E-ISSN 2581 - 7957
P-ISSN 2277 - 3916

CVR Journal of Science and Technology, Volume 19, December 2020
DOI: 10.32377/cvrjst 1915

Performance Prediction of Task Workloads in Work-Stealing Runtimes for NUMA Multi-core Architectures

J. Yashasree[1], A. Mallareddy[2] and Dr. B. Vikranth[3]
[1]Asst. Professor, CVR College of Engineering/CSIT Department, Hyderabad, India
Email: yashasree123@gmail.com
[2]Assoc. Professor, CVR College of Engineering/IT Department, Hyderabad, India
Email: malla.reddy@cvr.ac.in
[3]Professor, CVR College of Engineering/IT Department, Hyderabad, India
Email: b.vikranth@cvr.ac.in

Abstract: **Work stealing is a popular load balancing technique which is successfully adapted by various user level task parallel runtime systems such as Cilk, TBB etc. On the other hand, processor manufacturers are working on releasing processors with more than one socket on chip, to overcome the memory wall problem. If work stealing based run-time systems are ported onto such multi-socket machines, it may lead to performance issues. This paper has attempted to study the impact of such multi-socket multi-core architectures on performance of work stealing based run-times. In this paper, a work stealing parameter called remote steal count is introduced specially for assessing work stealing run-times on multi-socket architectures. This paper also proposes a simple regression model between work stealing parameters and performance of task-based applications. This model is helpful to predict the performance impact of task parallel applications while porting them into multi-socket hardware environment.**

Index Terms: **Multi-core, Multi-socket, Non-Uniform Memory Architecture, Load balancing, Work Stealing, worker threads, Regression.**

I. INTRODUCTION

A. Features of Modern Multi-Core Architecures

Modern High-Performance Computing processor consists of more than one integrated memory controller (IMC) on a chip to fill the gap between fast growing speeds of CPU and compatible data delivery rates of memory units [1]. Introducing more than one such IMC serves the data needs of threads pinned to cores belonging to different chips simultaneously. By deploying these multiple DRAM controllers, the memory bandwidth is improved and contention for single memory controller hub can be reduced [2]. The processors cores are grouped and deployed in a socket. These processors are High Performance Server processors such as Intel Xeon or AMD Opteron. The processors are connected with high speed links such as Quick Path Interconnect (QPI) links [2] [3] from Intel or Hyper transport links from AMD. These links allow more than one socket to be deployed in a single pack on high performance servers. The presence of multiple memory controllers make these processors behave as Non Uniform Memory Architecture (NUMA). A dual socket Xeon E5-2620 series processor architecture which is studied in this paper as an experimental platform is presented in Fig. 1.

Figure 1. Dual socket Xeon E5 2620 processor [4]

It can be observed from the Fig 1 that it is a two-socket (2 NUMA nodes) architecture since, two separate memory controllers (MC) are attached to individual sockets. It means that all the six cores present in a socket can access the memory bank's memory through the local memory controller. If these cores need to access any data from memory attached to a remote socket, they experience more memory latency. A thread which is pinned onto a CPU core, can access data from a memory bank connected to its local memory controller at a faster rate than that of a remote memory bank connected to different sockets. The ratio of the remote memory access latency to the local memory access latency is called NUMA ratio (R_{NUMA}) and is given by

$$R_{NUMA} = \frac{T_{RemoteAccess}}{T_{LocalAccess}} \quad (1)$$

CVR Journal of Science and Technology, Volume 19, December 2020
DOI: 10.32377/cvrjst 1915

where $T_{LocalAccess}$ represents the local memory access time and $T_{RemoteAccess}$ represents the memory access time on remote memory node. It is obvious that remote memory access latency is greater than local memory access time.

In the experimental setup, Intel's Memory Latency Checker [5] program is executed for calculating the memory latency values of local and remote memory accesses. These values are presented in TABLE I. This table clearly shows that a thread is pinned to one of the cores on socket 0. The latency involved in accessing a data item from socket 0 is 77.3 nano seconds whereas if the data item is located on socket 1 memory module, the latency is 124.7 nano seconds. The other way around is also approximately the same.

TABLE I.
. LOCAL AND REMOTE ACCESS LATENCIES IN XEON E5-2620

socket	0	1
0	77.3 nS	124.7 nS
1	122.8 nS	75.0 nS

R_{NUMA} factor is computed substituting the values of from TABLE I. R_{NUMA} is considered as the average of remote latency accesses by socket 0 and socket 1 which yield 1.625 for Xeon E5 2620 architecture.

B. Effect of NUMA architectures on Work Stealing run-time environment

Work stealing algorithm is a popular load balancing approach used in many user level run-time systems such as Cilk [6], Intel Threading Building Blocks (TBB) [7], Wool [8] and few implementations of OpenMP [9]. The common approach followed in work stealing is: during the initialization of the run-time system, a new worker thread is created which is associated with each processor core at hardware level. Associated with each worker thread, there is a task queue that contains the tasks spawned by the application program. These tasks are added to the queue. Each worker thread pops out one task at a time and executes the body of the task. Since the tasks arrive randomly as the user program instantiates tasks, the queues associated with worker threads may contain different counts of tasks, thereby causing imbalance in the load. Keeping the goal of balancing the load among worker threads as final objective, work-stealing strategy designates the worker thread which is under loaded as a *thief worker.* The thief worker attempts to *steal* tasks from another worker's queue. The worker from which one or more tasks are stolen is called a *victim worker*. If work stealing technique is applied in NUMA multi-core architectures:

- Identity affinity must be guaranteed. Individual worker thread is pinned to a processor core belonging to a particular node.
- If underlying hardware is NUMA, the memory locality of these worker queues is important since it is frequently accessed by the associated worker thread. In other words, the worker queue must be bound to the memory node(socket) where the worker thread is pinned to. Worker threads access these task queues in almost fully distributed way except in the instance of stealing occurrence i.e. regular job of the worker thread is to pop tasks from its own queue and execute the job on its processor. If locality of these queues is not considered, and if the worker thread and its task queue are mapped to different nodes due to the default first-touch policy of Linux [10] the overall performance may be affected due to increased remote memory access.

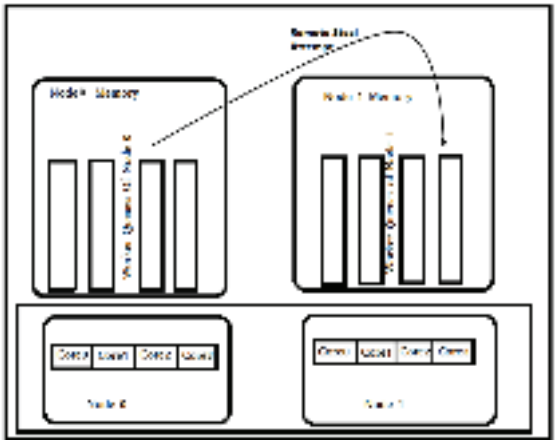

Figure 2. A thief worker attempting to steal task from remote node

- If a thief and victim are pinned to two different cores belonging to different sockets, the delays involved in stealing introduce additional performance overhead because of NUMA.

In this paper, the concept of remote stealing is analyzed, and a technique is proposed to predict the remote stealing attempts based on our experimental results.

C. Related Work

The effect of scheduling different threads that access different data and shared data is analyzed in [11] but the processors considered here are single socket machines with multiple cores. This approach is confined to cache miss values when data is shared among multiple threads on a multi core processor when more than one core has common last level cache. Threads that access shared data that is bound to different NUMA nodes and its effects is analyzed in [1]. Machine learning based performance prediction of applications on single socket multi-core processors by using the cache level parameters is done in [12]. Comparison and performance analysis of different data structures as task-queues in work stealing run-time systems is done in [13]. This proposed model in this paper is inspired by [12] but our work is confined to prediction of task parallel application's execution times on work stealing run-time systems with multi-socket processors. To the best of our knowledge, the work carried out is unique and the proposed model is useful in studying the effect of remote steal attempts on overall performance of an application.

D. Organization of the paper

Section II of this paper describes the motivation of the work behind the proposed strategy. In Section III, the experimental setup and the procedure followed in order to obtain work stealing parameters is elaborated. The mathematical model is proposed in Section III based on the empirical analysis. Section IV contains the concluding points of the proposed strategies in the paper.

E-ISSN 2581 - 7957
P-ISSN 2277 - 3916

CVR Journal of Science and Technology, Volume 19, December 2020
DOI: 10.32377/cvrjst 1915

II. Motivation

The work-stealing strategy proposed is also called as randomized work-stealing [14] since it follows a technique for choosing a victim by generating a random number. The random number generated indicates the victim worker. If the architecture on which the work stealing runtime is deployed has a multi-socket NUMA architecture, the processors are divided into equal portion on different sockets. In our experimental platform (Xeon E5-2620), there are two sockets each, with its own memory banks. A total of 12 cores are arranged as 6 cores per socket. All the 6 cores of the socket can access its local memory bank at low latency. If the work stealing is randomized, the random victim chosen by the algorithm may refer to a worker with remote socket affinity.

- In randomized work stealing strategy, whenever a worker thread finds no tasks in its own task queue, it becomes a thief, and it can randomly choose a worker queue as a victim for stealing tasks. But if randomly chosen worker thread is pinned to a core belonging to a different node, it is a remote steal attempt. This scenario is depicted in Fig. 2.
- As a result of random stealing, unrelated tasks stolen from other workers brought to execution on local worker thread may result in performance isolation problems [11].

III. Experimental Results

A. Platform description

MATMUL benchmark program was executed on a hardware platform with dual socket Xeon-E52620 processor where each socket has 6 cores (12 hyper threads). Hyper threading was intentionally disabled in BIOs while carrying out our experimentation to ensure that cache contention effects are minimized on the performance. The physical memory of the hardware platform is a total of 16 GB, where each socket is attached with its own integrated memory controller connected to 8GB RAM. The work-stealing layer of target experimental platform consists a total of 12 worker threads pinned to each core.as shown in Fig.3**.** Separate worker queues are associated with each worker thread. Dispatcher module is responsible for initial static-distribution of the tasks equally among all the worker queues. The actual load imbalance occurs because of run-time characteristics of tasks, since each task may execute for different quantum of time when it is scheduled on to a worker thread. These worker threads behave as software level virtual processors, within the work stealing infrastructure.

To analyze the remote work stealing effects, MATMUL benchmark [16] implemented using work stealing runtime is executed on the target experimental platform. The size of matrix is taken as 8192×8192 for the following reasons:

- The size of matrix is intentionally taken as 8192 to be greater than the kernel supported virtual memory page size, to ensure that the data section of the program occupies multiple pages in virtual memory system. Consideration of a large matrix also increases the possibility of pages mapping across multiple nodes of NUMA architecture.
- Huge number (8192×8192) of tasks must be generated quickly so that the worker queues filled quickly causing a load imbalance in terms of tasks thereby causing considerable number of task-stealing attempts by the worker threads.

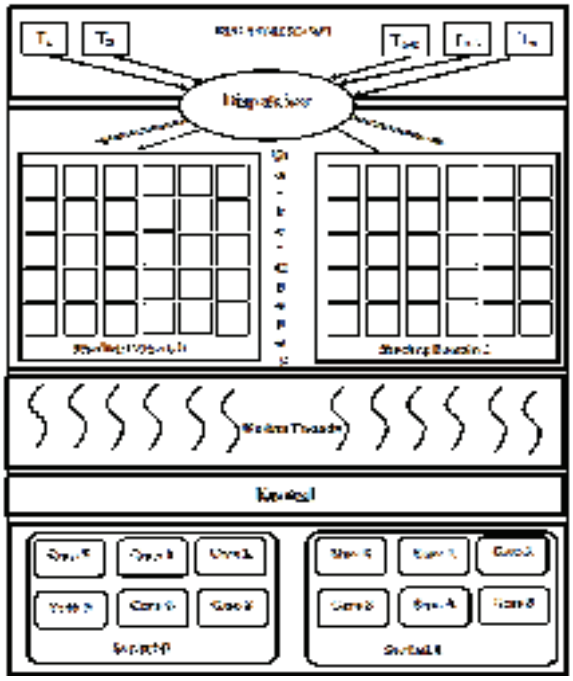

Figure 3. Modular Architecture of Experimental Platform

The MATMUL workloads are run using randomized work stealing policy with the additional feature of *thread to CPU pinning* policy. Remote steal count is measured for varying number of worker threads. It is ensured that the number of worker threads is even, such that two different halves of worker threads are bound to two different sockets during the execution of experiments. For instance, the first entry on the TABLE II represents the case with only two worker threads. In such a case, one thread has been explicitly bound on to a core belonging to one node and the other worker thread onto a different core belonging to different node. Explicit control of thread affinity is made possible using *sched_setaffinity* control functions of Linux environment.

While analyzing the remote steal attempts, special *work stealing parameters* were introduced in the source code to find the values of the following interested counters:

- *Remote Steal Attempts:* counts how many randomly generated victim indexes are leading to refer a worker thread on remote nodes.
- *Remote False Steals* count how many randomly generated victim indexes are leading to a failure to select a proper victim.

The common approach followed in randomized work stealing to choose a victim is:

$$victim\_id = random\ (seed)\ \%\ ncpus \qquad (2)$$

Where *ncpus* represents the number of processors or hardware-threads in the machine. On a multi-socket NUMA

CVR Journal of Science and Technology, Volume 19, December 2020
DOI: 10.32377/cvrjst 1915

multicore machine, n is the count of CPUs on all nodes. For instance, if the target platform is with 2 sockets where each socket has 6 processors, randomized work stealing strategy generates the victim index in range *[0:11]*. Among these generated indexes, the indexes generated in range [6:11] will result in a remote memory access. In the run-time's source code, a new parameter *Remote Steal Attempts* is used to count such remote event occurrences. Sample average values are presented in TABLE II after running the experiment for 10 times. It can be observed from the TABLE II that, 50% of the steal attempts are remote. Techniques to minimize the value of false steal count were proposed in [15]. Similar approaches can be useful for NUMA multi-core runtime; the second counter Remote False Steals can be minimized.

TABLE II.
REMOTE STEAL MISS RATIOS IN RANDOMIZED WORK STEALING

Number Of Worker Threads	Average Remote Steal Count	Average Local Steal Count
2	23	21.6
4	41	49
6	46	54
8	56	55
10	66	56
12	61	45
14	123	88
16	243	235
18	334	401
20	354	473
22	226	312
24	102	77

B. Model for performance prediction

To investigate the effect of remote steal attempts on the performance of the workload, the MATMUL benchmark [16] has been executed for 50 times on dual socket architecture. Sample values of this experiment are presented in TABLE III. Unlike considering different number of worker threads as shown in TABLE II, this time, the number of threads is confined to the number of cores (i.e. 12 cores with 12 worker threads) to maintain identity affinity. The remote *tasks-steal-attempt*s parameter is obtained from the experiments and respective execution times are noted. Using these results, an effort is put to study the impact of remote stealing attempts on overall performance of the task parallel matrix multiplication application.

TABLE III.
SAMPLE VALUES OF REMOTE STEAL ATTEMPTS ON EXECUTION TIME

S. No	No of worker threads	Local Steals count	Remote Steal count	Execution Time
1	12	90	81	1867
2	12	99	74	1966
3	12	75	51	1835
4	12	86	86	1848
5	12	47	20	1781

While proposing the mathematical model, Pearson Correlation Coefficient Calculator is used on the obtained data, the value of R was 0:9123 and it is a strong positive relationship between remote steals and execution time. The relationship is presented in the Fig. 4. The results of regression analysis are presented below where X represents the remote steal count and $\hat{y}$ indicates the execution time experienced by the MATMUL benchmark application.

Sum of X = 3555

Sum of Y = 91885

Mean X = 71.1

Mean Y = 1837.7

Sum of squares (SSX) = 127796.5

Sum of products (SP) = 205740.5

Regression Equation = $\hat{y} = bX + a$

b = SP/SSX = 205740.5/127796.5 = 1.60991

a = MY - bMX = 1837.7 - (1.61*71.1) = 1723.2356

$$\hat{y} = 1.60991X + 1723.2356 \quad (3)$$

It can be observed from (1) and (3) that the *b* value 1.6099 is approximately equal to R_{NUMA} value 1.625 of (1). This is very important conclusion which shows the strong relationship between R_{NUMA} factor and the performance. Fig. 4 shows the predictive model of performance given the number of remote steal counts as input. The error value between predicted performance (50 values) and emperical performance (50 values) is computed using root mean square method that yeilded error value 5.1683. This is a insignificant error with respect to execution time factor and the model is apt for prediction of performance.

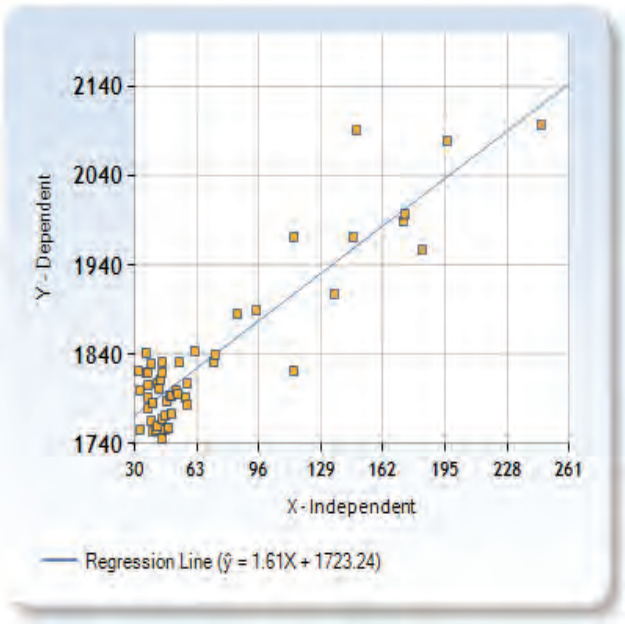

Figure 4. Regression relationship between remote steal count and execution time

CVR Journal of Science and Technology, Volume 19, December 2020
DOI: 10.32377/cvrjst 1915

IV. Conclusions

In this paper, the impact of multi-socket multi-core architectures on randomized work stealing load balancing algorithm implementation is analyzed. This paper also proposes a linear regression model to predict the performance of task-based application, based on the work stealing parameter called remote steal count. The proposed model confirms the dependency of performance of an application on R_{NUMA} value of the target architecture.

References

[1] Z. &. G. T. R. Majo, " Memory system performance in a NUMA multicore multiprocessor", in In Proceedings of the 4th Annual International Conference on Systems and Storage , (2011, May)..

[2] A. B. R. A. M. A. R. J. Dimitrios Ziakas, "Intel R quickpath interconnect architectural features supporting scalable system architectures", in IEEE 18th Annual Symposium on High Performance Interconnects, 2010.

[3] D. E. A. Ziakas, "Intel® quickpath interconnect architectural features supporting scalable system architectures.", 2010.

[4] "IntelR . Xeon E5 2620v3 processor architecture", Intel,2014.[OnlineAvailable: ProcessorE5-2620-v3-15M-Cache-2.40-GHz.

[5] K. K. A. T. W. V Viswanathan, ". Intel memory latency checker", Intel Technology Journal, 2015.

[6] C. F. J. B. C. K. Robert D Blumofe, "Cilk: An efficient multithreaded runtime system. Journal of parallel and distributed", Journal of parallel and distributed computing, vol. 37, no. 1, pp. 55-69, 1996.

[7] C. Pheatt, "Intel R threading building blocks," Journal of Computing, vol. 23, no. 4, pp. 298-298, 2008.

[8] K.-F. Faxén, "Wool-a work stealing library", 2009.

[9] F. F. N. G. B. N. R. &. W. P. A. .. Broquedis, "Dynamic task and data placement over NUMA architectures: an OpenMP runtime perspective", 2009.

[10] M. D. D. H. A. G. H. Martin J Bligh, " Linux on NUMA systems”., volume 1, in In Proceedings of the Linux Symposium, Pages:89-102, 2004.

[11] F. G. S. K. A. Y. S. Dhruba Chandra, " Predicting inter-thread cache contention on a chip multi-processor architecture”., pages 340-351. IEEE, 2005 in 11th IEEE International Symposium on High-Performance Computer Architecture HPCA-11. , 2005.

[12] A. N. A. R. W. Jitendra Kumar Rai, "Using machine learning techniques for performance prediction on multi-cores. In Applications and Developments in Grid, Cloud, and High Performance Computing", Applications and Developments in Grid, Cloud, and High Performance Computing. IGI Global, pp. 259-273, 2013.

[13] B. Vikranth, "Performance Analysis of Load Balancing Queues in User Level Runtime Systems for Multi-Core Processors", CVR Journal of Science and Technology, vol. 11, pp. 87-90, 2016.

[14] Robert D Blumofe and Charles E Leiserson, " Scheduling multithreaded computations by work stealing", Journal of the ACM (JACM), vol. 46, no. 5, pp. 720-748, 1999.

[15] B. R. W. A. C. R. R. Vikranth, "Topology aware task stealing for on-chip NUMA multi-core processors", in Procedia Computer Science, Barcelona, 2013.

[16] J. P. P. A. P. S. C. Burkardt, "MATMUL: An Interactive Matrix Multiplication Benchmark," IETE Journal, p. 640, 1995.

[17] I. Panourgias, " Numa effects on multicore, multi socket systems", The University of Edinburgh, 2011.