International Journal of Science and Research (IJSR)
ISSN (Online): 2319-7064
Index Copernicus Value (2013): 6.14 | Impact Factor (2013): 4.438

A Study on Setting Processor or CPU Affinity in Multi-Core Architecture for Parallel Computing

Saroj A. Shambharkar

Kavikulguru Institute of Technology & Science, Ramtek-441 106, Dist. Nagpur, India

Abstract: Early days when we are having Single-core systems, then lot of work is to done by the single processor like scheduling kernel, servicing the interrupts, run the tasks concurrently and so on. The performance was may degrades for the time-critical applications. ***There are some real-time applications like weather forecasting, car crash analysis, military application, defense applications and so on, where speed is important. To improve the speed, adding or having multi-core architecture or multiple cores is not sufficient, it is important to make their use efficiently. In Multi-core architecture, the parallelism is required to make effective use of hardware present in our system. It is important for the application developers take care about the processor affinity which may affect the performance of their application. When using multi-core system to run an multi threaded application, the migration of process or thread from one core to another, there will be a probability of cache locality or memory locality lost. The main motivation behind setting the processor affinity is to resist the migration of process or thread in multi-core systems.***

Keywords: Cache Locality, Memory Locality, Multi-core Architecture, Migration, Ping-Pong effect, Processor Affinity.

1. Processor Affinity

The processor affinity is a tendency for an application to run on particular processor and resist migration. The soft affinity is defined as, when the scheduler will prefer not to migrate a process to another CPU unless needed. This can be overridden with hard affinity assignments in source code. Hard affinity APIs allow the developer to make explicit assignments to a processor or a group of processors. We can decide that where our code runs, and on which processor our code will run that cannot be decided by user, but by setting a CPU bit mask for each thread with the help of calling functions Sched_setaffinity() and sched_getaffinity() in LINUX it can be done[18].

The implementation of multithreading, the master thread forks or generate child threads and then the system allocate task to these child threads. Each child thread executes the independent code in parallel, to schedule them among the existing multiple processors or cores in a multi-core systems, some technique or algorithm can be used. The technique or algorithm will take decision after doing the analysis of usage of each CPU, current load, and other factors, then the threads or task are assign to the processors or cores[12].

2. Need of Parallelization

Previously we are using workstation, now we are using desktop or laptop with multicore architecture. The parallel processing concept can be applied to many field of computer science. It is important to analyze the cpu usage, memory usage and how to efficiently make use of existing processors or cpu's present in our system. Parallel programming is not easy to implement without taking any parallel programming constraints.And also on which processor during parallelelism, where each thread is going to run that is a task of scheduler. Today, the systems are designed with multi-core architecture where multiple core or processors are available, and the designer and the existing operating system for that computer device must distribute the operations as possible as equally to all cores available with that device. Open MP, CUDA, Open Cl and so on provides parallel programming features to design the programs and and implement parallelism.

3. Literature Survey

The overview of work done by different researchers and understood after reading the research papers in the area of parallel processing is given in this section. The paper entitled "Using processor-cache affinity information in shared-memory multiprocessor scheduling" written by Squillante M.S. and Lazowska E.D. they were examined processor affinity in a shared-memory multiprocessor system. They said that in a shared-memory multiprocessor system scheduling a task from one processor than scheduling same task on another processor is efficient if the another processor is having relevant data is available in this processor's cache. They also observed that in shared-memory multiprocessor systems the tasks are continuosly executed and release by the processors. The tasks releases the processor when they have to do input or output operation or synchronization or premption or if time period expires[1].

The paper entitled "Characterization of Scientific Workloads on Systems with Multi-Core Processors", the authors Sadaf R. Alam, Richard F. Barrett, Jeffery A.Kuehn, Philip C. Roth and Jeffrey S. Vetter, they analyzed the behaviour of multi-core system, that is dual-core, of a company AMD opteron, its application kernels and other benchmarks they did for this system. They were also evaluated a different techniques for processor affinity tin order to manage existing memory in a multi-core systems. And they said by appropriate selection of MPI task and memory placements, they derived the conclusion that there will improvement of 25% for performing scientific calculations or application. The micro-benchmark and results they have obtained for this multi-core system[3].

According to William J.P Pilauds, the challenge for the industries is to locate the different technologies, architectures, and system topologies and when one work it amy lead to an invalid conclusions and results after performing or simulating their proposed approach on changing technologies, architecture and arrangement, are changing time to time. The embedded industry is searching and investigating an multi-core processors, which can be a good pltaform for digital Signal processor. The author also has pointed out in his paper about the difficulties may occur due to change in processors architectures and others chages in the specification of the computer systems. The paper written by William J.P.Pilauds x after the investigation of an Intel processor, proposed a processor FFTW (FFT in W) inorder to overview the changes in architectures[4].

In paper entitled as "Parallel 1-D FFT Implementation With TMS320C4x DSPs", the author Rose Marie Piedra presented two special cases the first is the size of input data for the large FFT does not fit in processors on-chip random access memory, and the execution in such case, the computation is performed but the performance is degrades, the large input data is off-chip. Due to this, the execution time required for this FFT computation grows exponontially with input data size. Second case is the same computation done by multiprocessing system's and more processors available and reduces the execution time. There is speed-up but it increases the inter-processors communication overhead [7].

The paper entitled as "An efficient Practical parallelization Methodology for Multi-core Architecture Simulation" wriiten by James Donald and Margaret Martonosi presented a programming methodology which is converting the uniprocessor simulators into multi-core simulators. The methodology used by them is required less development effort compared to other other programming techniques.

And another advantage of this programming methodology is after conversion, the obtained multi-core simulator retains a modular and comphrensible programming structure. They have used simple-scalar based and PTCMP frameworks. They have successfully these two frameworks and achieved the speed-up of 1.5X and 2.2X., and the opteron server used has been used as dual-cpu dual-core[9]. The NUMA architecture described by Gabriele Fatigati, the memory is divided in local or faster and slower or remote areas[10].

The paper entitled as "Achieving High Performance With TCP over 40 GbE on NUMA Architectures for CMS Data Acquisition", mentioned in their paper that the TCP and socket abstraction is changing time to time, but at the network layer, it has been found a gaint leap from a few megabits to 100 gigabits in bandwidth. Today, in multi-core architecture era, applications are expected to make full utilization of resources available to us. In a Non-Uniform Memory access architecture, the data acquistion based applications when running the standard socket library, they are unable to reach full efficiency and scalability, if the software is not aware about the 1)Interrupt request 2) CPU 3) memory affinities. In this paper, a new software component CMS online-framework is developed, which they are using for transferring data with sockets. The framework is event-based application with NUMA optimizations, and also it permits to transfer data across a large distributed system giving high throughput [11].

The paper entitled as "Sensmart Adaptive stack management for multitasking sensor Networks", introduced a SenSmart multitasking operating systems for sensor networks. It is having the multitasking capability, but the limitation when small-memory system, the traditional stack management techniques are not performed well. There is not having any support when using small-memory system's. In this paper, an efficient operating system with the help of combined binary translation and a new kernel runtime was developed. They introduced a new design of an operating system which has improved preemptive multitask scheduling memory isolation solves critical stack management problems, flexible multitasking. The operating system used by them having an advantage that the programmers don't have any burden of estimating tasks, stack usage, it helps to schedule and run more tasks as compared to other multitasking operating system's exist for sensor networks. They concluded after using the Sensmart multitasking operating systems for sensor network better capability to manage concurrent to other sensor network operating system[12].

4. Setting Processor Affinity

4.1 Tools and Ping- Pong Effect

A PI-Tool to achieve better performance of a Nqueen's Program. They have used existing FIFO and Round-Robin scheduling algorithms. They compared the PI-tool with other tools. They have discussed about the ping-pong effect. The ping-pong effect occurs in the situation when the scheduler bounces or migrates the processes between the multiple cores, each time the process schedule or rescheduled. This operation is costly due to cache invalidation and may degrades the performance of the system. Earlier, when one is using single core system, the people are not worry about ping-pong effect. In this paper, the PI-Tool used to reduce this ping-pong effect. The tool used is also increasing the priority of the process, changing the low to high priority and binding that process for longer period of time to any cpu. But, the drawback mentioned by them is other process waiting for the same CPU may delayed [2].

The other tools based on CUI, Character User Interface tools Top, Htop and Sysstat, not very user friendly, there is need of remembering commands. Other GUI (Graphics User Interface) based tool specified in this paper is Gnome or KDE system monitor, which is monitoring and changing the priority of a process, but there is no option available to utilize the benefits of CPU affinity[2].

The other tools specified are Perfsuite and Red Hat Linux Tuna, the Perfsuite tool make use of libraries to get the information related to the performance of the processes. The Red Hat Linux tuna GUI based tool and it is best tool, helping in maximizing the performance, but having some drawbacks. They are 1) there is option available to remove all processing from the processor and making it idle. 2)

installation requires registration 3) additional packages required. To avoid the problems associated with CUI and GUI based tools mentioned can be reduce by PI-tool. This tool is using the techniques of processor or CPU affinity and process priority. They have mentioned the advantages of using PI-Tool as avoiding the ping-pong effect, helps in increasing the priority of a process and can be used on any linux operating system. the disadvantage presented of PI-tool is as increasing the process priority the other process may delayed[2].

4.2 Multi-Core Architecture and Issues

The parallel processing is used in different engineering and application. Using multi-core system's, users can utilize all available cores, and more than one task can be perform at the same time. Without parallelization or without running

the code in parallel, and there will not be 100% CPU utilization unless the application code runs in parallel on different cores. They did experiment on FFT algorithm, and concluded using parallel approach, it requires less computation and memory resources, it suugested for multithreaded model[13].

The authors of paper "Characterization of Scientific workloads on systems with Multi-core Processors" investigated about the scaling behavior of a kernels, set of benchmarks, application on AMD opteron Dual-core Processor systems. In this paper, they have evaluated processor affinity techniques, and manage the memory resources on multi-core systems[14].

"Efficient threads mapping on multicore architecture" written by Iulan Nita and others, used a proper programmming strategy for optimal mapping of all processes to the resources available in the system, in parallel computing and comparision is done using efficient mapping algorithm and without this algorithm. The applications running on multiple cores on a single chip requires parallelism. The mapping algorithm, may lead to improve the performance of the system and less energy consumption. Their proposed algorithm simulation is done on Intel Core2 Duo T5879 2 Ghz and Intel Quad core Q6600 2.4 Ghz machines[15].

The authors of paper "Dynamic Load Balancing for real-time Video encoding encoding on heterogeneous CPU+GPU systems" targeted to achieve efficient parallelization and the RD performance analysis and done on multicore as well as on multi-GPU systems of H.264/AVC inter-loop modules. They proposed an dynamic load balancing algorithm which allows efficient and concurrent video encoding across several heterogeneous devices and relying on module-device execution affinities and on realistic run-time performance modeling. When they evaluated results using proposed algorithm for video inter-loop encoding on single GPU device and highly optimized multi-core CPU, the execution speed-up values found were 2.6 on a single GPU-device and 8.5 speed-up value on multi-core CPU [16].

The paper entitled as "A new Generation of Real-Time Systems in the JET Tokamak", the authors focused on the configuration aspects to enable the real-time capability in the system's. They have used the framework named as multithreaded application Real-Time executor to build the application particularly optimized for exploring multi-core architectures. A jitter analysis of this framework also presented in the paper. They compared the performance of this by running it on a Linux Vanilla Kernel and on a Messaging Real-Time Grid(MRG)Linux kernel[17].

5. Objectives of Setting Processor Affinity

There are some objectives for the proposed research study 1) Mapping or binding the running threads or tasks to specific core in multi-core systems and resolving issues related to it, 2) To reduce cache problems, 3) To resist migration of processes between the processors 4) Proper balancing of available cores through proper load distribution, allocating specific amount of work or task to each core, 5) To improve computational time, 6) To optimize cache performance, 7) To utilize the time quantum in a multhreaded application, 8) To improve the performance of running applications, The specific aspect after study is to design a new algorithm for scheduling a task or to assign it to specfic core among available cores in a multi-core architecture.

6. Significance of Processor Affinity

The question comes is whether the proposed research is having any significance or is its importance matters in today's multi-core systems. As we know Windows XP, Windows 7 automatically sets their affinity to different processors for laod distribution. But, if we have multiple cores and not utilizing them the CPU properly, that is only one application running at a time, then its a drawback of using multi-core systems[3].

With multiple cores on a single chip, the tasks were running in parallel, a proper algorithm is required to map the running processes/thread of a application by scheduling each on the multiple processors, without interrupting, or migrating them time to time may degrades the performance. The efficient mapping algorithm will set the processsor affinity for running processes.
There is 25 % performance improvement for scientific application using proper processor affinity techniques, proper selection of MPI task and proper memory management[13].

The results of parallel computing using efficient mapping algorithm, to map threads to specific cores with the parallel computing without using mapping algorithm were compared. In second approach that is without using efficient mapping algorithm, the thread mapping is done by Linux Thread scheduler. Mapping of running thread to specific core is decided by the algorithm, not by existing Linux Kernel scheduler[14].

It is advantageous for the real-time applications where time, the fast response is important. The user wants to run his/her application they desire to run them faster than the normal

execution. To achieve the maximum performance there is a need of setting priority of running processes, processor affinity, scheduling of task and so on. The processor affinity allows us to select the processor /CPU where user wish to run on our task or process.

7. Conclusion and Future Work

The conclusion after study the variuos research papers is, one of the important issue is to improve the performance of running applications with multi-core systems use proper tools programming to take benefits of multiple cores.And to resist the migration in a multi-core systems, design or implement some techinques, tools or approach to scheduling a task or process or threads if multi-threaded application or to set the processor affinity, to assign task or process to specfic core in a multi-core architecture.

References

[1] Squillante M.S, Lazowska E.D, "Using Processor-cache affinity information in shared-memory multiprocessor scheduling", IEEE Transactions on Parallel and Distributed Systems, pp-131-143, Vol. 4, Issue(2), Feb1993.
[2] P. Bala Subramanyam raju, P.Govindarajulu, "PI-Tool to improve performance of Aplication in Multi-core Architecture", International Journal of Computer Science and Security (IJCSS), Vol. 8, Issue(4), 2014.
[3] Sadaf R. Alam, Richard F. Barrett, Jeffery A.Kuehn, Philip C. Roth and Jeffrey S. Vetter"Characterization of Scientific Workloads on Systems with Multi-Core Processors", Computer Science and Mathematics Division Oak Ridge National Laboratory, Oak Ridge, TN, USA 37831.
[4] William J. Pilauds, "Improving FFTW Benchmark to Measure Multi-core Processor Performance ", Curtiss Wright Controls Embedded Computing, April 29, 2009.
[5] Duc Vianney, "HyperThreading Speeds Linux Multiprocessor performance on single Processor", Linux Kernel Performance Group, Linux Technology Center IBM, Software Group, 01 January 2003.
[6] Chi Zhang, Xin Yuan, Ashok Srinivasan, " Processor affinity and MPI performance on SMP-CMP Clusters", Department of Computer Science, Florida State University, Tallahassee.
[7] Rose Marie Piedra, "Parallel 1-D FFT Implementation With TMS320C4x DSPs", Digital Signal Processing — Semiconductor Group, SPRA108 February 1994.
[8] Gabriele Fatigati, "The affinity model", CINECA Supercomputing group.
[9] James Donald, Margaret Martonosi, " An efficient Practical parallilization Methodology for Multi-core Architecture Simulation", Department of Electrical Engineering, Princeton University, IEEE Computer Architecture Letters, Vol. 5, 2006.
[10] Tomasz Bawej, Ulf Behrens, James Branson, Olivier Chaze, Sergio Cittolin, Georgiana-Lavinia Darlea, Christian Deldicque, Marc Dobson, Aymeric Dupont, Samim Erhan, Andrew Forrest, Dominique Gigi, Frank Glege, Guillelmo Gomez-Ceballos, Robert Gomez-Reino, Jeroen Hegeman, Andre Holzner, Lorenzo Masetti, Frans Meijers, Emilio Meschi, Remigius K. Mommsen, Srecko Morovic, Carlos Nunez-Barranco-Fernandez, Vivian O'Dell, Luciano Orsini, Christoph Paus, Andrea Petrucci, Marco Pieri, Attila Racz, Hannes Sakulin, Member, IEEE, Christoph Schwick, Benjamin Stieger, Konstanty Sumorok, Jan Veverka, Christopher C. Wakefield, and Petr Zejdl, "Achieving High Performance With TCP over 40 GbE on NUMA Architectures for CMS Data Acquisition", IEEE TRANSACTIONS ON NUCLEAR SCIENCE, February 26, 2015.
[11] Rui Chu, Lin Gu, Yunhao Liu, Mo li and Xicheng Lu, "Sensmart Adaptive stack management for multitasking sensor Networks", IEEE TRANSACTIONS ON COMPUTERS, Vol. 62, No.1, January 2013, pp. 137-150.
[12] Faran Mahmood, Bilal A.Khan, "Parallelism and performance Comparison of FFT on Multi core Machines", Institute of Space Technology, Islamabad Highway.
[13] Iulian Nita, Adrian Rapan, Vasil Lazarescu, Tiberiu Seceleanu, "Efficient Threads mapping on Multicore architecture", 8th International Conference on Communications", 2010, pp. 53-56.
[14] Svetislav Momcilovic, Aleksandar Ilic, Nuno Roma, Leonel Sousa, "Dynamic Load Balancing for real-time Video encoding encoding on heterogeneous CPU+GPU systems", IEEE TRANSACTIONS ON MULTIMEDIA, Vol.16, No.1, JANUARY 2014.
[15] Diogo Alves, André C. Neto, Daniel F. Valcárcel, Robert Felton, Juan M. López, Antonio Barbalace,
[16] Luca Boncagni, Peter Card, Gianmaria De Tommasi, Alex Goodyear, Stefan Jachmich, Peter J. Lomas, Francesco Maviglia, Paul McCullen, Andrea Murari,
[17] Mark Rainford, Cedric Reux, Fernanda Rimini, Filippo Sartori, Adam V. Stephen, Jesus Vega, Riccardo Vitelli, Luca Zabeo, and Klaus-Dieter Zastrow, "A new Generation of Real-Time Systems in the JET Tokamak", IEEE TRANSACTIONS ON NUCLEAR SCIENCE, VOL. 61, NO. 2, APRIL 2014, pp. 717-719.
[18] Mike Aderson, "Understanding and Using SMP/Multi-Core Processors, New Hardware and How to use it", The PTR Group, Inc. 10/28/2008.
[19] Barbara Chapman, Gabriele Jost, Ruud van der Pas, " Using OpenMP Portable Shared Memory Parallel Programming, "The MIT Press, Cambridge, Massachusetts London, England, pp. 28-34.