ISSN: 0976-3376

Available Online at http://www.journalajst.com

ASIAN JOURNAL OF SCIENCE AND TECHNOLOGY

Asian Journal of Science and Technology
Vol. 07, Issue, 10, pp.3668-3672, October, 2016

RESEARCH ARTICLE

OPTIMIZING MATRIX MULTIPLICATION USING MULTITHREADING

***Harsh Patel, Anubhav Chaturvedi and Vishwas Raval**

Department of Computer Science & Engineering, Faculty of Technology & Engineering,
The MS University of Baroda, Baroda

ARTICLE INFO

Article History:
Received 17th July, 2016
Received in revised form
08th August, 2016
Accepted 20th September, 2016
Published online 30th October, 2016

Key words:

Multithreading, POSIX,
C Programming, Linux,
Time, Bash, Complexity.

ABSTRACT

Multicore applications can deliver better performance with Pthreads ,which is an API for writing multithreaded applications to boost the performance of a computer. With the examples presented in this paper for the multiplication of two NxN matrices with a serial application and a parallel application using p_threads,one can understand the power of the Pthread apps.

INTRODUCTION

Increasing the clock frequency of a microprocessor was considered smart-work in the IT industry once. But now a days, due to heat dissipation and energy consumption issues, processor developers are switching to a new model where microprocessors contain multiple processing units called cores. All the modern computing devices have several cores to meet the computing requirements. So to get the performance out of these cores, developers needs to write applications using parallel computing methods.

Serial:

//serial program to multiply two n*n matrices [matrix_mul_serial.c]

```
#include<stdio.h>
#define n 5120
int a[n][n],b[n][n],c[n][n];
int main()
  {
    int i,j,k;
    for(i=0;i<n;i++)    //data initialization
    for(j-0;j<n;j++)
    {
      a[i][j]=1;
      b[i][j]=1;
    }
  for(i=0;i<n;i++)   //multiplication
  for(j=0;j<n;j++)
  for(k=0;k<n;k++)
  c[i][j]=c[i][j]+a[i][k]*b[k][j];
  printf("\n The resultant matrix is \n");
  for(i=0;i<n;i++)
  {
    for(j=0;j<n;j++)
  {
  printf("%d",c[i][j]);
  printf("\n");
  }}}
```

```
harsh:->cc –o matrix_ser_5120 matrix_mul_serial.c
harsh:->time ./matrix_ser_5120
real   1m1.519s
user   1m1.582s
sys    0m0.002s
```

Parallel:

//parallel program to multiply two n*n matrices [matrix_mul_parallel.c]

****Corresponding author: Harsh Patel***
Department of Computer Science & Engineering, Faculty of Technology & Engineering, The MS University of Baroda, Baroda

```
#include <stdio.h>
#include<stdlib.h>
#include<pthread.h>
#define n 5120 //global space
#define nthreads 2
int a[n][n],b[n][n],c[n][n];
void *threadfun(void *arg)      // each thread
{
  int *p=(int *)arg;
  int i,j,k;
  for(i=*p;i<(*p+(n/nthreads));i++)
  for(j=0;j<n;j++)
  for(k=0;k<n;k++)
  c[i][j]=c[i][j]+a[i][k]*b[k][j];
}
int main()
{
  int i,j,k,r,rownos[nthreads];
  pthread_t tid[nthreads];
  for(i=0;i<n;i++){
  for(j=0;j<n;j++){
  a[i][j]=1;
  b[i][j]=1;
}}

//thread creations using pthreads API

for(i=0;i<nthreads;i++)
{
  rownos[i]=i*(n/nthreads);
  pthread_create(&tid[i],NULL,threadfun,&rownos[i]);
}

//making main thread to wait untill all other

for(i=0;i<nthreads;i++)
pthread_join(tid[i],NULL);

for(i=0;i<n;i++)
{
  for(j=0;j<n;j++)
  printf("%d",c[i][j]);
  printf("\n");
}
}

harsh:->cc –o matrix_par  matrix_mul_parallel.c -lpthresd
harsh:->time ./matrix_par
real   1m1.519s
user  2m4.824s
sys    0m0.087s
```

Brief introduction of Pthread API

A Phtreads API is a standard (IEEE Posix 1.3c) application program interface that could potentially be implemented on many different systems. Pthreads is a standard multi way multi-threaded support is offered.GNU/Linux implements the Pthreads API by keeping all thread functions and data-types in header file pthread. h. Pthread functions are not included in the standard C library. Instead, they are in libpthread so we need to compile and link out program as below:

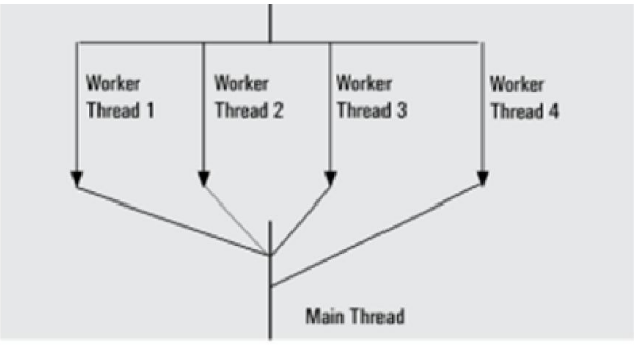

Figure 1: Multithreaded application design model

Developing a Multi-Threaded application with the Pthreads API

A multi-threaded application has multiple threads executing a shared address space. A thread is a lightweight process that executes within the same process. All the threads share the code and the data segment, but each thread will have its own program counter, machine registers and stack. The global and static variables present in the process are common to threads. Each thread will do a specific task assigned by the process. Consider the Boss and worker model working in the figure for a sample application to be written with the help of the Pthread API.The idea of this application is to have a single boss thread (main thread) that creates work and several worker threads that process the work. Typically, the boss thread creates a number of certain number of workers, distributes the work, and then wait for the workers to complete the work. The worker threads from the pool as shown in the Figure. This model works the best when the work items/data are independent of each other and we can schedule each worker thread on a separate core.

To create a thread we use the pthread_creation function present in the API. We need to call it with four parameters, which are listed below:

Parameter 1: It specifies the address of the variable where we want to store the ID of the newly created thread in the program for future reference.
Parameter 2: It specifies the thread attributes. NULL means default attributes.
Parameter 3: It specifies the address of the function according to which the newly created thread will perform the task.
Parameter 4: It specifies the data which the newly created thread gets from the caller.

Serial (sequential) application performance monitoring

We consider the application of two NxN matrices as the benchmark application .Matrix multiplication can be done in serial fashion as shown below:

We are executing the application on a system with Intel CoreTM i5-3450 CPU @3.10GHz x 4, 4GB RAM running a Linux OpenSUSE 42.1 Leap Version 64 bit operating system. We can use the htop command and system monitor to demonstrate the resource utilization of the application, as shown in Figures 3 and 4. It is very clear that, of the 4 cores available, the application is running on Core/CPU 1 is it is a serial application with one thread of control. The other cores

are underutilized because of this. With only 27.4 per cent of CPU utilization, the application failed to take advantage of its four cores, so the performance of the application is very poor.

design idea is to distribute the work equally among the threads, when N is an exact multiple of number of threads.

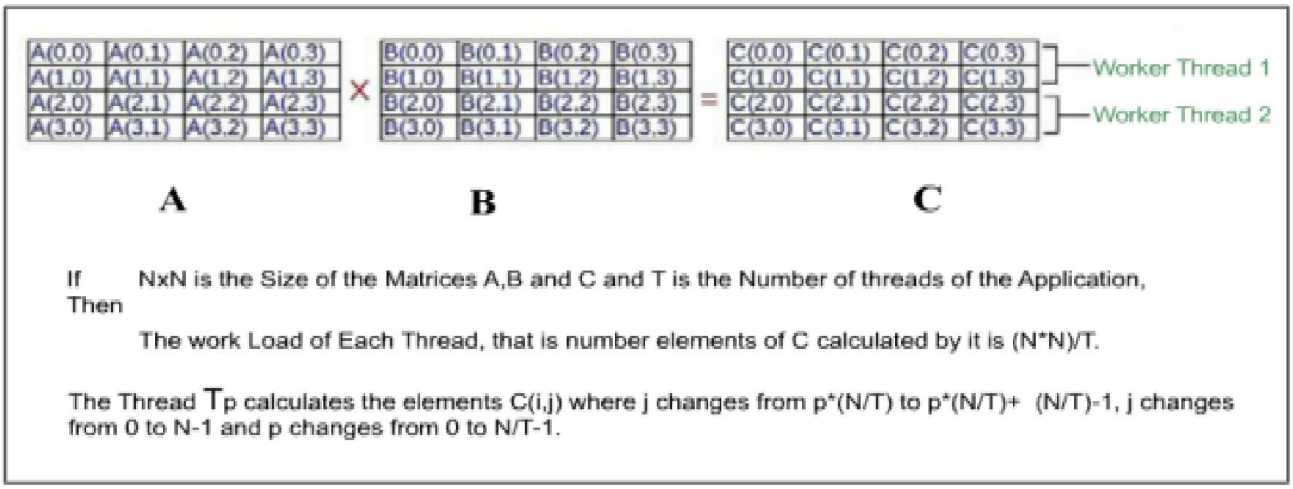

Figure 2. Matrix multiplication in parallel

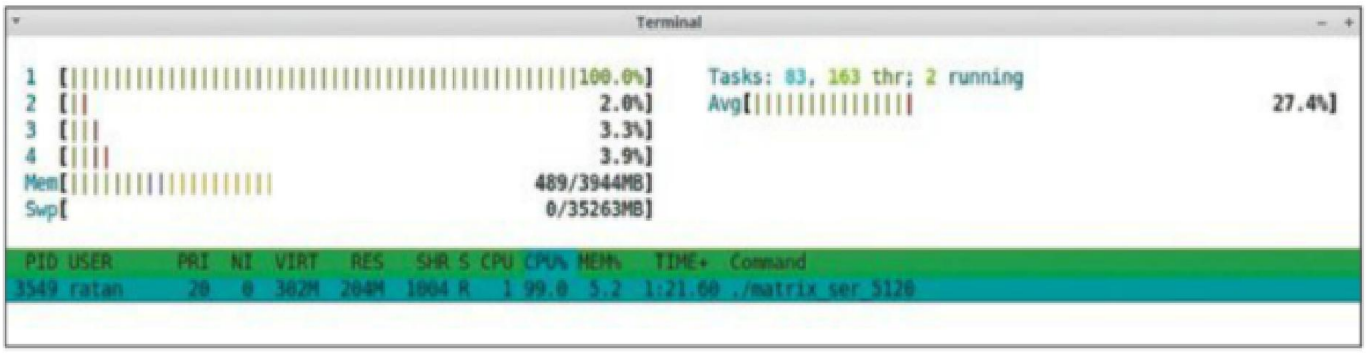

Figure 3. Serial application performance monitoring with htop

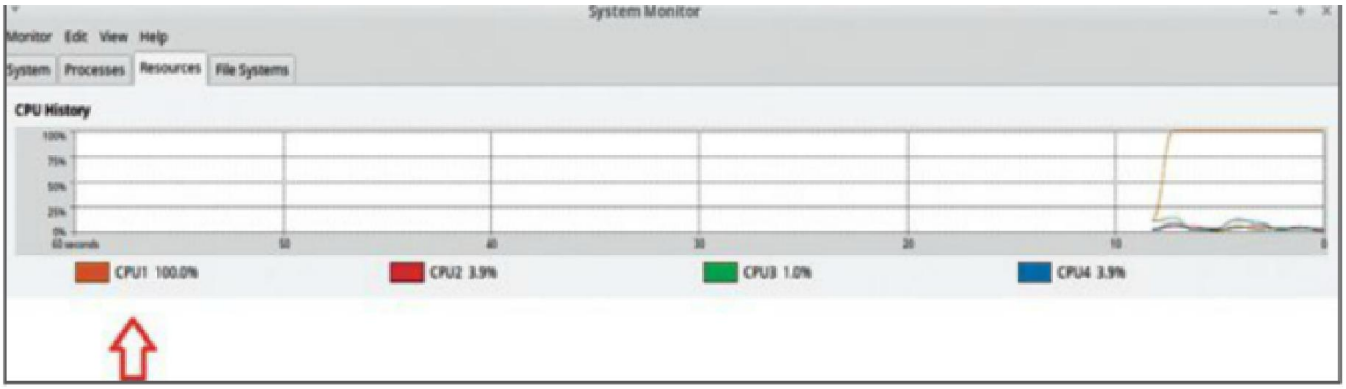

Figure 4. Serial application performance monitoring system monitor

Multi-threaded (parallel) application performance monitoring

Matrix multiplication of two NxN matrices, can be done in parallel, in many ways. We considered doing it in parallel, in many ways. We considered doing it in parallel as shown in Figure 2. If N is 4, then we can calculate two threads to calculate the C matrix, where Thread 1 computes the elements of the first two rows and thread 2 computes the elements of the next two rows in parallel. If we create four threads, then each thread computes elements of one row of C in parallel. The

The execution is as follows

This will create a multi-threaded application with two worker threads and one boss thread. The performance can be monitored by htop as shown in Figure 5. The two worker threads are running in parallel, simultaneously, on Cores 2 and 4, while the main thread (boss) is waiting as shown in Figure 5. The total CPU utilization is 55.4%. The application performance is better than the serial performance. If you replace the #define nthreads 2 in the above program with #define nthreads 4 and recompile it, then it generates a multi-threaded application with four worker threads and one boss

thread as shown in Figures 6 and 7.The four worker threads are running simultaneously on Cores 1,2,3 and 4 respectively, in parallel. It is very clear that CPU utilization has improved to more than 99 per cent. So the application's performance is very good.

The total CPU utilization is 55.4%.The application performance is better than the serial performance. If you replace the #define nthreads 2 in the above program with #define nthreads 4 and recompile it, then it generates a multi-threaded application with four worker threads and one boss

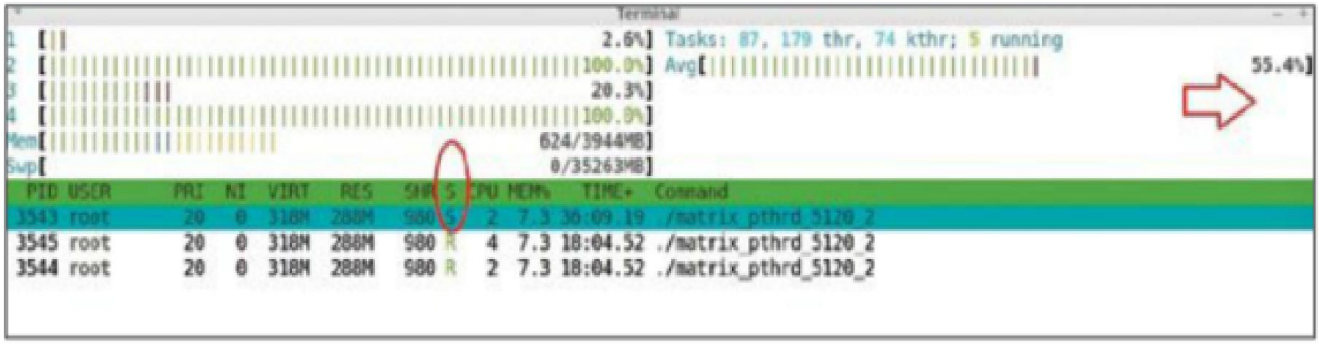

Figure 5. Multi threaded application with one boss and two works threads

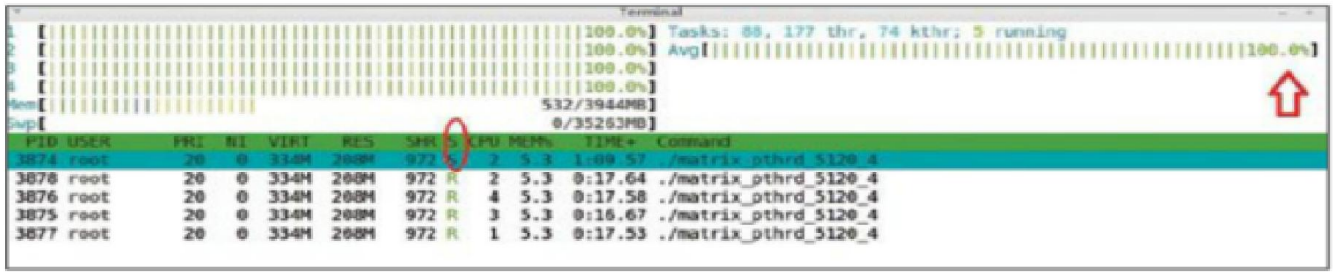

Figure 6. Multi thread application with one boss and 4 worker threads

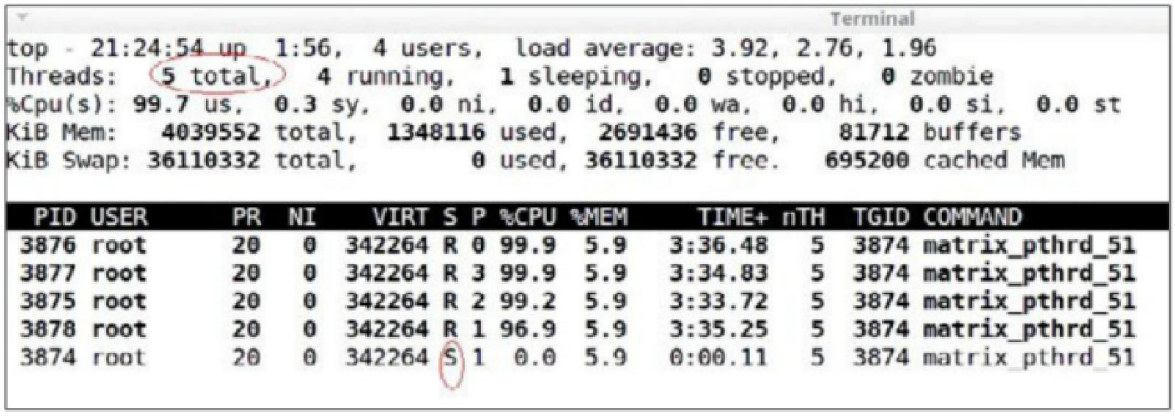

Figure 7. Multi threaded application with one boss and 4 worker threads

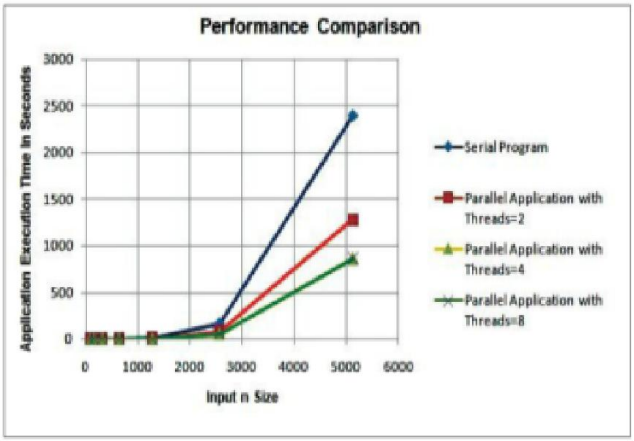

Figure 8. performance comparison

thread as shown in figures 6 and 7.The four worker threads are running simultaneously on Cores 1,2,3 and 4 respectively, in parallel. It is very clear that CPU utilization has improved to more than 99 per cent. So the application's performance is very good.

Conclusion

We have executed the serial application and multi-threaded applications by taking n=80,160,320,640,1280,2560,5120 and nthreads =2,4,8 in the case of multi-threaded applications. A comparison of the results is shown in Figure 8.

In terms of execution time, the multithreaded application clearly outperforms the serial applications performance. So, with the Pthreads API we can develop parallel applications to get a higher performance from the multi-core microprocessor.

REFERENCES

http://www.geeksforgeeks.org/
Linux Man Page

****