Performance and Energy Efficiency of Parallel Processing in Data Center Environments

Paul J. Kuehn(✉)

Institute of Communication Networks and Computer Engineering,
University of Stuttgart, Stuttgart, Germany
paul.j.kuehn@ikr.uni-stuttgart.de

Abstract. A novel approach is presented for the analysis of parallel processing of stochastic workload by multi-processor/multi-core processing resources in data center environments. The method is based on job workload descriptions by task graphs with generally-distributed task execution times and task scheduling under consideration of prescribed precedence and synchronization constraints. For the analytic performance evaluation, task graphs are restricted to the analysis of directed acyclic graphs which are reduced by stepwise aggregations of tasks. The reduction allows to aggregate the whole task graph under a given number n of processing elements to a single virtual job processing time with average value h_v and coefficient of variation c_v. By this way, the whole multi-processor system can be modeled by a queuing system of type GI/G/1 from which the response time T_R and the speedup factor S(n) is derived. Finally, the influence of the stochastic properties of the workload on the performance and on energy efficiency of parallel computing will be studied and compared with serial computing on a multi-processor system modeled by a queuing system of the type M/G/n.

Keywords: Parallel processing · Task graph · Graph reduction · Queuing system · Performance evaluation · Energy efficiency

1 Introduction

Parallel processing has received enormous attention in the last 50 years, c.f. fundamental presentations in the books as [1–3]. 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) [4] 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

S. Klingert et al. (Eds.): E^{2}DC 2014, LNCS 8945, pp. 17–33, 2015.
DOI: 10.1007/978-3-319-15786-3_2

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 [5]. 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 [6]. These deficiencies have partly been overcome by approximate evaluation methods and powerful computer tools for queuing network analysis and simulations, see, e.g., [7–9]. 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 [9, 10]. 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 [11–14]. 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 λ, G represents the virtual task execution time on the multi-processor system, and where $n = 1$ 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).

The remaining part of this paper is structured as follows: In Sect. 2, the problem of parallel execution of a job is defined by a task graph which is composed by four generic modeling constructs for serial processing, parallel processing, alternative task and repeated task executions. The description and reduction of the task graph follows in principle the modeling approach reported in [12, 14] and is illustrated by an example graph. Task execution times are described by generally distributed random variables and their probability density functions, respectively. For numerical analyses the generally distributed task execution times are represented by a phase-type model with 3 parameters only which allows the adaption to arbitrary mean values and coefficients of variation. The four generic modeling constructs are described generally through mathematical operations on random variables by distributions as well as by a two-moment characterizations. In Sect. 3, the reduction of the whole task graph by stepwise aggregation of tasks according to the 4 principal modeling constructs is discussed generally and for the example graph of Sect. 2. Section 4 addresses the performance of parallel processing in terms of the speed-up factor achieved by parallel processing and by the job response time by queuing analysis as well as an analysis of the energy consumption. Both performance and energy consumption are compared for two fundamentally different operation modes for multi-processor systems, serial processing of jobs on one processing element each and parallel processing of jobs on all available processing elements. The paper concludes by summarizing the current state of the project and gives an outlook on ongoing further work based on the presented methods.

2 Multi-processor Job Execution

2.1 Multi-processor Queuing Model

In Fig. 1, the considered queuing model is shown consisting of n processing elements representing processors of a multi-core or a multi-processor system. Jobs arrive

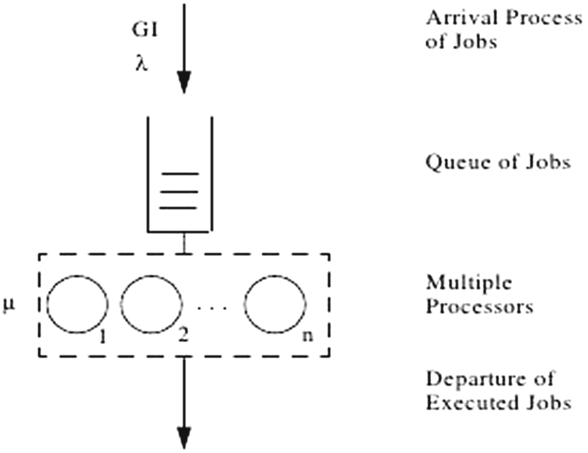


Fig. 1. Principal model of a multi-core/multi-processor processing system for parellel processing

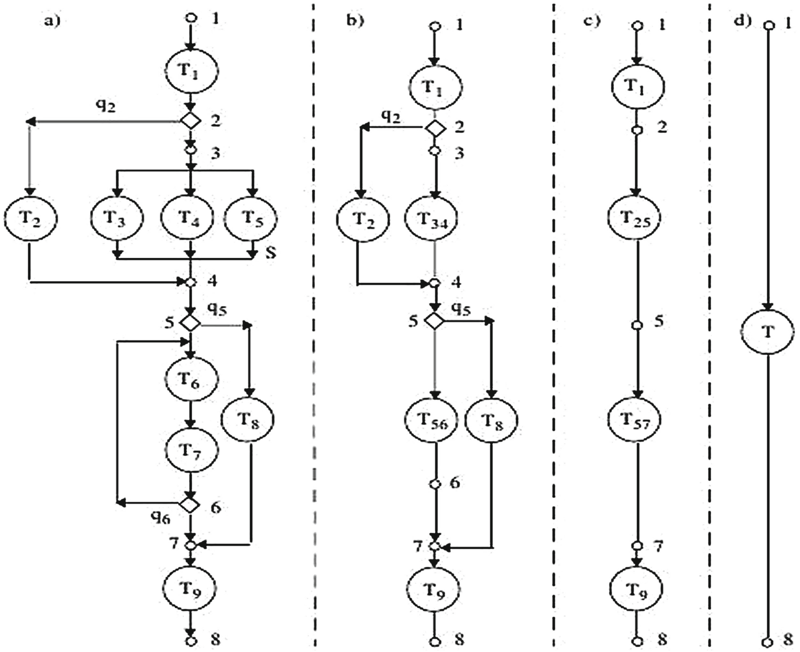


Fig. 2. Example of a task graph and its stepwise redution (a) Original task graph (b, c) Intermediate reduction steps (d) Results of aggregation

according to a general stochastic arrival process of type GI (generally- and independently-distributed arrivals) at an arrival rate of λ jobs/time unit. The model represents, e.g., the physical resources provided by a data center to a tenant (company) or to a group of users for a particular service. The model can be considered as a simplified model for an "Infrastructure as a Service" (IaaS) providing physical resources upon which a workload is processed defined by a Virtual Machine. The workload accompanied with each arrival is defined by a stochastic task graph, see e.g., Fig. 2. Jobs are processed on the multi-processor model according to the batch mode, e.g., by FIFO (First-In, First-Out) scheduling sequence and each job uses the n processors exclusively to avoid program context switching during a job being in execution.

2.2 Task Graph Job Model

Each job is represented by a task graph with stochastic task processing times. A simple model of a task graph is shown in Fig. 2(a) consisting of altogether 9 different tasks 1, 2, …, 9 and 4 generic task constructs. Terminals 1 and 8 represent the initial and final points of a job execution. The task graph belongs to the class of Directed Acyclic Graphs (DAG). Figures 2(b, c, d) represent reduced task graphs of the DAG of Fig. 2(a) and will be discussed later on in Sect. 3. Any execution path between Terminal 1 and Terminal 8 is a feasible production for a job execution.

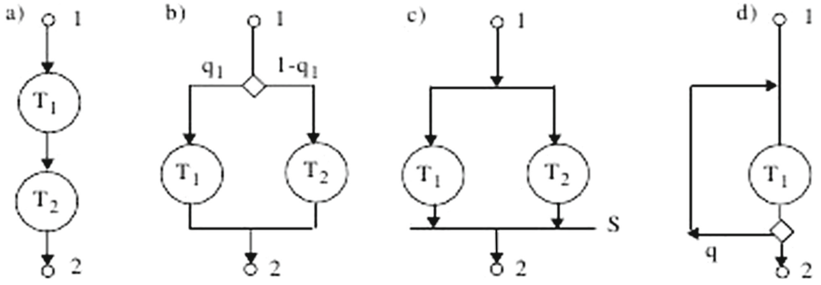


Fig. 3. Principal task graph elements (a) Sequential processing of two tasks 1 and 2 (b) Alternative split (Or-Split) of two tasks 1 and 2 (c) Parallel processing (Concurrency, And-Split) of two tasks 1 and 2 with synchronization S (And-Join) (d) Iteration loop for task 1 with parameter q

Figure 3 represents the four generic or principal constructs (task graph elements). The execution time T_i of a task i will be represented by its cumulative probability distribution function (DF) $F_i(t) = P\{T_i < t\}$ and its probability density function $f_i(t) = dF_i(t)/dt$, respectively. Individual task execution times T_i of a job are considered as being statistically independent of each other. The aggregated execution times T of the four principal task graph elements of Fig. 3, measured between the Terminals 1 and 2, can be mathematically expressed by their PDF f(t) as follows:

(a) **Sequential processing** of two tasks (concatenation)

$$f(t) = f_1(t) \otimes f_2(t) \tag{1}$$

where the symbol ⊗ indicates the mathematical convolution operator

(b) **Alternative Split** of two tasks (Or-Split)

$$f(t) = q_1 f_1(t) + (1 - q_1) f_2(t) \quad \text{(Or-Split followed by Or-Join, Choice)} \tag{2}$$

Remark: The OR-Split is the basic function for tree-structured execution paths.

(c) **Parallel processing** of two tasks with synchronization (Concurrency, And-Split followed y And-Join)

$$T = \max(T_1, T_2){:}\ f(t) = f_1(t)F_2(t) + f_2(t)F_1(t) \tag{3a}$$

$$T = \min(T_1, T_2){:}\ f(t) = f_1(t)[1 - F_2(t)] + f_2(t)[1 - F_1(t)] \tag{3b}$$

Remark: The maximum operator is applied if both tasks 1 and 2 have to be completed before continuation. The minimum operator applies, e.g., for a parallel search.

(d) **Iteration loop** for Task 1 (Repetition)

$$f(t) = \sum_{i=0}^{\infty} q^i (1-q) \cdot f_1(t) \otimes \underbrace{[f(t) \otimes \ldots \otimes f_1(t)]}_{ifactors} \tag{4a}$$

for a probabilistic iteration with probability q and

$$f(t) = f_1(t) \otimes \underbrace{[f_1(t) \otimes \ldots \otimes f_1(t)]}_{qfactors} \tag{4b}$$

for q deterministic iterations

2.3 Generic Task Execution Model

The operations are generally too complex to be programmed for arbitrary PDFs. Therefore, a generic task execution model will be used which is defined by 3 parameters only, see Fig. 4.

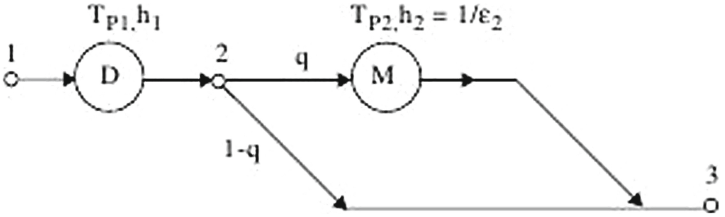


Fig. 4. Mixed phase-type model for task excution times

The substitute model of Fig. 4 consists of a series of a deterministic phase T_{P1} (D) with duration h_1 between Terminals 1 and 2 followed by a degenerated hyperexponential phase between Terminals 2 and 3 realized by a probabilistic alternative between a negative-exponentially distributed phase T_{P2} (M) with mean $h_2 = 1/\varepsilon_2$ chosen with probability q and a zero-phase chosen with probability (1−q). This model will be defined by two parameters for each task execution time T_P, its mean h_P and its coefficient of variation c_P:

$$h_P = E[T_P] \tag{5a}$$

$$c_P^2 = VAR[T_P]/E[T_P]^2. \tag{5b}$$

Parameters h_P and c_P can be arbitrarily prescribed, where $0 \le c_p < \infty$. This model allows to represent any task with arbitrary mean h_P and coefficient of variation c_P by a PDF $f_P(t)$ and a DF $F_P(t)$, respectively:

$$\mathrm{f_P(t)} = (1-\mathrm{q})\delta(\mathrm{t}-\mathrm{h_1}) + \mathrm{q}\varepsilon_2\exp(-\varepsilon_2[\mathrm{t}-\mathrm{h_1}])\mathrm{u}(\mathrm{t}-\mathrm{h_1}) \tag{6a}$$

$$\mathrm{F_P(t)} = (1-\mathrm{q})\mathrm{u}(\mathrm{t}-\mathrm{h_1}) + \mathrm{q}[1-\exp(-\varepsilon_2[\mathrm{t}-\mathrm{h_1}])]\mathrm{u}(\mathrm{t}-\mathrm{h_1}), \tag{6b}$$

where $\delta(t)$ indicates the impulse function (delta function) and u(t) the unit-step function, with

$$\mathrm{h_P} = \mathrm{h_1} + \mathrm{qh_2} \tag{6c}$$

$$\mathrm{c_P}^2 = \mathrm{q}(2-\mathrm{q})\mathrm{h_2}^2/(\mathrm{h_1}+\mathrm{qh_2})^2. \tag{6d}$$

The model has 3 parameters h_1, h_2 and q to be derived from Eq. (6c, d), i.e., there is one degree of freedom. For $h_1 \rightarrow 0$, we find $c_p \rightarrow \infty$ for $q \rightarrow 0$. The degree of freedom can be used by choosing $h_1 > 0$ to avoid trivial task execution times. Fixing h_1, parameters h_2 and q follow formally from Eq. (6c, d):

$$\mathrm{q} = 2\Big/\left[1+\left(\frac{\mathrm{c_P h_P}}{\mathrm{h_P}-\ \mathrm{h_1}}\right)^2\right] \tag{7a}$$

$$\mathrm{h_2} = \frac{1}{2}[\mathrm{h_P}-\ \mathrm{h_1} + c_{\mathrm{P}^2}\mathrm{h}_{\mathrm{P}^2}/(\mathrm{h_P}-\ \mathrm{h_1})] \tag{7b}$$

For a feasible solution, $0 < \mathrm{q} < 1$ has to be regarded as compatibility condition. A quite simple solution of the parameter fitting follows from (7a, b) by subdivision of the c_P-range:

(a) $0 \le c_P \le 1$ (**hypoexponential** characteristic)

$$\mathrm{q} = 1,\ \mathrm{h_1} = \ \mathrm{h_P}(1-\mathrm{c_P}),\ \mathrm{h_2} = \ \mathrm{c_P h_P} \tag{8a}$$

(b) $1 \le c_P < \infty$ (**hyperexponential** characteristic)

$$\mathrm{h_1} = 0,\ \mathrm{h_2} = \ \mathrm{h_P}\left(1+\ \mathrm{c_P}^2\right)/2,\ \mathrm{q} \ = 2/\left(1+\ \mathrm{c_P}^2\right) \tag{8b}$$

The solution (8a) represents a series of a deterministic phase and an exponential phase while (8b) represents a degenerated hyperexponential phase.

2.4 Performance of the Principal Task Graph Elements

Applying the mixed phase-type model for all task execution times, represented by the PDF $f_P(t)$ and DF $F_P(t)$ acc. to Eq. (6a, b), the execution times T for the principal task graph elements of Fig. 3 can be expressed explicitly by their PDF f(t) from Eqs. (1–4). These results are too voluminous and will be reported in detail in a forthcoming companion paper. The results for a two-moment representation are much easier to be derived by elementary moment operators on independent random variables (cases a, b and d) or on PDFs (case c). Tasks 1 and 2 are represented by the phase-type model parameters h_{i1}, h_{i2} and q_i, i = 1, 2. For the 4 principal task graph elements we find:

(a) **Sequential Processing** of two tasks 1 and 2 (Concatenation)

$$E[T] = E[T_1] + E[T_2] = h_{11} + q_1 h_{12} + h_{21} + q_2 h_{22} \tag{9a}$$

$$\begin{aligned} VAR[T] &= VAR[T1] + VAR[T_2] = q_1(2-q_1)h_{12}{}^2 + q_2(2-q_2)h_{22}{}^2 \\ c^2 &= VAR[T]/E[T]^2. \end{aligned} \tag{9b}$$

(b) **Alternative Split** of two tasks 1 and 2 followed by Or-Join (Choice)

$$\begin{aligned} E[T_i] &= q\, E[T_{1i}] + (1-q)\, E[T_{2i}],\ i = 1,2 \\ E[T] &= q(h_{11} + q_1 h_{12}) + (1-q)(h_{21} + q_2 h_{22}) \end{aligned} \tag{10a}$$

$$E[T2] = q\left(h_{11}{}^2 + 2q_1 h_{12}{}^2 + 2q_1 h_{11} h_{21}\right) + (1-q)\left(h_{21}{}^2 + 2q_2 h_{22}{}^2 + 2q_2 h_{21} h_{22}\right) \tag{10b}$$

$$VAR[T] = E\left[T^2\right] - E[T]^2 \tag{10c}$$

$$c^2 = VAR[T]/E[T]^2. \tag{10d}$$

(c) **Parallel Processing** of two tasks 1 and 2 with synchronization (Concurrency, And-Split)

In this case, the PDF f(t) of the aggregated random variable $T = \max(T_1, T_2)$ or $T = \min(T_1, T_2)$ acc. to Eqs. (3a, b) has to be derived first from which the moments

$$E[T^i] = \int_{t=0}^{\infty} t^i f(t)dt_{,\, i=1,2} \tag{11}$$

follow by integration. The variance VAR[T] and the coefficient of variation *c* follow acc. to Eqs. (10c, d). The explicit results are too voluminous and will be reported in a forthcoming paper.

(d) **Iteration Loop** for task 1 (Repetition)

Be J the RV of the number of executions of task 1 with average E[J]. Then we get for the aggregated RV T in general

$$\mathrm{E[T]} = \mathrm{E[J]} \cdot \mathrm{E[T_1]}\,. \tag{12a}$$

In case of a **geometrically-distributed** number of iterations with feedback probability q we get

$$E[J] = \sum_{j=0}^{\infty} (j+1)q^j(1-q) = 1/(1-q) \tag{12b}$$

$$\mathrm{E}\left[\mathrm{J}^2\right] = \sum_{\mathrm{j}=0}^{\infty} (\mathrm{j}+1)^2\mathrm{q}^\mathrm{j}(1-\mathrm{q}) = \frac{1+\mathrm{q}}{(1-\mathrm{q})^2} \tag{12c}$$

$$\mathrm{VAR[J]} = \mathrm{E}\left[\mathrm{J}^2\right] - \mathrm{E[J]}^2 \tag{12d}$$

$$\mathrm{VAR[T]} = \mathrm{E[J]} \cdot \mathrm{VAR[T_1]} + \mathrm{VAR[J]} \cdot \mathrm{E[T_1]}^2 \tag{12e}$$

$$\mathrm{c}^2 = \mathrm{VAR[T]}/\mathrm{E[T]}^2. \tag{12f}$$

If J is a **constant** E[J] = J and VAR[J] = 0, then

$$\mathrm{VAR[T]} = \mathrm{J} \cdot \mathrm{VAR[T_1]}\,. \tag{12g}$$

E[T] and c follow from Eqs. (12a) and (f).

Remark: If J is an RV, the statistics of T follow from the compound distribution of T_1 and J.

3 Task Graph Reductions

3.1 General Aspects and Application Cases

As outlined above, the workload by a specific job will be described by a directed acyclic graph (DAG) consisting of elementary structural elements expressing arbitrary workflows. Any DAG can be processed on a **single-processor system** by following any possible execution path through the DAG from the initial terminal to the final terminal. Any feasible execution path through the DAG can be considered as a thread which is scheduled by the operating system and processed on the single-processor system without any stop (except for memory I/0 which is not considered here). Processing of parallel executable instruction paths have to be executed serially in arbitrary sequence but continuation after the parallel paths depends on the synchronization condition. Parallel processing executed on a single processor does not cause any “slack times”.

In a **multi-processor environment**, parallel executable paths can be scheduled such that the thread is split into multiple parallel threads which can be scheduled by the operating system and processed simultaneously as long as the synchronization point is not reached. The degree of processing simultaneity depends on the individual execution path durations: The best case is if all parallel threads are of identical durations; however, with increasing variability, the degree of processing simultaneity decreases and some processing elements will become idle for a "slack time" until the next synchronization point. This reduces the performance by increasing the total job execution time. The idea of this paper is to quantify the efficiency of parallel processing by modeling task execution times by stochastic processes in order to express the influence on the performance as well as on the energy efficiency.

This can be achieved best if the DAG could be reduced stepwise by aggregating parallel or serially executable tasks to a single virtual task where the corresponding execution times of aggregated paths are obtained by application of the basic aggregation operations introduced in Sect. 2. Reducibility of graphs is a fundamental problem of graph theory. Graphs resulting out of the use of "go to" statements, e.g., jumps out of a loop or into another program branch, cause quite complex graph structures which cannot be reduced stepwise. Modern programming languages and programming styles result into well-structured programs which support graph reducibility.

From the application point of view, many problems are adequate for parallel execution and reducibility. Examples are:

(1) Parallel execution of a program for multiple input parameter sets. Each parameter set can be executed in parallel by a multi-processor system providing results of a whole parameter range instantaneously.

(2) Batch simulation methods where a simulation is subdivided in (typically) 10 "batches", each for a certain number of "events", e.g., 100.000 events. For such programs, we find a simple program structure of one task at the beginning to configure the "batch" programs and initializing counters for statistic data, then executing all "batch" simulations in parallel, and one task after the execution of all "batches" for processing of the final results out of each "batch" execution. Such a program consists of a simple DAG-structure of the form fork and join and allows a speed-up factor close to the number of "batches".

(3) Searching within large unstructured data sets, a typical problem of "Big Data". In such cases, the data sets can be partitioned and each partition can be executed in parallel. This approach can be repeatedly applied on reduced data sets, etc.

In most of such applications, execution times of parallel threads may depend on the properties of data or may affect the number of iterations for a certain precision; these execution time variations are modeled best by random variables.

3.2 Example of a Task Graph Reduction

A simple example of a model task graph as shown in Fig. 2(a) will be considered. In a first step, m serially or parallel executable tasks are combined successively by aggregation of two tasks in each step either in a linear sequence by (m−1) iterations or in a binary-tree fashion by aggregating each time 2 tasks resulting in log m iterations; in the latter case, the aggregations themselves can be executed in parallel, too. By these steps, the task graph Fig. 2(a) results in a reduced task graph shown in Fig. 2(b). In a second step, all loops are aggregated and finally replaced by one task resulting in the further reduced task graph shown in Fig. 2(c). In the final step, only a series of sequential tasks has to be aggregated in a single resulting “virtual task” for the whole job, see Fig. 2(d).

In a tree-structured task graph, the above outlined reduction strategy is applied at first on all branches of the tree or of subtrees, followed by combining the tasks representing the whole branch at the root point of subtrees, repeatedly in bottom-up direction, starting at the leaf-level.

Remark 1: All described steps are performed on two parameters of each task (the mean value and the coefficient of variation) as outlined in Sect. 2 of this paper. For this, we need only to program the basic operations which are implemented in a procedure and are applied repeatedly.

Remark 2: As outlined above, the virtual tasks include automatically the parallel execution on multiple processors. If the degree of task parallelism exceeds the number of available processors, the task graph has to be restructured first by combining maximally possible parallel aggregations and repeat these results sequentially for the remaining parallel tasks which (of course) adds to an increased task graph execution time.

4 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.

4.1 Performance Evaluation

The classical performance criterion for parallel computing was formulated by (13a) (Amdahl’s Law [15]): If α is the fraction of non-parallelizable parts of a program, the ideal speed-up factor is

$$S(n) = 1/[(1-\alpha)/n + \alpha]\,. \tag{13a}$$

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 $n = 1$ (single processor) and n, i.e.,

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

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. Under the special case of a constant parallelization degree n we get for α the result $\alpha = (\mathrm{nE[T|n]} - \mathrm{E[T|1]})/(\mathrm{n}-1)$; inserting this in Eq. (13b), the result coincides with Amdahl's Law Eq. (13a).

As a second performance metric, we will consider the response time $\mathrm{T_R}$ measured between the arrival instant of a job and the instant when the job is executed, i.e.,

$$\mathrm{T_R} = \mathrm{T_W} + \mathrm{T_v} \tag{14a}$$

where $\mathrm{T_W}$ is the waiting time and $\mathrm{T_v}$ the virtual processing time of the job.

Mode PP: Parallel Processing

The parallel processing system is represented by a virtual single-server system upon completion of the graph reduction method outlined in Sect. 3.2. $\mathrm{T_W}$ follows from a GI/G/1 queuing model, e.g., for Poisson job arrivals according to the Pollaczek-Khintchine formula for the M/G/1 delay system [6]:

$$\mathrm{E[T_W]} = \rho_\mathrm{V} \cdot \frac{\left(1 + c_\mathrm{V}^2\right)}{2(1-\rho_\mathrm{V})} \mathrm{E[T|n]}, \text{with load factor } \rho_\mathrm{V} = \lambda \cdot \mathrm{E[T|n]}. \tag{14b}$$

Note, that the maximum capacity of this system is reached for

$$\lambda_{\mathrm{max,PP}} = \frac{1}{E[T|n]} \tag{15a}$$

The average processing time of a job under Mode PP is E[T|n]. The average of the slack time $\mathrm{T_S}$ follows from the balance equation $\mathrm{E[T_S]} = \mathrm{n} \cdot \mathrm{E[T|n]} - \mathrm{E[T|1]}$ and reduces the capacity for high loads to $\lambda_{\mathrm{max,PP}}$ acc. to (15a).

Mode SP: Serial Processing

If each job is assigned to be processed by one processor, the n-processor system is modeled by a GI/G/n system, where G describes the job processing time T on a single processor which follows from the original task graph by adding all processing phases resulting in a mean processing time E[T|1]. The response time $\mathrm{T_R}$ follows from (14a), where $\mathrm{T_V}$ is represented by two moments h = E[T|1], c from a task graph reduction for 1 processor, and $\mathrm{T_W}$ from queuing system GI/G/n; for Poisson arrivals from the delay system M/G/n (exact closed-form solutions and tabled results are known for M/M/n and M/D/n only).

The maximum capacity of this n-server system is reached for

$$\lambda_{\mathrm{max,SP}} = \frac{\mathrm{n}}{\mathrm{E[T|1]}} = \frac{\mathrm{n}}{\mathrm{nE[T|n]} - \mathrm{E[T_S]}} > \lambda_{\mathrm{max,PP}} \tag{15b}$$

Note, that in Mode SP the full capacity of the n servers is available, where PP suffers from enforced idle times (slack times).

4.2 Energy Efficiency

As outlined in Sect. 2, we will assume for both Modes PP and SP that all jobs are processed on an n-processor machine in batch processing mode. Under low load, parallel processing results in shorter job execution times and is superior to job processing serially on a single processor. Idle phases of a processing element due to a limited parallelization degree $(1 - \alpha)$ will be considered as sleep phases ("slack times") with reduced power P_0; $P_1 > P_0$ denotes the power consumption of a running processor executing a task. Parallel processing of tasks and serial processing of tasks are neutral with respect to energy consumption as both modes require the same amount of energy in total, whereas parallel processing and serial processing differ with respect to the maximum job rate for saturation as well as with respect to the response time, i.e., there is a trade-off between parallel and serial processing. This effect has been observed in other energy-efficiency studies as well, where energy efficiency and performance reduction behave reciprocally [16, 17]. The energy consumptioncan, however, can be reduced by low-power operation of idle resources, e.g., by Dynamic Voltage and Frequency Scaling (DVFS) [18].

The energy consumption in an arbitrary large interval t_0 of time amounts for Modes PP and SP as follows:

$$E_{PP} = t_0 \cdot \rho_V \left[n \frac{E[T|1]}{nE[T|n]} \cdot P_1 + n \frac{E[T_S]}{nE[T|n]} \cdot P_0 \right] + t_0(1 - \rho_V) \cdot nP_0 \tag{16a}$$

$$E_{SP} = t_0[AP_1 + (n - A)P_0], \tag{16b}$$

where $\rho_V = \lambda \cdot E[T|n]$ denotes the utilization factor of the virtual GI/G/1 delay system and $A = E[X] = \lambda \cdot E[T|1]$ the offered (and carried) traffic value of the GI/G/n delay system, respectively. With these relationships both expressions for E_{PP} and E_{SP} of Eq. (16a, b) are identical $E_{SP} = E_{PP} = E$.

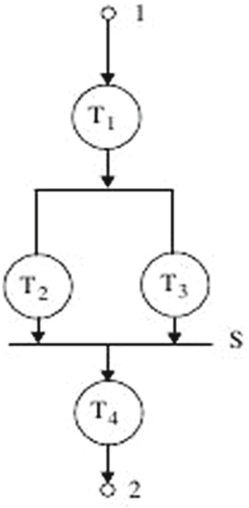
The energy efficiency η will be defined as the fraction of energy saved by DVFS relative to the energy consumption without DVFS:

$$\eta = 1 - \frac{E}{E_0} = 1 - \frac{A}{n} - \frac{n - A}{n} \cdot \frac{P_0}{P_1},$$

where $E_0 = nP_1t_0$ is the energy consumption without DVFS.

4.3 Trade-off Between the Operation Modes PP and SP

The observation that both operation modes differ in their maximum capacities gives rise for a trade-off discussion between them. This will be exemplified in the following by a numerical example for the generic task graph example for concurrency between two tasks 1 and 2 acc. to the model of Fig. 3c extended by constant common tasks at the beginning and end of the task graph with total length h_0. Two special cases of concurrent task execution times will be considered with identical average task execution times D (deterministic) and negative-exponentially distributed times (M) with parameters



Parameters

T_1, T_4	Type D, $E[T_1] = E[T_4] = h/2$
T_2, T_3	Type D, (Case 1), Type M (Case 2) $E[T_2] = E[T_3] = h = 1/\varepsilon$
h	Unit of Task Execution Times
PP	Parallel Processing of Job Tasks on both Processors (Mode 1)
SP	Serial Processing of Job Tasks on one Processor (Mode 2)
n	Number of Processors $n = 2$ (P_1, P_2)

Fig. 5a. Generic taskgraph for numeric example

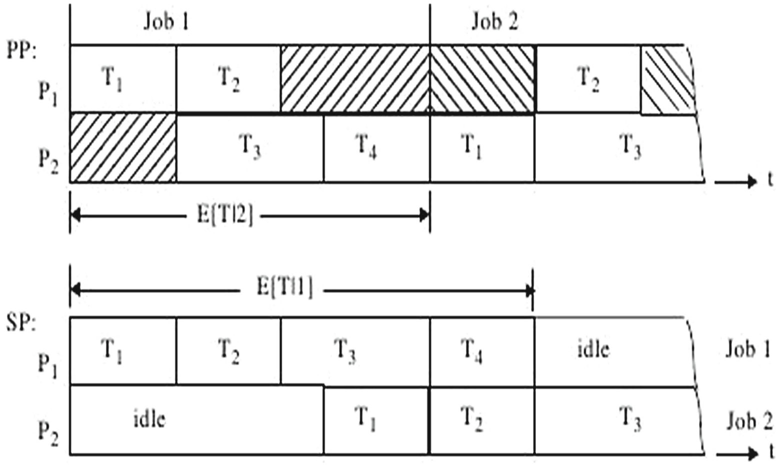


Fig. 5b. Grant-charts for job PP and SP models handed fields are slack times in PP

$h_1 = h_2 = h = 1/\varepsilon$ and $c = 0$ (deterministic) and $c = 1$ (Markovian), respectively. Jobs arrive in both cases according to a Poisson process with arrival rate λ. Figures 5a and 5b shows the example task graph and two Gantt-Charts for PP and SP schedules.

The performance analysis of Mode 1 (PP) follows the procedure of stepwise task graph reduction resulting in a virtual queuing system of the type GI/G/1. The analysis of Mode 2 (SP) follows from a standard queuing system of the type GI/G/n.

Table 1. Queuing system parameters

Mode	Case	Av. Serv. Time	SCOV	Utilization	λ_{max}	Qu.System
1 (PP)	1	$h_v = 2$ h	$c_v^2 = 0$	$\rho_v = 2\lambda h$	1/2 h	M/D/1
	2	$h_v = 2.5$ h	$c_v^2 = 0.2$	$\rho_v = 2.5\lambda h$	1/2.5 h	M/G/1
2 (SP)	1	E[T|1] = 3 h	$c^2 = 0$	$\rho = 3\lambda h$	1/1.5 h	M/D/2
	2	E[T|1] = 3 h	$c^2 = 2/9$	$\rho = 3\lambda h$	1/1.5 h	M/G/2

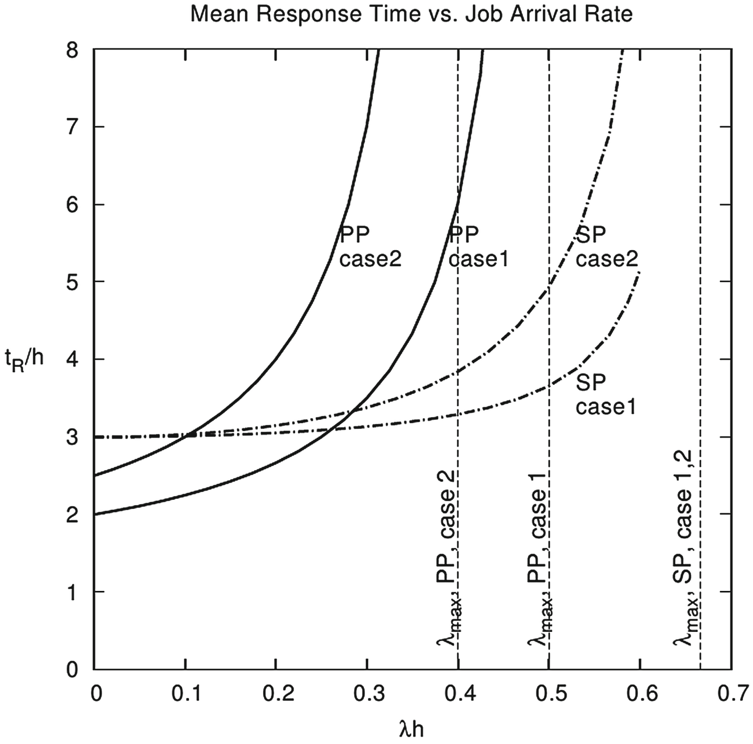


Fig. 6. Mean Response Time vs. Job Arrival Rate

The parameters of the corresponding queuing models are summarized in Table 1. Data center job arrivals are assumed to follow a Poisson distribution (Type M).

The results for the normalized average response times t_R/h are shown for both scheduling Modes 1 (PP) and 2 (SP) for the two parameter cases of constant task execution times (Case 1) and negative-exponentially distributed task execution times (Case 2) of Tasks 2 and 3, respectively (Fig. 6).

Note first the different maximum load levels for PP and SP and the different maximum load levels for PP for constant and for exponentially distributed virtual task times acc. to Case 1 and Case 2 which define the load limits of stationary system operation where the response times increase to infinity asymptotically.

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.

5 Conclusions

The main contribution of this paper is a novel method by which parallel and serial processing of jobs on a multi-core/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

1. Conway, R.W., Maxwell, W.L., Miller, L.W.: Theory of Scheduling. Addison-Wesley Publ. Comp., Reading (1967)
2. Coffman Jr., E.G., Denning, P.J.: Operating Systems Theory. Prentice-Hall Inc., Englewood Cliffs (1973)
3. Shirazi, B.A., Hurson, A.R., Kavi, K.M. (eds.): Scheduling and Load Balancing in Parallel and Distributed Systems. IEEE Computer Society Press, Los Angeles (1995)

4. Peterson, J.L.: Petri Net Theory and the Modeling of Systems. Prentice-Hall, Englewood Cliffs (1981)
5. Ajmone Marsan, M.: Stochastic Petri nets: An elementary introduction. In: Rozenberg, Grzegorz (ed.) APN 1989. LNCS, vol. 424, pp. 1–29. Springer, Heidelberg (1990)
6. Kobayashi, H., Mark, B.L.: System Modeling and Analysis: Foundations of System Performance Evaluation. Pearson/Prentice-Hall Inc. (2009)
7. Reiser, M., Lavenberg, S.S.: Mean-value analysis of closed multichain queuing networks. J. ACM **27**(2), 313–322 (1980)
8. Kuehn, P.J.: Approximate analysis of general queuing networks by decomposition. IEEE Trans. Commun. **27**(1), 113–126 (1979)
9. Whitt, W.: The queuing network analyzer. Bell Syst. Techn. J. **62**(9), 2779–2815 (1983)
10. Adve, V., Sakellariou, R.: 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)
11. Adve, V.S., Vernon, M.K.: Parallel program performance prediction using deterministic task graph analysis. ACM Trans. Comput. Syst. (TOCS) **22**(1), 94–136 (2004)
12. Ajwani, D., Ali, S., Morrison, J.P.: Application agnostic generation of synthetic task graphs for streaming computing applications. IBM Research Report RC 25181 (D 1107-003), 5 July 2011
13. Sadiq, W., Orlowska, M.E.: Applying graph reduction techniques for identifying structural conflicts in process models. In: Jarke, M., Oberweis, A. (eds.) CAiSE 1999. LNCS, vol. 1626, pp. 195–209. Springer, Heidelberg (1999)
14. Simon, J., Wierum, J.-M.: Accurate performance prediction for massively parallel systems and its applications. In: Fraigniaud, Pierre, Mignotte, A., Robert, Y., Bougé, Luc (eds.) Euro-Par 1996. LNCS, vol. 1124, pp. 675–688. Springer, Heidelberg (1996)
15. Sahner, R.A., Trivedi, K.S.: Performance and reliability analysis using directed acyclic graphs. IEEE Trans. Softw. Eng. **SE-13**(10), 1105–1114 (1987)
16. Sun, X.-H., Chen, Y., Byna, S.: Scalable computing in the multicore Era. In: Proceedings of International Symposium on Parallel Algorithms, Architectures and Programming (PAAP 2008) (2008)
17. Kuehn, P.J., Mashaly, M.: Performance of self-adapting power-saving algorithms for ICT systems. In: Proceedings of the IFIP/IEEE Symposium on Integrated Network and Service Management (IM 2013), Ghent, Belgium, 27–28 May 2013 (IEEE XPlore)
18. Mashaly, M., Kuehn, P.J.: Modeling and analysis of virtualized multi-service cloud data centers with automatic server consolidation and prescribed service level agreements. In: International Conference on Computer Theory and Applications (ICCTA 2013), Alexandria, Egypt, 29–31 October 2013
19. Wang, L., von Laszewski, G., Dayal, J., Wang, F.: Towards energy aware scheduling for precedence constrained parallel tasks in a cluster with DVFS. In: Proceedings of 10th IEEE/ATM International Conference on Cluster, Cloud and Grid Computing (CCGrid 2010), pp. 368–377, 17–20 May 2010