Average Execution Times of Series–Parallel Networks

W. J. Gutjahr and G. Ch. Pflug
Institute of Statistics and Computer Science
University of Vienna

Abstract

The papers investigates a model for series–parallel processing structures developed by E. Gelenbe. We show that, under the so–called combinatorial distribution assumption, the average total execution time of a series–parallel processing structure cannot grow essentially slower than $n^{1/2}$, where n is the number of primitive tasks in the structure.

1 Introduction

Since MacMahon's article [7], published exactly a hundred years ago, series–parallel networks have found continuous interest among combinatorialists (a survey and some new results are given in Moon [8]). In the present paper, we study a combinatorial parameter of such networks which plays an important role in the analysis of partially parallel processing structures.

In [2], chapter 5, Gelenbe develops a model for series–parallel processing structures. He supposes that a program is composed of (primitive) tasks; some of them are to be performed in series, others may be performed in parallel. The execution times of the tasks are independent identically distributed random variables. The question is: what can be said about the execution time of the whole program?

Obviously, an answer to this question will not only depend on the execution time distribution of a task, but also on the assumed distribution of processing structures. If we consider the processing structure as (deterministically) given, then the derivation of properties of the total execution time is a *computational* problem. This is the usual viewpoint in the theory of project planning, where processing structures with random execution times of primitive tasks are represented by *PERT–networks*. Recent results on this version of the problem can be found in Yazici-Pekergin and Vincent [9].

In Gelenbe's model, however, also the processing structure is considered as a random variable. He assumes that the structure originates as the result of a *branching process* (which models stepwise refinement of a program). Given the execution time distribution of a task and the characteristic parameters of the branching process, Gelenbe shows how to compute *numerically* the execution time distribution of the program.

This approach has two drawbacks:

- It does not supply information on the *average* total execution time, since there is no way to determine the (infinite) tail of the distribution.

- As the result of the branching process, processing structures of *different size* may occur. Their properties get "mixed" in the model. One does not obtain information on the behavior of processing structures of a *given size* n.

In this paper, we follow the usual line of investigation in the average case analysis of algorithms (see e. g. Kemp [5]): The average performance of the system is studied as a function of the "problem size" n, with the emphasis on the examination of the asymptotic behavior of this function. Applied to the present problem, this approach means that we have to *condition* the processing structure resulting from Gelenbe's branching process on a *fixed size* n.

(Series–parallel) processing structures will be represented by *series–parallel networks:*

Definition 1.1. A series–parallel network (abbreviated SPN) is defined by:

(i) A task, denoted by $\Box$, is a SPN.

(ii) If $N_1, \ldots, N_k$ $(k \geq 2)$ are SPNs, then also $\mathrm{series}\,(N_1, \ldots, N_k)$ and $\mathrm{parall}\,(N_1, \ldots, N_k)$ are SPNs.

$\mathrm{series}\,(N_1, \ldots, N_k)$ resp. $\mathrm{parall}\,(N_1, \ldots, N_k)$ means that $N_1, \ldots, N_k$ are executed in series resp. in parallel.

We measure the size of an SPN by the number of tasks contained in it. For example, $\mathrm{series}\,(\mathrm{parall}\,(\Box, \Box, \Box), \Box)$ is an SPN of size 4.

Without loss of generality, we may confine ourselves to SPNs where the operations "series" and "parall" always have exactly *two* operands. For example, the SPN $\mathrm{series}\,(N_1, \ldots, N_4)$ may also be represented as

$$\mathrm{series}\,(N_1, \mathrm{series}\,(N_2, \mathrm{series}\,(N_3, N_4))).$$

(Clearly, this representation is not unique.)

An SPN with this restriction may be represented in an obvious way by a binary tree, where the operations "series" and "parall" are internal nodes, and the tasks are leaves.

Gelenbe's probabilistic model, applied to SPNs of the described "binary" type, is the following:

a) Start with an unspecified symbol $\times$.

b) Replace each $\times$

  - by $\Box$ with probability $1 - w$,
  - by $\mathrm{parall}\,(\times, \times)$ with probability αw,
  - by $\mathrm{series}\,(\times, \times)$ with probability $(1 - \alpha)w$.

  $(0 < w < 1,\ 0 \leq \alpha \leq 1)$.

If $w < 1/2$, $w = 1/2$ and $w > 1/2$, this yields a *subcritical*, *critical* and *supercritical binary splitting process*, respectively (cf. Jagers [4], p. 20). In the subcritical and in the critical case, the resulting SPN is almost surely finite.

α may be interpreted as the *degree of parallelism*.

It can be shown (cf. Kolchin [6], Aldous [1], Gutjahr [3]) that conditioning the family tree of a Galton–Watson branching process to a fixed number m of nodes leads to *uniform distribution* on the family of trees with m nodes (the so–called *combinatorial distribution model*). Since in binary trees with n leaves the total number of nodes is $m = 2n - 1$, we obtain uniform distribution in our case also by conditioning on n leaves. So our distribution model can be described as follows:

- Choose a binary tree with n leaves (each such tree being equally likely),
- assign "parall" or "series" to its internal nodes, independently with probabilities α resp. $1 - \alpha$.

2 Growth of the average total execution time

Now let F_0 denote the distribution of the execution time of a single task, given by its distribution function. First of all, we consider the case where F_0 is the point mass in 1, i. e. we assume deterministic execution time 1 for each task. If one knows the SPN N, then the total execution time τ of N can be computed recursively in the following way:

$$\tau(\square) = 1, \tag{1}$$

$$\tau\,(\mathrm{parall}\,(N_1, N_2)) = \max\,(\tau(N_1), \tau(N_2)), \tag{2}$$

$$\tau\,(\mathrm{series}\,(N_1, N_2)) = \tau(N_1) + \tau(N_2). \tag{3}$$

For a binary tree t, let $\bar{\tau}(t)$ be the expected value of $\tau(N)$ for all SPNs N whose tree structure is t. In other words, $\bar{\tau}(t)$ is obtained by taking the expected value of $\tau(N)$ for all different assignments of "parall" resp. "series" to the internal nodes of t, weighted by their probabilities.

Definition 2.1. We denote by e_n the average value of $\bar{\tau}$ for all binary trees with n leaves. Thus, e_n is the average total execution time of SPNs of size n in our model.

For rational $\alpha = p/q$ (p, q integers), it is possible to compute e_n by means of a recursion: The numbers $a_{n,k}$ of SPNs N of size n with $\tau(N) \le k$ are given recursively by

$$a_{n,k} = \sum_{j=1}^{n-1} \left\{ p a_{j,k} a_{n-j,k} + (q-p) \sum_{l=1}^{k-1} (a_{j,l} - a_{j,l-1}) a_{n-j,k-l} \right\} \qquad (n > 1,\ 1 \le k \le n),$$

$$\begin{aligned} a_{n,0} &= 0 \qquad (n \ge 1), \\ a_{n,k} &= a_{n,n} \qquad (n > 1,\ k > n), \\ a_{1,k} &= 1 \qquad (k \ge 1), \end{aligned}$$

and one easily finds

$$e_n = n - \frac{1}{a_{n,n}} \sum_{k=1}^{n-1} a_{n,k}.$$

Hence the numbers e_n can be computed numerically. For $\alpha = 1/2$, e. g., one obtains:

n	10	20	30	40	50	60	70	80
e_n	4.989	8.598	11.866	14.929	17.852	20.667	23.395	26.051

It would be tempting to derive (e. g. by generating function methods) information on the asymptotic behavior of (e_n) from the recursion above, but this seems to be very difficult. So we follow another approach. We try to "translate" information on $\bar{\tau}$ from the corresponding *unconditioned* branching process (the binary splitting process described in Section 1) to the size–conditioned process.

The same approach can be applied to a far more general class of problems (see Gutjahr [3]). We shall use the following relation, which is proved in [3]:

Lemma 2.1. Let Π be a function assigning to each binary tree t a real value $\Pi(t)$. Let e^{crit} be the expected value of $\Pi(t)$ for family trees t of a critical binary splitting process, and let e_n be the expected value of $\Pi(t)$ for binary trees t with n leaves, where each such tree has the same probability. Then

$$\sum_{n=1}^{\infty} e_n n^{-3/2} < \infty \quad \text{iff} \quad e^{crit} < \infty.$$

We apply this lemma to the function $\Pi = \bar{\tau}$ defined above (such that Definition 2.1 is consistent with the definition of e_n in Lemma 2.1). This makes it possible to derive a kind of asymptotic upper bound for the growth of the numbers e_n. In order to formulate the result, the following definition is useful:

Definition 2.2. The *growth exponent* η of a series (e_n) of real numbers is the number

$$\eta = \inf\{y : e_n = O(n^y)\ (n \to \infty)\}.$$

For the proof of the result, we need a technical lemma concerning the recursion

$$a_{k+1} = a_k(1 - \gamma a_k), \tag{4}$$

which resembles the famous *logistic recursion*

$$a_{k+1} = \gamma a_k(1 - a_k)$$

investigated in biodynamics and chaos theory.

Lemma 2.2. Let $\gamma > 0$. For the numbers a_k defined by (4) and by an initial value a_1 with $0 < a_1 \le 1/(2\gamma)$, the estimation

$$a_k > a_2/k \qquad (k \ge 1)$$

holds.

Proof. Let

$$f(x) = x(1 - \gamma x). \tag{5}$$

$f(x)$ is increasing in the interval $[0,\ 1/(2\gamma)]$. We show by induction $a_k \le 1/(k\gamma)$ $(k \ge 2)$. The case $k = 2$ is clear. For $k \ge 2$, one finds

$$a_{k+1} = f(a_k) \le f\left(\frac{1}{k\gamma}\right) \le \frac{1}{(k+1)\gamma}.$$

Next, we show by induction $a_k \geq a_2/(k-1) \quad (k \geq 2)$. $k = 2$ is clear again. For $k \geq 2$,

$$a_{k+1} \geq a_k(1 - \gamma/(k\gamma)) \geq a_2/k = a_2/((k+1)-1).$$

Hence

$$a_k > a_2/k \quad (k \geq 2),$$

and evidently $a_1 > a_2$. ∎

Theorem 2.1. For each degree of parallelism $\alpha < 1$, the average total execution times e_n given by Definition 2.1 have a growth exponent $\eta \geq 1/2$.

Proof. The case $\alpha = 0$ is clear, so assume $0 < \alpha < 1$. Because of Lemma 2.1, it is sufficient to show that $e^{crit} = \infty$, since $\sum e_n n^{-3/2} = \infty$ implies that $e_n = O(n^y)$ cannot hold for an exponent $y < 1/2$. Consider the critical binary splitting process. The start symbol $\times$ is replaced by the symbol $\Box$ with probability $1/2$, by parall $(\times, \times)$ with probability $\alpha/2$, and by series $(\times, \times)$ with probability $(1-\alpha)/2$. If the start symbol is "split", i. e. not replaced by $\Box$, then the branching processes starting with the left resp. the right successor of the root are identical in distribution to the original binary splitting process. This yields the following recursion for the distribution function of the total execution time:

$$F(k) = \frac{1}{2} + \frac{\alpha}{2}\, F(k)^2 + \frac{1-\alpha}{2}\,(F * F)\,(k) \quad (k \geq 1), \qquad F(0) = 0, \tag{6}$$

where $*$ denotes the convolution of distribution functions. Since $\tau(N) \geq 1$, one has

$$\tau(N_1) + \tau(N_2) \geq \max\,(\tau(N_1), \tau(N_2)) + 1.$$

It will be shown that we get infinite average total execution time even if, in the definition of τ, we replace (3) by

$$\tau\,(\text{series}\,(N_1, N_2)) = \max(\tau(N_1), \tau(N_2)) + 1. \tag{7}$$

By this replacement, (6) turns into

$$F(k) = \frac{1}{2} + \frac{\alpha}{2}\, F(k)^2 + \frac{1-\alpha}{2}\, F(k-1)^2 \quad (k \geq 1), \qquad F(0) = 0. \tag{8}$$

Solution of the quadratic equation yields

$$F(k) = \frac{1}{\alpha}\left(1 - \sqrt{1-\alpha}\cdot\sqrt{1 - \alpha\, F(k-1)^2}\right). \tag{9}$$

With $g_k = 1 - F(k)$, the expected value of a random variable with distribution F is finite iff $\sum g_k < \infty$. In terms of g_k, (9) reads

$$g_{k+1} = \beta\left[\sqrt{1 + (2g_k - g_k^2)\,/\,\beta} - 1\right], \tag{10}$$

where $\beta = (1-\alpha)/\alpha \quad (0 < \beta < \infty)$.

It is not difficult to show that the solution F of the functional equation (8) exists and that F is a distribution function. Hence $g_k \downarrow 0 \quad (k \to \infty)$. Expansion of the square root in (10) and some straightforward lower bound estimations lead to

$$g_{k+1} \geq g_k(1 - \gamma g_k), \tag{11}$$

where $\gamma = (\beta+1)/(2\beta)$.

Because of $g_k \downarrow 0$, there is an m such that $g_{m+1} \le 1/(2\gamma)$. Let (a_k) be the numbers defined by (4) with $a_1 = g_{m+1}$. We show by induction that $g_{m+l} \ge a_l$ $(l \ge 1)$. With f given by (5),

$$g_{m+l+1} \ge g_{m+l}\,(1-\gamma g_{m+l}) = f(g_{m+l}).$$

Because of $g_{m+l} \le g_{m+1} \le 1/(2\gamma)$ and of f increasing in $[0,\ 1/(2\gamma)]$,

$$f(g_{m+l}) \ge f(a_l) = a_{l+1}.$$

So $g_{m+l} \ge a_l$ has been shown for all l.

Lemma 2.2 yields

$$a_l > a_2/l = c/l$$

with $c = f(g_{m+1})$. Thus also

$$g_{m+l} \ge c/l,$$

and hence $\sum g_k$ diverges. ∎

Now we return to the more general case of an arbitrary execution time distribution for the single tasks.

Corollary. Let the distribution F_0 of the execution time of a primitive task be arbitrary, except the point mass in zero, and $\alpha < 1$. Then Theorem 2.1 still holds.

Proof. Since F_0 is not the point mass in 0, there is an $\epsilon > 0$ such that $F_0(\epsilon) < 1$. Without loss of generality one may assume $\epsilon = 1$ (otherwise use another unit of time). Then if we replace F_0 by the Bernoulli distribution with $p = 1 - F_0(1)$, all average execution times will decrease (or at least remain constant). In this case we may use the "best case estimation"

$$\tau\,(\text{series}\,(N_1,N_2)) = \begin{cases} \max(\tau(N_1),\tau(N_2))+1 & \text{if } \tau(N_1)>0, \tau(N_2)>0, \\ \max(\tau(N_1),\tau(N_2)) & \text{otherwise} \end{cases}$$

instead of (7). This leads to

$$F(k) = \frac{1}{2} + \frac{\alpha}{2}\,F(k)^2 + \frac{1-\alpha}{2}\,\left\{(1-F(0))^2\cdot F(k-1)^2 + [1-(1-F(0))^2]\cdot F(k)^2\right\} \qquad (k \ge 1)$$

and

$$F(0) = \frac{1}{2}(1-p) + \frac{1}{2}F(0)^2.$$

Hence

$$F(k) = \frac{1}{2} + \frac{\tilde\alpha}{2}\,F(k)^2 + \frac{1-\tilde\alpha}{2}\,F(k-1)^2 \quad (k \ge 1), \qquad F(0) = 1-\sqrt{p},$$

with $\tilde\alpha = \alpha + (1-\alpha)(1-p)$. As in the proof of Theorem 2.1 one shows that F has infinite expectation. ∎

Without proof we remark that in the case $\alpha = 1$ (pure parallelism), the growth exponent is *not* always $\ge 1/2$; it turns out that for $\alpha = 1$ the numbers e_n have a growth exponent $\eta \ge 1/2$ iff

$$\int_0^\infty \sqrt{1-F_0(t)}\,dt = \infty.$$

3 Conclusion

Theorem 2.1 and its corollary yield only rough information on the growth of the average total execution times. Nevertheless, it seems remarkable that the result holds for arbitrary primitive execution time distribution F_0 and for any degree of parallelism $\alpha < 1$. Informally speaking, we have shown that the average total execution times of series–parallel processing structures cannot grow essentially slower than $n^{1/2}$, where n is the number of primitive tasks contained in the processing structure. In the purely sequential case ($\alpha = 0$), the average total execution time is of order n. So our result states that in the indicated situation a speed–up of an order larger than $n^{1/2}$ cannot be expected — even if an unlimited number of processors is available, no memory conflicts occur, the communication overhead is negligible and the degree of parallelism is high.

As to the mathematical technique of our investigation, we used a branching process approach which can also be applied to other recursively defined structures under the combinatorial distribution model.

References

[1] D. J. Aldous, The continuum random tree II: an overview, *Proc. Durham Symp. Stochastic Analysis 1990*, Cambridge University Press (1991), 23 – 70.

[2] E. Gelenbe, *Multiprocessor Performance*, Wiley (1989).

[3] W. J. Gutjahr, Expectation transfer between branching processes and random trees, accepted for publication in: *Random Structures and Algorithms*.

[4] P. Jagers, *Branching Processes with Biological Applications*, Wiley (1975).

[5] R. Kemp, *Fundamentals of the Average Case Analysis of Particular Algorithms*, Wiley–Teubner Series in Computer Science (1984).

[6] V. F. Kolchin, *Random Mappings*, Optimization Software, New York (1986).

[7] P. A. MacMahon, The combination of resistances, *The Electrician* 28 (1892), 601 – 602.

[8] J. W. Moon, Some enumerative results on series–parallel networks, *Annals of Discrete Mathematics* 33 (1987), 199 – 226.

[9] N. Yazici-Pekergin and J.-M. Vincent, Stochastic bounds on execution times of parallel programs, *IEEE Trans. Software Eng.* 17 (1991), 1005 – 1012.