User level Static and Dynamic Core assignment in Multi-core System

Dhruva R. Rinku[1] and Dr. M. Asha Rani[2]
[1] CVR College of Engineering/ECE Department, Hyderabad, India
Email: dhruva.rinku@cvr.ac.in
[2] Jawaharlal Nehru Technology University/ECE Department, Hyderabad, India
Email: ashajntu1@yahoo.com

Abstract: **There has always been a debate over the advantages of multi-core architecture over single core. It is obvious that the full advantage of multi-core can be achieved only if the programs are run in parallel [Amdahl's law] and the load is distributed evenly among all the available cores. In this context, processor or CPU affinity plays a major role in improving the performance of the system where time constraints are present. But, hard CPU affinity can also degrade the performance of the system in some scenarios. The alternative is to dynamically allocate the CPU core at application level. This paper studies these two techniques at application level to assign a particular CPU core once and stick to it, or switching among various cores dynamically by considering CPU usage.**

Index Terms—**multi-core, CPU affinity, static scheduling, dynamic load balancing, CPU utilization**

I. Introduction

To enhance the performance of an embedded system, multi-core architecture [7,10] is one of the possible solutions which allows the system to process numerous jobs simultaneously by parallel computation. At the same time the return on the investment has been a debating issue. As Amdahl proved that just by increasing the number of cores, the processing power can't be multiplied arithmetically, [1] but it depends on various factors like program parallelization. Even considering all favorable conditions for Kernel scheduling for multi-core few more factors affect the performance of the system.

Linux kernel has been giving significant importance to multiprocessor architecture [4,9,13,15]. Modern operating systems provide multi-core aware infrastructure [19], interrupt load-balancer, affinity facilities [3], CPUSETS [6], and CPU isolation [5]. These functions help running tasks adapt to system characteristics including SMP scheduler, synchronization very well by considering CPU utilization. Still there is enough scope to improve the performance of the system by providing user level control to decide a particular process to be run on a pre-defined CPU core.

'CPU or processor affinity' increases the performance of the system by assigning the process to one particular core, that avoids cache miss [17,18]. Processor affinity takes advantage of the fact that some remnants of a process that was run on a given processor may remain in that processor's memory state (for example, data in the CPU cache) after another process is run on that CPU. Scheduling that process to execute on the same processor could result in an efficient use of process by reducing performance-degrading situations such as cache misses [15]. This approach would be useful, if the application has multiple-instances and is non-threaded, such as some graphics-rendering software.

In some applications (particularly Real-Time control applications) [2], it may be desirable to specifically assign a task to its own dedicated core. For example, a time critical task such as a control loop can be implemented in this manner.[14] This allows the remaining tasks in the system to share the other processor resources among themselves—and ensures that nothing interferes with the time critical process.

On the other hand, there are limitations with CPU affinity. Most importantly, CPU affinity wastes the CPU usage if the task goes in to wait or sleep mode. This degrades the performance of the system, as once the core has been assigned, the process can't run after a wait or sleep unless the assigned core is free.

Load balancing on the cores is an alternative to achieve better performance. Utilization of a processor is usually considered as the criterion in load balance [8]. To generate the maximum balanced load, tasks should be assigned to the processor core with the lowest utilization [12]. Beyond OS schedulers that try to balance the load,[16] it is advantageous to provide monitoring and CPU switching at user level, using system calls within a program. On a Linux platform, a program in C can achieve this.

II. Methodology

This paper is based on literature available as mentioned in the references, as well experimental study on systems running with Linux operating system. Implementation of static (CPU affinity) and dynamic process allocation to a CPU core is done using qTcreator GUI and C language in Linux environment. To give reasonable understanding to the user, pseudo code is provided, where the original programs are tried and tested. The examples cited are to demonstrate the successful running of programs and to prove the concept. In the real world scenario, the context and applications may vary widely, thus differ in results.

III. Implementation

A. *Static Core Assignment*

Static or off line scheduling to achieve process affinity[11]:

A process can be given affinity [18] to a particular CPU core as per the user choice. This can be controlled by a GUI built using qtCreator, which in turn runs the kernel level commands to assign the affinity to a particular process, as

doi:10.32377/cvrjst1113

shown in fig 1. Any number of processes can be listed in the drop down list of the application, and the n numbers of cores are given as buttons. Whenever the user selects a particular process from the drop down list and assigns to a particular core, the process is assigned to that core. System calls 'system' and 'taskset' are used to achieve processor or CPU affinity.

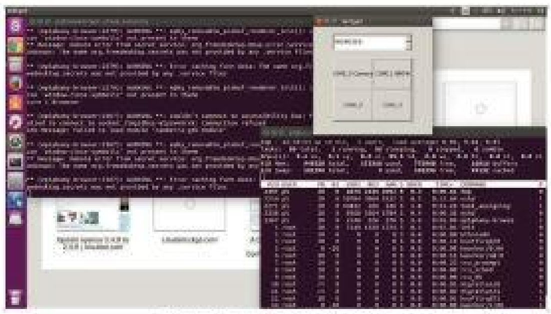
Figure 1 quad core with process

Pseudo code for task affinity on multi-core:

```
• create four core options as four buttons;
• create 'process' options as dropdown menu items;
• read 'process' choice from dropdown menu into
  "process name";
• read the 'core selection' from button;
• assign task on core 'n' using system call: system
  ("taskset n+1 "process name" &");
• 'process' runs on core 'n'
• update the label on button with "process name"
```

B. *Dynamic Core Assignment*

Load balancing on a multi-core system can be achieved by continually monitoring the loads on various CPU cores and assign the desired program to the least loaded core. The proc file system is a pseudo-file system which provides an interface to kernel data structures. It is commonly mounted at /proc. Most of it is read-only, but some files allow kernel variables to be changed. The PID of the desired process is read from process status file of the kernel by tracing the name of the process. With this, the desired process can be fetched. The same way the load on each CPU core at current state is read from the kernel data structures and identified the least loaded core. After these two steps, the desired process can be assigned to the least loaded core using the PID of the process and appropriate affinity mask in the 'taskset' system call.

Proc/stat file contains the CPU usage statistics in ASCII file format. The same can be read into the program and decode the fields to identify the total CPU usage and idle time, and also for all the existing cores. Each core details are given in each line in the file.

In a quad core system, the load balancing program runs on core 0, while monitoring and assigning the loads to cores 1, 2 and 3. This has been implemented through an algorithm as given here.

Simplified load balancing algorithm.

```
//check the process pid of process name from PID list
through /proc
• process_pid = get_pid("process name");
• read file proc/stat;
• read each field using the file pointer;
• while cpu count is less than 5
• read CPU usage for each core
• store the values of CPU core usage data
• increment count until it reaches number of cores
• CPUx usage = time lapsed between reads - CPUx idle
  time
• compare the loads to find the least loaded core
• assign the process to this core using taskset -p
  lowest_load, "process pid"
```

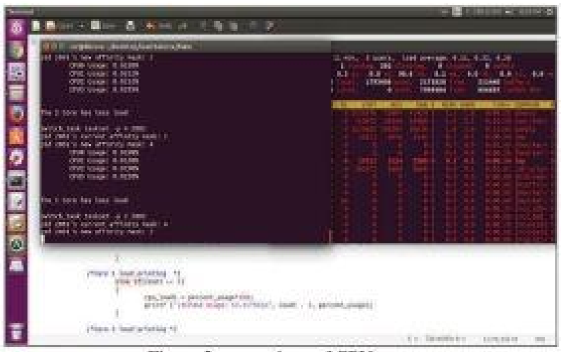
Figure 2 comparison of CPU usage

With this, as shown in figure 2 the load balancing scheduler monitors the load on various CPU cores continually, and assigns the desired process with its process ID to the least loaded core. This ensures that the desired process can be run continuously without waiting for the CPU.

IV. Output

A. *Static Core Assignment*

GUI program is run on the system. Browser, camera, and media player have been given in the drop down list of the program. Four buttons were provided to select any core among 0 to 3. Once Browser was selected and is assigned to core 1, the browser was switched and continually run on core 1. The same was checked using top kernel command from terminal application. Similarly core 0 was assigned to Camera application. The buttons were also updated with their number along with the assigned application. The glitch in the video streaming disappeared after the camera application had the affinity with core 0. Hence it is proven that applications which are non- threaded run better with CPU affinity.

doi:10.32377/cvrjst1113

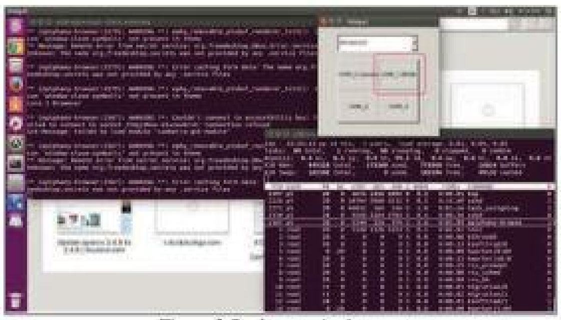
Figure 3 Static core Assignment

Figure 3 shows output of Static core assignment of processes. It shows browser process is running on core 1.

B. *Dynamic Core Assignment*

The main program was compiled using gcc along with other source files. Camera application, 'cheese' was taken as the program that should be assigned dynamically to the least loaded core. The cheese program was launched, and then the load balancer program 'task_switch' was also launched in another terminal. The 'task-switch' program started displaying the loads of various cores, and identified the least loaded core. Camera application was assigned to the least loaded core by changing the affinity mask, which could be either 2, 4, or 8 for core 1, 2, or 3 respectively.

The period of monitoring has been taken as three seconds, and for every three seconds the terminal window was updated with current load values of all cores, and affinity mask was also updated accordingly. The same was checked using top command running on another terminal. The screen shots of the same were given below in figure 4. All the cores were used efficiently with fine load balancing.

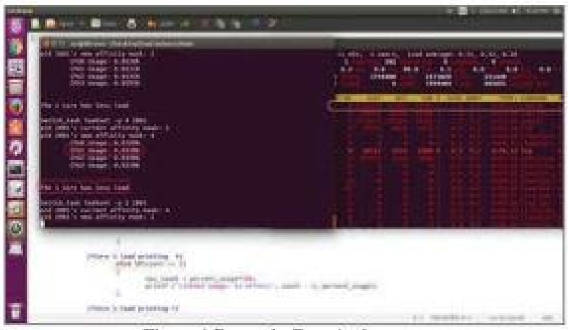
Figure 4 Dynamic Core Assignment

V. Conclusions

This paper tried to touch couple of enormous possibilities to achieve optimal performance of a system, beyond OS kernel level scheduling. Also demonstrated that the programs at application level may be used to enhance efficiency of scheduling further using various monitoring techniques and assignment methods. Further, considering static and dynamic core assignments, to get the best of the both worlds, an algorithm can be developed to mix them, and achieve in a single program, giving the user the control to run the program.

References

[1] Amdahl (1967) "Validity of the Single Processor Approach to Achieving Large-Scale Computing Capabilities" ://www inst.eecs.berkeley.edu/~n252/paper/Amdahl.pdf

[2] J.H. Anderson. Real-time scheduling on multicore platforms. In Real-Time and Embedded Technology and Applications Symposium, 2006.

[3] Robert A Alfieri. Apparatus and method for improved CPU affinity in a multiprocessor system. http: //www.google.com/patents/US5745778

[4] S Brosky. Shielded cpus: real-time performance in standard linux. In Linux Journal, 2004Y.

[5] Knauerhase. Using os observations to improve performance in multicore systems. In Micro IEEE, May 2008.

[6] SimonDerr,PaulMenage.CPUSETS.http://http://www.kernel. org/doc/Documentation/cgroups/cpusets.txt.

[7] "A survey of Multicore Processors", Geoffrey Blake, Ronald G. Dreslinski, and Trevor Mudge IEEE SIGNAL PROCESSING MAGAZINE NOVEMBER 2009 1053-5888/09/IEEE

[8] "A High Performance Load Balance Strategy for Real-Time Multicore Systems,"The Scientific World Journal Volume 2014 (2014), Article ID 101529, 14 pages,Keng-Mao Cho, Chun-Wei Tsai, Yi-Shiuan Chiu, and Chu-Sing Yang

[9] "Chip Multi Processing aware Linux Kernel Scheduler"Suresh Siddha Venkatesh Pallipadi,Asit Mallick,2006 Linux Symposium, Volume Two ,Page Nos. 330-340

[10] "Multi-core and Many-core Processor Architectures", A. Vajda, Programming Many-Core Chips, DOI 10.1007/978-1-4419-9739-5_2, ©Springer Science+Business Media, LLC 2011,Page Nos.- 9 to 36

[11] "A Study on Setting Processor or CPU Affinity in Multi-Core Architecture for Parallel Computing", International Journal of Science and Research ISSN (Online): 2319-7064 ,Volume 4 Issue 5, May 2015, Page Nos.- 1987 - 1990

[12] "Parallel Task Scheduling on Multicore Platforms", Department of Computer Science,The University of North Carolina at Chapel Hill,

[13] "Multi-core and Linux* Kernel ,Intel Open Source Technology center", Suresh Sidhdha

[14] "Real-Time Scheduling on Multicore Platforms", Issue Date: 04-07 April 2006 On page(s): 179 - 190 Print ISBN: 0-7695-2516-4 doi: 10.1109/RTAS.2006.35 Date of Current Version: 24 April 2006

[15] "The Linux Scheduler: a Decade of Wasted Cores", Nice Sophia, Justin Funston.

[16] "A Hierarchical Approach for Load Balancing on Parallel Multi-core Systems",La´ercio L. Pilla , Christiane Pousa Ribeiro , Daniel Cordeiro , Chao Mei , Abhinav Bhatele ,pages 119-129

[17] "Benefits of Cache-Affinity Scheduling in Shared-Memory Multiprocessors: A Summary,"Josep Torrellas, Andrew Tucker, and Anoop Gupta Computer Systems Laboratory, Stanford University, CA 94305,page Nos: 272-274

[18] White Paper Processor Affinity Multiple CPU Scheduling , November 3, 2003

[19] Knauerhase. Using os observations to improve performance in multicore systems. In Micro IEEE, May 2008.

doi:10.32377/cvrjst1113