Interval Based Workload Characterization for Distributed Systems*

M. Braun, G. Kotsis

Institut für Angewandte Informatik und Informationssysteme
Universität Wien, Lenaugasse 2/8, A-1080 Vienna, Austria
Email [braun,kotsis]@ani.univie.ac.at

Abstract. In this paper we analyze a graph model representing the coarse grain dependency and communication structure of a distributed application. The model is called Timed Structural Parallelism Graph (TSPG). Nodes represent program components, arcs represent dependencies among components. This workload model differs from well known task graphs in two ways: (1) arcs can either have dependence or activation semantics and (2) timing parameters associated to arcs and nodes are given as intervals. Besides describing this new workload model, we sketch the issues and problems in corresponding evaluation techniques. In particular, we investigate techniques for estimating the total execution time and for deriving potential parallelism profiles. The proposed techniques are illustrated by example.

1 Introduction

To analyze and predict the performance of computer systems and networks, it is necessary to characterize the load of the system in terms of a workload model (see e.g. [CS93] for a survey). In this paper we propose a graph model representing the (typically coarse grain) dependency and communication structure of a distributed application. In contrast to well known graph models characterizing parallel applications (e.g. task graphs), our approach tries to model applications at a higher granularity. In previous work [CHK+95], a Structural Parallelism Graph (SPG) was proposed, where nodes represent program components and arcs represent either dependencies (e.g. the preceding component must have finished before its successor) or activation (the preceding component activates its successor, but continues its own execution).

We extend this SPG model to Timed Structural Parallelism Graphs (TSPGs). Timing information is associated to both, arcs and nodes in the graph. The timing parameters are specified as intervals with an associated probability. Intervals associated to a node represent possible durations of the component's execution time. Timing parameters associated to arcs represent the possible timing interval during which the successive component will be activated.

Both, the concept of activation arcs and the concept of associating timing intervals instead of average values or distributions contribute to the suitability of the model for the characterization of distributed applications. The components of a distributed application are typically coarse grain and it is much likely, that

* This work is supported by the Austrian National Science Foundation (FWF).

a component will activate another component during its execution. Furtheron, it might be difficult to give an accurate estimate of the execution time of a component in terms of a mean value or a distribution, but the analyst might have a good guess for the approximate duration specified as an interval. Variabilities and uncertainties in the execution time motivate the specification of timing information by means of intervals.

The idea of using intervals as timing parameters has also been applied to other models. In [vdA94] interval timed colored Petri nets are investigated. Solution techniques for queuing network models with interval-based input parameters are for example presented in [MR95] or in [LKMH96]. Client-Server Architectures are analyzed using interval arithmetic in [WMN91].

In this work, we present three techniques for estimating the total execution time and for deriving potential parallelism profiles from TSPGs. The first analysis technique is based on determining all possible "states" of a timed SPG, where a state is defined by the state of its components (not started, active or terminated). This information can be transformed into a parallelism profile by aggregating the number of active components at a certain instance in time. Using this approach, the most detailed results can be obtained but the computational complexity limits the practical use. The second approach derives parallelism profiles under minimum or maximum execution time assumption. The profiles are easy to compute, but only give a rough estimate on system behavior. In the third approach, a technique similar to networking techniques is applied. For each component, the possible start and end times are calculated. By comparing the start and end times for the components, the parallelism profiles can be derived. The techniques are demonstrated on simple examples. The computational complexity of the proposed algorithms is investigated and techniques for reducing the complexity are discussed. A set of tools supporting the analysis is currently under development.

2 Description of the Workload Model

2.1 Definition of an SPG

The Structural Parallelism Graph (SPG) has been proposed as part of a hierarchical approach for workload characterization of parallel applications:

Definition 1 *A* **Structural Parallelism Graph** *is an acyclic directed graph* $SPG = (V, D, W)$*, where* V *is a set of vertices corresponding to the components (parts of the application defined at the desired granularity level) and* $D = (D^p \cup D^a)$ *is a set of directed arcs defining two different types of relations between the components.*

$d_{i,j} \in D^p$ *denotes a precedence relation between vertices (components)* i *and* j *such that* j *is activated (starts execution) after* i *has finished. This activation can also involve sending of data (communication).* i *and* j *may never run in parallel, but are strictly sequential. We will call this type of arc a precedence arc. Graphically such an arc is depicted as a thin arrow.*

$d_{i,j} \in D^a$ *denotes an activation relation between vertices (components) i and j such that j is activated (starts execution) somewhen during the execution of i. This activation can also involve sending of data (communication). Thus i and j may run in parallel, but j must not start before i. We will call this type of arc an activation arc. The graphical representation is a thick arrow.*

W is a set of weights assigned to outgoing arcs, which represent branching probabilities.

Figure 1 shows two very simple examples. Both examples consist only of three nodes. Nodes 1 and 3 are connected via a precedence arc ($d_{1,3} \in D^p$), nodes 1 and 2 are connected via an activation arc ($d_{1,2} \in D^a$). In example (a) the semantic of the outgoing arcs is AND, in example (b) either one OR the other branch is taken. Incoming arcs can also have either AND or (exclusive) OR semantics (not shown in this example, as each node has only 0 or 1 incoming arcs).

The components of an SPG can be specified at different levels of detail. In a fine grain representation, each statement constitutes a component, thus the graph model will have a large number of nodes. In this work, we assume a coarse grain representation, i.e. a large number of statements is grouped together in a single component. The grouping is defined by the structure of the application. A component could therefor be seen as a part of the application, e.g. a certain (sub)function or procedure, which will be executed on a particular computer in the distributed system. We assume, that no parallelism within a component will be exploited, but that a component may activate other components which can then be executed in parallel

The SPG in Figure 1 (b) could for example represent a distributed application, where component 1 performs some initial computations. During its execution, either a second component can be activated resulting in two components being executed in parallel (e.g. with probability 0,4 component 1 distributes some of its load to another processing element). But it is more likely (prob=0,6), that component 1 terminates and component 3 will finish the computation.

2.2 Histogram Based Timing Parameters

For a quantitative analysis, it is necessary to associate timing information to the nodes in the SPG. We define a timed SPG (see Figure 1 (c)) as follows:

Definition 2 *A* **Timed Structural Parallelism Graph** *is an acyclic directed graph $TSPG = (V, D, W, T)$, where V, D, and W are defined as in an SPG and $T = \{T^V \cup T^{D^a}\}$ is a set of timing parameters. A timing parameter $t_i^v \in T^V$ is associated to a node and is given by a histogram which specifies the duration of the corresponding program component. A timing parameter $t_{i,j}^{D^a} \in T^{D^a}$ is associated to an activation arc connecting nodes i and j and is given by a histogram. It defines the moment (or interval) within the duration of the activating node when activation of node j will happen.*

Communication and contention costs are assumed to be either incorporated in the task execution time or are neglected.

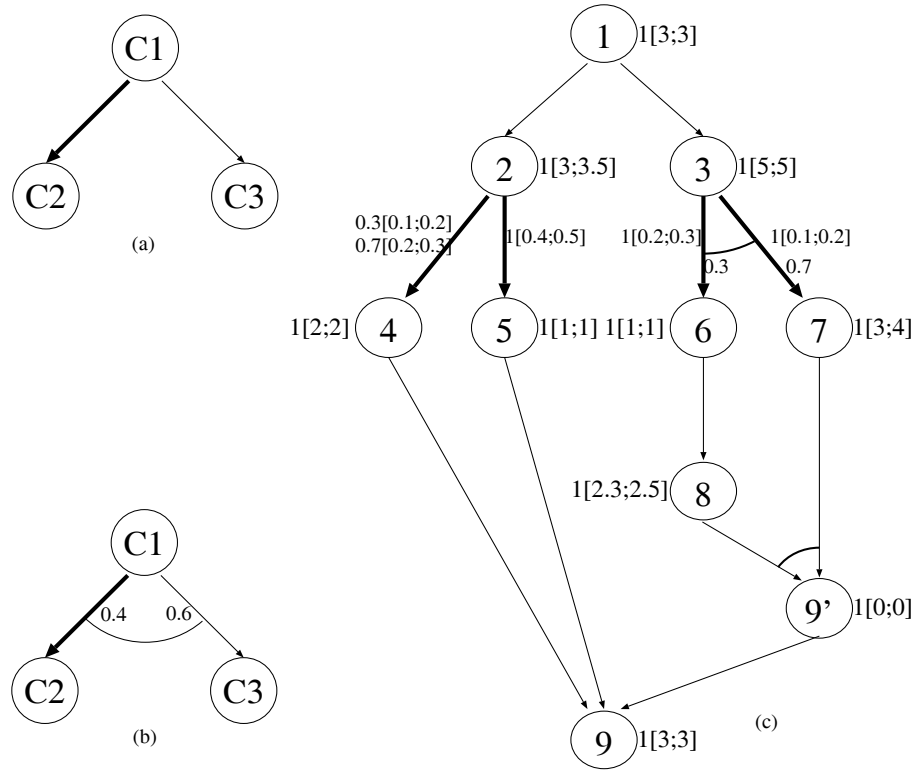


Fig. 1. Examples of a SPGs (a,b) and a Timed SPG (c)

Mathematically the timing parameter t_i^v which is associated to a node is defined as follows:

$$t_i^v : \; p_{i,k}^v[\underline{t_{i,k}^v}, \overline{t_{i,k}^v}], \; k = 1 \ldots K$$

where $\underline{t_{i,k}^v}$ is the lower and $\overline{t_{i,k}^v}$ the upper bound of the execution time interval and $p_{i,k}$ represents the probability that the execution time of component i is within the interval k. The timing parameter $t_{i,j}^{D^a}$ is defined as

$$t_{i,j}^{D^a} \; : \; p_{i,j,l}^{D^a}[\underline{x_{i,j,l}^{D^a}}, \overline{x_{i,j,l}^{D^a}}]$$

where $x_{i,j,l}^{D^a}$ denotes a percentage and is the lower bound of the relative part in the execution time of component i where i can activate component j, $x_{i,j,l}^{D^a}$ represents the corresponding upper bound and $p_{i,j,l}^{D^a}$ represents the probability that component j is activated by component i in the interval l. To transform these relative activation intervals into starting intervals for component j the following computation has to take place:

$$\underline{a_{i,j,m}} = \underline{t_{i,k}^v} * \underline{x_{i,j,l}^v} \tag{1}$$

$$\overline{a_{i,j,m}} = \overline{t_{i,k}^v} * \overline{x_{i,j,l}^v} \tag{2}$$

where $M = 1 \ldots K * L$ denotes all possible combinations of execution intervals and activation intervals. As an example consider the activation arc in Figure 1 between component 2 and component 4. The following starting intervals are computed for component 4:

$$\underline{a_{1.2,1}} = 3,3;\ \overline{a_{1.2,l}} = 3,7$$

$$\underline{a_{1.2,2}} = 3,6;\ \overline{a_{1.2,2}} = 4,05$$

We assume, that within an interval, all values are equally likely.

This timing model is a compromise between the assumption of deterministic values and a detailed specification of timing parameters as random variates following a certain distribution function. A single deterministic value may not sketch a very realistic picture of the timing demands of a node, because many factors will cause non-determinism in a distributed system (e.g. non-deterministic processing requirements or delays due to inter communication events and contention for shared hardware and software resources). On the other hand, the analyst might not have enough information to specify in detail the distribution of the component duration, but he or she might be able to give a good estimate of the execution time in terms of intervals, representing the variability in the execution time. If more information is known, the timing information can be specified as a set of intervals and a probability associated to each interval. Using these histograms, also distributions can be approximated.

Thus we give the analyst the necessary degree of freedom in deciding between model accuracy and model complexity.

3 Analysis Techniques

3.1 General Idea

Although efficient solution techniques and tool implementations exist for conventional task graphs (e.g. SHARPE [ST86] or PEPP [HM92]), these approaches cannot be applied in the analysis of TSPGs, because of the new concept of activation arcs. As an initial approach, a set of analysis tools has been presented in [BHK96] for the analysis of SPGs (without considering timing information). Based on the SPG representation, a state graph (SG) of the program is generated. A state in the SG is characterized by a vector containing the state of each of the components of an SPG (not started=0, active=1, or terminated=2). A state transition ($0 \rightarrow 1$ or $1 \rightarrow 2$) is characterized by the change of the state of one of the components. The state transitions are discretized, i.e. there is a hypothetical time axis and at each clock tick a transition may occur. A sequence of state transitions from an initial state to a final state is a possible path in the execution. Such a path can be represented in a Gantt Chart from which both, the execution time as well as potential parallelism profiles can be derived. At this level of abstraction, only the number of active components at a given instant in time is of importance, no matter which components they are exactly. This information may be helpful in the initial design phase of the application and might provide information for scheduling decisions.

When considering timing information, state transitions will occur with a certain probability in a certain time interval. Thus, changes in the parallelism profile occur with a certain probability in a certain time interval. The problem is now, how to derive the state transition times and probabilities.

We propose three different techniques for deriving parallelism profiles:

1. The SG model can be extended to a timed SG model. But this approach wouldn't be very efficient, because of its computational complexity. It can be shown, that the transition probabilities and the transition times depend on the history, i.e. on the previous nodes on the path in the TSG. This approach would give the most detailed information on potential parallelism profiles. However, it can already be seen for the simple example, that the number of possible intervals grows rapidly, thus resulting in a large number of different parallelism profiles.
2. To obtain a more general view of the potential parallelism profile, either the minimum or the maximum execution times can be considered for each node. Thus, each node has only a single value timing parameter and the computation of the potential parallelism profile is comparatively fast and simple.
3. The Activity Interval Approach (discussed in detail in Section 3.2) is a technique similar to techniques used in project planning. For each component, the possible start and end times are calculated and by comparing the start and end times for the components, the parallelism profiles can be derived.

3.2 The Activity-Interval Approach

The Activity Interval Approach is a four step procedure which determines potential parallelism profiles at a more detailed level than the MIN/MAX approach, but is of less computational complexity than the TSG approach. It will be explained on the example graph in Figure 1 (c). This graph represents a distributed application, where component 1 performs some initial computations. After its termination it activates component 2 and 3. During its execution component 2 activates components 4 and 5 (e.g. two subfunctions or subprocedures of component 2) at the specified activation intervals. Component 3 also activates its successor components during it's execution. It either activates component 6 (with probability 0.3) or component 7 (with probability 0.7). Component 9' is just a dummy node to reunite the OR branches in the TSPG. The execution of component 9 starts after the execution of component 4, 5, 8 or 4, 5, 7 has finished.

The four steps of the Activity-Interval approach are:

Step 1 Firstly all alternative paths through the TSPG are determined. The number of paths depends on the number of outgoing arcs with OR semantic. The following paths are identified in the TSPG in Figure 1 (c).

```
path 1:  1;2;3;4;5;6;8;9';9
path 2:  1;2;3;4;5;7;9';9
```

Step 2 For all nodes the possible starting and terminating time intervals and the corresponding probabilities have to be determined. The input to this step

is the specification of the TSPG (its structure and the timing parameters). The output is a set of interval pairs for each node i

$$p_{i,j}([\underline{s_{i,j}};\overline{s_{i,j}}],[\underline{e_{i,j}};\overline{e_{i,j}}])$$

giving the earliest possible starting time $\underline{s_{i,j}}$, the latest possible starting time $\overline{s_{i,j}}$ and the earliest possible end time $\underline{e_{i,j}}$ and the lates possible end time $\overline{e_{i,j}}$ where $p_{i,j}$ represents the probability that the execution takes place in the denoted interval.
Figure 2 shows the generated starting and terminating time intervals for path 1 of the TSPG in Figure 1.

Node	Starting	Terminating	Probability
1	[0.0;0.0]	[3.0;3.0]	1.0
2	[3.0;3.0]	[6.0;5.5]	1.0
3	[3.0;3.0]	[8.0;8.0]	1.0
4	[3.3;3.7]	[5.3;5.7]	0.3
4	[3.6;4.05]	[5.6;6.05]	0.7
5	[4.2;4.75]	[5.2;5.75]	1.0
6	[4.0;4.5]	[5.0;5.5]	1.0
8	[5.0;5.5]	[7.3;8.0]	1.0
9'	[7.3;8.0]	[7.3;8.0]	1.0
9	[7.3;8.0]	[10.3;11]	1.0

Fig. 2. Starting and Terminating Intervals

Step 3 After determining starting and terminating time intervals for each node, these intervals have to be aggregated in order to derive parallelism profiles. Each combination of a starting and a terminating interval is reduced to one interval taking the mean value of the starting interval as the lower bound and the mean value of the terminating interval as the upper bound of the possible execution time. Formally this transformation step can be described in the following way:

$$p_{i,j}([\underline{s_{i,j}};\overline{s_{i,j}}],[\underline{e_{i,j}};\overline{e_{i,j}}]) \Rightarrow p_{i,k}([\underline{q_{i,k}};\overline{q_{i,k}}])$$

with

$$\underline{q_{i,k}} = \underline{s_{i,j}} + \frac{\overline{s_{i,j}} - \underline{s_{i,j}}}{2}$$
$$\overline{q_{i,k}} = \underline{e_{i,j}} + \frac{\overline{e_{i,j}} - \underline{e_{i,j}}}{2}$$

Step 4 Finally parallelism profiles are generated for all possible combinations of component execution times and the corresponding probabilities for the combination are calculated. By interval splitting it is determined how many components are active at a given time.

We will now discuss in detail the computations involved in these steps and give closed expressions for calculating the corresponding values.

First, we will show, how the starting and terminating time intervals and the associated probabilities are computed. We assume without loss of generality, that there is a single component which starts the execution[2]. For this single component (node 0), the following equations hold:

$$\underline{s_0} = \overline{s_0} = 0$$

and

$$\underline{e_{0,j}} = \underline{s_{0,j}} + \underline{t_{0,j}}^v \tag{3}$$

$$\overline{e_{0,j}} = \overline{s_{0,j}} + \overline{t_{0,j}}^v \tag{4}$$

for $j = 1 \ldots J$, where J denotes the number of timing intervals associated to node 0. The probabilities associated to the J different intervals are given by the probabilities associated to the terminating time intervals in the TSPG specification $p^v_{0,j}$

For all other nodes, the end times are calculated analogously:

$$\underline{e_{i,m}} = \underline{s_{i,k}} + \underline{t_{i,l}}^v$$

$$\overline{e_{i,m}} = \overline{s_{i,k}} + \overline{t_{i,l}}^v$$

For each of the K possible starting times, L different time intervals are associated, thus $M = K * L$ possible end times are obtained.

For computing the starting times and the interval probabilities, we have to distinguish four cases:

1. node j has exactly one predecessor i and
   (a) i and j are connected via a dependence arc
   (b) i and j are connected via an activation arc
2. node j has more that one predecessor i
   (a) its input has OR semantic
   (b) its input has AND semantic

If component j has just one predecessor i and is connected with this predecessor by a

"dependence" -arc the starting time correspond to the terminating time of the predecessor node i:

$$\underline{s_{j,k}} = \underline{e_{i,k}}$$

$$\overline{s_{j,k}} = \overline{e_{i,k}}$$

The terminating times are determined using equations 3 and 4. For each of the K starting time, there are L different terminating times. The corresponding probabilities for each of the $M = K * L$ combinations of starting time and terminating time is computed by multiplying the probability of the

[2] If there is more than one starting component, a dummy component with execution time 0 can be introduced.

starting time (which corresponds to the probability of the terminating time of the predecessor node) with the probability of the execution time of the node itself:

$$p_{j,m} = p^v_{i,k} * p^v_{j,l}$$

"activating" -arc the starting times correspond to all possible activating times and are given by

$$\underline{s_{j,o}} = \underline{s_{i,k}} + \underline{a_{i,j,l}}$$
$$\overline{s_{j,o}} = \overline{s_{i,k}} + \overline{a_{i,j,l}}$$

where $\underline{a_{i,j,l}}$ and $\overline{a_{i,j,l}}$ are computed as shown in equations 2 and 2. The end time intervals are computed using equations 3 and 4. The probabilities for each start/end combination are computing by multiplying the probability of the starting time with the probability of the activating time and of the execution time of the node itself:

$$p_{j,o} = p^v_{i,k} * p^{D^a}_{i,j,m} * p^v_{j,l}$$

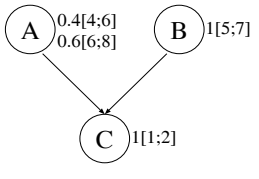
resulting in $O = K * L * M$ combinations.

If component j has more than one predecessor and is connected with its predecessors by

"OR"-semantic the execution times and probabilities are computed analogously to the case of just one predecessor since we assume an exclusive OR-semantic. Depending on the previous decision in branching at the outgoing OR-node, the corresponding incoming OR branch will be considered.

"AND"-semantic to cases can be distinguished:

1. If there exists one predecessor whose terminating time dominates (i.e. is larger than) the intervals of all other predecessor components, the starting times of the successor component correspond to the terminating times of this predecessor. The terminating times and probabilities are computed as if the component would have only one predecessor.
2. If the starting time of a component is determined by more than one predecessor an interval splitting has to take place to determine the latest possible end times of all predecessors (see Figure 3).



Node	Start.	Term.	Prob
A	[0.0;0.0]	[4.0;6.0]	0.4
A	[0.0;0.0]	[6.0;8.0]	0.6
B	[0.0;0.0]	[5.0;7.0]	1.0
C	[5.0;6.0]	[6.0;8.0]	0.2
C	[6.0;7.0]	[7.0;9.0]	0.5
C	[7.0;8.0]	[8.0;10.0]	0.3

Fig. 3. Example for a TSPG with an AND connection

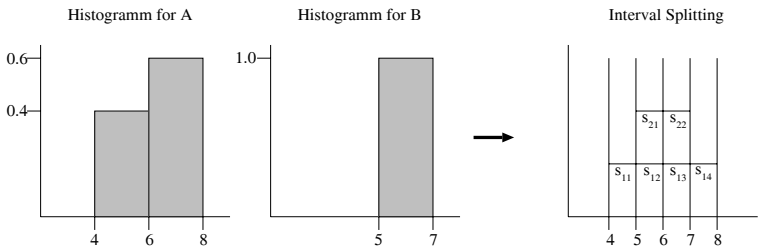


Fig. 4. Histograms and Interval Splitting for Node 1 and Node 2

The terminating times can be depicted as histograms (the values on the x-axes correspond to the terminating time intervals and the values on the y-axes correspond to the probabilities of the interval) (see Figure 4). To compute the respective probabilities for all possible combinations of execution times, the interval parameters $(\underline{e_{i,j}}, \overline{e_{i,j}})$, are transformed into subintervals $(S_{i1}, \ldots, S_{im}; \ldots; S_{k1}, \ldots, S_{kn})$ such that for each two S_{im}, S_{kn} of these subintervals it holds that either $S_{im} = S_{kn}$ or $S_{im} \cap S_{kn} = 0$.

This allows for a total ordering of the subintervals. To these subintervals corresponding probabilities are assigned, representing the share of subinterval of its original interval. Then all possible combinations of subintervals together with their probabilities of occurrence are analyzed and for each possible combination the probabilities are aggregated. These probabilities correspond to the probability that the starting time of the successor component is determined by a specific time interval.

3.3 Results

Figure 5 shows the parallelism profiles for the TSPG of Figure 1.

Figure 5 shows the parallelism profiles for the TSPG of Figure 1. The shapes of all profiles are similar, but they differ slightly in terms of execution times. When calculating hypothetical execution times for $1, 2, \ldots 5$ processing elements, speed up curves can be derived as shown in Figure 6. To determine the optimum number of processing elements, further metrics (e.g. efficacy) can be derived easily.

The presented analysis techniques thus provide a convenient method for doing rough estimates on execution time and for deriving parallelism profiles. The execution time of the analysis technique presented depends on the number of nodes, the number and size of the timing intervals, but mostly on the number of OR branches as each OR branch causes the generation of a different path through the TSPG. But since step two to four of the Activity Interval Approach are applied to each path independently the processing of these procedure steps could be distributed to different processing elements and therefor run in parallel.

Profile Number: 1		Profile Number: 2		Profile Number: 3		Profile Number: 4	
Time Stamp	# Tasks	Time Stamp	# Tasks	Time Stamp	# Tasks	Time Stamp	# Tasks
0.000	0	0.000	0	0.000	0	0.000	0
3.000	1	3.000	1	3.000	1	3.000	1
3.500	2	3.825	2	3.500	2	3.750	2
4.250	3	4.250	3	3.750	3	3.825	3
4.475	4	4.475	4	4.475	4	4.475	4
5.475	5	5.475	5	5.475	5	5.475	5
5.500	4	5.825	4	5.500	4	5.825	4
6.250	3	6.250	3	6.250	3	6.250	3
8.000	2	8.000	2	8.000	2	8.000	2
10.650	1	10.650	1	10.250	1	10.250	1
Profile Prob.: 0.09		Profile Prob.: 0.21		Profile Prob.: 0.21		Profile Prob.: 0.49	

Fig. 5. Parallelism Profiles

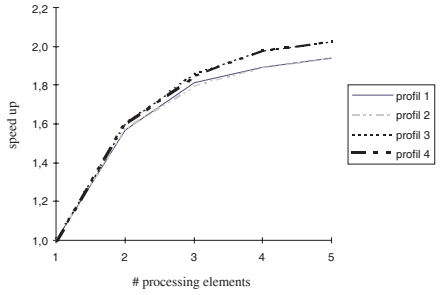


Fig. 6. Speed-up Profile

4 Conclusions and Future Work

In this work, we have addressed the problem of workload modeling for distributed systems. A graph model – the Timed Structural Parallelism Graph – has been proposed. This model provides the appropriate level of detail for modeling the typically coarse grain parallelism in distributed applications. In addition, component's execution times can be given as intervals, thus supporting flexibility in model parametrization.

We have discussed three approaches for deriving estimates on the total execution time and for deriving potential parallelism profiles. The first approach offers a very detailed result at the cost of prohibitively high computational complexity. The MIN/MAX approach can be computed easily, but the resulting parallelism profile is only a rough estimate of the actual profile. As a compromise, we have developed an activation-interval approach. The algorithm is sketched in the paper. Also this approach can become rather complex if the number of com-

ponents connected via activation arcs is large. Therefore, future work will focus on optimization techniques. Currently, the aggregation of "similar" intervals is investigated: When generating starting or terminating intervals, some intervals are rather close to each other or even overlapping. Others may only have a very small probability of occurrence. In these cases, it might be helpful to combine similar intervals into a single interval or to omit intervals, if the probability is very small. Ideally, the analyst should have the possibility to specify the degree up to which aggregation should be allowed. The actual influence of this optimization is subject to future research.

References

[BHK96] Markus Braun, Guenter Haring, and Gabriele Kotsis. Deriving parallelism profiles from structural parallelism graphs. In *Proc. of the TDP'96*, 1996.

[CHK+95] Maria Calzarossa, Guenter Haring, Gabriele Kotsis, Alessandro Merlo, and Daniele Tessera. A hierarchical approach to workload characterization for parallel systems. In B. Hertzberger and G. Serazzi, Ed., *High Performance Computing and Networking, LNCS vol. 919*, pages 102–109, 1995.

[CS93] Maria Calzarossa and Guiseppe Serazzi. Workload characterization: A survey. *Proc. of the IEEE*, 81(8):1136–1150, August 1993.

[HM92] Franz Hartleb and Vassilis Mertsiotakis. Bounds for the mean runtime of parallel programs. In Rob Pooley and Jane Hillston, Eds., *Proc. of the 6th Int. Conf. on Modelling Techniques and Tools for Computer Performance Evaluation*, pages 197–210, 1992.

[LKMH96] Johannes Lüthi, Gabriele Kotsis, Shikharesh Majumdar, and Günter Haring. Bounds-based performance analysis for distributed systems with variabilities and uncertainties in workload. In *Proc. of DAPSYS'96*, pages 51–58, Hungarian Academy of Sciences Report KFKI-1996-09/M,N, October 1996.

[MR95] Shikharesh Majumdar and Revathy Ramadoss. Interval-based performance analysis of computing systems. In Patrick Dowd and Erol Gelenbe, Eds. *Proc. MASCOTS 95*, pages 345–351. IEEE CS Press, Jan. 1995.

[ST86] Robin A. Sahner and Kishor S. Trivedi. *SHARPE: Symbolic Hierarchical Automated Reliability and Performance Evaluator – Introduction and Guide for Users.* Gould CSD, Urbana, 1101 E. University, Urbana, IL 61801, Sep. 1986.

[vdA94] W. M. P. van der Aalst. Using interval timed coloured petri nets to calculate performance bounds. available upon request from wsinwa@win.tue.nl, 1994.

[WMN91] C. Murray Woodside, Shikharesh Majumdar, and J. E. Neilson. Interval arithmetic for computing performance guarantees in client-server software. In F. Dehne, F. Fiala, and W. W. Koczkodaj, Eds. *LNCS 497: Proc. ICCI '91*, pages 535–546, Berlin, et al., 1991. Springer-Verlag.

This article was processed using the LaTeX macro package with LLNCS style