Performance Prediction of Data-Dependent Task Parallel Programs

Hasyim Gautama and Arjan J. C. van Gemund

Faculty of Information Technology and Systems, Delft University of Technology
P.O. Box 5031, NL-2600 GA Delft, The Netherlands
{H.Gautama, A.J.C.vanGemund}@ITS.TUDelft.NL

Abstract. Current analytic solutions to the execution time prediction Y of binary parallel compositions of tasks with arbitrary execution time distributions X_1 and X_2 are either computationally complex or very inaccurate. In this paper we introduce an analytical approach based on the use of lambda distributions to approximate execution time distributions. This allows us to predict the first 4 statistical moments of Y in terms of the first 4 moments of X_i at negligible solution complexity. The prediction method applies to a wide range of workload distributions as found in practice, while its accuracy is better or equal compared to comparable low-cost approaches.

1 Introduction

A well-known problem in the performance analysis of parallel and distributed systems is to predict the execution time of a parallel composition of tasks having stochastic execution time. Parallel task compositions can be distinguished into n-ary compositions and binary compositions. The n-ary compositions typically result from data parallelism (e.g., parallel loops) where each task essentially involves the same computation on different data. Binary compositions, in contrast, typically result from task parallelism where the computation involved in the composition may be totally *different*. Consequently, performance prediction of task parallel programs frequently requires the evaluation of binary (heterogeneous) parallel compositions, a problem that, unlike n-ary compositions, has not received much attention.

Consider a binary parallel composition of two tasks having execution time X_1 and X_2, respectively. The resulting execution time Y of the parallel composition is given by

$$Y = \max(X_1, X_2) \tag{1}$$

Many authors have used Eq. (1) as part of a compile-time static prediction technique [1,3,11,14]. In these approaches X_i (and Y) are implicitly assumed to be deterministic. While Eq. (1) indeed yields a correct prediction when X_i are deterministic, interpreting Eq. (1) in terms of mean values when X_i are *stochastic* (i.e., $\mathsf{E}[Y] = \max(\mathsf{E}[X_1], \mathsf{E}[X_2])$) introduces a severe error which increases monotonically with the variance of X_i [7]. For example, consider the

R. Sakellariou et al. (Eds.): Euro-Par 2001, LNCS 2150, pp. 106–116, 2001.

binary composition of two tasks whose execution time are independent, normally distributed with $\mathsf{E}[X_1] = \mathsf{E}[X_2] = 1$, $\mathrm{Var}[X_1] = 1$ and $\mathrm{Var}[X_2] = \sigma^2$. Our measurements show that the relative error of the predicted $\mathsf{E}[Y]$ is almost 40% and 70% for $\sigma = 1$ and 5, respectively. Yet, in many practical circumstances X_i are typically modeled as stochastic parameters, reflecting the execution time distribution of possibly time-critical tasks over a large spectrum of possible input data sets, and/or the inherent stochastic behavior of the underlying virtual machines. While such a stochastic approach is more effective and realistic than using deterministic parameters [15], solving Eq. (1) now becomes a non-trivial problem well-known in the field of order statistics.

There are a number of approaches to express an execution time distribution, the choice of which largely determines the trade-off between accuracy and cost involved in solving Y. An exact, closed-form, solution for the distribution of Y can be obtained using the cumulative density function (cdf). Let $F_{X_i}(x)$ denote the cdf of X_i. For independent X_i from order statistics [17] it follows that $F_Y(x)$ is given by

$$F_Y(x) = F_{X_1}(x) F_{X_2}(x) \tag{2}$$

While Eq. (2) is exact, only parametric solutions are of practical use.

Recently a method has been proposed [4] where a distribution is represented in terms of a limited number of statistical moments. The kth moment of X, denoted $\mathsf{E}[X^k]$, is defined by

$$\mathsf{E}[X^k] = \int_{-\infty}^{\infty} x^k \mathrm{d}F_X(x) \tag{3}$$

This method has been successfully applied to the analysis of sequential and conditional task compositions [5], such that the first four moments of the execution time ($\mathsf{E}[Y^k]$) of an arbitrary composition of loops and branches can be recursively expressed in terms of the first four moments of each loop bound, branch probability, and basic block execution time ($\mathsf{E}[X^k]$) at $O(1)$ cost. Although the approach is straightforward in the sequential domain, for parallel composition there is in general no analytic, closed-form solution for $\mathsf{E}[Y^k]$ in terms of $\mathsf{E}[X^k]$ due to a fundamental integration problem [4].

In this paper we present a method based on the use of lambda distributions as intermediate approximation of an execution time distribution in terms of the first four moments. Our contribution has the specific advantage that the approximation of $\mathsf{E}[Y^k]$, now readily expressed in terms of the input moments $\mathsf{E}[X_i^k]$, has $O(1)$ solution complexity while the approximation error is acceptable. Recently the use of lambda distributions has already been successfully introduced to solve a related integration problem for n-ary parallel compositions of tasks with independent, identically distributed (iid) execution times [4]. Experiments show that the estimation error of the mean value of the parallel execution time is less than 4% for parallel sections comprising up to 10,000 tasks whose execution times are normally distributed. Measurements on real programs (NAS-EP benchmark, PSRS sorting program, and WATOR simulator) confirm these results provided the task execution distributions are independent and unimodal [4].

However, the requirements of X_i being identical restricts these results to binary parallel compositions with *identical* work loads while *non*-identical work loads frequently occur in practice.

The results for binary compositions of *non*-identical tasks as presented in this paper therefore extends the general applicability of the moments approach from the data parallel computing domain to the task parallel computing domain, where task heterogeneity is common. Combining with the results for sequential programs the moment method constitutes an integrated approach to the analysis of any parallel program that can be modeled by Series-Parallel stochastic graphs (SP-graphs). To the best of our knowledge such an approach towards solving $\mathsf{E}[Y^k]$ for binary parallel task compositions in terms of $\mathsf{E}[X^k]$ has not been described elsewhere.

The remainder of the paper is organized as follows. In Section 2 we review current approaches towards performance estimation of binary parallel task composition. In Section 3 we present our approximation method using lambda distributions. In Section 4 we test the our method using well-known standard distributions as well as distributions measured from real applications.

2 Related Work

There have been a number of analytic approaches to predicting the execution time of a binary parallel composition of tasks having stochastic execution time. One approach is to restrict the type of distributions allowed for X to those, for which exact analytical solutions can be derived, such as the class of discrete distributions using the traditional z-transform [9], exponential and uniform distributions [10], and the class of exponomial distributions [13]. While the solution is exact, such execution time distributions are seldom found in practical programs.

Another approach is by approximating X in terms of increasing failure rate random variables [8] or in terms Gram-Charlier series of type A from which the integration problem can be solved [4]. Again, let $\mathsf{E}[X^k]$ be the moments that characterize the distribution of X. Then the approximating probability density function (pdf) of X, denoted by $f(x)$, can be expressed by the Gram-Charlier series [4]. While asymptotically exact, it can be shown, unfortunately, that the number of Gram-Charlier terms needed for a sufficiently accurate approximation is prohibitive [4].

While the above approaches allow X_i to have different distributions, other methods approach the binary composition problem from the n-ary perspective [4,6,7], i.e., by solving $Y = \max(X_1, \ldots, X_N)$ for $N = 2$. However, all these approaches assume X_i to be iid which significantly narrows the application space.

As a result of the difficulties in finding a low-cost, accurate, analytical solution to solving the binary parallel composition problem, also a number of heuristic approaches have been proposed. A good example of such an approach is found in [15]. Y is calculated by simply choosing X_i with the largest mean or by selecting the stochastic value with the largest magnitude value in its entire range.

For example (adapted from [15]), to compute the maximum of $X_1 = 4 \pm 0.5$ and $X_2 = 3 \pm 2$, X_1 has the largest mean, and X_2 has the largest value within its range. On average however, the values of X_1 are likely to be higher than the values of X_2. We formulate the heuristic as described in [15] as follows

$$Y = \begin{cases} X_1, & \mathsf{E}[X_1] > \mathsf{E}[X_2] \\ X_1, & \mathsf{E}[X_1] = \mathsf{E}[X_2] \text{ and } \mathsf{E}[X_1^2] > \mathsf{E}[X_2^2] \\ X_2, & \text{otherwise.} \end{cases} \tag{4}$$

Being a simple heuristic, despite its attractive low-cost property, Eq. (4) only takes into account the first two moments $\mathsf{E}[X_i^k]$, and, amongst other things, does not compute the offset in $\mathsf{E}[Y]$ as established by the order statistics. Nevertheless, next to our positive prior experience with lambda distributions, this heuristic has partly been the inspiration for our low-cost, analytic solution to the binary parallel composition problem.

3 Methodology

In this section we present our approximation approach using lambda distributions. Due to space limitations the background of lambda distributions is omitted, while the interested reader is referred to [12]. First, we describe the principle behind our approach, after which we present our main result for binary parallel composition.

3.1 Principle

To illustrate the principle of our approach consider the following example. Let X_1 and X_2 be uniform distributions with sample spaces $[0, 1/m]$, where $m > 0$ is a shape parameter, and $[1, 3/2]$, respectively. Figure 1 shows F_{X_1} for $m = 1$ and F_{X_2} which represents a trivial case to obtain Y. From Eq. (2) we immediately obtain $F_Y = F_{X_2}$ since $F_{X_1} = 1$ for $F_{X_2} > 0$. In contrast, Figure 2 shows the resulting F_Y (dashed line) using Eq. (2) for $m = 1/2$. Due to the integration problem this exact solution F_Y cannot be evaluated explicitly in terms of $\mathsf{E}[X_i^k]$. Applying heuristic Eq. (4) would simply yield $Y = X_2$ independent of $\mathrm{Var}[X_1]$. Consequently this heuristic causes a large estimation error shown by area between F_{X_2} and F_Y which increases monotonically as function of $\mathrm{Var}[X_1]$.

A better approximation can be obtained by taking the *minimum* of F_{X_1} and F_{X_2} as shown by the bold solid line in Figure 2. In contrast to heuristic Eq. (4) this approach implicitly takes $\mathrm{Var}[X_1]$ into account such that the estimation error is less sensitive to $\mathrm{Var}[X_1]$.

Since the execution time distribution in our method is expressed in terms of statistical moments, we have to evaluate the moments of Y which can be time consuming because the cdf of Y may range from $-\infty$ to ∞. Due to the specific (inverse) formulation of the lambda distribution, however, the moments can be easily obtained while the cdf range is significantly reduced from 0 to 1 (as shown

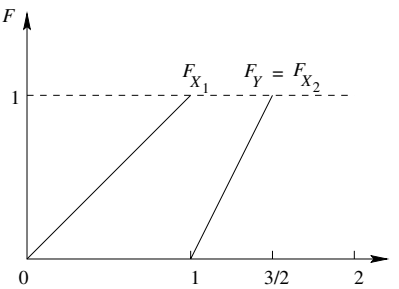

Fig. 1. $Y = \max(X_1, X_2)$ for $m = 1$

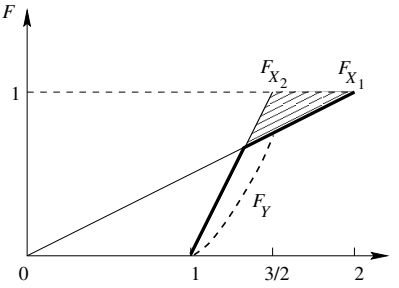

Fig. 2. $Y = \max(X_1, X_2)$ for $m = 1/2$

in Section 3.2). A second reason for using the lambda distribution is that its parameters (λ values) can be obtained from $\mathsf{E}[X^k]$ in a straightforward manner. Thus $\mathsf{E}[Y^k]$ can be easily evaluated directly from $\mathsf{E}[X^k]$ using the λ values as intermediate parametric representation.

3.2 Binary Parallel Composition

We now present our new result for general binary parallel compositions in Theorem 1 based on the use of the lambda distribution.

Theorem 1. *Let random variable Y be defined as*

$$Y = \max(X_1, X_2)$$

where X_i are independent random variables for which $\mathsf{E}[X^k]$, $k = 1, 2, 3, 4$ exists. Let X_i be expressed in terms of the lambda distribution as

$$X_1 = R_{X_1}(F) = \alpha_1 + \frac{(F^{\alpha_3} - (1-F)^{\alpha_4})}{\alpha_2}$$
$$X_2 = R_{X_2}(F) = \beta_1 + \frac{(F^{\beta_3} - (1-F)^{\beta_4})}{\beta_2}$$

where α_i and β_i are constants evaluated from a simple function of X_1 and X_2, respectively. Then the moments of Y are approximated by

$$\mathsf{E}[Y^k] = \int_0^1 (\max(R_{X_1}(F), R_{X_2}(F)) + \Delta x)^k \mathrm{d}x \tag{5}$$

where Δx is the approximation error. □

Due to space limitations the proof of Theorem 1 is given in [4]. It can be proven that Δx reaches a maximum when $\mathsf{E}[X_1] = \mathsf{E}[X_2]$ and approaches zero when $|\mathsf{E}[X_1] - \mathsf{E}[X_2]|$ is large. As mentioned earlier, due to specific formulation of λ distribution the distribution of Y may range from 0 to 1 rather than from $-\infty$ to ∞ which significantly reduces the solution cost. Note that Eq. (5) conserves the commutative property of the binary parallel composition.

4 Synthetic Distributions

This section describes the quality of our prediction approach when applied to some of the frequently-used standard distributions. The estimation quality is defined by the relative error ε_k evaluated from the predicted moments $\mathsf{E}[Y_p^k]$ in Eq. (5) and the measured moments $\mathsf{E}[Y_m^k]$, according to

$$\varepsilon_k = \frac{|\mathsf{E}[Y_p^k] - \mathsf{E}[Y_m^k]|}{\mathsf{E}[Y_p^k]} \tag{6}$$

In our applications we use ε_k to determine the estimation error of our method (Eq. (5)) as well as the heuristic (Eq. (4)).

The first synthetic distribution is the continuous uniform distribution. Let X_1 and X_2 be continuous uniform distributions with sample spaces $[0, 1/m]$ and $[1, 3/2]$, respectively, where m in X_1 is introduced to evaluate the relative error between X_1 and X_2 for various scenarios. For $0.1 \leq m < 2/3$ our method is much better than the heuristic as shown in Figure 3 while for $2/3 \leq m < 1$ both methods yield the same error. For $m \geq 1$ both methods have no error since the cdf's are disjunct (i.e., $Y = X_2$). The maximum error of our method is 15% while ε_1 is even less than 5%.

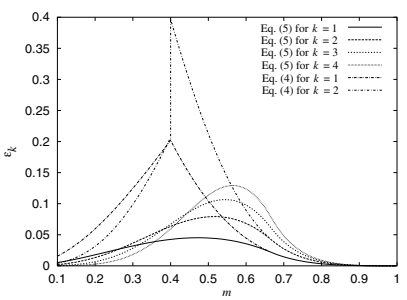

Fig. 3. ε_k for uniform distribution

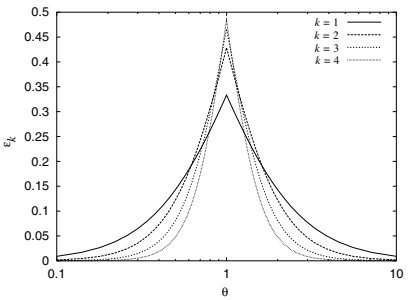

Fig. 4. ε_k for exponential distribution

A similar scenario is also used for the exponential distribution where $\mathsf{E}[X_1] = 1$, and $\mathsf{E}[X_2] = 1/\theta$ with θ varying from 0.1 to 10. The relative error is shown in Figure 4. Again as a consequence of the prediction principle described in Section 3.1, the error is the largest when $\mathsf{E}[X_1] = \mathsf{E}[X_2]$. Note that in this case the heuristic Eq. (4) has the same performance since $F_{X_2} < F_{X_1}$ in the entire range causes both methods to return the X with the greatest mean value. Note also that in Figure 4 ε_k is symmetric around $\theta = 1$ when θ is plotted in logarithmic scale. The maximum error for $\mathsf{E}[Y]$ is 33%, which sharply decreases for diverging workloads.

The third synthetic distribution is the normal distribution where $\mathsf{E}[X_1] = 1$ and $\mathrm{Var}[X_1] = 1$, while $\mathsf{E}[X_2] = \mu$ and $\mathrm{Var}[X_2] = \sigma^2$. We vary μ and σ as shown in Figures 5 and 6, respectively. Again, the estimation error decreases as the workloads diverge, while for larger variance our method outperforms the heuristic. In Figure 5 both methods have the same error since the predicted Y is X_2. Note that in Figure 6 the decreasing ε_k for $k = 2$ and $\sigma = 2$ is due to $\mathsf{E}[Y^2] > \mathsf{E}[X_2^2]$ for $1 < \sigma < 2$ while $\mathsf{E}[Y^2] < \mathsf{E}[X_2^2]$ for $\sigma > 2$. The maximum error of $\mathsf{E}[Y]$ is 35%, while the error decreases with $\mathrm{Var}[X_2]$ whereas the error of heuristic Eq. (4) steadily increases.

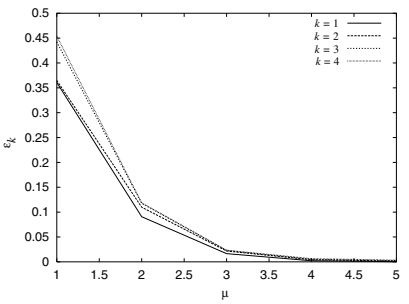

Fig. 5. ε_k for normal distribution $\sigma = 1$

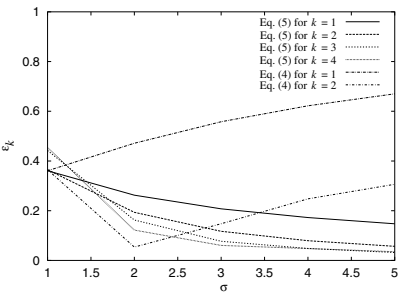

Fig. 6. ε_k for normal distribution $\mu = 1$

Summarizing this section, we conclude that the maximum error of our method occurs when $X_1 = X_2$, while the error quickly *decreases* with increasing workload unbalance. Although the maximum error ranges into tens of percents, the heuristic is a significant improvement over the previous heuristic Eq. (4), and of course, over the commonly used deterministic approach (Eq. (1)) of which the error reaches of $\mathsf{E}[Y]$, e.g., 70% for $\sigma = 5$ in the normal distribution case. Furthermore, the difference between the prediction accuracy of our method and the previous heuristic and deterministic approaches increases in favor of our method for increasing task execution time variance.

A significant point of our results is that our method *gains* accuracy for *different* workloads, a situation typical for task parallelism, which indeed is the focus of our method. In this sense our method is complementary to our n-ary (data parallel) prediction method which adequately covers the situation where $\mathsf{E}[X_1]$ is *close* to $\mathsf{E}[X_2]$ [4].

5 Empirical Distributions

5.1 Pipeline

In this section we determine the quality of our approximation when applied to a pipelined application of two tasks whose execution times are denoted by X_1

and X_2, respectively. The pipeline comprises a Gaussian random generator task that supplies one-dimensional, real-valued vectors of length N to a PSRS sorting task. In steady state the total execution time per vector of the pipelined application is given by $Y = \max(X_1, X_2)$, assuming both tasks use different resources.

The generator task is implemented using the data parallel NAS-EP benchmark [2]. The PSRS sorting task is a data parallel application that sorts the input array by first dividing the array in P equal subarrays where P also denotes the number of processors. Each partition is sorted in parallel, after which a number of global pivots are determined. Based in these pivots, the subarrays are cyclically merged. As a result, a sorted array is obtained. A description of the algorithm can be found in [16].

To have an interesting scenario for our experiment (i.e., the worst case for Eqs. (5) and (4)), we adjust the parameters such that both tasks have approximately equal workload (which also balances the pipeline). We choose $P = 1$ processor for NAS-EP and $P = 24$ for PSRS. The pipeline processes $6,000$ arrays per experiment. A number of experiments, based on $6,000$ simulation runs, are performed where N varies from $N = 4\ 10^4$ to $N = 10^5$. For $N = 10^5$ the pdf's of X_1 and X_2 are given in Figure 7. In this figure X_1 is approximately normal while X_2 has a long right tail. Furthermore, the variance of X_2 (the sorting task) is much larger than that of X_1.

Figure 8 shows ε_k of our method and the heuristic. For both methods ε_k is decreasing for increasing N due to the disjunction of X_1 and X_2. Although $\mathsf{E}[X_1] = \mathsf{E}[X_2]$ ε_k is excellent since the coefficient of variation is small (in practice, task variance is indeed much smaller than we have assumed in our theoretic evaluation in Section 4).

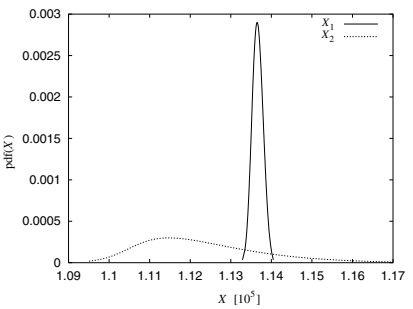

Fig. 7. pdf(X_1) and pdf(X_2) for $N = 10^5$

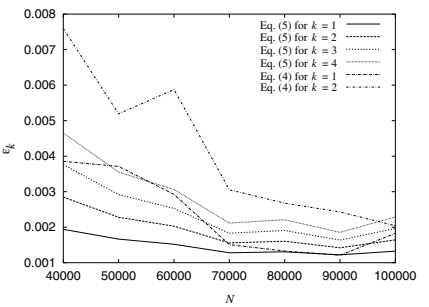

Fig. 8. ε_k for pipeline

5.2 WATOR

In this section we predict the performance of WATOR, an application where the task workload balance changes significantly over the time. WATOR is a Monte Carlo simulation in which idealized fish and sharks live, move randomly, breed, and eat one another in a two-dimensional ocean with toroidal topology. The characteristic of such an algorithm is that the workload can be severally unbalanced because of the computational scenario. Hence the workload in each processor is changing with increasing iteration number i (i.e., simulated time). The load unbalance comes about naturally because of the dynamics of the problem: the fish and sharks tend to aggregate in schools as the they breed, move and eat each other. More information on the algorithm can be found in [4].

In our experiment we perform a simple rectangular subdomain decomposition of the ocean for $P = 2$ processors such that each processor is assigned to process 2,500 grid points. As workload X we choose the number of fish within each processor, which on initialization is generated randomly over each location in the ocean. The workload X_1 for i ranging from 0 to 256 is given in Figure 9 in terms of central moments based on $6,000$ simulation runs. The pdf of X_1, where X_1 is normalized, is given in Figure 10. As shown by the table and the figure, the workload changes over time, exhibiting a bimodal distribution for $i = 256$. The estimation error is given in Figure 12 for iteration number $i = 2, \ldots, 128$. Although initially below 5%, the error quickly increases with the iteration number i as a result of the large correlation between X_1 and X_2, which implies that X_1 and X_2 are no longer independent (populations in different processors influence one-another). This dependence also causes the error to be the same for both methods.

In Figure 11 the correlation is shown between P_1 and P_j for $P = 16$ processors. Despite the small coefficient of correlation the actual covariance is large due to large variance values. As to be expected Figure 11 shows that the correlation is the largest for the processors nearest to P_1 (P_2 and P_{16}, the processors are interconnected in a 1-D torus). Figure 11 also shows that the correlation increases with the iteration number i.

i	E$[X_1]$	Var$[X_1]$	Skw$[X_1]$	Kur$[X_1]$
0	$1,000$	594	-0.02	2.99
4	491	175	0.07	3.01
16	895	$7,421$	0.36	2.80
64	$1,762$	$208,695$	-0.40	2.50
128	$1,278$	$230,641$	1.02	3.32
256	$1,487$	$495,116$	0.22	1.86

Fig. 9. The first four central moments of X_1 for WATOR ($P = 2$)

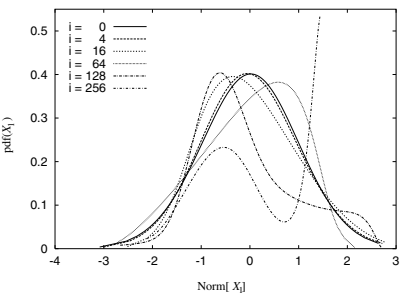

Fig. 10. Normalized pdf(X_1) for WATOR

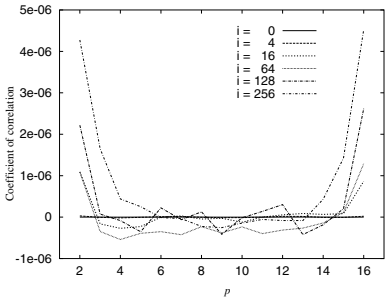

Fig. 11. Measured correlation for WATOR

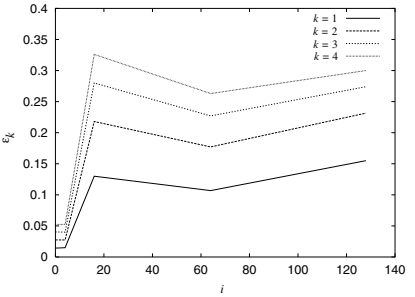

Fig. 12. ε_k for WATOR

In summary, our measurements of the two above applications show that the error of $\mathsf{E}[Y]$ is in the percent range, rather than in the ten percent range, as in practice task execution time variance is less than our theoretic experiments might suggest. Provided the task execution times are independent these results suggest that this is even a worst case as in the corresponding experiments the tasks had equal workloads.

6 Conclusion

In this paper we have presented an analytical model of the execution time distribution Y of binary parallel composition of tasks X_1 and X_2 with stochastic workloads. Our approach is based on approximating the distributions X and Y in terms of the first four moments, in conjunction with the use of the lambda distribution as an intermediate vehicle, to derive a closed-form, $O(1)$ complexity expression for $\mathsf{E}[Y^k]$ in terms of $\mathsf{E}[X^k]$.

We have investigated to what extent the moments approximation and the lambda distribution approximation have affected the accuracy of our model. Measurements using three well-known synthetic workload distributions show that the worst case error of $\mathsf{E}[Y]$ is 35%, occurring for equal workloads, while sharply decreasing for diverging workloads. Empirical data obtained from two real programs suggest that the worst case error may actually be much less (our measurements indicate 5%), provided the task workloads are independent.

As an adequate solution already exists for parallel compositions where tasks have equal workloads, our method focuses on filling the gap where workloads are different. The results show that in this sense our approach provides a low-cost solution that outperforms comparable methods known to date.

References

1. Allen, F., Burke, M., Cytron, R., Ferrante, J., Hsieh, W., and Sarkar, V.: "A Framework for Determining Useful Parallelism." In *Proc. 1988 Int. Conf. Parallel Proc.*, IEEE, Aug. 1988, pp. 207–215. 106

2. Bailey, D. *et al.*: The NAS Parallel Benchmarks, Tech. Rep. RNR-94-007, NASA Ames Research Center, Moffett Field, CA, March 1994. 113
3. Fahringer, T. and Zima, H. P.: "A static parameter-based performance prediction tool for parallel programs." In *Proc. 7th ACM Int'l Conf. on Supercomputing*, Tokyo, July 1993, pp. 207–219. 106
4. Gautama, H.: On the Use of Lambda Distributions in the Prediction of Parallel Program Execution Time, Tech. Rep. 1-68340-44(2000)02, Delft University of Technology, Delft, The Netherlands, Feb. 2000. 107, 108, 110, 112, 114
5. Gautama, H. and van Gemund, A. J. C.: "Static Performance Prediction of Data-Dependent Programs." In *ACM Proc. on The Second International Workshop on Software and Performance (WOSP 2000)*, Ottawa, Canada, Sept. 2000. 107
6. Gelenbe, E., Montagne, E., Suros, R.: "A Performance Model of Block Structured Parallel Programs." In Cosnard, M. *et al.* (eds.): *Parallel Algorithms and Architectures*, Elsevier Science Publishers B. V. North-Holland, 1986, pp. 127–138. 108
7. Gumbel, E. J.: "Statistical theory of extreme values (main results)." In *Contributions to Order Statistics* (Sarhan, A. E. and Greenberg, B. G. eds.), John Wiley & Sons, 1962, pp. 56–93. 106, 108
8. Kruskal, C. P. and Weiss, A.: "Allocating Independent Subtasks on Parallel Processors." In *IEEE Transactions on Software Engineering* Vol. 11, Oct. 1985, pp. 1001–1016. 108
9. Lester, B. P.: "A system for Computing The Speedup of Parallel Programs." In *Proc. 1986 Int. Conf. Parallel Proc.*, IEEE, Aug. 1986, pp. 145–152. 108
10. Madala, S. and Sinclair, J.: "Performance of Synchronous Parallel Algorithms with Regular Structures." In *IEEE Trans. on Parallel and Distributed Systems*, Vol. 2, No. 1, Jan, 1991, pp. 105–116. 108
11. Mendes, C. L. and Reed, D. A.: "Integrated compilation and scalability analysis for parallel systems." In *Proc. Int'l Conf. on Parallel Architectures and Compilation Techniques (PACT '98)*, Paris, Oct. 1998, pp. 385–392. 106
12. Ramberg, J. S. *et al.*: "A Probability Distribution and Its Uses in Fitting Data." In *Technometrics*, Vol. 21, No. 2, May, 1979, pp. 201–214. 109
13. Sahner, R. A. and Trivedi, K. S.: "Performance and Reliability Analysis Using Directed Acyclic Graphs." In *IEEE Transactions on Software Engineering*, Vol. SE-13, No. 10, Oct. 1987, pp. 1105–1114. 108
14. Sarkar, V.: "Determining Average Program Execution Times and their Variance." In *Proc. 1989 ACM SIGPLAN Conf. Prog. Language Design and Implementation*, pp. 298–312. 106
15. Schopf, J. M. and Berman, F.: "Performance Prediction in Production Environments." In *Proc. of 1998 IPPS/SPDP, 12th Int. Parallel Processing Symp. on Parallel and Distributed Processing*, Orlando, Florida, 1998, pp. 647–653. 107, 108, 109
16. Shi, H. and Schaeffer, J.: "Parallel Sorting by Regular Sampling." In *Journal of Parallel and Distributed Computing*, Vol. 14, No. 4, 1992, pp. 361-372. 113
17. Trivedi, K. S.: *Probability and Statistics with Reliability, Queueing, and Computer Science Applications*, Prentice-Hall, INC. , Englewood Cliffs, NJ. 1982. 107