International Journal of Advances in Computer Science & Its Applications – IJCSIA
Volume 4: Issue 3 [ISSN 2250-3765]
Publication Date : 30 September,2014

Threads and Computer Performance in OpenMP

Arsen Kurti, Igli Tafa, Ermela Cekani Hysa

Computer Science, Faculty of Information and Technology, Polytechnic University Tirana

***Abstract*--- In our days we want to run multiple jobs in the same time in the system. Most of computers systems have multiple processors or multiple execution cores. This makes the concurrency higher. I choose the theme about threads because I wanted to know more about the basic unit of software that operating systems deal with "threads". In this article I am going to describe these threads and I am going to do an experiment to explain better how they affect the load of CPU and the performance too.**

***Keywords* --- multi processors, threads, CPU performance**

I. Introduction

A very common question that comes is: Are thread faster? Faster for what? What they actually do? First I will explain each of them.

Thread exist inside of processes. In Linux when a program is being executing automatically a new process is created, and the process create a thread that runs sequentially. That thread may create other threads, that run the same program but each of them execute different part of it. Thread and its process are different from each other because processes group resources together and threads are the entities scheduled for execution on the CPU. Multiple threads may run parallel inside a process, this is equal as many processes running parallel in one computer [2]. Hyper-threading technology is used first in February 2002 on Xeon server processors and on November 2002 on Pentium 4 desktop CPU. After it took place in Itanium Atom and Core 'i' series CPUs [1].

Why to use thread?

If there is an application that have some activities going on at once, an blocking will occur. The situation will be much more easer if decomposing this in threads that will run in quasi-parallel. They are easy to be created and to be destroyed too. To create a thread is much more faster than creating a process. And they are really very powerful in systems with multiple CPUs because a real parallelism happens.

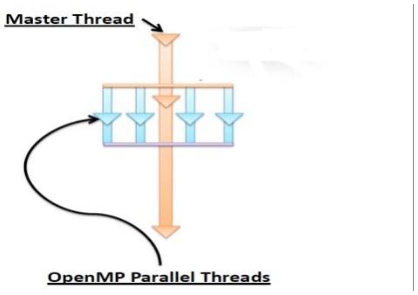


Fig 1 Fork and Join Model

In the figure 1 at first when a program starts to run only one thread exist and that is master thread. This thread by using the directives, create parallel threads.

After all of instructions are executed the result synchronization happens till the program complete.

The other questions is where is better to use threads?

It is found that OpenMP has some performance advantages over Windows this is related with startup cost[3]. OpenMP is an API Application Programming Interface for parallel applications in architectures with shared memory.

The other thing that confuses us is deciding how to thread my code?

First is to separate the program into parts that can run independently , and for each of these parts to create threads. For example to separate it for different functions or to thread that part of the code that takes more time. In lower level of code OpenMp makes so easy the threading process.

How many parts could we implement to thread this code?

OpenMP library helps me to decide the best number of threads to be used in a code, this will help me for the best speedup. We mustn't spend time creating threads that won't be used.[9]

II. Related works

OpenMP when first presented, lot of scientist find the best solution for complex applications instant of using supercomputers . It published its first API IN 1997 for Fortran one year later for C/C++. [4]. To improve thread level parallelization, synchronization has been tested to extract thread in hardware or in software then they are executed in parallel. [5] Another important thing is Amdahl's Law. This law expected to find maximum improvement to overall system. His law confirm the maximum speeding up of the parallelized version is 1.36 times faster that non parallelized implementation [6]. Another research has been done to compare OpenMP with Pthreads. OpenMP has the advantage of less memory usage and less direct inter-thread communication, Pthreads were OpenMP in lower level. They have almost the same assembly code but their interfaces are very different. [7] Every concept must be very clear to do a experimental work so I will explain two models of threading

User-level threading is when a library support thread creation, joining, termination, scheduling. But if a containing process is blocked every other threads of the same process are blocked.

Kernel –Level threading are slower than user threads, but blocking one thread will not cause the other threads of the same block to be blocked . On Linux C Library implements threads [8]

III. Theory of experiment

In this section we are going to do two programs, one serial program and the other parallel program. And we are going to study how the loops are parallelize and how the performance is affected. The algorithm and its implementation are explained at section 4. In the end of this article (section 5) we have summarizes our findings and also the future prospects.

A. Environment

I executed my code on Ubuntu OS 12.04. chooseOpenMP to do the experiment because it provide parallel executions also it has a high capacity which grows up every day. The hardware I used was Pentium(R) Dual-core CPU with 2 GB of RAM. The operating system was 64 bit.

B. Programming Languages

Two create two programs with the same soft but one serial execution and one parallel execution I choose C programming language because I was very familiar with it, and Open multi processing supports very well programming in C language.

IV. Experiment Phase

In the next section of the article a quick view of the experimental phase is presented. Two programs are executed in Ubuntu to find the maximum in a vector but one of them in serial and one in parallel. We measured their time of execution. The parallelization is done by #pragma omp directory. We include the <time.h> library to be able to call the function omp_getwtime() We cached the time before and after execution, find the difference between them and that is exactly the time of execution.

A. Algorithm of the experiment

The experiment is performed in a multiprocessor computer.

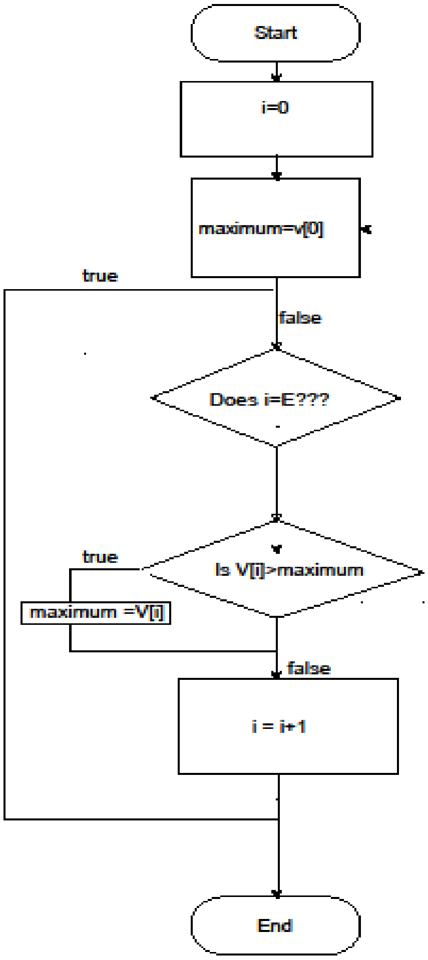


Fig 2: Algorithm scheme

1. Start the program
2. Let`s suppose that the first element of the array is maximum.
3. The variable goes till the end of an array.
4. If another element of the array is bigger than the first maximum, that element becomes the maximum of an array.
5. End of the program

V. Experim. Environment

First of all we have installed UBUNTU OS in my computer, then execute

sudo apt-get install gcc-4.6

command in the terminal to install OpenMp Library.

To execute the program we used these command:

1- Open terminal

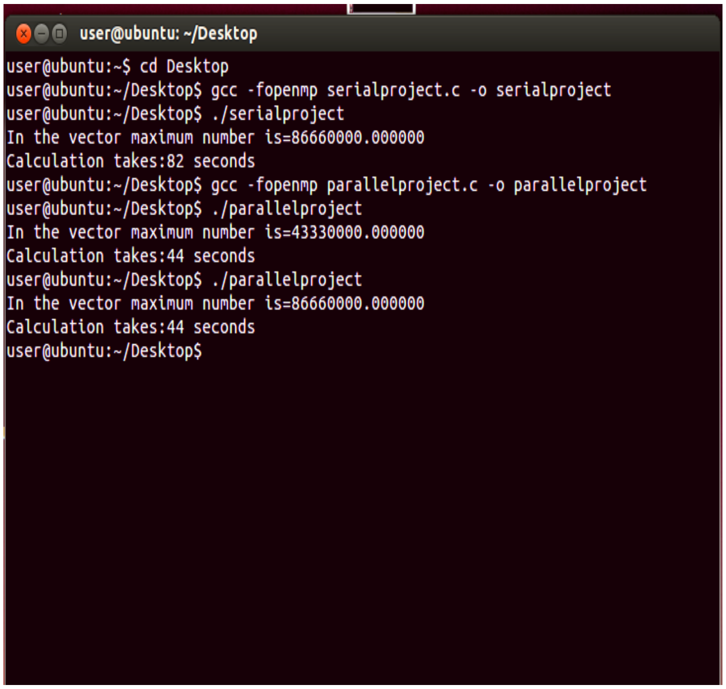
2- cd Desktop

3- gcc –fopenmp serialproject.c –o serialproject

4- ./serialproject

5- gcc –fopenmp parallelproject.c –o parallelproject

6- ./parallelproject

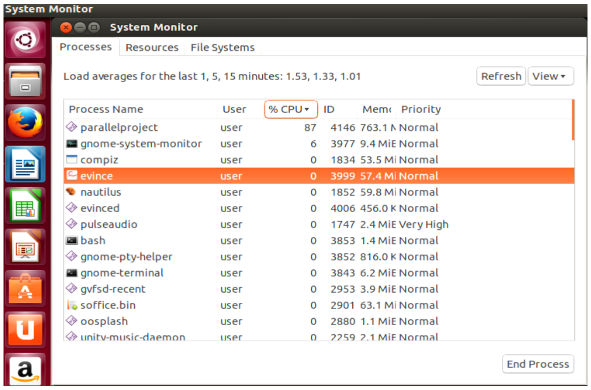


```
user@ubuntu:~$ cd Desktop
user@ubuntu:~/Desktop$ gcc -fopenmp serialproject.c -o serialproject
user@ubuntu:~/Desktop$ ./serialproject
In the vector maximum number is=86660000.000000
Calculation takes:82 seconds
user@ubuntu:~/Desktop$ gcc -fopenmp parallelproject.c -o parallelproject
user@ubuntu:~/Desktop$ ./parallelproject
In the vector maximum number is=43330000.000000
Calculation takes:44 seconds
user@ubuntu:~/Desktop$ ./parallelproject
In the vector maximum number is=86660000.000000
Calculation takes:44 seconds
user@ubuntu:~/Desktop$
```

Fig 3: Executing two programs

A. Observation of Experiment

While the program was running we directly open the system monitor to see what happens.

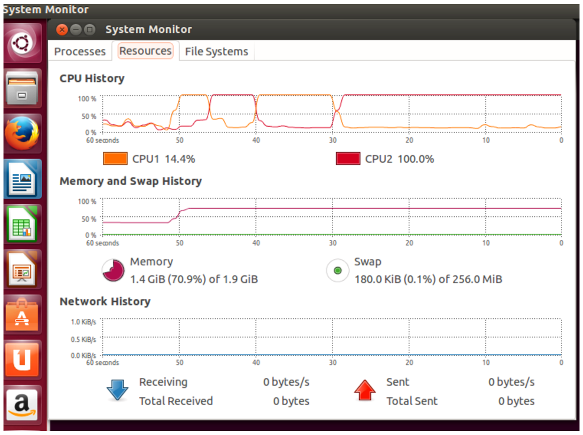


Load averages for the last 1, 5, 15 minutes: 1.53, 1.33, 1.01

Process Name	User	% CPU ▾	ID	Mem	Priority
parallelproject	user	87	4146	763.1 M	Normal
gnome-system-monitor	user	6	3977	9.4 MiE	Normal
compiz	user	0	1834	53.5 Mi	Normal
evince	user	0	3999	57.4 Mi	Normal
nautilus	user	0	1852	59.8 Mi	Normal
evinced	user	0	4006	456.0 K	Normal
pulseaudio	user	0	1747	2.4 MiE	Very High
bash	user	0	3853	1.4 MiE	Normal
gnome-pty-helper	user	0	3852	816.0 K	Normal
gnome-terminal	user	0	3843	6.2 MiE	Normal
gvfsd-recent	user	0	2953	3.9 MiE	Normal
soffice.bin	user	0	2901	63.1 Mi	Normal
oosplash	user	0	2880	1.1 MiE	Normal
unity-music-daemon	user	0	2259	2.1 MiE	Normal

Fig 3 :Linux processes System Monitor

Process table in graph form shows CPU performance memory and network performance too. By clicking over processes overview all the programs currently running on system because a lot of them run in background. To see which process uses larger CPU time we click *% CPU* so it will sorted automatically. In figure 3 parallel project uses the most % of CPU



CPU1 14.4%
CPU2 100.0%

Memory
1.4 GiB (70.9%) of 1.9 GiB

Swap
180.0 KiB (0.1%) of 256.0 MiB

Receiving 0 bytes/s
Total Received 0 bytes
Sent 0 bytes/s
Total Sent 0 bytes

Fig 4: Linux Resources history

Resources history: There are two processors in computer and each of it has different history line. During the real time running is shown CPU utilization. Memory and Swap History we can see two running real-time graphs, one part of memory used by the user and the other swap space used by user. Network history: shows how many data will be send or will be received in the network interfaces

VI. Results

The reason why I worked with this theme was to be well informed about parallelism in computer systems and to see better in practice how it works. First I measure the time for the first execution, **parallel for directive**, then we measured it twice .The execution time was approximately two times faster. Every application must be thread because the same program, the same numbers, the same logic, analogically by few words the same work is done faster by using parallel region than serial region. Here is the advantages of parallel version.

VII. Future Work

By seeing how technology has evolved till nowadays the future seems to be everything computerized. So the matter of time is the most critical point. OS communities will be expanded much more by using multithreading for application-level parallelism. The sequential thread-based programming will be less efficient since the numbers of cores are increasing more and more.

References

[1]http://www.xbitlabs.com/articles/cpu/displat/pentium4-3066.html

[2]Modern Operating Systems Tanenbaum

[3]Basic OpenMP Threading Overhead

[4]http://en.wikipedia.org/wiki/OpenMP#History

[5] A Clustered Approach to Multithread Processors

[6] Validity of the single processor approach to the achieving Large scale computing capabilities.

[7]Using OpenMP,Portable Shared Memory Parallel Programming Ruud Van Der Pas.

[8]http://en.wikipedia.org/wiki/Thread_(computing)

[9] Programming on Parallel Machines Califonia University

APPENDIX

Serial programming

```
#include<stdio.h

#include<stdlib.h>

#include<time.h>

#include<omp.h>

#define E 200000000

float v[E];

int main(void)

{

float maximum=-10000;

intstart_time,end_time;

longi,x=0; int y=0;

start_time=omp_get_wtime();  // put the time at the beginning of execution

for(i=0;i<E;i++)

v[i]=i*0.4333;

for(i=0;i<E;i++)

{

if (maximum<v[i])

maximum=v[i];

for (x=0;x<100;x++)

y=x*0.7;

}

end_time=omp_get_wtime(); // time since a fixet point in the past

printf("In the vector maximum number is=%f\n",maximum);

printf("Calculation takes:%d seconds\n", end_time-start_time);

}
```

Parallel Programming

```
#include<stdio.h>
#include<stdlib.h>
#include<time.h>
#include<omp.h>
#define E 200000000
#define NUM_THREAD 3
float v[E];
int main(void)
{
float maximum=-10000;
intstart_time,end_time;
longi,x=0;
int y=0;
intnthr;
floatth_max[NUM_THREAD]={0.0,0.0,0.0};
start_time=omp_get_wtime();
#pragma omp parallel private(i,x,y,maximum),shared(v,th_max) //  code inside this reagon runs  in parallel
{
#pragma omp for
for(i=0;i<E;i++)
v[i]=i*0.4333;
nthr=omp_get_num_threads();
#pragma ompfor  //parallel keyward is needed to create some new threads
for(i=0;i<E;i++)
{
if(maximum<v[i])
maximum=v[i];
for (x=0;x<100;x++)
y=x*0.7;
}
th_max[nthr]=maximum;

}

for(i=0;i<NUM_THREAD;i++)

if(maximum<th_max[i])

maximum=th_max[i];

end_time=omp_get_wtime();

printf("In the vector maximum number is=%f\n",maximum);

printf("Calculation takes:%d seconds\n", end_time-start_time);

}
```

Bibliography

Arsen Kurti was born at 1977 in Vlora. He finished high school at hometown. At 2000 he was graduated as Computer Engineering Specialist. He was IT Header at Raiffeissen Bank in Tirana from 2001-2005. At 2005-2013 he was Chief at Microsoft Albania. Now he is Phd student and he is preparing thesis for Cloud Efficiency.