COMPUTER SYSTEMS LABORATORY

STANFORD ELECTRONICS LABORATORIES
DEPARTMENT OF ELECTRICAL ENGINEERING
STANFORD UNIVERSITY · STANFORD, CA 94305

SEL 78-034

PERFORMANCE CHARACTERIZATION OF PARALLEL COMPUTATIONS

by

Ruby Bei-Loh Lee

September 1978

Technical Report 158

The work herein was supported in part by the Ballistic Missile Defense Systems Command under contract no. DASG60-77-C-0073

SEL 78-034

PERFORMANCE CHARACTERIZATION OF PARALLEL COMPUTATIONS

by

Ruby Bei-Loh Lee

September 1978

Technical Report 158

Digital Systems Laboratory
Departments of Electrical Engineering and Computer Science
Stanford University
Stanford, CA 94305

The work herein was supported in part by the Ballistic Missile Defense Systems Command under contract no. DASG60-77-C-0073.

SEL 78-034

Digital Systems Laboratory
Departments of Electrical Engineering and Computer Science
Stanford University
Stanford, CA 94305

Technical Report 158

September 1978

PERFORMANCE CHAPACTERTZATTON OF PARALLEL COMPUTATIONS

by

Ruby Rei-Loh Lee

ABSTPACT

This paper defines and interprets quantitative measures by which we may characterise the absolute and relative performance of a parallel computation, compared with an equivalent serial computation. The absolute performance measures are the Parallelism Index, PI(P), the Utilization, U(P), and the maximum Quality, Q(P). The corresponding relative performance measures are the Speedup, S(P,l), the Efficiency, E(P,l), and the Quality, Q(P,l). We show how the corresponding absolute and relative performance measures are related via the Redundancy measure, R(P,l). We also examine the range of permissible values for each performance measure.

Ideally, we would like to compare an optimal parallel computation with an optimal equivalent serial computation, in order to determine the performance improvements due solely to parallel versus serial processing. Toward this end, we define optimal parallel and serial computations, and show how such optimality may be approximated in practice.

In order to facilitate the calculation of the above performance measures, we show how the complexity of modclling an arbitrary parallel computation may be reduced substantially to two simple canonical forms, which we denote the computation's Parallelism Profile and TOP-form.

Finally. we show how all the canonical forms and performance measures may be generalised from one computation to a set of computations, to arrive at aggregate canonical and performance descriptions.

The work. herein was supported in part by the Ballistic Missile Defense Systems Command under contract no. DASG60-77-C-0073.

TABLE OF CONTENTS

1. INTRODUCTION

The purpose of this paper is to characterise the performance potential of parallel computations. We are interested, for example, in the speed improvements obtained by a single computation or a set of computations when the operations in the computation/s may be executed in parallel. By not limiting the maximum number of operations which may be executed simultaneously in any step of a computation, we hope to obtain the minimum execution-time possible for the computation by fully exploiting its inherent parallelism. It is intuitively clear that some types of computations will have more inherent parallelism than others. Hence , even if the same total number of operations are executed, some computations will have a smaller execution-time, since more operations may be executed simultaneously. Of course the minimum possible execution-time of any computation with at least one operation to be performed, is one step, or one time-unit. Rut this is almost never attained in practice due to data and control dependencies in the program, which force a sequential. chain of execution for the dependent operations. In general. there may be many separate dependency chains running through the program. Since operations in different dependency chains may be executed in parallel. the minimum parallel execution time, in steps, is really the maximum number of dependency levels in any of the dependency chains of the program. This minimum parallel execution-time, which we denote T_{min} in this paper, in turn gives rise to the maximum speedup of the computation.

There are many reasons why one may be interested in the unconstrained performance potential of parallel. computations. Our main motivation stems from the desire to characterize the performance improvements attainable by parallel processor organisations compared with uniprocessor, or serial, organisations.

INTRODUCTION

By a parallel processor organisation, we mean a computer organisation with $p \geq 1$ processors, which behave identically, each being able to execute one of many operations in one time-unit. Hence, the processors need not be exactly identical, but they appear to be identical because they can each execute the same repetoire of operations in the same amount of time. At any time-unit, or step of the execution of a computation, any number from one to p of the processors may be busy executing an operation each. For our purposes, the parallel processor organisation may be assumed to be synchronous. This means that the p parallel processors execute in lock-step fashion. so that the instructions in step (i+1) of the computation are not started until all those in step (i) are completed, each step taking one time-unit for execution.

Flynn[FLY72] has classified parallel processor organisations into two types: SIMD (Single Instruction Multiple Data) organisations, and MIMD (Multiple Instruction Multiple Data) organisations. In SIMD organisations, also known as array processors, a single type of instruction is carried out by some or all of the processors in parallel, but on different pairs of operands. In MIMD organisations, many different types of instructions may be carried out simultaneously by different parallel. processors. It is clear that SIMD organisations are just a special class of restricted MIMD organisations, and the speed. improvements obtained for MIMD organisations will be an upper bound for those obtainable in SIMD organisations. The canonical forms and performance measures described in this paper apply equally well to the characterisation of parallel computations written for SIMD or MIMD organisations, since they are independent of the type or types of operations performed in any step.

Intuitively, it is clear that a parallel processor organisation with p parallel processors will not, in general, attain

a speed improvement of p times that of a uniprocessor system. This is because the computation being executed usually cannot utilize all p parallel processors at every step of execution. This points out a fundamental difference between the speed improvements obtained via technological improvements and those obtained via computer organisational techniques. Technological speed improvements are quite independent of the computation being executed, since they merely serve to decrease the clock. cycle time. On the other hand, computer organisational techniques are very much dependent on the demands of the computation being executed. Hence, it is not very illuminating to talk about the maximum speed, operations per time-unit, of a parallel processor organisation, especially if this is hardly ever obtai.ned. or even approached, in practice. Rather, we need to characterize the average speed of a parallel processor organisation with respect to a single computation, or a set of representative computations. In other words, the performance of a parallel processor organisation is constrained not only by the "physical parallelism" avail able in the form of parallel processors, but a! so by the "logical parallelism" inherent in the computations which are executed on it. Hence, before we can characterize the true merits of parallel processor organisations, we must learn how to effectively characterise parallel computations (section 2) and their performance with respect to different criteria (sections 3 - 7).

On the other hand, the unconstrained performance potential of parallel computations describe the performance potential of a "virtual" parallel processor organisation, where there are always as many physical processors as needed in any step of the computations executed. In practice, this performance potential is further constrained by the actual number of processors available in the computer organisation, on the type of parallel processor organisation (like SIMD or MIMD), and on the delays caused by the

interactions between processors, memories and input-output devices. Again, we observe that there are both logical and physical constraints on the performance attainable by parallel processing: the former caused by the dependencies in computations which Limit their inherent parallelism, and the later caused by the limitations of the physical resources and their interactions in the hardware organisation itself. Tn this paper, we wish to characterise the logical constraints, which are in a sense intrinsic constraints on parallelism, since they are intrinsic to the set of problems to be solved on the computer organisation.

2. CANONICAL FORMS OF PARALLEL COMPUTATIONS

The main difference between ordinary serial computations and parallel computations is that in the former, only one operation may be executed at any one time, whereas in the latter, more than one operation may be executed simultaneously. The class of parallel computations that we are interested in has the same "computational power" as the class of all serial computations, i.e., any problem solvable by a parallel computation must also be solvable by a serial computation, though not necessarily in the same way or in the same amount of time. Hence, since the functions computable by the class of all serial programs are precisely the Turing-computable functions [HOP69,MAN74], these are also all the functions computable by our class of parallel computations. In short, the primary motivation in this paper for going from serial to parallel computations is not for increased computational power, but rather for the engineering consideration of increased speed of execution.

In this section, we want to find suitable representations for parallel computations, which satisfy the following requirements:

(1) Any parallel computation must be reducible into this representation, by the application of a canon, or general rule. (The representation must be a canonical form.)

(2) The representation must clearly exhibit the parallelism in the computation, and/or facilitate the calculation of the desired performance measures.

Before reducing parallel computations into such suitable canonical forms, we should first define exactly what we mean by a parallel computation.

Definition 2.1:

A parallel computation is a sequence of steps, where step$_k$

has i operations which may be performed simultaneously. i. is called the degree of parallelism of step k. The maximum degree of parallelism in any step is denoted P, also called the logical parallelism inherent in the computation.

Assumimg that each step takes one time-unit for execution, the execution-time, T(P), is defined as the total number of steps encountered in executing the computation. The size, O(P), of the computation, is defined as the total number of operations performed in all the steps encountered during execution. The size O(P) is also called the length of the computation, especially when P=1.

It is clear that serial computations are merely a restricted subset of parallel computations, where the logical parallelism,P, is always equal to one. A graphic illustration of a parallel computation is given in Figure 1. Each unit-square in the processor--time space is called a "processor-step", since it involves one processor for a tine-duration of one step. Processor-step (i,j) is said to be "busy" if processor i is required to execute an operation in step j, and "free" otherwise. The parallel computation is then the area covered by the busy processor-steps.

PARALLELISM PROFILE

Since we are primarily interested in the speed of the parallel computation, we make the following observations which enable us to simplify the representation of any parallel. computation:

(1) The type of operation is not relevant, if all. operations can be completed in one step, or one time-unit. N-tat is relevant is the NUMBER of operations in each step. Hence, we can represent all operations by a single symbol, say an "x".

CANONICAL FORMS

(2) It is not relevant which particular processors are used in a step, just the NUMBER of processors used in the step. Hence, we can move all operations as far left as possible, and redefine a "busy" processor-step (i,j) to be one where step j has a degree of parallelism greater than or equal to i.

(3) The particular order of the steps is not relevant. What is relevant is the NUMBER of steps with any given degree of parallelism. Hence we can permute the steps so that they are arranged in increasing (or decreasing) order of their degrees of parallelism.

Figure 2 shows the graphic representation of the parallel computation of Figure 1, after the reductions outlined in (1) and (2) above. Figure 3 shows the final graphic representation after reduction (3) is applied.

We observe that the relevant aspects of the computation left after reductions (1) through (3) may be summarised as "the frequency distribution of the degrees of parallelism In the computation"*

Definition 2.2:

The Parallelism Profile of a computation is a canonical form, representing the frequency distribution of the degrees of parallelism in the computation, or the number of steps having degree of parallelism i, for $1 \leq i \leq P$.

It may be given in the format:

$$1^{x_1}.2^{x_2}.....i^{x_i}.....P^{x_P}$$

where i is the degree of parallelism, and

x_i is the number of steps with degree of parallelism i.

CANONICAL FORMS

We note that the frequency distribution of the degrees of parallelism in a computation may also be given in the format of a P-tuple:

$$(x_1, x_2, \ldots, x_i, \ldots, x_p)$$

where x_i is the number of steps whose degree of parallelism is i.

We will usually use the format of Definition 2.2 for the Parallelism Profile of a computation, since unlike the P-tuple format, we can see at a glance which frequency xi corresponds to which degree of parallelism, i. In addition, if frequency x_i for degree of paralleism i is zero, we may omit the term i^{x_i} since symbolically $i^0=1$ and a.1.b = a.b. This results in a more compact representation than the P-tuple format, especially when there are many zero frequencies.

TOP-FORM

Very often. we do not need all the information given by the Parallelism Profile. We often do not need to know the exact frequency distribution of the parallel computation. Hence, we have an even simpler canonical form for parallel computations*

Definition 2.3:

The TOP-form of a computation is a 3-tuple:

$$(T, O, P),$$

where T is the execution time in steps,
0 is the size of the computation in operations, and
P is the maximum degree of parallelism.

CANONICAL FORMS

We note that the TOP-form can be derived from the Parallelism Profile but not vice-versa. In particular,

$$T = \sum_{1 \le i \le P} x_i,$$

$$0 = \sum_{1 \le i \le P} i \cdot x_i$$

and P is the largest degree of parallelism with nonzero frequency.

Pence, by successively reducing the amount of pertinent information kept, we can reduce every computation first to its Parallelism Profile and then to its TOP-form.

Example 2.1:

In the example given in figure 1, the Parallelism Profile is:

$$1^3.2^2.3^1.4^4.8^2$$

From this we can calculate:

$T = 3+2+1+4+2 = 12$ steps

$0 = 1.3+2.2+3.1+4.4+8.2 = 42$ operations

$P = 8$

so the TOP-form is:

$(12,42,8)$.

We remark that the TOP-form of a serial computation where P=1 is always of the form:

$(x,x,1)$,

where x is a positive integer. Because of this, it is not surprising that the terms "execution-time" and "size" or "length" are sometimes used interchangeably for serial computations, since

they always have the same value. Tn fact, finding the TOP-form of a serial computation reduces to finding its length, O(l), since it will always be true that T(l)=O(l) and P=l.

The TOP-form of a parallel. computation where P>l is of the form:

(x,y,z),

where x,y,z are positive integers, and x is never equal to y when z>l. The possible ranges for each of x,y and z are given in the following theorem:

Theorem 2.1:

For any given TOP-form = (t,o,p), the following relationships hold:

(i) $o/p \leq t \leq o-p+1$

fii) $p+t-1 \leq o \leq p.t$

(iii) $o/t \leq p \leq o-t+1$

Proof:

(i) The shortest execution-time occurs when the computation is fully parallel, i.e., each step has the maximum degree of parallelism p. Then the execution-time is just $\lceil o/p \rceil$ steps. Clearly, $\lceil o/p \rceil \geq o/p$.

The longest execution-time occurs when each step has degree of parallelism = l. Rut there can be at most (o-p) such steps, since at least one step must have degree of parallelism p, by the

definition of the TOP-form. Hence, t < o-p+l.

(ii) and (iii) follow directly from (i).

We state the following corollary, which will he used in the proofs given in subsequent sections:

Corollary 2.2:

T(P) $\leq$ O(P) with equality iff P=l

Proof:

From theorem 2.1(i), O(P)/P < T(P) < O(P)-P+l

So, for P=1, O(1)/1 $\leq$ T(1) $\leq$ O(1)-1+1

hence, T(1)=O(1).

for P$\geq$ 2, T(P) $\leq$ O(P)-2+1 = O(P)-1

hence, T(P) < O(P).

FIGURE 1: A PARALLEL COMPUTATION

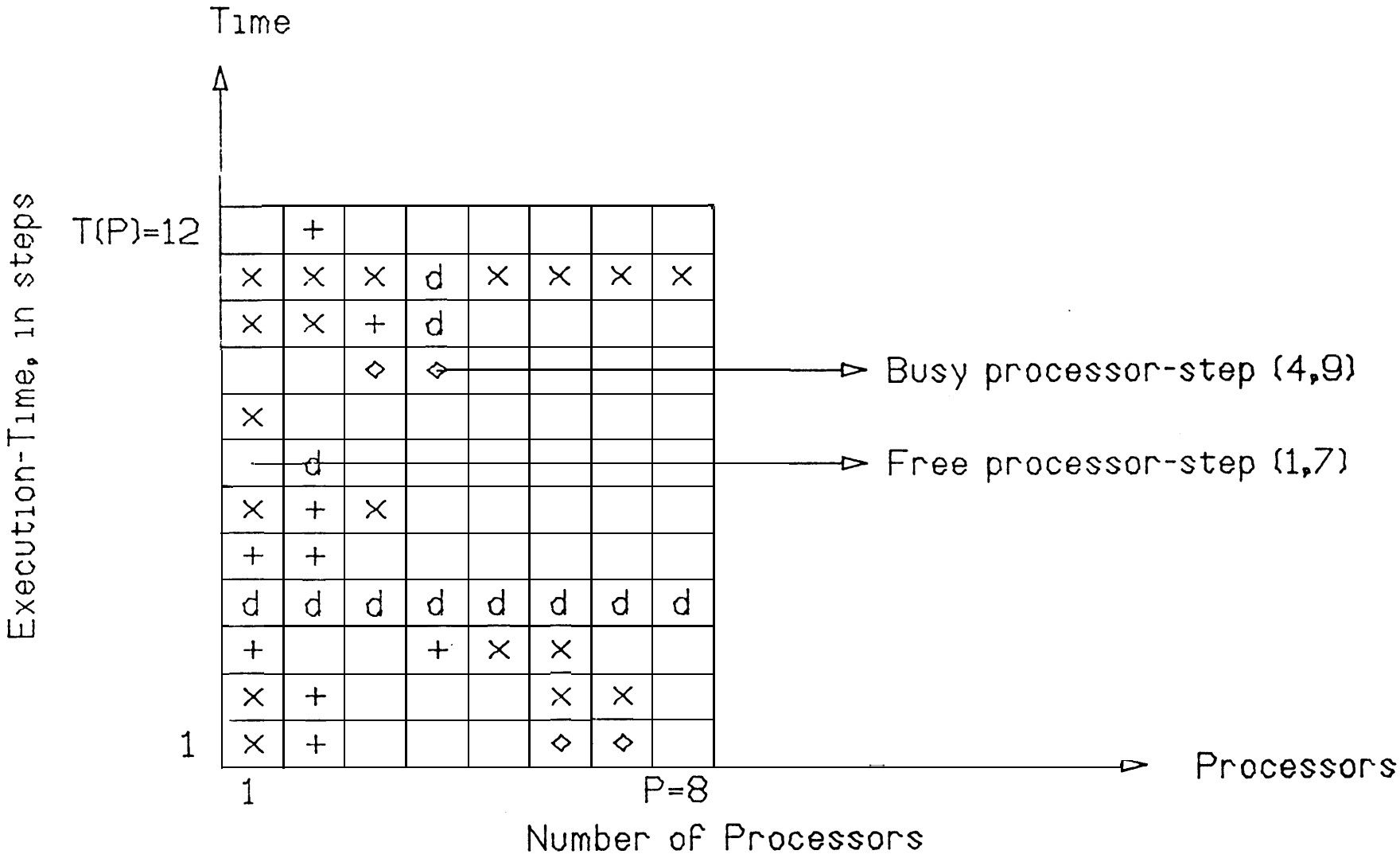


FIGURE 2: A REDUCED PARALLEL COMPUTATION

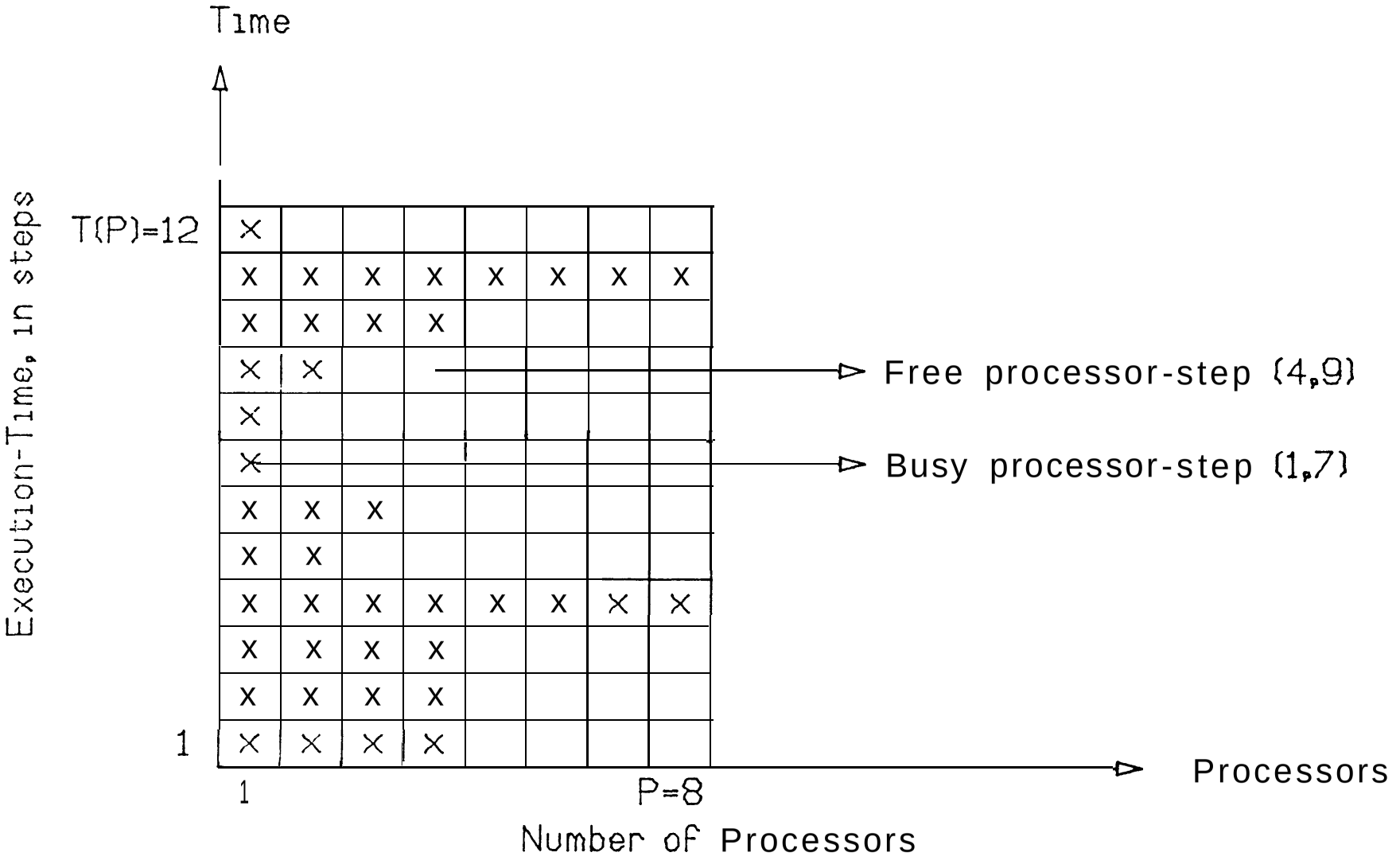
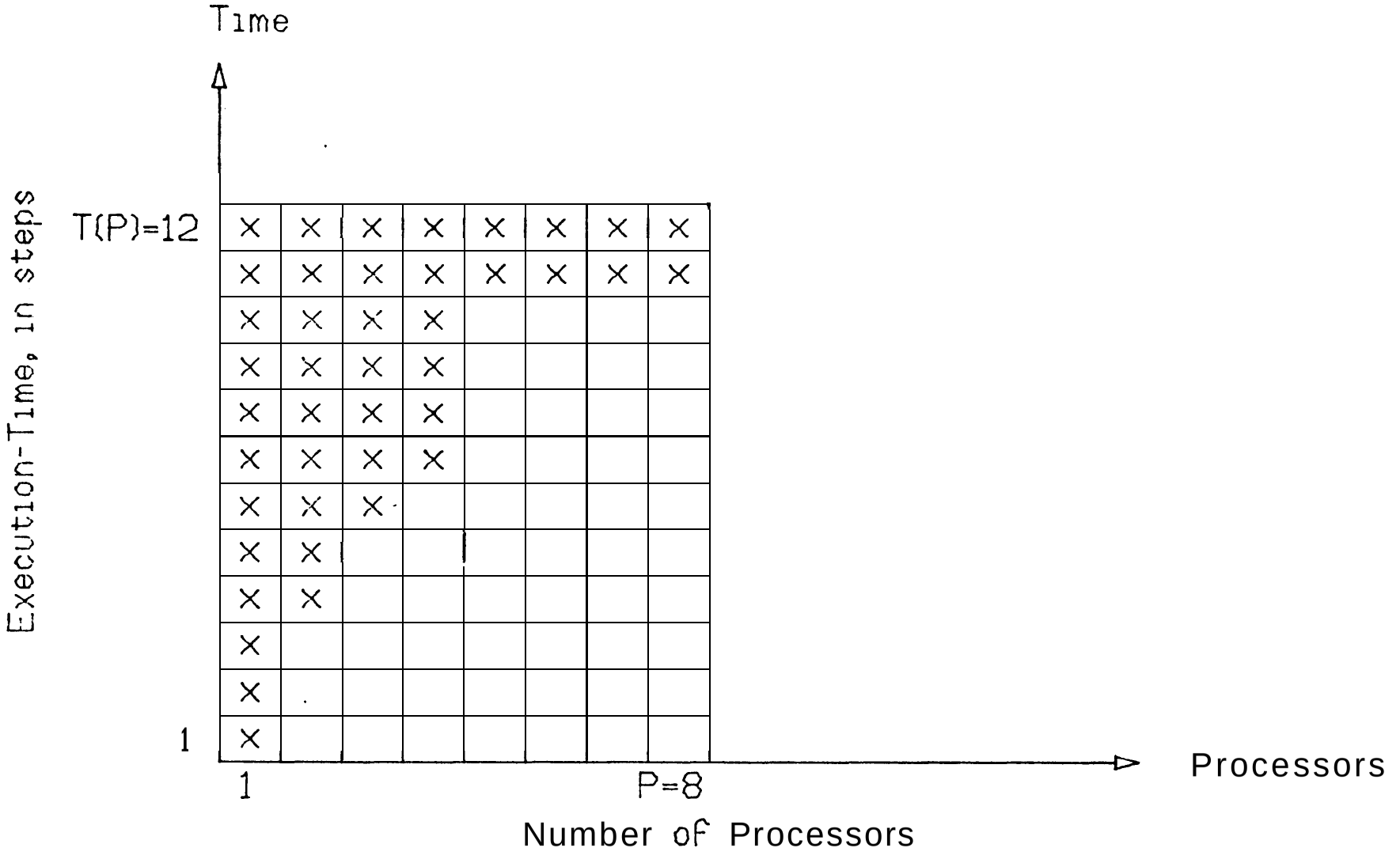


FIGURE 3: A PARALLELISM PROFILE



3. PARALLELISM INDEX AND UTILIZATION

The last component of the TOP-form of a parallel computation gives us the maximum degree of parallelism P of the computation. This is useful since it tells us how many parallel processors would be needed to execute this parallel computation as given. However, the maximum degree of parallelism is not as good an indicator of the parallelism inherent in a computation as the average degree of parallelism per step.

Example 3.1:

Suppose we have two computations, A and B, each with 10 steps. Suppose A has 9 steps with degree of parallelism 1 and one step with degree of parallelism 16. Suppose B has all 10 steps having degree of parallelism 12. This can be represented concisely by the Parallelism Profiles of A and B:

A: $1^9.16^1$

B: 12^{10}

If we knew only the maximum degree of parallelism of A and B, we might say that A has more inherent parallelism than B. But this does not agree with our intuitive understanding of "inherent parallelism", which is better described as a kind of overall, or average, degree of parallelism. In fact, the average degree of parallelism of A is only 2.5, while that of B is 12. Hence, B has more inherent parallelism than A in terms of the average degree of parallelism.

PARALLELISM TNDEX AND UTILIZATION

Definition 3.1:

The Parallelism Index (abbreviated PI or π) is defined as the average degree of parallelism per step. It is the ratio of the total size, O(P), of the computation in operations, over the total time, T(P), taken for execution in steps.

$$PT = \frac{O(P)}{T(P)}$$

The Parallelism Index may be interpreted as the average processor bandwidth, in operations per step, of the parallel computation. The maximum processor bandwidth is the maximum degree of parallelism, P.

The Parallelism Index may also be interpreted as the average speed of the computation, since it measures the total size (or length) in number of operations executed, divided by the total time taken. If we let dt be one step of the computation, then the instantaneous speed is interpreted as the degree of parallelism in each step. The maximum instantaneous speed is the maximum degree of parallelism P.

Since speed is the magnitude of velocity, it is interesting to consider the interpretation of negative velocities in the parallel computation. A negative velocity may be interpreted as the undoing of operations already executed. For example, if the result of a conditional branch instruction is true, execution continues along path (a) of the program, and if it is false, execution continues along path (b). Suppose we did not want to hold up execution to

wait for the result of the conditional branch instruction, we might guess that the result would be true and continue execution along path (a). If it turned out that the result was false, then we would have to undo the operations performed along path (a), resulting in a negative velocity for this duration. It is interesting to speculate on whether certain types of computations might have a higher average speed if negative velocities are allowed. This is certainly worth some future investigation, but for the purposes of this paper, we may assume that all velocities are positive.

The interpretation of the degree of parallelism in a step as its instantaneous speed implies that serial computations always have a constant speed of one operation per step. In fact, any computation with a constant speed of P operations per step, for $P \geq 1$, is said to be "fully parallel" at degree P.

Definition 3.2:

A computation is said to be fully parallel at degree P, $P \geq I$, if every step in the computation has degree of parallelism P. Then, the average degree of parallelism, PI, is equal to the maximum degree of parallelism P.

The interpretation of the degree of parallelism in a step as its instantaneous speed also implies that every parallel computation with $P \geq 2$ can also be executed as a serial computation by going at a constant speed of execution of one operation per step. Hence, if a parallel computation has a size of O(P) operations, an equivalent serial computation which solves the same problem, need never have a size greater than O(P) operations.

PARALLELISM INDEX AND UTILIZATION

If a parallel computation with TOP-form = (T,O,P) is executed on a parallel-processor organisation with P parallel processors, then the maximum number of operations which could have been executed would be P.T operations. However, the actual size of the computation executed is only 0, which is less than P.T except when the computation is fully parallel. The ratio of the actual size of the computation to the maximum possible size describes the "utilization of the available parallelism" by the computation.

Definition 3.3:

The Utilization (abbreviated U) of available parallelism by a given computation is the ratio of actual computation size over the maximum possible computation size.

$$U = \frac{O(P)}{P.T(P)} = \frac{PI(P)}{P}$$

The units of Utilization may be regarded as operations per processor step, where one processor step is the work done by one processor in one step. In Figures 1,2 and 3, a processor-step is a unit-square in the processor-time graph, and the Utilization is the ratio of the area covered by the "busy" processor-steps over the area of the rectangle with sides P and T(P). Hence, if all the processors in the system are identical and used with equal frequency over the time-duration T(P), then the Utilization may'be regarded as the probability that any single processor will be busy in any given step. Pence, we offer the following hypothesis, which will be examined in detail in a separate paper:

Hypothesis 3.1:

Let U be the utilization of a parallel-processor system, as in Definition 3.3. Then the probalility, q_i, that i processors will be used in any step is a truncated binomial. distribution

$$q_i = \frac{\binom{P}{i} U^i \cdot (1-U)^{P-i}}{1 - (1-U)^P}, \quad \text{for } 1 \leq i \leq P.$$

The following theorem examines the possible ranges of values for PI and U for serial and parallel computations.

Theorem 3.1:

(i) $PI(1) = U(1) = 1$, for serial computations where P=1

(ii) $1 < PI(P) \leq P$, and
$1/P < U(P) \leq 1$, for parallel computations where P>1.

Proof:

We will deal with PI first, then U.

PI* From corollary 2.2,

$T(P) \leq O(P)$, with equality iff P=1.

So, for P=1, $PI(1) = \frac{O(1)}{T(1)} = 1$

and for P>1, $PI(P) = \frac{O(P)}{T(P)} > 1$

Futhermore, since the maximum possible computation size is P.T(P), we have

$$PI(P) = \frac{O(P)}{T(P)} \leq \frac{P.T(P)}{T(P)} = P$$

U: For P=1, $U(1) = \frac{PI(1)}{1} = \frac{1}{1} = 1$

For P>1, $U(P) = \frac{PI(P)}{P} > \frac{1}{P}$,

and $U(P) = \frac{PI(P)}{P} \leq \frac{P}{P} = 1$

---

4. SPEEDUP AND EFFICIENCY

The performance measures, PI and U, of the previous section may be calculated from knowledge of just the TOP-form of a parallel computation. They give us a good indication of the computation's average processor bandwidth and its utilization of available parallelism.

Mow, we would like to compare the performance of a parallel computation with an equivalent serial computation. This will indicate the performance improvement obtained when the given problem is executed on a parall el-processor organization rather than on a uniprocessor organization. Rut what do we mean when we say that two computations are equivalent?

Definition 4.1:

Two computations are said to be equivalent if given the same inputs, they always produce the same final outputs.

This definition of equivalency does not require intermediate results to be the same. This means that the two computations are equivalent because they produce the same solution-set for the same problem, but they may involve entirely different algorithms with entirely different sets of intermediate results.

Unfortunately, there are many equivalent serial computations and many equivalent parallel computations. In order to make a fair comparison, if we choose an optimal parallel computation, then we should also choose an optimal equivalent serial computation. An optimal serial computation is a "minimum--size serial computation", or one which has the minimum size, 0_{min}, among all equivalent

computations. We note that there may be many different minimum-size serial computations, in terms of the algorithms implemented and the operations performed. However, the TOP-form of all these equivalent minimum-size serial computations is the same.

Definition 4.2:

An optimal serial computation is a minimum-size serial computation with the TOP-form:

$$(T(1'), O_{min}, 1),$$

where the minimum size is defined as

$$O_{min} \leq \min \{O(P), P \geq 1\}$$

and the minimum serial execution time is

$$T(1') = O_{min}.$$

Rut finding the minimum size, or length, of a computation for any given problem is a well-known undecidable issue [MAN74, MIN67, HOP69] However, we can always estimate the minimum-size, O_{min}, from the sizes of known serial and parallel computations.

Definition 4.3:

If we know only the size, O(P), of a parallel computation, and the size, O(I), of an equivalent serial computation, then we can estimate the minimum size, O_{min}, as follows:

$$O_{min} \leq \min \{O(1), O(P))$$

Some readers may be wondering why O(1) is not always equal to O(P). There are two reasons for this. The first is that the given serial computation is not an efficient one, and a better serial computation may be obtained by just executing the given parallel computation at a constant speed of one operation per step, as discussed in the previous section. In this case, the given O(P) is less than the given O(l), and we approximate O_{min} with O(P).

The second reason why O(1) and O(P) may not be the same is because redundant operations are often introduced into a parallel computation in order to increase its speed of execution. In this case, O(P) is greater than O(l), and we approximate O_{min} with O(1).

Sometimes, if we are not careful, a lot of unnecessary redundant operations are introduced into the parallel. computation so that the parallel execution-time, T(P), is in fact greater than a known serial execution-time, T(1). Clearly, if we are to choose an optimal parallel. computation, then we must allow the introduction of redundant operations only if this reduces the execution-time when compared with known serial computations. An optimal parallel computation is then a "minimum-time minimax-parallel computation", defined below. Again, there may be many different minimum-time minimax-parallel computations in terms of the algorithms and operations performed, but the TOP-form of all these equivalent optimal parallel computations is the same.

Definition 4.4:

An optimal parallel computation is a minimum-time minimax-parallel computation with the TOP-form:

$$(T_{min}, O(P'), P'), \text{ for } P' \geq 1,$$

where the minimax degree of parallelism, P', is the smallest

maximum degree of parallelism of any computation having the minimum execution-time, T_{min}, defined as follows:

$$T_{min} \leq \min \{T(P), P \geq 1)$$

In particular, T_{min} is less than or equal to the minimum serial execution-time, $T(1')$, and $O(P')$ is greater than or equal to the minimum size. O_{min}. Hence, we have

$$T_{min} \leq T(1')=O_{min} \leq O(P').$$

The definition of the minimum time, T_{min}, implies that sometimes an optimal parallel computation is in fact a serial. one with $P'=1$. Hence, when the minimax degree of parallelism is one, the TOP-form of a minimum-time minimax-parallel computation is identical to that of an equivalent minimum-size serial computation:

$$(T_{min}, O(1'), 1') = (T(1'), O_{min}, 1'),$$
where $O(1')=T_{min}=T(1')=O_{min}$.

For computations where this is true, there is absolutely no speed improvement when executing the computation on a parallel-processor organisation since the computation cannot utilize the available parallelism.

Just as finding O_{min} is an unsolvable problem in general, finding T_{min} and P' is also an unsolvable issue [BER66]. Rut again, we can try to estimate T_{min} and P' from known equivalent computations.

Definition 4.5:

If we know only the execution-time, T(P), of a parallel computation, and the execution-time, T(1), of an equivalent serial computation, then we can estimate the minimum execution-time, T_{min}, and the minimax degree of parallelism, P', as follows:

$$T_{min} \leq \min \{T(1), T(P)\}$$

$$\text{and } P' = \begin{cases} 1, & \text{if } T(1) \leq T(P) \\ P, & \text{otherwise.} \end{cases}$$

Ideally, we would like to minimize the execution-time of the computation, minimize the total number of operations that have to be performed, and minimize the maximum number of processors required to execute the computation. In other words, we would like to minimize each component of the computation's TOP-form. The minimum-time minimax-parallel computation would satisfy these criteria except that its size, O(P'), may contain redundant operations, and hence be greater than the minimum size, O_{min}. We should take this redundancy into account in determining the speed improvement of parallel execution over serial execution.

The performance measures, PI and U, of the previous section were defined with respect to the actual computation size, O(P). Hence, they required knowledge of only the TOP-form of the parallel computation being measured. We will now define the analogous performance measures of Speedup and Efficiency with respect to O(1), the size of an equivalent serial computation. Therefore, in addition to the TOP-form of the parallel computation, we also need to know the serial size, O(1). It is easy to see that the TOP-form of any serial computation reduces to just the serial size, O(1), since T(1)=O(1) and P=1.

Definition 4.6:

The Speedup, S, of any given parallel computation over any equivalent serial computation is defined as:

$$S(P,1) = \frac{T(1)}{T(P)} = \frac{O(1)}{T(P)}$$

If the serial computation is an equivalent minimum-size serial computation, then:

$$S(P,1') = \frac{O_{min}}{T(P)} = \frac{T(1')}{T(P)}$$

In addition, if the parallel computation is a minimum-time minimax-parallel computation, then:

$$S(P',1') = \frac{O_{min}}{T_{min}} = \frac{T(1')}{T(P')}$$

The definition of Speedup as a dimensionless ratio of execution-times is similar to definitions of speedup found in the literature [KUC73,KUC76,KUC78,LEE76,TOW76]. However, the definition of Speedup in terms of operations per step is very useful since it enables us to relate it directly to the definition of the Parallelism Index of any parallel computation. This enables us to interpret the speedup, $S(P,1')$, as the average effective processor bandwidth, whereas the Parallelism Index is the average actual processor bandwidth, and the logical parallelism P is the maximum processor bandwidth.

Similarly, we can also interpret the Speedup, $S(P,1')$, as the average effective speed of the parallel computation, whereas the Parallelism Index is the average actual speed, and the logical parallelism P is the maximum (average or instantaneous) speed.

Theorem 4.3:

(a) $0 < S(P,1) \leq O(1)$

(b) $0 < S(P,1') \leq PI(P) \leq P$

(c) $1 \leq S(P',1') \leq PI(P') \leq P'$

Proof:

(a) For any arbitrary equivalent parallel and serial computations,

$S(P,1) = \frac{T(1)}{T(P)} > 0$, since both $T(1)$ and $T(P)$ are positive.

$S(P,1) = \frac{O(1)}{T(P)} \leq O(1)$, since $T(P) \geq 1$.

(b) For an arbitrary parallel computation and an equivalent minimum-size serial computation, the lower bound is unchanged, but now

$$S(P,1') = \frac{O_{min}}{T(P)} \leq \frac{O_{min} \cdot P}{O(P)} \leq P, \quad \text{since } O(P) \geq O_{min}$$

(c) For a minimum-time minimax-parallel computation and an equivalent minimum-size serial computation, the upper bound is the same as in (b), except that P is replaced by the minimax degree of parallelism, P'. However, we now have a better lower bound:

$S(P',1') = \frac{T(1')}{T_{min}} \geq I$, since $T(1') \geq T_{min}$

SPEEDUP AND EFFICIENCY

We note that for arbitrary parallel and serial computations, the speedup, S(P,1) is not necessarily greater than or equal to one, since T(P) may in fact be greater than T(1), as in a nonoptimal parallel. computation. This speedup, S(P,1), is also not bound from above by P, since O(1) may be greater than O(P), as in a nonoptimal seria! program, so that:

$$S(P,1) = \frac{O(1)}{T(P)} \leq \frac{O(1).P}{O(P)}$$

is not necessarily less than P.

Next, we define the Efficiency of a parallel computation, which is just its utilization of available parallelism with respect to the serial computation size, O(1).

Definition 4.7:

The Efficiency, E, of any parallel computation over any equivalent serial. computation is defined as:

$$E(P,1) = \frac{O(1)}{P.T(P)} = \frac{S(P,1)}{P}$$

If the given serial computation is an equivalent minimum-size serial computation, then:

$$E(P,1') = \frac{O_{min}}{P.T(P)} = \frac{S(P,1')}{P}$$

In addition. if the parallel computation Is a minimum-time minimax-parallel computation, then*

$$E(P',1') = \frac{O_{min}}{P'.T_{min}} = \frac{S(P',1')}{P'}$$

The Efficiency may be interpreted as a measure of the "cost-effectiveness" of a parallel computation. Cost-effectiveness is generally given as a ratio of "effectiveness", or performance, over "Cost". It is important that the "scales" over which effectiveness and cost are measured are compatible. For example, if the effectiveness is given as the average performance for any step, then the cost should be measured as the average cost per step. If we let the effectiveness be represented hy the Speedup S, which is the average effective processor bandwidth or the average effective number of operations per step, then a compatible cost measure, C, would be a measure of the average cost per step.

Definition 4.8:

The Cost-Effectiveness, CE, of a computation is defined as

$$CE = \frac{S(P,1')}{C} = \frac{S(P,1')}{P.t} = \frac{E(P,1')}{t}$$

where $S(P,1')$ is the speedup with respect to an optimal eauivalent serial computation,

P is the maximum degree of parallelism, or the number of processors required,

t is the average time for one step,

and $C=P.t$ is the average cost per step.

Hence, if each step takes one time-unit for execution, as in definition 2.1, then t=l, and CE reduces to the efficiency, $E(P,1')$.

The only other serious attempt to define a similar cost-effectiveness measure for a parallel computation can be found

in [KUC74]. Kuck defined a Speedup over Cost measure:

$$CE_{Kuck} = \frac{S(P,1)}{C} = \frac{S(P,1)}{P.T(P)} = \frac{E(P,1)}{T(P)}$$

where $S(P,1)$ is the speedup with respect to any equivalent serial computation,

P is the maximum degree of parallelism, or the number of processors required,

and $T(P)$ is the execution-time of the parallel computation in number of steps.

Unfortunately, the scales of definition in the numerator and denominator are not compatible, so that CE_{Kuck} is usually very much smaller than one, and most parallel programs have a "cost-effectiveness" less than 2 percent. Furthermore, the speedup is defined with respect to any, not necessarily optimal, equivalent serial computation, so that speedups would appear larger for choosing more poorly-constructed serial computations. These problems have been ratified in our definition of cost-effectiveness as given in Definition 4.8, which we feel is more useful. and more accurate.

The following theorem gives the permissible ranges for the efficiency with respect to different combinations of equivalent serial and parallel computations.

Theorem 4.2:

(a) $0 < E(P,1) \leq O(1)/P$

(b) $0 < E(P,1') \leq U(P) \leq 1$

(c) $1/P' \leq E(P',1') \leq U(P') \leq 1$

The proof of theorem 4.2 is derived by substituting the corresponding values of S. as given in theorem 4.1.

We note that for arbitrary parallel and serial computations, the efficiency need not have an upper bound of 1 since the speedup may be greater than P (see theorem 4.1). as in the case of a nonoptimal serial program where $O(1)>O(P)$.

From theorem 3.1 we know that serial. computations always have a utilization of one. But, by substituting P=1 in theorem 4.2 (b), we see that the efficiency, $E(1,1')$, of a serial computation compared with an equivalent minimum-size serial. computation, may be less than one.

From theorem 4.2(c), we see that a minimum-time minimax-parallel computation need not (and usually does not) have an efficiency of one. In the following corollary, we state the conditions under which the maximum utilization or efficiency of one is attained.

Corollary 4.3:

A computation has a utilization, $U(P)$, of one hff it is fully parallel. It has an efficiency, $E(P,1')$, of one iff it is fully parallel AND has minimum size, i.e., $O(P)=O_{min}$.

---

5. REDUNDANCY

We mentioned in the previous section that redundant operations are often introduced into a parallel computation to speed up its execution time. We would now like to investigate this concept of redundancy more thoroughly.

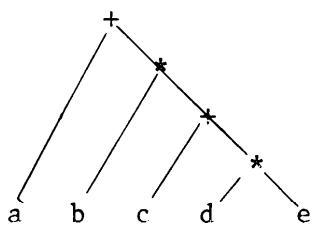
As a rule, redundant operations should not be introduced into a parallel computation unless this decreases the parallel execution time, as compared with the minimum known size of known equivalent serial computations. Although it is beyond the scope of this paper to show how and when redundant operations may profitably be introduced, we can illustrate with a simple example.

Example 5.1:

The following expression cannot be sped up by parallel execution if left as it stands:

a + b (c + d * e)

The execution tree for this expression is:



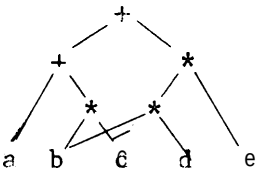
Step	Degree of Parallelism
4	1
3	1
2	1
1	1

Here, T(1) = O(1) = 4.

However, by using the distributive law of multiplication over addition, we transform the expression to:

a + b*c + b*d*e

The execution tree is now:



Step	Degree of Parallelism
3	1
2	2
1	2

Here, $T(2) = 3 < T(1)$, and
$O(2) = 5 > O(1)$.

We note that the execution time has decreased from 4 to 3 steps, but the size has increased from 4 to 5 operations. Hence, the second (parallel) computation is said to have:

a parallelism index, $PI(2) = 5/3 = 1.67$,

a speedup, $S(2,1') = 4/3 = 1.33$,

a utilization, $U(2) = 5/6 = 0.83$,

an efficiency, $E(2,1') = 4/6 = 0.67$,

and a redundancy, $R(2,1') = 5/4 = 1.25$.

We now give a formal definition of redundancy:

Definition 5.1:

The Redundancy, R. of any parallel computation over any equivalent serial computation is defined as:

$$R(P,1) = \frac{O(P)}{O(1)}$$

If the given serial computation is an equivalent minitnum-size serial computation. then:

$$R(P,1') = \frac{O(P)}{O_{min}}$$

In addition, if the parallel computation is a minimum-time minimax-parallel computation, then:

$$R(P',1') = \frac{O(P')}{O_{min}}$$

Theorem 5.1:

(a) $0 < R(P,1) \leq 1/E(P,1)$

(b) $1 \leq R(P,1') \leq 1/E(P,1')$

(c) $1 \leq R(P', 1') \leq 1/E(P',1') \leq P'$

Proof:

(a) For any arbitrary parallel and serial computations, there is no restriction on the sizes, O(1) and O(P). All we can say is R(P,1) is positive, since both O(1) and O(P) are positive integers. For the upper bound, we have:

$$R(P,1) = \frac{O(P)}{O(1)} \leq \frac{P.T(P)}{O(1)} = \frac{1}{E(P,1)}$$

Unfortunately, since E(P,1) > 0, from theorem 4.2 we cannot give an upper bound for R(P,1) from this reciprocal of the efficiency.

(b) For an arbitrary parallel computation and an equivalent minimum-size serial computation, we get a better lower bound:

$$R(P,1') = \frac{O(P)}{O_{min}} \geq 1 \quad \text{since } O(P) \geq O_{min}$$

(c) For a minimum-size minimax-parallel computation, we have the same lower bound as in (b). In addition, we now have the

following upper bound:

$$R(P',1') = \frac{O(P')}{O_{min}} \leq \frac{P'.T_{min}}{O_{min}} \leq \frac{1}{E(P',1')} < P',$$

since $E(P',1') \geq J/P'$

We note that the redundancy has a lower bound of one iff we compare the computation size O(P) with the minimum size O_{min}. The redundancy may be less than one if we compare O(P) with any arbitrary equivalent serial computation size, O(1).

One of the main significances of the redundancy measure is that it relates the performance measures (PI,U) of section 3 with the performance measures (S,E) of section 4. In theorems 4.1 and 4.2, we saw that $(S,E) \leq (PI,U)$. We now show that they are related exactly via the reciprocal of the Redundancy.

Theorem 5.2:

The Spcedup and Efficiency (S,E) are equal to the Parallelism Index and Utilization (PI,U) respectively, divided by the Redundancy.

$$(S,E) = \frac{1}{R} \cdot (PI,U)$$

Proof:

$$S(P,1) = \frac{Q(1)}{T(P)} = \frac{O(P)}{R(P,1).T(P)} = \frac{1}{R(P,1)} \cdot PI(P)$$

Clearly, the same relationships hold if we replace 1 by I' and/or P by P'.

In the following, we assume consistent values for 1 and P are used throughout:

$$E = S/P = (1/R) \cdot (PI/P) = (1/R) \cdot U$$

Theorem 5.2 is very useful since we can estimate the Speedup and Efficiency without knowing the minimum size O_{min}, which may be quite hard to determine. This is because experimental results have shown that in about eighty percent [KUC78] of the parallel computations examined, the Redundancy lies within a narrow range of $1 \leq R(P) \leq 3$. Hence, from the TOP-form of a parallel computation, we can calculate PI and U, and estimate S and E as follows:

$$(PI,U)/3 \leq (S,E) \leq (PI,U)$$

The reader may feel that a redundancy of 3, or two hundred percent more operations, is actually quite large. But, considering that redundancies on the order of 10^4 [KUC78] have been found for some minimum-time minimax-parallel computations, a range of $1 \leq R \leq 3$ is relatively small.

Alternately, if the expected value of the Redundancy is known, say it is 2, then we can estimate S and E as follows:

$$(S,E) = (PI,U) / 2$$

Another significance of the redundancy measure was given by Sameh in [SAM77]: "The larger the redundancy of a parallel

algorithm, the larger the probability that the parallel algorithm is not as numerically stable as its sequential counterpart." Hence, it is not just aesthetically desirable to minimize the total number of operations performed to solve a given problem, but this is also important to maximize the speedup and efficiency, and to preserve numerical stability.

6. QUALITY

The Speedup, Efficiency and Redundancy give us three different aspects of the performance of a parallel computation with respect to an equivalent serial computation. Ideally, we would like to maximize the Speedup to obtain the greatest. speed improvement, maximize the Efficiency to obtain the most cost-effective computation, and minimize the Redundancy to reduce the total number of operations that must be executed to solve the given problem and to avoid numerical instability. It would be useful to have a single performance measure that incorporates the effects of speedup, efficiency and redundancy. This would be a measure of the "quality" of a computation, which is directly proportional to its Speedup and Efficiency, and inversely proportional to its Redundancy.

Definition 6.1:

The Quality, Q, of any parallel computation with respect to any serial computation is defined as:

$$Q(P,1) = \frac{S(P,1).E(P,1)}{R(P,1)} = \frac{O(1)^3}{T(P)^2.O(P).P} = \frac{T(1)^2.O(1)}{T(P)^2.O(P).P}$$

If the serial computation is an equivalent minimum-size serial computation, then:

$$Q(P,1') = \frac{S(P,1').E(P,1')}{R(P,1')} = \frac{(O_{min})^3}{T(P)^2.O(P).P}$$

In addition? if the parallel computation is a minimum-time minimax-parallel computation. then:

$$Q(P',1') = \frac{S(P',1').E(P',1')}{R(P',1')} = \frac{(O_{min})^3}{(T_{min})^2.O(P').P'}$$

We note that the units for quality may be regarded as "q-units per processor", since

$$Q = \frac{T^2.O}{T^2.O.P} = \frac{\text{steps}^2.\text{operations}}{\text{steps}^2.\text{operations.processors}} = \text{q-units/processor}$$

Theorem 6.1:

(a) $0 < Q(P,1)$

(b) $0 < Q(P,1') \leq S(P,1') \leq PI(P) \leq P$

(c) $1/(P'^2) \leq Q(P',1') \leq S(P',1') \leq PI(P') \leq P'$

The proof may be obtained by substituting the values of S, E and R from theorems 4.1, 4.2 and 5.1 respectively.

We include the definition of quality for arbitrary parallel and serial computations for the sake of completeness. In fact, we feel that in order for Q to be a meaningful measure of the quality of a parallel computation, it should be defined with respect to the minimum serial size, O_{min}. When the quality is so defined, it is always less than or equal to the corresponding speedup measure, as can be seen from parts (b) and (c) of theorem 6.1.

We notice that the numerator of $Q(P,1')$ is $(O_{min})^3$, which is "fixed" as far as the parallel computation is concerned. Hence, we may increase $Q(P,1')$ only by decreasing each component of the denominator, which is $T(P)^2.O(P).P$. This is equivalent to decreasing each component of the TOP-form of the parallel computation. Since there are two factors of the parallel execution-time, T(P), decreasing this is more important than decreasing the size, O(P), or the maximum degree of parallelism, P. This is reasonable since T(P) is the only component of the TOP-form

that is directly related to the speed improvement obtained by the parallel computation. The primary purpose for going from serial execution to parallel execution is to increase the speed of execution. which is the same as decreasing the parallel execution time. Reducing the number of processors, P, required for execution and reducing the size. O(P), are only of secondary importance. In other words, increasing the speedup is our primary goal, after which we try to increase the efficiency and reduce the redundancy.

Theorem 6.1 (parts (b) and (c)) also indicates that we have four successively refined measures of the performance improvement of a parallel computation with respect to an equivalent minimum-size serial computation.

First, we have the maximum processor bandwidth, or maximum speed, of P operations per step. This is the maximum degree of parallelism in the computation, and also indicates the maximum number of parallel processors required to execute the parallel computation.

Then, we consider the average actual processor bandwidth or average actual. speed of the computation. This is the average degree of parallelism per step, called the Parallelism Index, PI(P), of the parallel computation.

Third, we realise that parallel computations often perform more operations than the equivalent minimum-size serial computation, in order to increase the speed of execution. Taking this redundancy into account, we find the average effective processor bandwidth, or the average effective speed, of the computation. This is the Speedup, S(P,1'), measure, defined with respect to the minimum serial size, O_{min}.

QUALITY

Finally, in an attempt to define a single performance measure that takes into account mainly the speed improvement, but also the efficiency (or cost-effectiveness) and redundancy of parallel processing over serial processing, we defined the Quality, Q(P,l'), measure.

The maximum and average degrees of parallelism, P and PI, are characteristics of the parallel computation itself, and do not depend on its serial equivalent. The speedup and quality do depend on the equivalent minimum serial size, but they may still be used to compare any two computations, not just equivalent ones. This is possible because each computation will have been "normalised" with respect to its equivalent minimum-size computation, which has a speedup and quality of one.

Now, suppose we did not know the equivalent minimum serial size, $0_{min'}$ can we still say something about the quality of a parallel computation? Clearly we can do this by ignoring the redundancy, if any, and defining the quality in terms of the parallelism index and utilization, which are analogous to the speedup and efficiency, respectively.

Definition 6.2:

The maximum quality, Q(P), of a parallel computation is defined as

$$Q(P) = PI(P).U(P)$$

Theorem 6.2:

(i) $1/P \leq Q(P) \leq P$

(ii) $Q(P,l') \leq Q(P)$

Proof:

(i) From theorem 3.1, $1 \leq PI(P) \leq P$ and $1/P \leq U(P) \leq 1$, so

$$1/P \leq Q(P) \leq P$$

(ii) $$Q(P,1') = \frac{S(P,1').E(P,1')}{R(P,1')} \leq PI(P).U(P) = Q(P)$$

Clearly, the same result holds when P=P'.

Perhaps one of the main significances of the quality, Q(P',1'), measure is that it gives us a basis upon which we can decide whether a computation will benifit from parallel processing rather than serial processing. The quality of an optimal serial computation is one. We suggest, for example, that if the quality of an equivalent parallel computation is greater than one, than parallel processing is preferable to serial processing, since the quality of the computation is increased. It is interesting to note that about half the programs examined in [KUC78] have a quality less than one. Hence, parallel processing is preferable for half of the 355 programs examined, and serial processing produces the higher quality computation for the other half.

In practice, a computation center may choose a higher value of Q as the cutoff point for executing a given problem as a parallel computation rather than a serial one. For example, it may require a Quality of at least 1.5, or at least 50 percent increase i-n quality, before it considers that parallel processing is indicated.

We have seen in section 5 that increased redundancy is often correlated with numerical instability. It is clear that no matter how much we speed up a computation, this is futile if the required degree of accuracy in the results is lost. Even if accuracy were

not jeopardised, it seems that there must be some cutoff function involving at least the speedup and the redundancy, which determines when the increased redundancy does not iustify the increased speedup. For example, it seems unjustified to use a parallel computation that achieves a speedup of three at the cost of a redundancy of one hundred. One possible cutoff function is that the parallel computation is not justified if the redundancy is greater than the speedup. We suggest another possible cutoff function based on the quality measure, i.e., a parallel computation is "justified" if its quality, Q. satisfies $Q > 1$, or $S.E/R > 1$. This means that if, the redundancy is greater than or equal to the product of the speedup and the efficiency then the parallel computation is not justified.

Of course the question of whether parallel processing is "justified" for any given computation is highly dependent on the goals of the application concerned. In some applications, the reduction of execution-time is the only, often critical, goal, so that parallel processing is "justified" as long as the speedup, $S(P',1')$ is greater than one, no matter what the values of the redundancy, efficiency or quality. But in general, if we have to choose just one measure, then the quality is probably a better indicator of the overall performance improvement obtainable by parallel processing over serial processing, than just the speedup alone.

7. GENERALISATION TO A SET OF PARALLEL COMPUTATIONS

All the canonical forms and performance measures defined in sections 2 through 6, have been defined for a single parallel computation. In practice however, we are often more interested in the overall performance improvement for a representative set of computations, than just the performance improvement for any single computation. We are still interested in the same types of performance measures of PI,U,S,E,R and Q. Hence, in this section, we show how we may generalise the definitions to apply to a set of computations rather than to a single one.

First, we notice that all the performance measures PI,U,S,E,R and Q depend on knowledge of the TOP-form of a parallel. computation. Some, like S,E,R and Q also require knowledge of the size, f?(l) or O_{min}, of an equivalent serial computation. This is equivalent to knowing the TOP-form of an equivalent serial computation. Hence, all we need to do is to define the TOP-form for a set of computations, rather than for a single computation. We call this an aggregate TOP-form. Then, all the definitions for the performance measures, PI,U,S,E,R and Q willhold with respect to this aggregate TOP-form.

Definition 7.1:

The aggregate TOP-form of a set of n computations is defined as:

$$TOP^a(n) = (T^{mean}, O^{mean}, P^{max})$$

where T^{mean} is the average execution-time per computation, in steps,

O^{mean} is the average size per computation, in operations,

and P^{max} is the maximum degree of parallelism in all the computations.

We feel that P^{max} should be used rather than P^{mean}, since this is the maximum number of processors required to execute the computation/s with this largest maximum degree of parallelism. If each computation is optimised so that it is a minimum--time minimax-parallel computation, then P = P', the minimax degree of parallelism, for each computation. Then, it is reasonable to define the aggregate TOP-form in terms of P^{max} rather than P^{mean}, since if we execute the set of computations using $p < P^{max}$ parallel processors, we would not obtain the maximum possible speed improvement, for each computation.

In the case of the aggregate execution-time and size, T^{mean} and O^{mean}, if each of the computations in the set is executed with equal frequency, then we could just as well use T^{sum} and O^{sum}, since none of the performance measures, PI,U,S,E,R and Q, will be changed.

Definition 7.2:

Let $(r_1, r_2, \ldots, r_n)$ be the probability that computations $A_1, A_2, \ldots, A_n$, will be executed next. Then, if

$$r_1 = r_2 = \ldots = r_n,$$

we can define the aggregate TOP-form of this set of n computations as:

$$\text{"TOP}^a(n)\text{"} = (T^{sum}, O^{sum}, P^{max}),$$

where T^{sum} is the sum of the execution-times of all the computations, in steps,

O^{sum} is the sum of the sizes of all the computations, in operations,

and P^{max} is the maximum degree of parallelism in all the computations.

Theorem 7.1:

If the frequency of execution of each computation in a set of n computations is the same, then each of the performance measures, PI, U, S, E, R and Q has the same value whether it is defined with respect to the aggregate TOP-form, $TOP^a(n)$, or to the alternate aggregate TOP-form, "$TOP^a(n)$".

Proof:

It is only necessary to show true for PI and R, since S can be defined in terms of PI and R, U and E can be defined in terms of PT. and S respectively, and Q can then be defined in terms of S, E and R.

Let the performance measures defined with respect to the alternate aggregate TQP-form, "TOP^a", he enclosed within "quote" marks. Then,

$$\text{"PI}(P^{max})\text{"} = \frac{O^{sum}}{T^{sum}} = \frac{O^{sum}.n}{n.T^{sum}} = \frac{O^{mean}}{T^{mean}} = PI(P^{max})$$

$$\text{"R}(P^{max},1)\text{"} = \frac{O^{sum}}{O(1)^{sum}} = \frac{O^{sum}.n}{n.O(1)^{sum}} = \frac{O^{mean}}{O(1)^{mean}} = R(P^{max},1)$$

We have introduced the aggregate TOP-form defined in terms of the sums of the execution-times and sizes because this is often easier to understand and compute in practice. For example, the aggregate speedup, $S^a(P,1)$, is then the total time taken by all the computations executed serially divided by the total time taken by all the computations executed in parallel. But, because of theorem 7.1, we feel that the definition of the aggregate TOP-form

in terms of the means of the execution-times and sizes, as in Definition 7.1, is more powerful than that in terms of the sums, as in Definition 7.2. The former can describe the aggregate TOP-form of a set of computations, even when each computation does not have the same frequency of execution. The latter is only valid in the case of equally frequent computations, where it produces the same value for each of the performance measures, PI, U, S, E, R and Q, as the former.

We now examine the possible ranges of each component of the aggregate TOP-form defined with respect to means.

Theorem 7.2

Assuming that each computation in the set of computations is consistent with Theorem 2.1, then the aggregate TOP-form,

$TOP'' = (T^{mean}, O^{mean}, P^{max}) = (t,o,p)$

satisfies the following relationships:

(i) $o/p \leq t \leq o$

(ii) $t \leq o \leq p.t$

(iii) $o/t \leq p$

Proof:

(i) Let $\{r_1, r_2, \ldots, r_n\}$ be the probability distribution of the frequency of execution of the n computations in the set. Then,

$$t = r_1.t_1 + r_2.t_2 + \ldots + r_n.t_n$$

$$\geq \sum_{1 \leq i \leq n} r_i . \frac{o_i}{p_1}$$

$$\geq \frac{1}{p} \sum_{1 \leq i \leq n} r_i o_i$$

$$= o/p$$

$$t \leq \sum_{1 \leq i \leq n} r_i(o_i - p_i + 1)$$
$$= \sum_{1 \leq i \leq n} r_i \cdot o_i - \sum_{1 \leq i \leq n} r_i \cdot p_i + 1$$
$$= o - pmean + 1$$
$$\leq o$$

(ii) and (iii) follow directly from (i).

We see that the broader ranges given in the above theorem for aggregate TOP-forms include the ranges given in Theorem 2. I for individual TOP-forms. Also, by examining the proofs of Theorems 3. 1, 4. 1 and 5. I, we see that the ranges of the aggregate performance measures, PI^a, U^a, S^a, E^a, R^a and Q^a are the same as the ranges of the performance measures, PI, U, S, E, R and Q, for a single computation, when the values of the aggregate TOP-form and equivalent aggregate serial size O(1) are used.

We note that the overall performance measures, PI", U^a, S^a, E^a, R^a and Q^a, defined with respect to the aggregate TOP-form, $TOP^a(n)$, are quite different from just the averages of the performance measures for each computation. This is because, in general,

$$PI^a = \frac{O^{mean}}{T^{mean}} = \frac{r_1 \cdot O_1 + r_2 \cdot O_2 + \ldots + r_n \cdot O_n}{r_1 \cdot T_1 + r_2 \cdot T_2 + \ldots + r_n \cdot T_n}$$
$$\neq \frac{r_1 \cdot O_1 + r_2 \cdot O_2 + \ldots + r_n \cdot O_n}{\frac{T_1}{} \quad \frac{T_2}{} \quad \frac{T_n}{}}$$
$$= PI$$

where $(r_1, r_2, \ldots, r_n)$ is the probability distribution of the frequency of execution of the n computations in the set.

It seems reasonable that the overall performance should be weighted not only by the relative frequencies of each computation in

the set, but also by the magnitudes of the sizes and execution-times of each computation. Surely the overall performance is more affected by a longer computation than by a shorter one. The definition of an aggregate performance measure with respect to the aggregate TOP-form, $TOP^a(n)$, takes this into account, unlike the simple averages of the performance measure for each computation.

Example 7.1:

Let A and B be two parallel computations with TOP-forms:

A: (100,1000,40)

B: (5, 200,100)

Suppose A and B are executed with equal frequency,

then $PI_A(40) = \frac{1000}{100} = 10$

$PI_B(100) = \frac{200}{5} = 40$

and $\overline{PI} = (10+40) / 2 = 25.$

but $PI^a = \frac{1200}{105} = 11.43$

We note that the aggregate measure. PI^a, gave more weight to the parallelism index of A, the larger computation, in terms of both the size C(P), and the execution-time, T(P).

8. SUMMARY

In this paper, we have tried to find simple and effective ways to represent the parallelism inherent in computations, which facilitate the calculation of important performance characteristics. We introduced. the canonical form called the Parallelism Profile which substantially reduced the complexity of modelling an arbitrary parallel computation. The Parallelism Profile retains only the information on the frequency of different degrees of parallelism in the computation. The most succinct representation of a parallel computation, for the purposes of calculating performance measures defined in this paper, is given by the TOP-form of the computation.

We defined quantitative measures which characterise the absolute and relative performance of a parallel computation, compared with an equivalent serial computation. The absolute performance measures are the Parallelism Index, PI (P), the Utilization. U(P), and the maximum Quality, Q(P). The corresponding relative performance measures are the Speedup, S (P, 1), the Efficiency, E(P,1), and the Quality, Q(P,1). We also examine the range of permissible values for each performance measure.

Ideally, we would I ike to compare an optimal. parallel computation with an optima! equival ent serial computation, in order to determine the performance improvements due sol ely to parallel versus serial processing. Toward this end, we define an optimal serial computation as a "minimum-size" computation, where the size or length of a computation is the total number of operations executed. We define an optimal parallel computation to be a "minimum-time minimax-parallel" computation, or a computation which achieves the minimum execution-time using the least number of

parallel processors. We also show how such optimality may be approximated in practice.

We discussed how the logical parallelism, P, may be interpreted as the maximum speed of the computation, in operations executed per step. Then, the Parallelism Index, PI(P), may be interpreted as the average actual speed of the computation, and the speedup, S(P,1′), may be interpreted as the average effective speed of the computation. This is because PI(P) is the speed measured with respect to the actual number of operations executed in the parallel computation, or the actual size of the parallel computation, whereas S(P,1′) is the speed measured with respect to the minimum size of an equivalent optimal serial computation. Hence, the speedup measure takes into account the redundant operations in the parallel computation, which were introduced in order to reduce the parallel execution time. The Parallelism Index is an upper bound for the range of attainable values of the Speedup, S(P,1′), of any parallel computation when compared to an optimal equivalent serial computation. This is very useful since it is often not possible to calculate S(P,1′) exactly as this requires finding the minimum size, O_{min}. PI(P) may be calculated from knowledge of just the size and execution-time of the parallel computation itself.

Although speed is the performance characteristic we are most interested in, a measure of the cost-effectiveness of executing a computation on a parallel processor organization is also important in practice. We defined Utilization as the proportion of the total processor-time space, P*T(P), actually used in executing operations of the parallel computation. The corresponding performance measure relative to the minimum size of an equivalent serial computation is the Efficiency measure, E(P,1). Again, if the serial computation chosen for comparison is an optimal one, then the Utilization, U(P), is an upper bound for the Efficiency, E(P,1′).

SUMMARY

It is Important to realise that if we measure speedup, S(P,1), and efficiency, E(P,1), with respect to a nonoptimal. serial computation, then it is possible to have a speedup greater than P, and an efficiency greater than one. In these cases. the values do not reflect the performance improvements due solely to parallel versus serial. processing, since part of the improvements are due to optimising the serial computation itself.

We maintain that if the parallel computation is an optimal one, then the equivalent serial computation chosen for execution should also be optimal. This means that we are interested in S(P′,1′) values but not in S(P′,1) values, for reasons given in the previous paragraph. However, we are interested in the case S(P,1′), where the serial. computation is optimal but the parallel computation is not. This is because we often have to use a nonoptimal parallel computation in practice, if the minimax logical parallelism P' of the computation exceeds that of the physical parallelism in terms of the number of parallel processors available. The case of S(P,1), where both serial and parallel computations may he nonoptimal is also studied in this paper, since these are the results usually given in the literature [KUC73,KUC76,KUC78,LEE76].

We showed that the absolute and relative speeds and efficiencies are related by the appropriate redundancy measure:

$$(PI,\ U) = R\ .\ (S,\ E)$$

Ideally, we would like to maximize the speed of the parallel computation, maximize its efficiency and minimize its redundancy. We defined the Quality, Q(P,1), of a parallel computation to be the product of the speedup and the efficiency divided by the redundancy. Assuming that the serial computation used for comparison is optimal.,

then the quality, $Q(P,1')$ has an upper bound given by the speedup, $S(P,1')$. Since optimal serial computations have speeds and qualities of one, it is possible to have parallel computations with speeds greater than that of the optimal equivalent serial. computations but qualities less than the corresponding optimal serial computations. Hence, the quality measure is a more stringent measure of the overall performance improvement obtained by going from serial to parallel processing of any given computation.

Finally, we show how al.1 the canonical forms and performance measures may be generalised from one computation to a set of computations, to obtain "aggregate" canonical forms and performance charaterizations. We show that the aggregate performance measures are different In value from the average performance measures, although both are useful characterizations of the performance potential of a set of computations.

We summarize the definitions and ranges of the performance measures under the categories of speed, efficiency, redundancy and quality, in Table 8.1.

ACKNOWLEDGEMENTS:

I am deeply indebted to Professor David Kuck, Robert Kuhn, Michael Wolfe, Penn Chung, Bruce Leasure and other members of the "Parafrase" project at the University of Illinois, Urbana-Champaign, for permission to use raw data from an unpublished database of about 163 different parallel programs. Some of these programs with loops

were repeated for varying numbers of iterations of these loops to arrive at a grand total of 355 parallel. programs. All these parallel programs were automatically generated from ordinary serial Fortran programs by a sophisticated "compiler", or analyser, which incorporates the best algorithms developed over the past eight years by Professor Kuck and his graduate students. The "raw data" which we have used to calculate the few performance values quoted in this paper form a very small subset of the data collected in the Illinois database. We merely extracted the values for what we have defined as the TOP-form of each parallel computation, and the equivalent (possibly nonoptimal.) serial size, O(1). The database contains a lot of other information pertaining to the code characteristics of the program, like the number and sizes and degrees of nesting of DO-loops, the numbers of various categories of IF-statements, etc. This database represents probably the largest and most significant experimental work on the speedup of existing serial programs by parallel processing. We are indeed grateful_ for the opportunity to analyse this excellent experimental data, and bave gained much insight from it.

I would also like to thank my research adviser, Professor Michael Flynn, for many interesting discussions and useful comments, for his help in arranging suitable research and computing facilities, and especially for his constant implicit and explicit encouragement.

I would also like to thank my husband and my one-year-old son, Howard and Patrick, for their understanding and cheerful help whenever I had to work.

TABLE 8.1: SUMMARY OF PERFORMANCE MEASURES

Required Information	$(T(P), O(P), P)$	$(T(P), O(P), P)$ and $O(1)$	$(T(P), O(P), P)$ and O_{min}	$(T_{min}, O(P'), P')$ and O_{min}
Computations	any parallel no serial	any parallel any serial	any parallel optimal serial	optimal parallel optimal serial
SPEED	$PI(P)=\frac{O(P)}{T(P)}$	$S(P,1)=\frac{O(1)}{T(P)}$	$S(P,1')=\frac{O_{min}}{T(P)}$	$S(P',1')=\frac{O_{min}}{T_{min}}$
	$1 \leq PI(P) \leq P$	$0<S(P,1) \leq O(1)$	$0<S(P,1') \leq PI(P)$	$1 \leq S(P',1') \leq PI(P')$
EFFICIENCY	$U(P)=\frac{PI(P)}{P}$	$E(P,1)=\frac{S(P,1)}{P}$	$E(P,1')=\frac{S(P,1')}{P}$	$E(P',1')=\frac{S(P',1')}{P'}$
	$\frac{1}{P} \leq U(P) \leq 1$	$0<E(P,1) \leq \frac{O(1)}{P}$	$0<E(P,1') \leq U(P)$	$\frac{1}{P'} \leq E(P',1') \leq U(P')$
REDUNDANCY		$R(P,1)=\frac{O(P)}{O(1)}$	$R(P,1')=\frac{O(P)}{O_{min}}$	$R(P',1')=\frac{O(P')}{O_{min}}$
		$0<R(P,1)$	$1 \leq R(P,1')$	$1 \leq R(P',1') \leq 1/E(P',1') \leq P'$
QUALITY	$Q(P)=PI(P).U(P)$	$Q(P,1)=\frac{S(P,1).E(P,1)}{R(P,1)}$	$Q(P,1')=\frac{S(P,1').E(P,1')}{R(P,1')}$	$Q(P',1')=\frac{S(P',1').E(P',1')}{R(P',1')}$
	$\frac{1}{P} \leq Q(P) \leq P$	$0<Q(P,1)$	$0<Q(P,1') \leq Q(P)$ (also $\leq S(P,1')$)	$\frac{1}{P'^2} \leq Q(P',1') \leq Q(P')$ (also $\leq S(P',1')$)

9. REFERENCES

[AHO76] Aho,A.,Hopcroft,J.,Ullman,J., "The Design and Analysis of Computer Algorithms", Addison-Wesley, June 1974.

[BER66] Bernstein,A., "Analysis of Programs for Parallel Processing", IEEE Transactions Electronic Computers PC-15, pp.757-763, 1966.

[CAX72] Caxton,C., and Riseman,E., "Percolation of Code to Enhance Parallel Dispatching and Execution", IEEE Transactions on Computers, December 1972.

[CHE75] Chen,S.C., and Kuck,D., "Time and Parallel Processor Bounds for Linear Recurrence Systems", IEEE Transactions on Computers, c-24 no. 7, pp.701-717, July 1975.

[FLY72] Flynn,M., "Some Computer Organizations and Their Effectiveness", IEEE Transactions on Computers, c-21, pp.948-960, Sept. 1972.

[FLY74] Flynn,M., "Trends and Problems in Computer Organizations", IFIPS Congress 1974 (invited), Stockholm, Sweden, August 1974, North-Holland Pub. 1975, pp.2-10.

[HOP69] Hopcroft,J., and Ullman,J., "Formal Languages and Their Relation to Automata", Addison-Wesley, 1949.

[KAR66] Karp,R., and Miller,R., "Properties of a Model for Parallel Computations: Determinacy, Termination, Queuing", SIAM Journal Appl. Math., Vol 14 No 4, November 1944.

REFERENCES

[KUC73] Kuck,D.,Budnik,P.,Chen,S.C.,Davis,E.,Han,J.,Kraska,P.,Lawrie,D., Muraoka,Y.,Strebendt,R.,and Towle,R., "Measurements of Parallelism in Ordinary Fortran Programs", 1973 Sagamore Computer Conference on Parallel Processing.

[KUC74] Kuck,D., "On the Speedup and Cost of Parallel. Computation", in "Complexity of Computational Problem Solving", (R. Anderson and R. Brent, eds.), Australian National University, Canberra.

[KUC76] Kuck,D., "Parallel Processing of Ordinary Programs", in Advances in Computers, Vol. 15, Academic Press,Inc., 1976.

[KUC78] Kuck,D.,Kuhn,R.,Wolfe,M.,Chung,P.,and Leasure,B., "A Database of Parallel Programs Automatically Generated from Ordinary Fortran Programs", informal data provided September 1978, unpublished, used with permission.

[LEA76] Leasure, B., "Compiling Serial Languages for Parallel Machines", Report No. 805, Department of Computer Science, University of Illinois at Champaign-Urbana, November 1976.

[LEE76] Lee,R., "Performance Bounds for Parallel Processors", Stanford University DSL Technical Report No. 125, Nov. 1976.

[LEE77] Lee,R., "Performance Bounds in Parallel Processor Organizations", Proceedings of the Symposium on High Speed Computer and Algorithm Organization, April, 1977. Academic Press pub. 1977, (D. Kuck,D. Lawrie,A. Sameh, eds.).

[LEE78] Lee,R., "Optimal Program Control Structures Based on the Concept of Decision Entropy", Stanford University DSL Technical Report No. 156, July 1978.

REFERENCES

[MAN 74] Manna,Z., "Mathematical Theory of Computation", McGraw-Hill, 1974.

[MIN 67] Minsky,M., "Computation: Finite and Infinite Machines", Prentice-Hall,Inc., 1967.

[RIS 72] Riseman,E., and Caxton,C., "The Inhibition of Potential Parallelism by Conditional Jumps", IEEE Transactions on Computers, December 1972.

[SAM 77] Sameh,A., "Numerical Parallel Algorithms - A Survey", Proceedings of the Symposium on High Speed Computer and Algorithm Organization, April, 1977, (invited), Academic Press pub. 1977, (D. Kuck, D. Lawrie, A. Sameh, eds.).

[TOW 76] Towle,R., "Control and Data Dependence for Program Transformations", Report No. 788, Department of Computer Science, University of Illinois at Champaign-Urbana, March 1976.