 multicore processor[18]. The results are detailed below

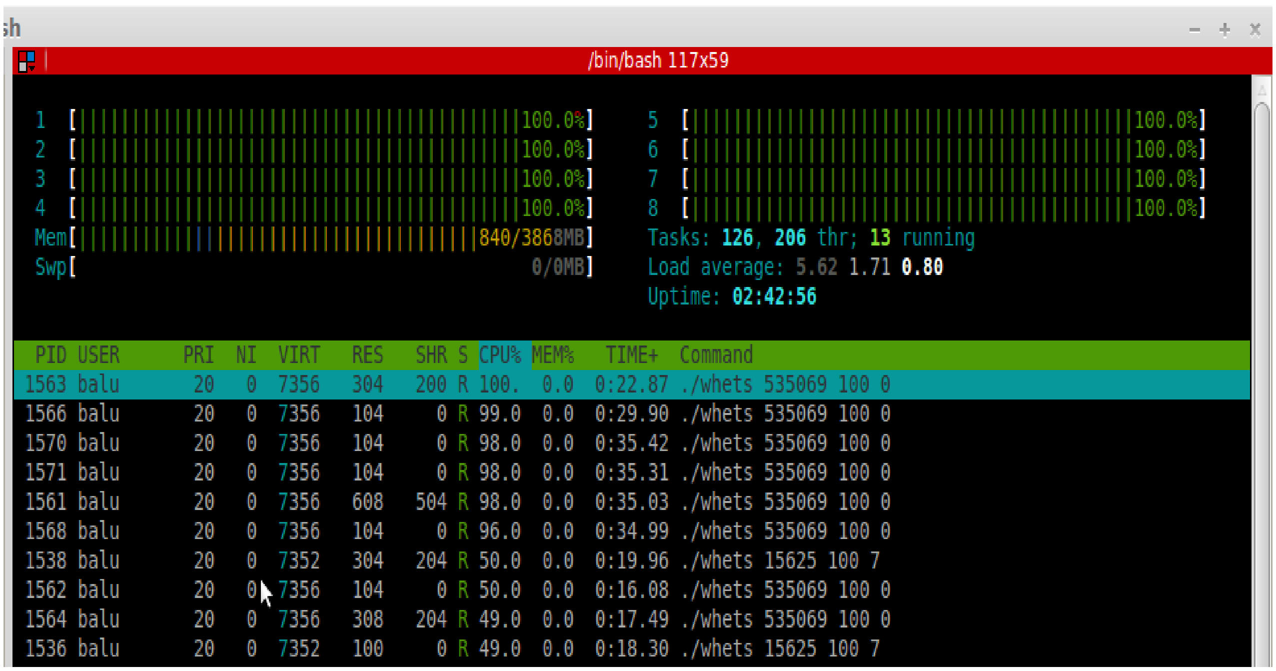
Below table shows calibration values for Whetstone application

Before Modification		**After Modification**	
Time (in Sec)	**No of Passes(x 100)**	**Time (in Sec)**	**No of Passes(x 100)**
0.00	1	0.00	1
0.02	5	0.00	5
0.11	25	0.00	25
0.56	125	0.02	125
2.69	625	0.09	625
		0.48	3125
		2.38	15625
Use23263 passes(x100)		**Use 656471 passes(x 100)**	

The below table shows time, expression values of whetstone application for unmodified and modified							
		Unmodified Program			**Modified Program**		
Loop content	**Result**	**MFLOPS**	**MOPS**	**Sec**	**MFLOPS**	**MOPS**	**Se**
N1 floating point	-1.12475013732910156	674.655		0.662	467.987		26.933
N2 floating point	-1.12274742126464844	628.792		4.972	443.159		199.092
N3 if then else	1.00000000000000000		958.428	2.512		548.810	123.804
N4 fixed point	12.0000000000000000		1254.68	5.840		736.464	280.786
N5 sin,cos etc.	0.50000000000000000		101.883	18.997		72.295	755.495
N6 floating point	0.99999982118606567	336.100		37.334	285.823		1007.01
N7 assignments	3.00000000000000000		394.524	10.897		269.024	450.948
N8 exp,sqrt etc.	0.75110864639282227		44.222	19.569		33.767	723.209
	MWIPS	**2308.19**		**100.78**	**53025.13**		**123.804**

The below figures shows the core utilization for unmodified and modified whetstone application during program runtime.





7. Conclusion and Future Work

The above results shows that after modification the whetstone application has produced more MWIPS and increased no of passes by utilizing all the existing cores and threads. While at the same time it produced less expression values by concentrating more on MWIPS this may be obtained by further modifying the program to concentrate on expression values.

When utilizing all the cores the system will produce more heat and power consumption is more than normal utilization.

Note: The order of results and values may change depending on load of the system, processes creation and execution by the Operating System.

References

[1] Jeffrey C. Mogul, Andrew Baumann, Timothy Roscoe, Livio Soares Mind the Gap: Reconnecting Architecture and OS Research, HotOS'13 Proceedings of the 13th USENIX conference on Hot topics in operating system,s USENIX Association Berkeley, CA, USA ©2011

[2] White Paper, Altera Corporation, Accelerating High performance computing with FPGAs, 2007

[3] www.kth.se/upload/234/Moores_law.pdf
[4] Linux Kernel Development, Robert Love 3rd Edition
[5] www.saneeshththottamkara.wordpress..com/2011/01/15/advantages and disadvantages-of-linux-operating-system-2/
[6] www.ubuntuartists.deviantart.com/journal/8-Advantages-of-using-Linux-over-Windows-291681914
[7] Weicker, R.P., "An overview of common benchmarks," *Computer*, vol.23, no.12, pp.65, 75, Dec. 1990 doi: 10.1109/2.62094
[8] www.roylongbottom.org.uk/whetstone.htm
[9] www.keil.com/benchmarks/whetstone.asp
[10] www.cse.dmu.ac.uk/~bb/Teaching/ComputerSystems/SystemBenchmarks/BenchMarks.html#introduction
[11] Roy, A.; Jingye Xu; Chowdhury, M.H., "Multi-core processors: A new way forward and challenges," *Microelectronics, 2008. ICM 2008. International Conference on* , vol., no., pp.454,457, 14-17 Dec.2008doi: 10.1109/ICM.2008.5393510
[12] John Fruene, Dell Power Solutions. Reprinted from Dell Power Solutions, May 2005. ..... tions of migrating applications to multicore processor technology.www.dell.com/downloads/.../power/ps2q05-20050103-Fruehe.pdf
[13] Shameem Akhter, Jason, Roberts Multi-Core Programming Increasing Performance through Software Multi-threading, April 2006, ISBN 0-9764832-4-6.
[14] Bryan Schauer Multicore processors-A Necessity *proQuest 2008*
[15] Balaji venu Multi-core processors-An overview, arXiv: 1110.3535 (October 2011)
[16] Shameem Akhter , Jason Roberts ,Multi-Core Programming, © 2006 Intel Corporation
[17] http://ark.intel.com/products/53469
[18] www.roylongbottom.org.uk/whetstone%20results.htm#anchornopt
[19] www.roylongbottom.org.uk/whetstone%20results.htm#anchorLinux
[20] www.roylongbottom.org.uk/linux%20multithreading%20benchmarks.htm#anchor3
[21] www.cse.dmu.ac.uk/~bb/Teaching/ComputerSystems/SystemBenchmarks/BenchMarks.html#introduction
[22] http://www.tux.org/~balsa/linux/benchmarking/articles/html/Article1d-3.html

P. Bala Subramanyam Raju received B.Sc(M.E.Cs) . and M.C.A. degrees from Sri Venkateswara University in 2005 and 2008 respectively. He is a Research Scholar in the department of Computer Science, Sri Venkateswara University, Tirupati,A.P India. His research focus is on Parallel Computing.

P.Govindarajulu, Professor,Department of Computer Science, Sri Venkateswara University,Tirupati, India. He received his M.Tech. from IIT Madras(Chennai), Ph.D. from IIT Bombay(Mumbai). His area of research:Databases, Data Mining, Image Processing, Intelligent Systemsand Software Engineering,Parallel Computing.