*Journal of Advances and Scholarly Researches in Allied Education*
*Vol. XIV, Issue No. 1, October-2017, ISSN 2230-7540*

# Performance and Energy Efficiency of Parallel Processing in a Multiprocessor System

**Mahak Dhanda***

M. Tech (Computer Science) Net Qualified

***Abstract – Performance of the multi-processor system analysed by them employed multitasking in case, there are fewer programs in the system requiring CPU service Than the number of CPUs. However, if there are alleast as many programs requiring CPU service as there are CPUs, then each CPU works on an independent program. In Their model, each program is broken up into two completely independent tasks when tv/o processors cooperate on a single program, A review of the work dona in the field of performance analysis of multiprocessor systems based on queueing models is contained in £ 110J, Sauer and Chancly have studied the behaviour of a multiprocessor system under various scheduling strategies and CPU service distributions.***

***Keywords:- Multiprocessor, CPU, Multiprogramming, Processors, Independent Program etc.***

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - x - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

## INTRODUCTION

It is possible to reduce the execution time of a program if it is decomposed into a number of independent tasks for simultaneous execution on different processors of a multiprocessor system, A method for partiti_oning a 'serial' program into tasks consisting of a single statement or a group of statements has been discussed in the previous chapter. Studies have revealed that for a single FORTRAN program? The number of independent tasks obtained by program decomposition and capable of simultaneous execution may not be sufficient for utilising the entire processing capacity of a multiprocessor system. A more meaningful use of such a system would be the sharing of the processor resources among a number of programs each of which is decomposed into a number of Independent tasks. This reduces idle time of the processors. A CPU will find a task waiting for execution most of the time. Multiprogramming a number of jobs in such parallel processing mode is essential for increasing the overall utilization of the system bpt may cause serious contention for processor resources and an inordinate delay to the program execution time if the degree of multiprogramming is too high. Here we present a study based on probabilistic model of a method for scheduling the tasks on different processors with a view to reducing the delay caused by processor contention. Parallel processing has received enormous attention in the last 50 years, c.f. fundamental presentations in the books as (Marsan, 1989. Kobayashi and Mark, 2009. Reiser and Lavenberg, 1980). Most of the studies in the early times of computer science addressed problems of scheduling tasks with given task processing times to be processed on a single or few processing elements under various scheduling strategies based on constant processing times, order of arrival, priority classes or deadlines for the execution. Many results are known from this research on optimum scheduling with respect to the shortest possible execution duration until completion of a given workload. For the description of more complex systems with precedence and synchronization constraints, Petri Nets (PN) have proved as an excellent modeling methodology to guarantee the correct execution and to detect deadlock situations by the control of state transitions using places and tokens but were not able to express performance phenomena as a result of the absence of time. This deficiency was later corrected by the introduction of timed Petri Nets and stochastic Petri Nets (SPN) where state transitions were extended by deterministic or stochastic durations. For the processing of such generalized Petri Nets, powerful tools were developed either for the simulation or for an analytical evaluation under Markovian process assumptions. Simultaneously to the developments of scheduling parallel computing as described before, queuing network theory has progressed extensively within the last 5 decades which is expressed by the phenomenon of “product-form” queuing networks and efficient algorithms for their numerical performance evaluation. Queuing networks allow for modeling of parallelism at large but are severely limited with respect to synchronization constraints and generalized stochastic arrival and service processes beyond Markovian assumptions. These deficiencies have partly been overcome by approximate evaluation methods and powerful computer tools for queuing network analysis and simulations, see, e.g. (Adve and Vernon, 2004). Apart from the state-of-the-art reached in queuing theory and through SPN, main problems remained open as decomposition methods to reduce complexity in the evaluation and how to apply the results practically as, e.g., to detect parallelism in the program execution path (at instruction or task level) or in the data automatically as a basis for scheduling and program execution. More recent developments in microelectronics and in program languages give rise to a re-thinking of parallel processing: Through microelectronics powerful multi-core processors with 16 or 32 cores are integrated on chip-level; multi-processor computer racks provide thousands of processors within a cloud data center. Developments in high-level programming languages allow for parallel program constructs which can be explicitly expressed by the program developer and which support compilation and scheduling by the operating system, in computing and communication.

In this paper, a novel and practical approach to the evaluation of parallel processing will be presented which is based on processing jobs described by reducable task graphs. A task graph models all possible execution paths of a program (computation job) and can be described by a directed acyclic graph (DAG) with generally-distributed task execution times, precedence conditions and synchronization constructs for parallel executable tasks.

From the viewpoint of analysis it is important to derive task graphs automatically, to generate task graphs synthetically and to reduce the complexity by graph reduction methods. For the analytic performance analysis of this paper it is important that the task graph can be reduced stepwise by elementary aggregations of two tasks at each step. By this approach, it is possible to reduce the whole task graph for a given number n of processing elements to one “virtual” task with a corresponding generally-distributed virtual processing time. Thus, the execution of a specified job stream on a multi-core or multi-processor system can be modeled by a virtual queuing system of type GI/G/1 where GI represents the job arrival stream with arrival rate k, G represents the virtual task execution time on the multi-processor system, and where n ¼ 1 server represents a “virtual processor”. Jobs are served by the virtual processor in a batch processing mode, i.e., one at a time only, to avoid context switching overhead and cache splitting in case of simultaneous processing of multiple jobs in a time-sharing mode.

Temporally idle processors are turned in a low-power sleeping mode to save energy consumption during enforced “slack times” for concurrent processes or idle periods which can be accomplished by dynamic voltage and frequency scaling (DVFS).

## REVIEW OF LITERATURE

A review of the work dona in the field of performance analysis of multiprocessor systems based on queueing models is contained in £ 110J,

Sauer and Chancly (2008) have studied the behaviour of a multiprocessor system under various scheduling strategies and CPU service distributions. They have also studied the impact of the level of multiprogramming on throughput.

Multitasking (i,e, 5 when a program is decomposed into a partially ordered set of tasks which may be processed in parallel) in a multiprocessor system has been studied by Browne et al. (1997) when two or more CPUs cooperate on a single program. Performance of the multiprocessor system analysed by them employed multitasking in case, there are fewer programs in the system requiring CPU service than the number of CPUs. However, if there are alleast as many programs requiring CPU service as there are CPUs, then each CPU works on an independent program. In their model, each program is broken up into two completely independent tasks when tv/o processors cooperate on a single program, The time required to complete CPU service for that program is 1/K times the time required to complete CPU service on a single CPU whore K (defined as cooperation factor) is between 1 and 2, K was assumed to have the value 2 assuming perfect cooperation between processors. OPU-I/O, I/O - I/O overlap models have been formulated by I'owsley et al., Their paper presents models for multiprogrammed systems iu which programs may either partially or completely overlap CPU and I/O processing and where two I/O activities may partially or completely overlap among themselves and CPU processing. Their models require tight synchronisation i.e, both the CPU and I/O (or the tv/o I/O) tasks must complete execution before further processing can continue„ Recently Heidelberger and Trivedi (1999)have attempted to model parallel processing system in which a job subdividee into two or more tasks at some point during its execution and do not require synchronisation. In they have discussed a model for parallel processing in which a job consists of a primary task and two or more secondary tasks which execute concurrently. Al^ the secondary tasks must complete execution before the primary task becomes active again. In , their primary interest has been in developing approximate analytical solution methods for the performance prediction of such system .

## METHODS AND METHODOLOGY

In tills chapter, we formulate a performance model of parallel processing on a multiple CPU system when each program is partitioned into a number of independent tasks capable of simultaneous execution but require tight synchronization i.e., all the independent tasks must complete execution before further processing of the program can continue. The model suggested

here for performance analysis is different from those that have been proposed by others and assumes that for each program the number of CPU tasks capable of simultaneous execution under SIM or TOG type ordering [ a§] is a random variable. The multiple server queuing model proposed here has been used for studying the throughput of a multiprocessor system with variation in program characteristics, degree of multiprogramming etc, as discussed in the paper Before we formulate a model for performance analysis of a multiprocessor system exploiting implicit parallelism in programs, we discuss, a method for scheduling the independent tasks on different processors. A scheme for implementing this scheduling method by table interpretation is also suggested which is different from the one proposed by Gonzalez and Ramamurthy.

## HARDWARE ORGANISATION

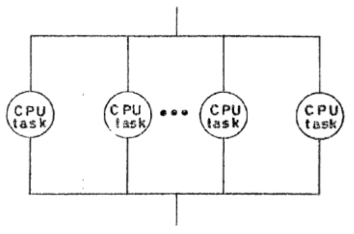
Here we will be concerned with the performance analysis of a multiprocessor system consisting of a number of homogeneous processors sharing access to a global memory. The performance of such 'closely coupled* multiprocessors suffer because of contention for the shared memory when the CPUs attempt to access the main memory.

As discussed in Chapter 5» one way to avoid the frequent access to main memory is by providing each of the processors with a private cache so that instructions and data may be accessed from cache and reference to main memory has to be done only in case the required item is not present in cache. Here we analyse the performance of multiprocessors with a two level memory hierarchy with each CPU having a private cache attached to it. We have assumed that the oache size and the block size have been chosen to yield a high cache hit ratio. We have dismissed main memory interference when writing and reading from cache. Her main memory is assumed to be interleaved. The choice of the processor memory interconnection network to reduce the communication time has been an important issue in designing of multiprocessor systems.

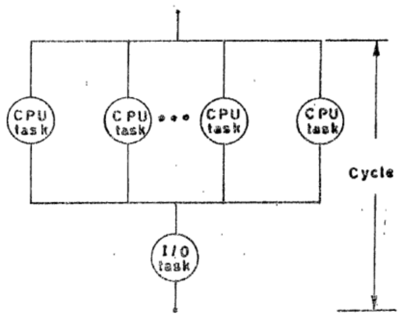
Here we will restrict our analysis to multiprocessor systems provided with processor memory interconnection network having a high bandwidth such thatj there is no serious delay due to contention for processor memory interconnection. Thus the hardware organisation of the multiprocessor system chosen to be analysed is similar to Gmmp type system. The analysis is not applicable for distributed processing systems or Cm ^ type multiprocessors.

**Program Model**

The conceptual flow of execution in TOG or SIM type of program block is shown in figure 7*1 * Cnee this type of block is entered all individual tasks within this block can proceed simultaneously, dependent only upon the availability of resources. Control is not allowed to pass beyond the block until all individual tasks in this block have been completed. A program alternates between CPU and I/O activities. A program may be thought of as repeating cycles where each cycle consists of an arbitrary number of tasks requiring CPU service simultaneously under SIM or TOG type execution ordering followed by one requiring I/O service. The effect of overlapping of CPU and I/O activities is not considered in this model.



TOG (do together) or SIM (simultaneous) execution ordering of CPU tasks



## PREDECESSOR TABLE

The information contained in the Predecessor Table is needed by the operating system for correctly initiating tasks after a Branch task is encountered. A Branch task can have more than one possible successor, which may be contained in the same row of the Parallel Task Table indicating that they may be executed simultaneously. But as successors of the same Branch task, control may flow to only one of them depending on the outcome of the testing' of the branch condition. Thus the Operating System will not be able to get the entire information about task initiation merely from the Parallel Task Table but will also need information about the tasks which should not be initiated depending on the branch direction. It should be noted that edges drawn in the Bata Flow Graph (as discussed in the previous chapter) prevent any task within an IF block or successors of GO TO statements to be initiated in an incorrect sequence. The information of the Predecessor Table is kept in two columns. The first column contains a task number and the next column named as Task Mot Reachable column (TNH) contains the numbers of the tasks to which control cannot flow once the task whose number appears in the first column is initiated.

The Information kept In the Predecessor Table is obtained as follows:

(1) Construct a Control Flow Graph depleting the sequence of execution of tasks which would be followed if the program is run in a uniprocessor environment, The Control Flow Graph is represented by an Adjacency Matrix An entry in this matrix is a 1 if and only if control flows from task i to task j, A node in the graph from which more than one directed arte emanates represent a Branch task which has more than one potential successor

(2) Construct the Reachability Matrix from the Adjacency Matrix and transpose the matrix obtained.

## PERFORMANCE EVALUATION AND ENERGY EFFICIENCY

Task graph processing on a multi-processor system will be considered under two different aspects, performance and energy efficiency. For comparison, we will distinguish between two modes in each case:

Mode PP: Parallel processing of each job on the n-processor system.

Mode SP: Serial processing of each job on one processor of the n-processor system.

### Performance Evaluation

The classical performance criterion for parallel computing was formulated by (13a) (Amdahl's aw): If a is the fraction of non-parallelizable parts of a program, the ideal speed-up factor is

$$S(n) = 1/[(1-\alpha)/n + \alpha]. \qquad (13a)$$

As a consequence of the introduced job description by a DAG, the speed-up factor has to be re-defined as the fraction of job processing times for n ¼ 1 (single processor) and n, i.e.,

$$S(n) = \frac{E[T|n=1]}{E[T|n]} \qquad (13b)$$

Note, that this approach is more general as individual task execution time variations and limitations in parallelization are taken into consideration, where (13a) holds for an idealized case of constant parallelization degree of n only.

## RESULTS & DISCUSSION

The results underline the following general properties

- The maximum capacities for PP are lower than for SP.
- The maximum capacity for PP reduces with increasing slack times caused by task execution time variations.
- Trade-off of the performance results between PP and SP with smaller response times for PP in the low-load region and for SP in the high-load region.

The main contribution of this paper is a novel method by which parallel and serial processing of jobs on a multi ore/multi-processor system can be analyzed for generally-distributed task execution times by stepwise reduction of directed acyclic task graphs. The reductions are performed by task aggregations for four principal structure elements of computation programs: concatenation, alternative splitting, iterative repetitions, and concurrency of tasks. The mathematical operations are based on generally distributed random task execution times. The principal four structure elements are used for the exact aggregation based on the theory of functions of random variables. For an efficient computational implementation, the generally-distributed task execution times are represented by a mixed phase-type model for the first and second order moments. The method allows to represent the multi-core/multi-processor system to standard queuing models of the types GI/G/1 and GI/G/n, where the service times are represented by their averages and coefficients of variation. From the application's point of view, the new method allows the extension of Amdahl's Law to more realistic conditions of random task execution times and arbitrary degrees of parallelization as well as real-time performance metrics as the average job response times. The trade-off between the two major schedules for parallel and serial processing of jobs on an n-server system leads to the most important conclusion, that parallel processing is only superior for low- and medium-load ranges, while serial processing outperforms parallel processing for high loads with respect to the maximum capacity and response times. Both job execution modes are neutral with respect to energy efficiency; the only way to increase energy efficiency is by low-power operation of idle processors through Dynamic Voltage and Frequency Scaling (DVFS). The current paper reflects the status of "work in progress"; ongoing work addresses the development of general analysis tools for the analytical solution as well as for simulations.

## REFERENCES:-

Adve, V. S., Vernon, M. K. (2004). Parallel program performance prediction using deterministic task graph analysis. ACM Trans. Computer System (TOCS) 22(1), pp. 94–136.

Adve, V., Sakellariou, R. (2000). Compiler synthesis of task graphs for parallel program performance prediction. In: Proceedings of 13th International Workshop on Languages and Compilers for High-Performance Computing (LCPC 2000), Yorktown Heights, N. J. (2000).

Ajmone Marsan, M. (1989). Stochastic Petri nets: An elementary introduction. In: Rozenberg, Grzegorz (ed.) APN 1989. LNCS, vol. 424, pp. 1–29. Springer, Heidelberg (1990).

Kobayashi, H., Mark, B. L. (2009). System Modeling and Analysis: Foundations of System Performance Evaluation. Pearson/Prentice-Hall Inc.

Kuehn, P. J. (1979). Approximate analysis of general queuing networks by decomposition. IEEE Trans. Commun. 27(1), pp. 113–126.

Reiser, M., Lavenberg, S. S. (1980). Mean-value analysis of closed multichain queuing networks. J. ACM 27(2), pp. 313–322.

Whitt, W. (1983). The queuing network analyzer. Bell Syst. Techn. J. 62(9), pp. 2779–2815.

**Corresponding Author**

**Mahak Dhanda***

M. Tech (Computer Science) Net Qualified

**E-Mail –** mahak0570@gmail.com