Job scheduling on a parallel shared memory bus computer

Dedicated to Cor Baayen, with esteem and admiration

H.J.J. te Riele

Centrum voor Wiskunde en Informatica, Kruislaan 413,
1098 SJ Amsterdam, The Netherlands.

The advent of vector computers in the beginning of the eighties, and of parallel computers a few years later has triggered the development of new algorithms, especially tailored to these new architectures. One of the many initiatives of Cor Baayen was the stimulation of research activities in this new field and the provision of the necessary equipment, both at CWI and at SARA.
In this paper we study a problem which is typical for these new developments, namely the scheduling of jobs on *bus*–type parallel computers. The processing elements of such computers communicate with a common memory through channels called buses. Usually, there are less buses than processing elements, so that several processing elements have to share the same bus. It is a consequence of this restriction, as we show in this paper, that the *total processing time* of a parallel job may depend on the *order of execution* of the communication parts of the different subjobs. Unfortunately, this order of execution can *not*, in general, be influenced by the programmer. Therefore, this phenomenon must be accepted as an inherent uncertainty in the timing and reproducibility of jobs on parallel bus–type computers.

AMS Subject Classification (1991): Primary 69C12; Secondary 69D51
CR Subject Classification (1991): B.4.3, C.1.2
Keywords & Phrases: Bus–type parallel computers

1 Introduction

Consider a parallel bus – type computer with a shared memory having $p * b$ processing elements (PEs), where b is the number of buses and p is the number of PEs per bus; see Figure 1.
Apart from the main shared memory, each PE has its own small local memory, called *cache*. It is important to re-use cache data as much as possible, in order to minimize transport of data between the cache and the main memory. Processing elements which share the same bus cannot send/receive data

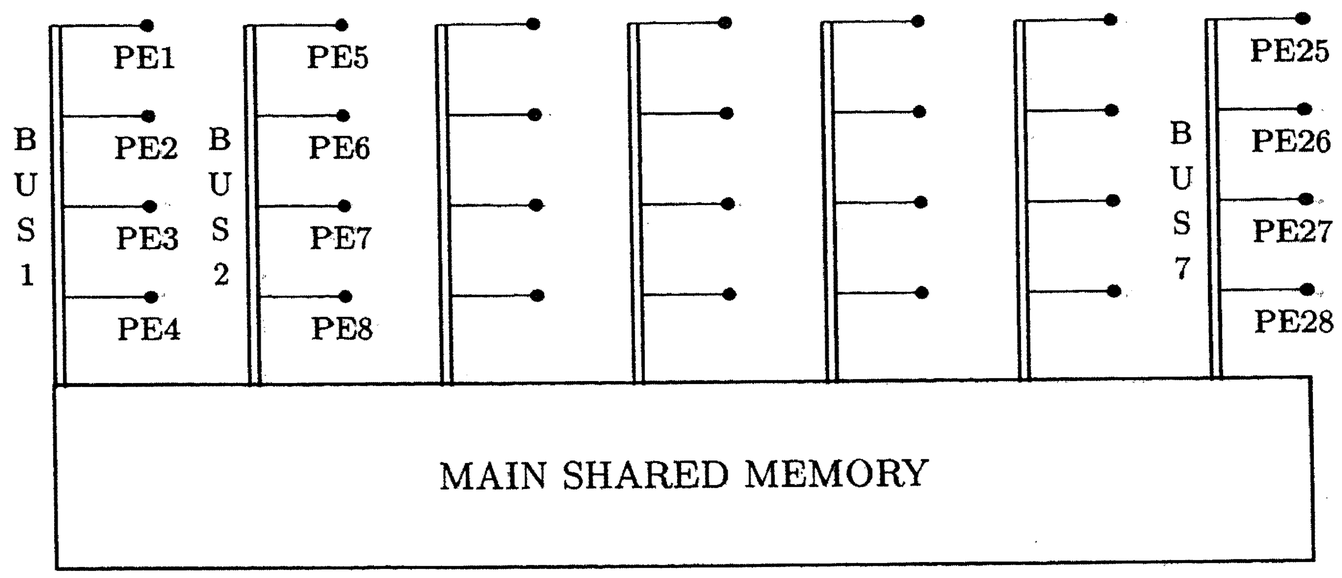
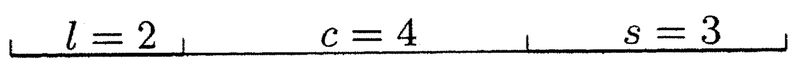


FIGURE 1. Parallel bus – type shared memory computer, $b = 7$, $p = 4$

to/from the main memory simultaneously. CWI has at least two computers of this kind, viz., the Cray S-MP ($b = 7$, $p = 4$; each PE has a data cache of 8 Kbytes; the size of the shared memory is 256 Mbytes) and the SGI Challenge ($b = 1$, $p = 4$; each PE has two data caches: a primary cache of 16 Kbytes, and a secondary (slightly slower) cache of 1 Mbytes; the shared memory has a size of 256 Mbytes).

We make the simplifying assumption that a job for our parallel shared memory bus computer can be split up in S equal subjobs. Not many real-life application jobs satisfy this condition, but basic building blocks like matrix-vector multiplication do. Each subjob consists of one part where data are *loaded* from the shared memory into the cache, a second part where *computations* are done with these data, and a final part where the results are *stored* from the cache into the shared memory. The times (in seconds) for these three parts are denoted by l, c, and s, respectively. Schematically, we will represent a subjob as



where the lengths of the line segments have the ratio $l : c : s$. Communication parts (l and s) are marked by thick lines.

The total time T of a job depends on b, p, S, l, c, and s, so

$$T = T(b, p, S, l, c, s).$$

By $\underline{T}(b, p, S, l, c, s)$ we will denote the *minimal* time needed to complete the job. In general, it is easy to give upper and lower bounds for T. For example,

$$p\,(l + s) \leq T(1, p, p, l, c, s) \leq p\,(l + s) + c.$$

The lower bound just counts all the communication times (neglecting the computing times), and for the upper bound we assume that after the last PE has

loaded its data, the computing parts of all the subjobs are carried out simultaneously.

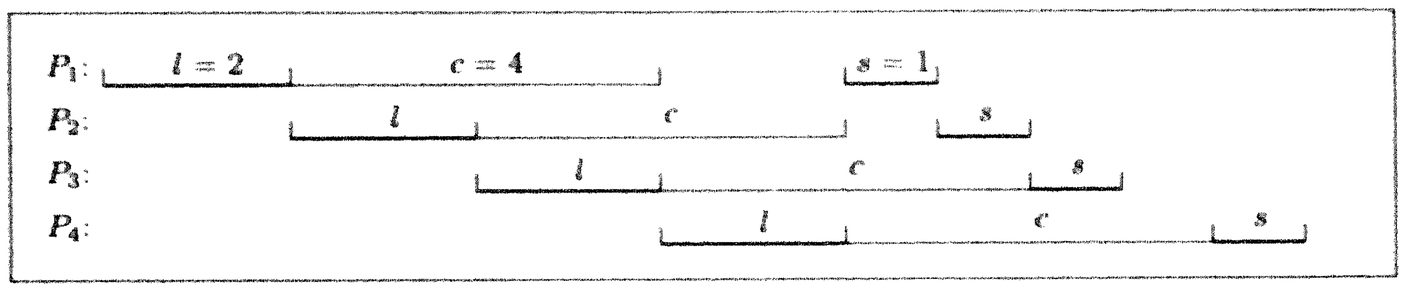


FIGURE 2. Job schedule with $T(1, 4, 4, 2, 4, 1) = 13$

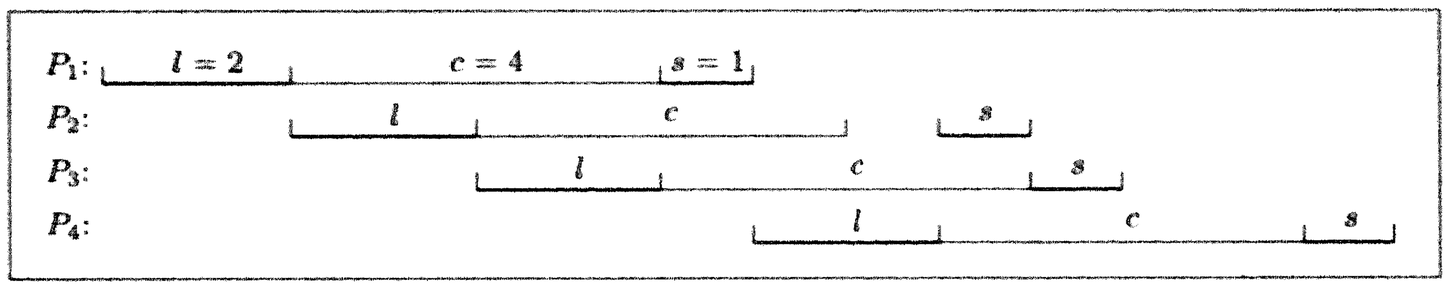


FIGURE 3. Job schedule with $T(1, 4, 4, 2, 4, 1) = 14$

Figures 2 and 3 illustrate for the case $b = 1$, $p = 4$, $S = 4$, $l = 2$, $c = 4$, $s = 1$, that T may depend on the *order* by which the different PEs execute their communicating parts. In the schedule of Figure 2, processing element P_1 only starts with storing the data into the shared memory after all the PEs have loaded their data. In the schedule of Figure 3 processing element P_1 starts with storing the data as soon as it has completed its computing part (and the bus channel is free). Consequently, we find $T = 13$ and $T = 14$, respectively.

In this paper, we shall analyze the case $b = 1$ in Section 2, and partly generalize this in Section 3. We present theorems which give the minimum times needed to execute a job on a bus-type parallel computer, under the assumption that the total job can be split up into a number of equal subjobs. Proofs will appear elsewhere, but no doubt the reader will be able to construct some of them without too much effort.

2 THE CASE $b = 1$

We start by assuming $S = p$, and give three examples with $p = S = 4$, viz., $l = 1$, $c = 2$, $s = 2$ (Figure 4), $l = 2$, $c = 2$, $s = 1$ (Figure 5), and $l = 1$, $c = 4$, $s = 2$ (Figure 6).

These examples suggest that in some cases $\underline{T}(b, p, S, l, c, s) = \underline{T}(b, p, S, s, c, l)$, i.e., that the time T remains fixed, if we interchange l and s. Define

$$\underline{m} = \min(l, s), \quad \text{and} \quad \overline{m} = \max(l, s),$$

then we have the following

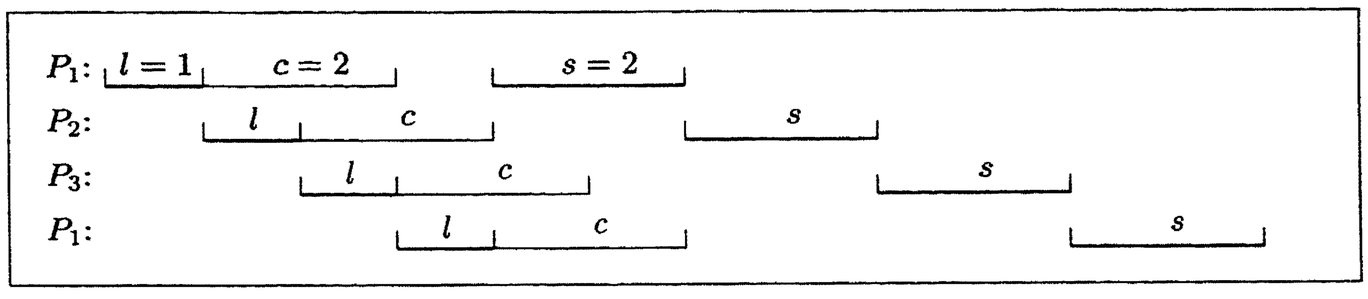


FIGURE 4. Job schedule with $\underline{T}(1,4,4,1,2,2) = 4(l+s) = 12$

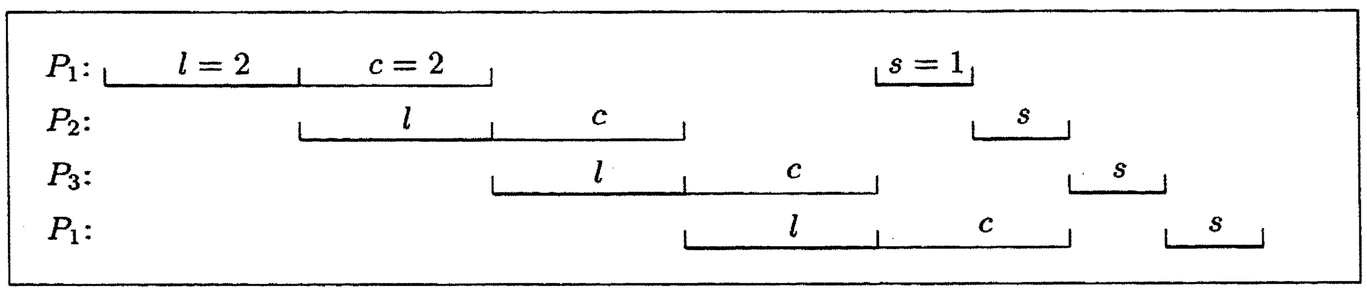


FIGURE 5. Job schedule with $\underline{T}(1,4,4,2,2,1) = 4(l+s) = 12$

THEOREM 1 *Let* $b = 1$ *and* $S = p$;
i. *if* $c \leq (p-1)\underline{m}$, *then* $\underline{T} = p\,(\underline{m} + \overline{m}) = p\,(l+s)$;
ii. *if* $c > (p-1)\underline{m}$, *then* $\underline{T} = \underline{m} + c + p\overline{m}$.

Figures 4 and 5 correspond to Theorem 1.i and Figures 2 and 6 correspond to Theorem 1.ii.

The next case we consider is $S = k * p$ for some integer $k \geq 2$. In that case, each processing element will execute k subjobs. One possibility is that the schedule of Theorem 1 is just repeated k times, so that the total time is: $kp\,(\underline{m} + \overline{m})$ if $c \leq (p-1)\underline{m}$, and $k(\underline{m} + c + p\overline{m})$ if $c > (p-1)\underline{m}$. However, it turns out to be more efficient in general, if a PE continues with the loading part of the next subjob, as soon as the storage part of its previous subjob has been finished. This concentrates the communication parts of the work done by one PE, and therefore gives more freedom to carry out the computing parts in between them. An example with $b = 1, p = 4, k = 2 (S = 8), l = 1, c = 4, s = 2$ is given in Figure 7. Counting from the end of the job back to the beginning we find that

$$T(1,4,8,1,4,2) = 4s + 4(l+s) + c + l = 25$$

(vs. $T = 26$ if we repeat the schedule of Theorem 1.ii two times). Notice that if we fix the schedule of the load and storage parts, the first computing task

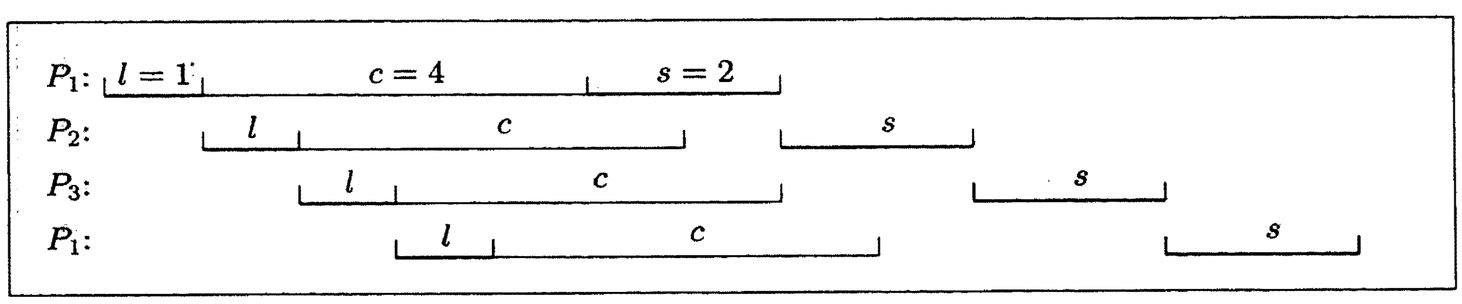


FIGURE 6. Job schedule with $\underline{T}(1,4,4,1,4,2) = l + c + 4s = 13$

of processing elements P_2, P_3, and P_4 could have been scheduled somewhat later, and the second computing task of *all* the four PEs could also have been scheduled somewhat later, without effect on the total computing time T.

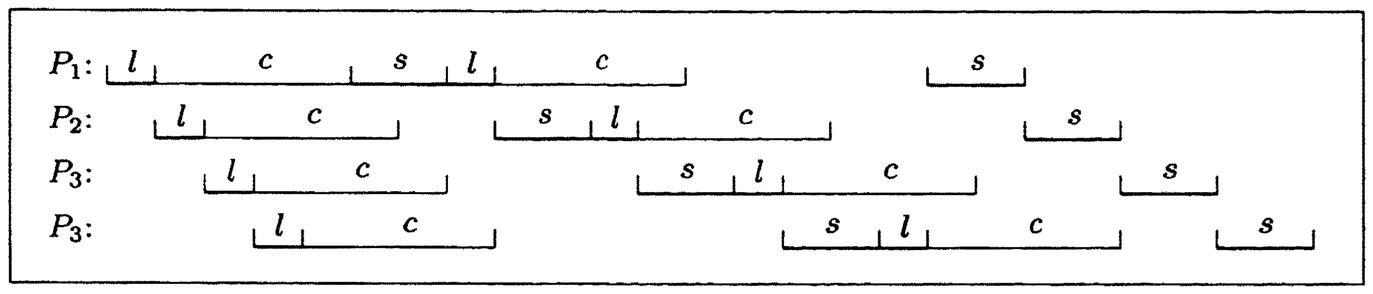


FIGURE 7. Job schedule with $T(1, 4, 2 * 4, 1, 4, 2) = 25$

We have the following

THEOREM 2 *Let $b = 1$ and $S = k * p$ for some integer $k \geq 2$;*
i. *if $c \leq (p-1)\underline{m}$, then $\underline{T} = kp\,(\underline{m} + \overline{m})$;*
ii. *if $(p-1)\underline{m} < c \leq (p-1)\overline{m}$, then $\underline{T} = c + \underline{m} + p\overline{m} + (k-1)p\,(\underline{m} + \overline{m})$;*
iii. *if $(p-1)\overline{m} < c \leq (p-1)(\underline{m} + \overline{m})$, then $\underline{T} = 2c + \underline{m} + \overline{m} + (k-1)p\,(\underline{m} + \overline{m})$;*
iv. *if $(p-1)(\underline{m} + \overline{m}) < c$, then $\underline{T} = k(c + \underline{m} + \overline{m}) + (p-1)(\underline{m} + \overline{m})$.*

Figure 7 corresponds to Theorem 2.ii: we find

$$\underline{T}(1, 4, 8, 1, 4, 2) = 4 + 1 + 4.2 + (2-1)4(1+2) = 25,$$

so the schedule of Figure 7 yields the minimal time. To further illustrate this theorem, we consider case iv., and compare its time with that obtained by just repeating Theorem 1.ii k times. We find, assuming that $(p-1)(\underline{m} + \overline{m}) < c$,

$$\frac{\underline{T}_{\text{Thm2.iv}}}{\underline{T}_{\text{Thm1.ii}}} = \frac{k(c + \underline{m} + \overline{m}) + (p-1)(\underline{m} + \overline{m})}{k(c + \underline{m} + p\overline{m})} \to \frac{c + \underline{m} + \overline{m}}{c + \underline{m} + p\overline{m}}, \quad \text{as} \quad k \to \infty.$$

For example, for $p = 4$, $c = 20$, $l = 1$, $s = 2$ this gives

$$\frac{\underline{T}_{\text{Thm2.iv}}}{\underline{T}_{\text{Thm2.iv}}} = \frac{23k + 9}{29k} \to \frac{23}{29} = 0.793, \quad \text{as} \quad k \to \infty.$$

Now we study, for another example, how the total time T depends on c, if the other parameters are kept fixed. Assume $b = 1$, $p = 4$, $k = 10$ $(S = 40)$, $\underline{m} = 1$, $\overline{m} = 2$. If we simply repeat Theorem 1 ten times, we find that $T = 120$ if $c \leq 3$ and $T = 10c + 90$ if $c > 3$. Theorem 2 gives the *minimum* times, with $\underline{T} = 120$ for $c \leq 3$, $\underline{T} = c + 117$ for $3 < c \leq 6$, $\underline{T} = 2c + 111$ for $6 < c \leq 9$, and $\underline{T} = 10c + 39$ for $c > 9$. This is represented graphically in Figure 8. It follows that an efficiency–loss of nearly 40% is possible (for $c = 9$ we have a worst/best times ratio of $180/129 \approx 1.40$).

3 THE GENERAL CASE

For the general case for b we present two theorems. In the next theorem, we assume that subjobs on different buses update different parts of the main

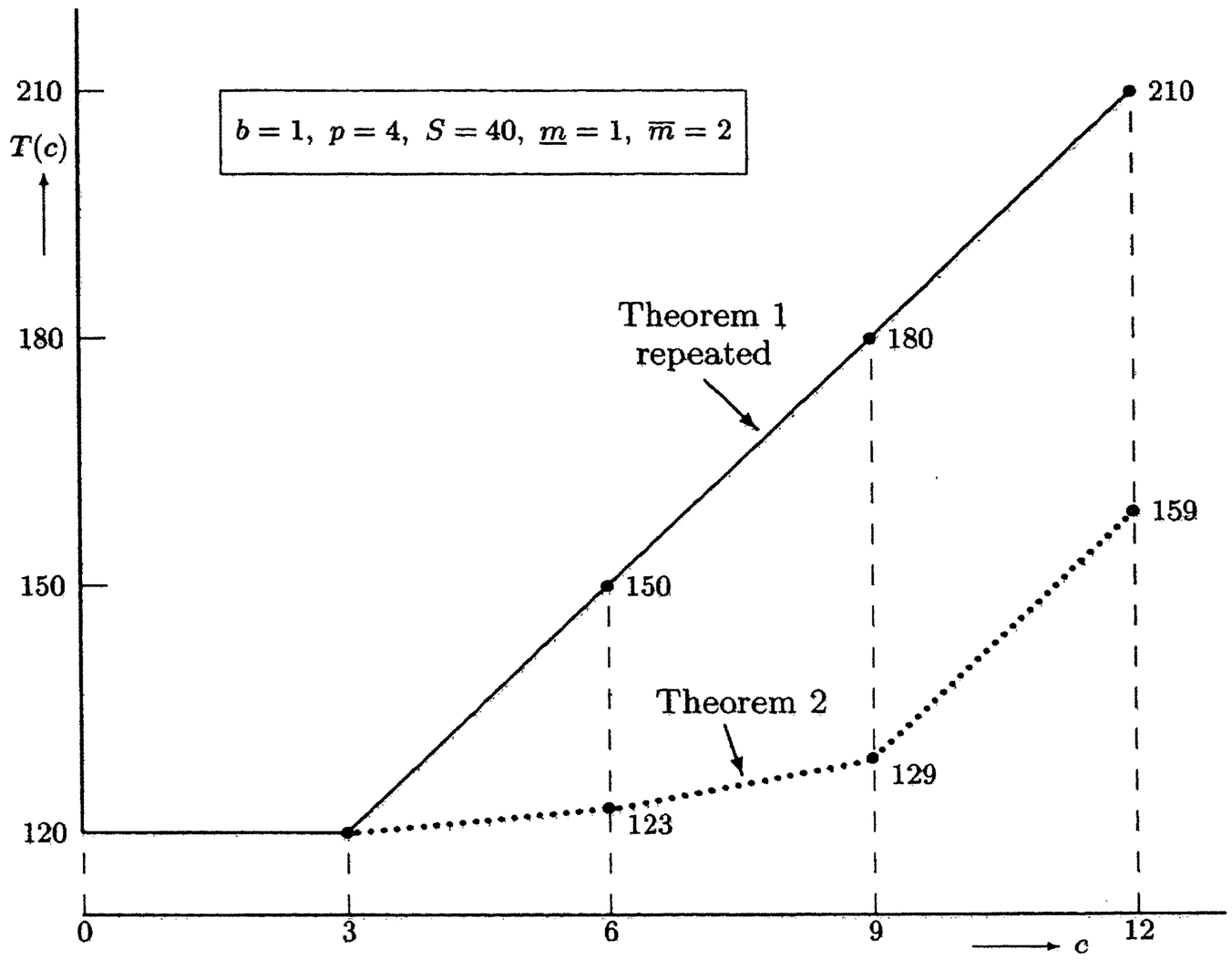


FIGURE 8. Total time $T(c)$ obtained with Theorem 1 (drawn line) and Theorem 2 (dotted line)

memory, so they can update the main memory at the same time. Moreover, we restrict ourselves to the condition of case iv. in Theorem 2.

THEOREM 3 *Let* $(p-1)(\underline{m}+\overline{m}) < c$, $S = k_1 pb + k_2$ *with* $0 \le k_2 < pb$, *and* $k_2 = k_3 b + k_4$ *with* $0 \le k_4 < b$; *assume that subjobs on different buses update different parts of the main memory.*
i. *If* $k_2 = 0$, *then* $\underline{T} = k_1(c + \underline{m} + \overline{m}) + (p-1)(\underline{m} + \overline{m})$;
ii. *if* $k_2 \neq 0$ *and* $k_4 = 0$, *then* $\underline{T} = (k_1+1)(c + \underline{m} + \overline{m}) + (k_3 - 1)(\underline{m} + \overline{m})$;
iii. *if* $k_2 \neq 0$ *and* $k_4 \neq 0$, *then* $\underline{T} = (k_1+1)(c + \underline{m} + \overline{m}) + k_3(\underline{m} + \overline{m})$.

In our final theorem we assume that subjobs on different buses *not necessarily* update different parts of the main memory; this means that if one PE is updating the main memory, all the others can not (neither those connected to the same bus, nor those connected to other buses).

THEOREM 4 *Let* $S = p*b$, *so we have precisely one subjob for each PE; assume that if one PE updates the main memory, the others can not.*
i. *If* $c \le (p-1)\underline{m}$, *then* $\underline{T} = pl + pbs$;
ii. *if* $c > (p-1)\underline{m}$ *and* $l \le s$, *then* $\underline{T} = l + c + pbs$;
iii. *if* $c > (p-1)\underline{m}$ *and* $l > s$, *then* $\underline{T} = l + c + pbs + (p-1)(l-s)$.

4 Conclusion

We have shown that the order of execution of communication parts of subjobs on a parallel shared memory bus–type computer can influence the total processing time of a parallel job unfavourably. Since, in general, the programmer can *not* influence this order of execution, this phenomenon must be accepted as an inherent uncertainty in parallel processing. Examples illustrate that an efficiency-loss of 40% is not uncommon.