1988

On the Effects of Synchronization in Parallel Computing

Dan C. Marinescu

John R. Rice
Purdue University, jrr@cs.purdue.edu

Report Number:
88-750

ON THE EFFECTS OF SYNCHRONIZATION IN PARALLEL COMPUTING

Dan C. Marinescu
John R. Rice

CSD TR-750
March 1988

ON THE EFFECTS OF SYNCHRONIZATION IN PARALLEL COMPUTING

*Dan C. Marinescu**
*John R. Rice**
Computer Science Department
Purdue University

CSD-TR-750
March 15, 1988

Abstract

We present a non-deterministic model of parallel computation that includes the effects of communication costs, computation control costs and synchronization effects. Synchronization may be the most important effect in many important applications. Our model is particularly suited for coarse grain parallelism, as in Same Program Multiple Data (SPMD) computations. Using this model we derive exact expressions for synchronization costs, where the parallel tasks have execution times that are uniformly or exponentially distributed. We show that efficient massive parallelism is possible with the uniform distribution, but the synchronization costs for exponentially distributed execution times lead to a logarithmically decaying efficiency.

1. Introduction

As the number of multiprocessor systems increases, practical questions related to their efficient use as well as questions of how to design new systems capable of high performance stimulate performance related studies in parallel processing. Studies related to problem partitioning, [2], [3], [8], [12], [13], [15] investigate methods for decomposing a large application into smaller subproblems, and for allocation of such subproblems to processing elements of a multiprocessor system. Mapping of known applications to existing multiprocessor systems has produced experimental results [2], [7], [13] used to validate models of parallel execution. System designers use the models of a parallel execution and the empirical data for comparative analysis of different architectural options for highly concurrent multiprocessor systems [5], [10], [16], [19].

* Work supported in part by the Strategic Defense Initiative under Army Research Office contract DAAL03-86-K-0106.

It is generally recognized that accurate models of parallel execution need to describe the complex interactions among concurrent activities and to estimate their corresponding overhead which limits the actual performance of parallel system executing a given application.

The inefficiency associated with parallel execution usually is attributed to two causes: communication among the tasks executed concurrently, [2], [3], [6], [8], and control of parallel activities including scheduling of tasks [2], [9], [14]. Another potential source of inefficiency which has received less attention is the synchronization among concurrent tasks. In case of synchronized execution, it is required that all computations running concurrently complete, before the next set of activities can proceed. Intuitively we expect that synchronization leads to an increase in the execution time, hence any model ignoring synchronization is an optimistic one.

In this paper, we present a non-deterministic model for parallel computation and provide some answers to questions about to the cost of synchronization. We show that in the general case the overhead associated with synchronization depends upon two factors, namely, the number of PEs running in parallel, and the actual distribution of the execution time on PEs. For particular distributions the overhead associated with synchronization is independent of the number of PEs running in parallel when this number is large, hence massive parallelism does not become prohibitively expensive solely due to synchronization. This is the case for the uniform and normal distributions, when only the coefficient of variation of the distribution determines the synchronization overhead. For other distributions, like the exponential one, the synchronization overhead grows logarithmically in the number of PEs. It follows that any modeling technique using Markovian models will overestimate the synchronization overhead if there is not strong empirical evidence that execution times are indeed exponentially distributed. But, due to instrumentation difficulties, studies of empirical distributions of execution times will probably not be available very soon. For this reason we present upper bounds for the synchronization costs which can be computed based upon the mean and the variance of the execution time.

The paper is organized as follows. The next section provides background and motivation for the model. An important class of applications is reviewed where this model is appropriate, others are identified. It involves domain decomposition techniques for physical world simulations in multiprocessor systems with multiple levels of memory. This provides a concrete example for the unified model introduced in the third section. This model takes into account communication, control and synchronization effects in parallel computation. In the last section we develop approximations and bounds for some performance measures including total execution time, speed-up and synchronization.

2. Background and Motivation for the Model

There are cases in parallel processing when synchronization cannot be avoided. For example, in case of domain decomposition techniques for partial differential equations [2], [12], [17] all subcomputations active in a given sub-domain have to exchange

boundary values. In such a case a large problem is partitioned in a set of subproblems and all processing elements (PEs) execute the *same program* on different data (SPMD - Same Program Multiple Data). If the execution times are strictly deterministic and equal, then all PEs would complete at the same time and synchronization would have no effect upon the performance of the system. But a deterministic and equal time model for program execution is unrealistic, since, due to data dependences, different PEs may execute different sequences of instructions. Even when they execute the same sequence of instructions but on different data, their execution times are slightly different. Real applications will have significant variations in the execution times due to data dependencies.

Modeling and analysis of synchronization in parallel computing raises difficult questions. Empirical data are largely unavailable, due to the present state of the art in the instrumentation of parallel systems. Only measurements related to the aggregate program behavior as total execution time, processor utilization, etc., can be carried out routinely, while detailed data concerning communication and synchronization costs are usually unavailable. Theoretical results in the analysis of Fork-Join queues which model synchronized parallel computations are beginning to emerge. Recently Bacelli and his co-workers [1] have shown that for homogeneous Fork-Join systems, under renewal assumptions, the moments of the system response time grow logarithmically in the number of parallel processors. This result is extremely important, since it seems to indicate that the cost of synchronization cannot be ignored in case of massively parallel systems independent of the effort to match as closely as possible the execution times of computations running in parallel.

Among the modeling and analysis techniques used to study the performance of parallel computations on multiprocessor systems, the ones based upon Stochastic Petri Nets (SPNs) have become popular in recent years. Their main advantage seems to be derived from the high descriptive power of Petri net based model description languages and from their two-dimensional syntax suitable for visual representations. The limitation of these techniques for the analysis of multiprocessor systems with a large number of processors is due to the explosion of the state space of the model. A possible way to overcome this limitation is to carry out a structural analysis of the net representing the model, rather than to attempt to identify all possible states of the system and then to use standard queueing methods for the analysis.

The authors have investigated [12], [17] a particular application and a multiprocessor system described in [16] using a class of SPNs, the Stochastic High Level Petri Nets (SHLPNs), which support in a simple way the aggregation of the states of the model, and in the same time provide a convenient support for modeling of synchronized execution. The application was the solution of partial differential equations (PDEs) based upon the Schwartz splitting algorithm using domain decomposition techniques. The results of our analysis show that models of synchronous execution [17], predict a considerable lower average processor utilization than the ones of asynchronous execution [12]. We outline this application to motivate our investigation of the effects of synchronization in parallel computing.

Most physical phenomena (e.g., heat flows, electromagnetic forces, stresses, air flows) are modeled by differential equations in 1 to 4 physical dimensions. These phenomena are inherently local in time and space (only gravity is thought to act instantaneously at a distance). These phenomena are inherently synchronized in time, but loosely synchronized in space; space synchronization comes through the time for effects to propagate through space via local interactions.

Computations modeling these phenomena can exploit this loose coupling in space to achieve parallelism. The principal technique is called *domain decomposition,* where physical space is decomposed into a large number of domains. Since interactions between these domains is local, this allows one to use locally connected computer architectures effectively. See [12], [17] and [18] for previous work of ours, which include descriptions of this approach at a fairly high level. There is an enormous literature on the mathematical analysis of specific instances of this general approach, this is currently one of the most intensively studies areas of numerical computation.

The basic technique is to compute the results in the interior of a particular domain and then communicate the new state to neighboring domains for their use. Important characteristics of such computation are as follows:

1. The interior computations and data are usually large compared to data to be communicated. One of the objectives of algorithm design to be sure this is so, it follows naturally if one chooses domain shapes that have small "surface" compared to "volume" (e.g., nearly spheres or cubes).
2. The interior computations are usually similar (use the same program), but rarely identical (have different data) due to variations in shapes, materials, intensities of physical effects, etc.
3. Synchronization in time is essential. Some models of the physics may compensate for small time asynchronizations locally, and more as domains become separated. Many algorithms have an artificial time (e.g., iteration numbering) which has the same characteristics as real world time.

Thus the real world seems well suited to partitioned or hierarchical parallel machines such as the Butterfly, Cedar [5], hypercubes, Multi-FLEX [13], PASM [16], as well as the shared memory, "homogeneous" machines such as the Elexi, RP3 or Sequent.

We have made a detailed analysis [17] of the effects of synchronization for a particular domain decomposition algorithm running on a particular machine. The effect of synchronization is shown in Figure 1; it is quite substantial, reducing utilization from nearly 100% to 56%. The effect in Figure 1 is entirely that of synchronization. Utilization increases with problem size because the ratio of computation to communication increases. However, this increase does not reduce the effect of synchronization.

Other application areas with similar characteristics include analysis of organizational phenomena, molecular structures and searching schemes.

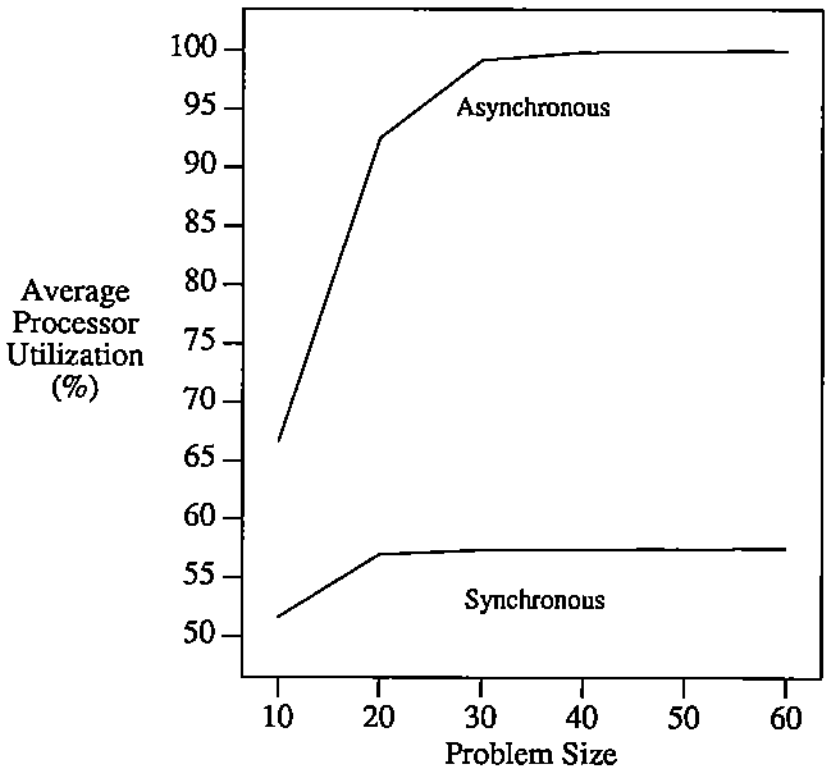

Figure 1. Processor utilization for synchronous and asynchronous execution of a particular domain decomposition algorithm on a 63 processor machine.

3. A Unified Model of Parallel Computation

Let us now consider a computation **C** represented as a graph, each node in the graph corresponds to a dispatchable unit of work (task) and arcs connecting nodes represent communication and synchronization among tasks running concurrently. We assume in the following that **C** exhibits a certain degree of parallelism, namely subsets of tasks can be executed in parallel, and that synchronization conditions are imposed.

Such a graph for a computation **C** is shown in Figure 2. In this figure, we recognize a sequence of n *synchronization epochs*, Π_i with $1 \leq i \leq n$, each consisting of I_i parallel tasks, $t_{i,j}$ with $1 \leq j \leq I_i$. In terms of parallel programming constructs a synchronization epoch corresponds to a *Fork-Join* pair. Each synchronization epoch Π_i is preceded by the execution of a control flow task Φ_i. The computations done by Φ_i are twofold: (1) the algorithm related flow of control computation, and (2) the execution environment related computations setting up the next computations. In case of parallel execution, the second component reflects the scheduling overhead associated with the I_i tasks $t_{i,j}$ running in parallel in Π_i.

Let us now examine the execution environment for computation **C**. We model it by assuming that the parallel system has P processing elements (PEs) and one or more system control elements (SCEs). Each SCE controls a group of PEs. Each processing node

(PE or SCE) consists of a processor, a local memory and possibly an I/O subsystem. This model is illustrated in Figure 2.

For simplicity, we consider a system with one SCE, and P PEs, PE_1, PE_2 ... PE_P. We assume that $I_i \leq P$ for all $1 \leq i \leq n$. The mapping of tasks to processors is straightforward, and the control tasks are executed sequentially by the system control element and the I_i tasks of the synchronization epoch Π_i are assigned such that task $t_{i,j}$ is executed by PE_j. The model of execution is: SCE executes Φ_1 which spawns the computational tasks in Π_1, $t_{1,1}$ to t_{1,I_1}. When all of them complete their execution, Φ_2 is started on the SCE and in turn it spawns $t_{2,1}$ to t_{2,I_2} and so on.

Note the model of the parallel architecture described here is quite general and it describes a wide class of parallel machines. At one extreme are machines with identical processors interconnected some way. The FLEX/32 uses a high speed bus and one processor acts as the SCE; PASM [19] and the Butterfly use a high speed interconnection network; the hypercube machines have a separate SCE and a set local connecting busses. At the other extreme are shared memory machines where the local memory of a processor is merely a cache, and a bus or network connects to the main memory (e.g., Sequent, RP3 and Elexi). Here one processor acts as the SCE. Also modeled well are machines with hierarchical forms of control, such as the Multi-FLEX [16], Cedar [5] or a partitioned PASM machine. The model illustrated in Figure 2 becomes more complex through nesting of the epochs (e.g., task $t_{1,1}$ may itself be represented by computation as shown in Figure 2). That is, several synchronization epochs may be active concurrently on different partitions of a parallel machine. In this case a control task, say Φ_1, is executed on the system controller and determines the partitioning of the P processing elements into q groups with $P_1, P_2, \ldots, P_q$, PEs respectively, such that $\sum_{i=1}^{q} P_i \leq P$. The control tasks $\Phi_{11}, \ldots, \Phi_{q1}$, are then executed on a separate SCE for each group.

The communication model

As noted earlier, several communication models have been discussed in the literature. We base our model on an observation which seems intuitively correct and which is supported both by empirical evidence [7] and consideration of applications [18]; namely, that some form of *locality of communication* is expected as a result of any decomposition.

More specifically, we expect a *temporal locality of communication,* only tasks active in a given synchronization epoch communicate among themselves. In addition, a *spatial locality of communication* exists with two properties:

P1: During the synchronization epoch Π_i with I_i tasks running in parallel, the j-th task $t_{i,j}$ communicates only with the tasks $t_{i,k}$ with k in the range

$$\left\lceil j - W \right\rceil \bmod I_i \leq k \leq \left\lceil j + W \right\rceil \bmod I_i .$$

P2: The number of messages exchanged by $t_{i,j}$ and $t_{i,k}$ is proportional with the distance between the two of them defined as $|j - k|$. Figure 3 presents the number of messages exchanged by a given task with other tasks in its synchronization epoch function of the distance between them.

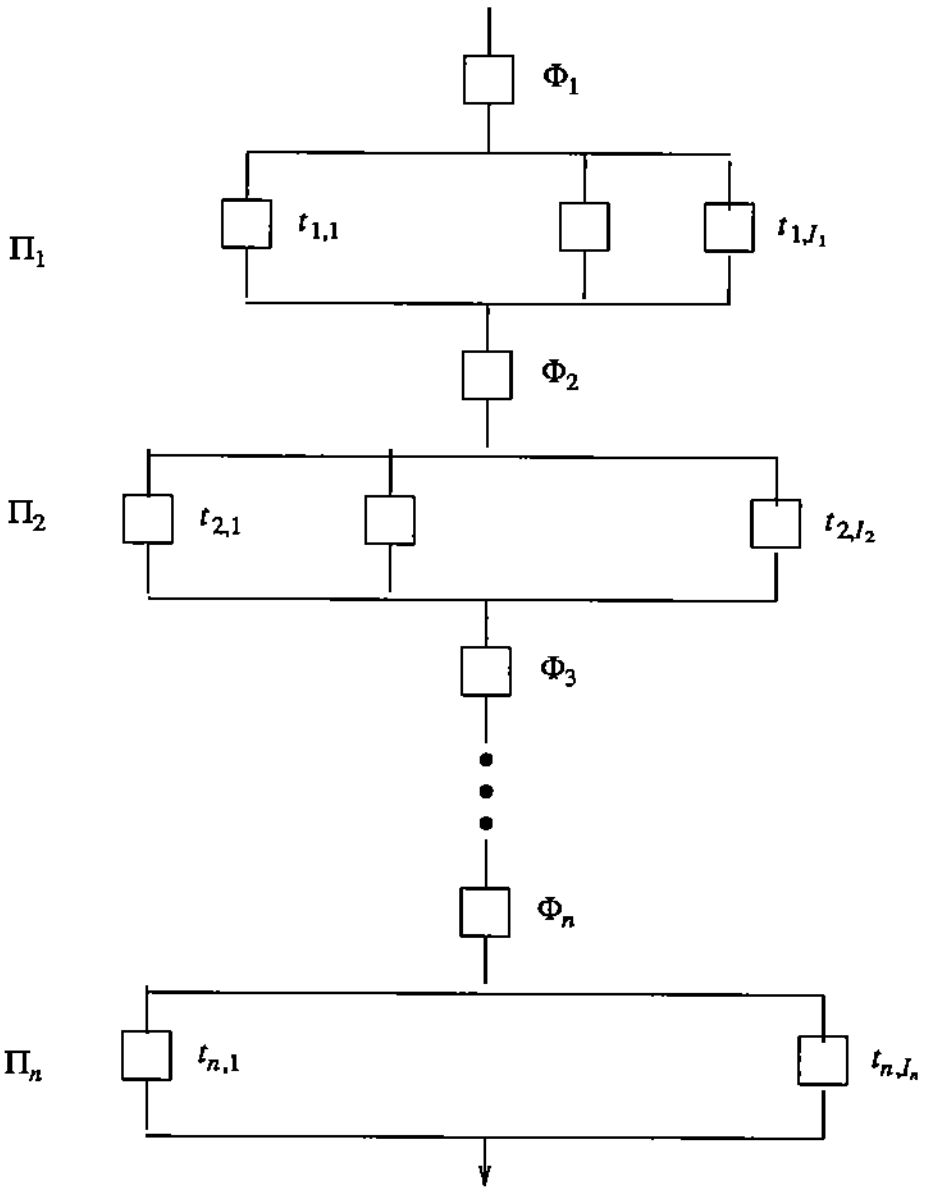

Figure 2. Graphical representations of a computation with sets of parallel subcomputations plus synchronization conditions.

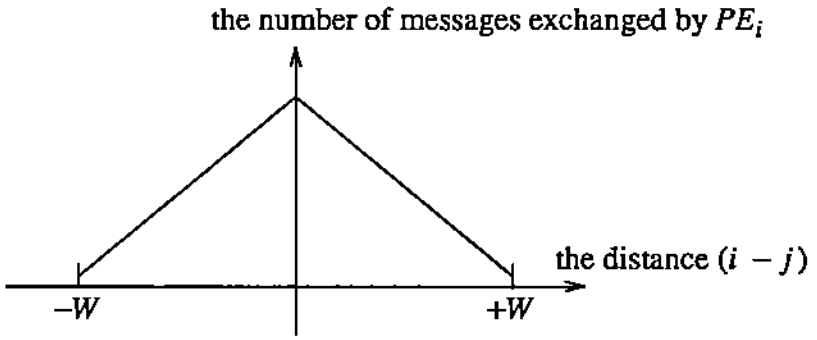

Figure 3. The spatial locality of communication. Any task in a synchronization epoch Π_i communicates only with tasks in a window of size $2W \pmod{I_i}$.

In this case the total number of messages exchanged by task $t_{i,j}$ is

$$N_m = 4 \sum_{i=1}^{+W} (W - i) = 2W(W - 1)$$

with $W \leq I_i$. We assume that the communication between any pair of tasks involves messages of equal length, l and that the total time c_m for transmission of a message is the same for all pairs of communicating tasks.

A good model for c_m is

$$c_m = c_s + l \times c_c$$

with

c_s = the start-up time of the transmission,
c_c = communication time per message of unit length.

Finally, the total time spent by task $t_{i,j}$ for communication is:

$$\beta_{i,j} = N_m \; c_m = 2W(W - 1)(c_s + l \times c_c)$$

In our analysis the communication costs are assumed independent of the task numbering, the synchronization epoch and to have a common value

$$\beta = \beta_{i,j} \qquad 1 \leq i \leq n \;, \qquad 1 \leq j \leq I_j$$

We note that the model presented here is a pessimistic model, it does not take into account possible overlapping of different activities. In particular, the communication between PE_l and PE_k may overlap with execution of tasks $t_{i,j}$ and $t_{i,k}$ on the two PEs. A similar overlapping may exist between the synchronization epoch Π_i and its associated control task Φ_i.

Quantitative model analysis

The qualitative analysis of the parallel execution model discussed in the previous section points out that the expected execution time of a parallel computation C on a system with P processing elements will have the components of control, communication and PE runtime, including synchronization. The expected total execution time T^* may be expressed as

$$T^* = E(T_e) = T_p^{\alpha} + T_p^{\beta} + T_p^{\gamma}$$

with

T_e - a random variable representing the actual execution time of C ,

T^* - the expected value of T_e,

T_p^{α} - the expected execution time attributed to control,

T_p^{β} - the expected execution time attributed to communication,

T_p^{γ} - the expected execution time attributed to PE runtime including synchronization.

By examining the program graph (Figure 2), we can express the total execution time as

$$T_e = \sum_{i=1}^{n} (\pi_i + \phi_i)$$

with

π_i - the duration (a random variable) of the synchronization epoch Π_i, $1 \le i \le n$, consisting of parallel tasks $t_{i,j}$ with $1 \le j \le I_i$. I_i is the number of active tasks in Π_i and $I_i \le P$,

ϕ_i - the execution time (a random variable) of the control task, Φ_i,

n - the number of synchronization epochs.

Based upon the previous discussion, we recognize the execution time ϕ_i is the sum of the time of the algorithmic flow of control, ϕ_i^a, and the execution time related to the architecture dependent flow of control, ϕ_i^c, that is

$$\phi_i = \phi_i^a + \phi_i^c .$$

We will investigate a simple model for the flow of control with two properties, first the total flow of control computation time is independent of the parallelism. Thus we have

$$\sum_{i=1}^{n} \phi_i^a = \alpha_1 \qquad \text{(P1)}$$

with α_1 the execution time required for the flow of control in case of serial execution.

Second, it is assumed that the architecture dependent flow of control associated with each synchronization epoch depends only upon the number of parallel tasks in the epoch

$$\phi_i^c = \alpha_2 I_i \quad \text{for all} \quad i \leq n \ . \tag{P2}$$

With these two assumptions the expected execution time attributed to control becomes

$$T_p^{\alpha} = E\left[\sum_{i=1}^{n} \phi_i\right] \equiv \alpha_1 + \alpha_2 \left[\sum_{i=1}^{n} I_i\right]$$

We model, π_i, the actual duration of a synchronization epoch Π_i, by

$$\pi_i = \underset{1 \leq j \leq I_i}{\text{MAX}} (X_{i,j} + Y_{i,j})$$

with

$X_{i,j}$ - the execution time (a random time) of task $t_{i,j}$ on PE_j,

$Y_{i,j}$ - the communication time (a random time) for task $t_{i,j}$.

This model overestimates the total execution time since it does not take into account the possible overlapping of communication and control activities with execution of computations. For the simple communication model introduced earlier we have

$$Y_{i,j} = \beta \quad \text{for all} \quad i \leq j \leq I_i$$

and hence

$$\pi_i = \beta + \underset{i \leq j \leq I_1}{\text{MAX}} (X_{i,j})$$

We introduce γ_i, the *average duration of a synchronization epoch excluding communication*,

$$\gamma_i = E(\pi_i - \beta) = E\left[\underset{1 \leq j \leq I_1}{\text{MAX}} (X_{i,j})\right]$$

which can be expressed as

$$\gamma_i = \mu_i(1 + \Delta_i)$$

with

μ_i - the expected value of $X_{i,j}$, $1 \leq j \leq I_1$. Note that in case of identical deterministic execution times we have $\gamma_i = \mu_i$ since all tasks running in parallel finish their execution in the same time.

Δ_i - represents the cost of synchronization. In the SPMD case Δ_i is a random variable due to data dependency effects in the computation $t_{i,j}$.

The final expression for the expected execution time, when control, communication and synchronization are taken into account, becomes

$$T^* = E(T_e) = \alpha_1 + \alpha_2 \sum_{i=1}^{n} I_i + n\ \beta + \sum_{i=1}^{n} \mu_i\ (1 + \Delta_i) \tag{3.1}$$

In case of serial execution, the terms corresponding to architecture dependent control, communication and synchronization are zero, that is, $\alpha_2 = 0$, $\beta = 0$ and $\Delta_i = 0$ for all i and the μ_i times are summed in each epoch.

Thus the expected *serial execution time* of C is

$$T_S = \alpha_1 + \sum_{i=1}^{n} \mu_i\ I_i$$

Consequently, the speedup factor for our model of parallel computation becomes

$$S_p = \frac{\alpha_1 + \sum_{i=1}^{n} \mu_i\ I_i}{\left[\alpha_1 + \alpha_2 \sum_{i=1}^{n} I_i\right] + n\ \beta + \sum_{i=1}^{n} \mu_i\ (1 + \Delta_i)} \tag{3.2}$$

When all synchronization epochs Π_i, $1 \le i \le n$ exhibit identical behavior, namely

$$\mu_i = \mu \quad \text{and} \quad \Delta_i = \Delta$$

and when the number of parallel tasks in each Π_i is equal to the number of processing elements P, namely $I_i = P$, for $1 \le i \le n$ then previous expressions for T^*, T_S and S_p become

$$T^* = \alpha_1 + \alpha_2 nP + n\ \beta + n\mu(1 + \Delta),$$

$$T_S = \alpha_1 + nP\mu,$$

$$S_p = \frac{\alpha_1 + nP\mu}{\alpha_1 + \alpha_2 nP + n\ \beta + n\mu(1 + \Delta)}.$$

Using our model, we can compare the costs of synchronization, communication and control related to parallel execution. By rewriting (3.1) as

$$T^* = \alpha_1 + n\ \beta + \sum_{i=1}^{n} (\mu_i + \alpha_2 I_i + \Delta_i)$$

we see that the cost of synchronization in Π_i can be neglected when

$$\mu_i + \alpha_2 I_i >> \Delta_i$$

This corresponds to the case when the average execution time of $t_{i,j}$ is large and when the overhead associated with dispatching of all tasks in Π_i is relatively significant.

A model of parallel computation which takes into account the cost of communication, but ignores synchronization, is accurate only when

$$\beta >> \frac{\sum_{i=1}^{n} \Delta_i}{n}.$$

In conclusion, to construct an accurate model of the parallel execution of a computation on a multiprocessor system it is necessary to estimate the cost of synchronization, and only after comparing it with the overhead associated with communication, and control it is possible to decide which effects can be ignored.

4. Approximations and Bounds For Performance Measures

In this section we examine the effects of synchronization upon the expected execution time and upon processor utilization. From our previous qualitative analysis and experience with domain decomposition algorithms, we know that synchronization can lead to a substantial increase in the execution time of a parallel computation. We now give exact expression and bounds on Δ_i, the quantitative measure of the synchronization cost.

A standard assumption in the analysis [1], [8] is that the random variables $X_{i,j}$ for $1 \le j \le I_i$, representing the execution times of the parallel tasks $t_{i,j}$ are independent and identically distributed. Under these assumptions we are able to use results from [11] to derive exact expressions for Δ_i in the cases of uniform, normal and exponential distributions of the execution times $X_{i,j}$. For other distributions we develop bounds for Δ_i. We then compare these bounds with the actual values for the previous cases in order to illustrate how accurate the bounds are. Finally we relax the independence assumption and derive additional upper bounds on the Δ_i.

We expect Δ_i to be a function of the distribution of $X_{i,j}$ and of the number of tasks running in parallel in Π_i, I_i. A related analysis without synchronization has been given

by Kruskal and Weiss [9] for the simpler case of allocating independent tasks to processors.

Uniform distribution

In [11] we have examined the case when the $X_{i,j}$ are uniformly distributed in $[a_i, b_i]$ with $a_i \leq b_i$ and we have derived the following expression for γ_i, the expected duration of the synchronization epoch Π_i with I_i tasks running in parallel

$$\gamma_i = b_i - (b_i - a_i)\frac{1}{I_i + 1}$$

In this case, the expected value μ_i and its variance σ_i are given by

$$\mu_i = \frac{a_i + b_i}{2}$$

$$\sigma_i^2 = \frac{(b_i - a_i)^2}{12}$$

Hence γ_i can be expressed as:

$$\gamma_i = \mu_i \left[1 + C_i \frac{\sqrt{3}(I_i - 1)}{I_i + 1} \right]$$

with C_i the *coefficient of variation* of the distribution given by

$$C_i = \frac{\sigma_i}{\mu_i} = \frac{(b_i - a_i)}{\sqrt{3}(b_i + a_i)}$$

The expression for Δ_i for the uniform case is

$$\Delta_i = C_i \sqrt{3} \frac{I_i - 1}{I_i + 1} = \frac{(b_i - a_i)(I_i - 1)}{(b_i + a_i)(I_i + 1)} \tag{4.1}$$

For large values of I_i, Δ_i depends only upon the coefficient of variation of the distribution and is

$$\Delta_i \simeq C_i \sqrt{3} = \frac{b_i - a_i}{b_i + a_i}$$

Exponential distribution

In the case of exponential distribution with parameter λ, we have derived in [11]

$$\gamma_i = \frac{1}{\lambda} (\log I_i + C + O(I_i^{-1}))$$

with $C = 0.577$ the Euler's constant. In this case

$$\mu_i = \frac{1}{\lambda} \text{ and } C_i = 1 .$$

Hence γ_i can be expressed as

$$\gamma_i = \mu_i(1 + \Delta_i)$$

with

$$\Delta_i = \log I_i + C - 1 + \mathbf{O}(I_i^{-1}) \tag{4.2}$$

Standard normal distribution

As shown in [9], Δ_i for the standard normal distribution has the value

$$\Delta_i = (2 \cdot \log I_i)^{1/2} - \frac{1}{2} (2 \log I_i)^{-1/2} (\log \log I_i + \log 4\pi - 2c) + \mathbf{O}(\log I_i^{-1}) \tag{4.3}$$

Upper bounds for Δ_i

The distribution functions $F_i(x)$ of the random variables $X_{i,j}$ for $1 \le j \le I_i$ are seldom available. More often the first two moments of the distribution can be estimated. In the followings we present upper bounds for Δ_i in terms of the mean value of the execution time, μ_i and the variance of the execution time σ_i.

A first upper bound is derived under the assumption that $F_{X_i}(x)$ is a continuous, strictly increasing cumulative distribution function. In this case we have

$$\Delta_i \le C_i \frac{I_i - 1}{\sqrt{2I_i - 1}} \tag{4.4}$$

with

C_i - the coefficient of variation of the distribution = σ_i/μ_i,

I_i - the number of parallel tasks in the synchronization epoch π_i.

The derivation follows immediately from the results presented in David [4].

A sharper bound exists for symmetric distributions. In this case

$$\Delta_i \le C_i \frac{1}{2} I_i \left\{ \frac{2\left[1 - 1/\left[\begin{matrix} 2I_i - 2 \\ I_i - 1 \end{matrix} \right] \right]}{2I_i - 1} \right\}^{1/2} \tag{4.5}$$

For large values of I_i, the Stirling approximation can be used to obtain

$$\left[\begin{matrix} 2I_i - 2 \\ I_i - 1 \end{matrix} \right] \approx \frac{1}{\sqrt{2\pi(I_i - 1)}} 2^{2I_i - 3/2}$$

It follows that

$$\Delta_i \le C_i \frac{1}{2} I_i \left\{ \frac{2\left[1 - \sqrt{2\pi(I_i - 1)}\, 2^{-(2I_i - \frac{3}{2})} \right]}{2I_i - 1} \right\} \tag{4.6}$$

The accuracy of these bounds decreases when I_i increases. Table 1 (see also [4]) shows the exact values $\frac{\Delta_i}{C_i}$ for the uniform and the normal case compared with the values obtained using the two bounds presented above.

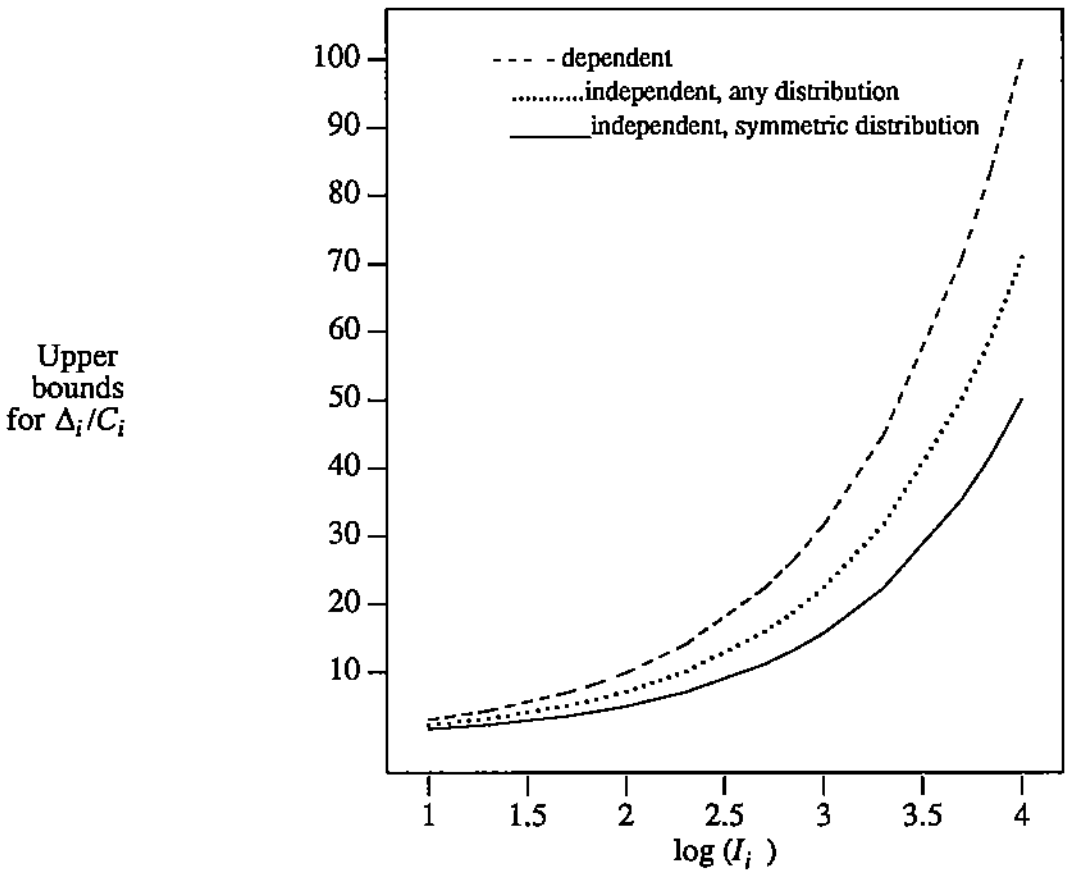

Figure 4. The cost of synchronization as function of the number of processors active in a synchronization epoch. Three upper bounds for Δ_i/C_i are shown: (a) dependent random variables case (4.7), (b) independent random variables with an arbitrary distribution (4.4), and (c) independent random variables with symmetric distribution (4.6). C_i is the coefficient of variation of the distribution.

When $X_{i,j}$, $1 \leq j \leq I_i$ are dependent random variables, then a yet weaker upper bound exists for Δ_i, namely

The dependency of the synchronization costs upon the number of processors active in a synchronization epoch is summarized in Figure 4. The values Δ_i/C_i are plotted versus $\log(I_i)$ for the three bounds presented above.

$$\Delta_i \leq C_i \sqrt{I_i - 1} \tag{4.7}$$

I_i	Uniform Distribution (4.1)	Normal Distribution (4.3)	First Upper Bound (4.4)	Upper Bound for Symmetric Distribution (4.6)
5	1.1547	1.1630	1.3333	1.1701
10	1.4171	1.5388	2.0647	1.6222
20	1.5671	1.8673	3.0424	2.2645
50	1.6641	2.2491	4.9247	3.5533
100	1.6978	2.2491	7.0179	5.0125

Table 1. Comparison of the bounds with the exact values for $\frac{\Delta_i}{C_i}$.

Processor Utilization

The average processor utilization in the the synchronization epoch Π_i with I_i PEs active can be computed using the method described in detail in reference [9]. We present here only the results for two cases, the uniform distribution and the exponential distribution.

The average processor utilization in the case of the uniform distribution is

$$U_i = \frac{I_i + 1}{(I_i + 1) + C_i \sqrt{3}(I_i - 1)}$$

with C_i the coefficient of variation of the distribution of the execution time. From (4.1) it follows that:

$$U_i = \frac{1}{1 + \Delta_i}$$

The results are plotted in Figure 5 for several values of C_i. They support the previous observation that for uniform distribution the synchronization overhead does not prevent massive parallelism and only the coefficient of variation of the distribution is important.

In case of exponential distribution the average processor utilization is given by

$$U_i = \frac{1}{K_i} (1 - e^{-K_i})$$

with

$$K_i = \log (I_i) + C + \mathrm{O}(n^{-1}) = \Delta_i - 1$$

Figure 6 presents the average processor utilization as function of the number of PEs for the exponential distribution. For a large number of PEs the synchronization costs become prohibitive.

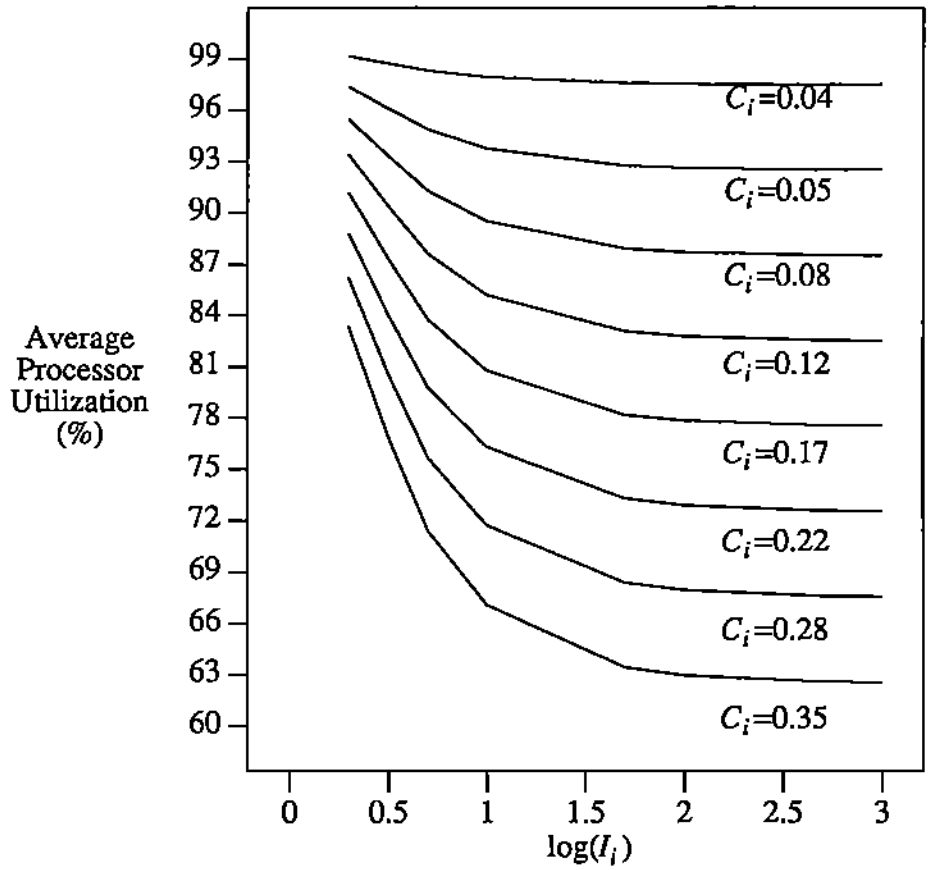

Figure 5. The average utilization of a processor versus the number of processors for uniform distribution of processing time.

A comparison of Figures 5 and 6 shows the impact of the distribution of the execution time upon the actual cost of synchronization reflected in this case by the utilization of the PEs. At one extreme the uniform distribution allows massive parallelism, since by carefully matching the PE execution time inside a Fork-Join pair one can theoretically achieve any level of performance desired, while at the other extreme, such as the case of exponential distribution, the performance depends solely upon the number of PEs running in parallel.

The results of [11] can be used to show that massive parallelism is also efficient for the normal distribution, processor utilization again depends only on the variance of the distribution.

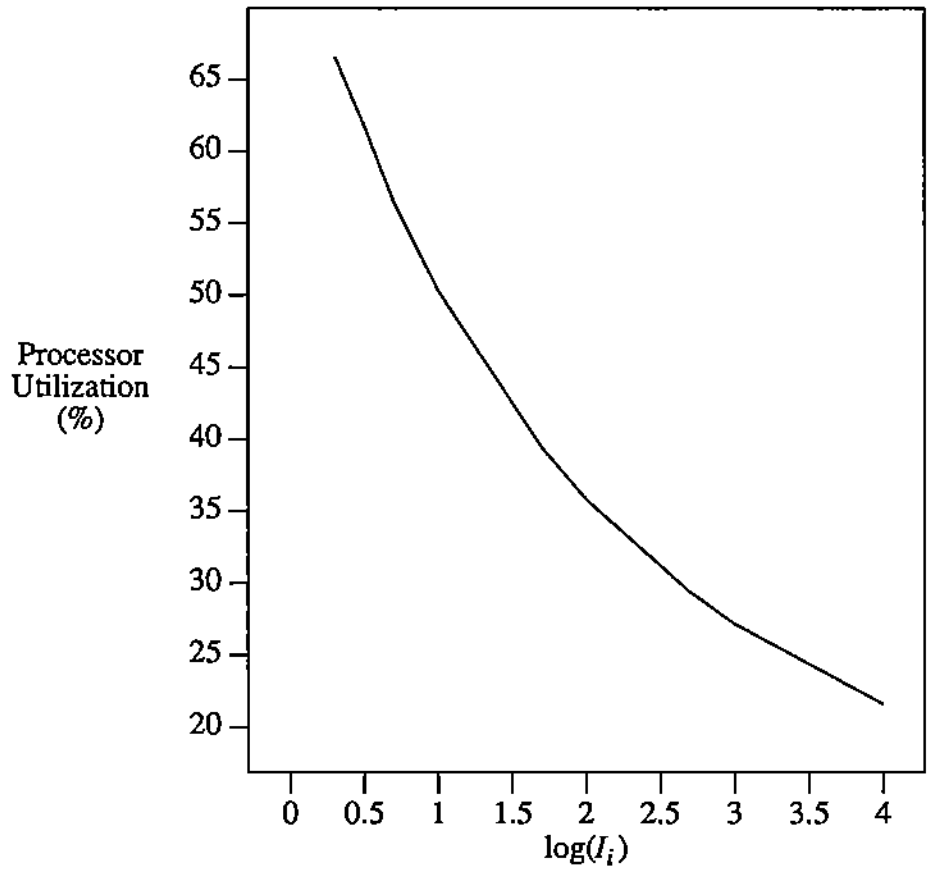

Figure 6. The average utilization of a processor versus the number of processors for the exponential distribution of processing time.

5. Conclusions

Comparisons of synchronous and asynchronous models of solutions of PDEs on a multiprocessor system with multiple levels of memory show that synchronization cost cannot be ignored in the analysis of parallel computations.

In this paper we develop a unified model of a computation to take into account the overhead associated with communication, control and synchronization. Based upon this model we derive exact expressions for the actual synchronization costs for uniform, normal and exponential distributions. Then we compute bounds for the actual cost of synchronization for dependent and independent execution times of the computations carried out in parallel in terms of the mean value of the execution time and of its variance.

Empirical studies of the distribution of the execution times inside a Fork-Join pair are necessary in order to establish whether the actual distribution can be approximated by uniform distributions, so that massive parallelism is still possible with a low synchronization overhead.

Literature

[1] F. Bacelli, A.M. Makowski and A. Schwartz, "The fork-join queue and related systems with synchronization constraints, stochastic ordering, approximations and computable bounds", preprint, January 1986.

[2] L. Brochard, "Communication and control costs of domain decomposition on loosely coupled multiprocessors", *Proceedings of the 7th International Conference on Distributed Computing Systems*, Berlin, pp. 200–205, 1984.

[3] Z. Cvetanovic, "The effects of problem partitioning allocation and granularity on the performance of multiple-processor systems", *IEEE Trans. Computers*, Vol. C-36, pp. 421–432, 1987.

[4] H.A. David, *Order statistics,* Second Edition, Wiley, 1981.

[5] D. Gajski, D. Kuck, D. Lawrie and A. Sameh, "Cedar", in *Supercomputers: Design and Applications* (K. Hwang, ed.) IEEE EH0219-6, pp. 251–275, 1984.

[6] D.B. Gannon and J. van Rosendale, "On the impact of the communication complexity on the design of parallel numerical algorithms", *IEEE Trans. Computers*, Vol. C-33, pp. 1180–1194, 1984.

[7] D.C. Grunwald and D.A. Reed, "Benchmarking hypercube hardware and software", Report No. U1UCDCS-R-86-1303, University of Illinois, 1986.

[8] B. Indurkhaya, H.S. Stone and L. Xi-Cheng, "Optimal partitioning of randomly generated distributed programs", *IEEE Trans. Software Engineering*, Vol. SE-12, pp. 483–495, 1986.

[9] C.P. Kruskal and A. Weiss, "Allocating independent subtasks on parallel processors, in *Intl. Conf. Parallel Processing,* IEEE, pp. 236–240, 1984.

[10] S.L. Lundstrom, "Applications considerations in the system design of highly concurrent multiprocessors", *IEEE Trans. Computers*, Vol. C-36, pp. 1292–1309, 1987.

[11] D.C. Marinescu and J.R. Rice, "Synchronization of nonhomogeneous parallel computations", Proceedings of the SIAM Conference on Parallel Processing for Scientific Computing, December 1987 (to appear). Also CSD-TR-683 Computer Sciences, Purdue University, May 1987.

[12] D.C. Marinescu and J.R. Rice, "Domain oriented analysis of PDE splitting algorithms", Information Sciences, 43, pp. 3–24, 1987.

[13] D.M. Nicole, "Performance issues for domain-oriented time-driven distributed simulation", Technical Report 87-49, ICASE, 1987.

[14] C.D. Polychronopoulous and U. Banerjee, "Processor allocation for horizontal and vertical parallelism and related speed-up bounds", *IEEE Trans. Computers*, Vol. C-36, pp. 410–420, 1987.

[15] D.A. Reed, L.A. Adams and M.L. Patrick, "Stencils and problem partitioning, their influence on the performance of multiple processor systems", *IEEE Trans. Computers*, Vol. C-36, pp. 845–858, 1987.

[16] J.R. Rice, "Multi-FLEX machines preliminary report", CSD-TR-612, Computer Science, Purdue University, 1986.

[17] J.R. Rice and D.C. Marinescu, "Analysis and modeling of Schwartz Splitting algorithms for elliptic PDEs", in *Advances in Computer Methods for Partial Differential Equations*, VI (Stepleman and Vishnevetsky, eds.), IMACS, Rutgers University, pp. 1–6, 1987.

[18] J.R. Rice, "Parallel methods for partial differential equations", in *The Characteristics of Parallel Computation* (Jamieson, et. al., eds.), MIT Press, pp. 209–231, 1987.

[19] H.J. Siegel, T. Schwederski, J.K. Kuehn and N.J. Davis IV, "PASM: A reconfigurable parallel system for image processing" in Parallel Computing, Theory and Comparisons by G. J. Lipovski and M. Malek, John Wiley, 1987.