Performance Measures for Evaluating Algorithms for SIMD Machines

LEAH J. SIEGEL, MEMBER, IEEE, HOWARD JAY SIEGEL, MEMBER, IEEE, AND PHILIP H. SWAIN, SENIOR MEMBER, IEEE

Abstract—This paper examines measures for evaluating the performance of algorithms for single instruction stream–multiple data stream (SIMD) machines. The SIMD mode of parallelism involves using a large number of processors synchronized together. All processors execute the same instruction at the same time; however, each processor operates on a different data item. The complexity of parallel algorithms is, in general, a function of the machine size (number of processors), problem size, and type of interconnection network used to provide communications among the processors. Measures which quantify the effect of changing the machine-size/problem-size/network-type relationships are therefore needed. A number of such measures are presented and are applied to an example SIMD algorithm from the image processing problem domain. The measures discussed and compared include execution time, speed, parallel efficiency, overhead ratio, processor utilization, redundancy, cost effectiveness, speed-up of the parallel algorithm over the corresponding serial algorithm, and an additive measure called "price" which assigns a weighted value to computations and processors.

Index Terms—Algorithm evaluation, parallel processing, performance measures, reconfigurable systems, SIMD machines.

I. INTRODUCTION

ADVANCES in technology have made feasible the design of large-scale parallel processing systems. Machines with as many as 9216 processors currently exist [6], and systems with 2^{14} processors have been proposed [3], [14]. SIMD (single instruction stream–multiple data stream) machines [8] represent one type of parallel processing system. An *SIMD* machine typically consists of a control unit, a set of N processing elements (*PE*'s) (each a processor with its own memory), and an interconnection network [17]. The control unit broadcasts instructions to all PE's, and each active PE executes each of these instructions on the data in its own memory. Each instruction is executed simultaneously in all active PE's. The *interconnection network* allows data to be transferred among the PE's. *Masks* are used to enable and disable PE's.

For application areas such as image processing, where the same operation is often repeated thousands of times in processing the image array, and speech processing, where computationally intensive matrix and vector operations are frequently used, SIMD machines promise to be valuable tools for reducing the amount of computation time needed. It has been shown that SIMD algorithms can realize substantial time savings over the corresponding serial algorithms for image processing tasks such as smoothing [19], histogram calculation [19], two-dimensional FFT [12], [23], image correlation [24], and feature extraction [27], and speech processing tasks such as linear predictive coding [21] and the one-dimensional FFT [23].

The complexity of SIMD algorithms is, in general, a function of the problem size (number of elements in the data set to be processed), machine size (number of PE's), and the interconnection network used to provide communications among the PE's. For example, an algorithm which uses N PE's to execute some operation on an $M \times M$ image will exhibit different "performance" for different values of N. Measurements of this performance can be used in various ways.

The most obvious use of performance measures is for selecting from alternative SIMD algorithms. For a given SIMD machine, different algorithms for performing a particular task can be compared. The algorithm which performs best based on the desired measurement criteria can then be chosen.

As another use for performance measures, consider the situation where an SIMD machine is being designed for a particular application or class of applications, so that typical values of M are known. Then a measure of the way in which the machine size affects the performance of the application algorithms will be useful in deciding how many PE's the system should have.

Lastly, consider proposed reconfigurable systems in which it is possible to vary the number of PE's which act together as a virtual SIMD machine (e.g., [13], [16], [18], [19], [28]). Given such a reconfigurable system, the machine size can be tailored to the problem size for execution of a given algorithm if there exist performance criteria for comparing different choices of machine size.

One difficulty in quantifying the effect of changing the problem-size/machine-size/network-type relationships arises because a number of different performance measures can be applied. The goal of this paper is to collect measures that have been proposed, define some new measures, and study the relationships among the various measures. In the following sections, nine such measures are presented, compared, and discussed.

Manuscript received March 12, 1981; revised November 25, 1981. This work was supported by the National Science Foundation under Grant ECS-7909016, the U.S. Air Force Office of Scientific Research, Air Force Systems Command, under Grant AFOSR-78-3581, and the Defense Mapping Agency, monitored by the U.S. Air Force Rome Air Development Center, Information Sciences Division, under Contract F30602-78-C-0025 through the University of Michigan, Ann Arbor. Portions of this work were presented at the Workshop on Applications of Nonconventional Computers in Image Processing: Algorithms and Programs, Madison, WI, May 1981.

The authors are with the School of Electrical Engineering, Purdue University, West Lafayette, IN 47907.

0098/5589/82/0700-0319$00.75

In Section II the performance measures to be considered are defined. These measures include execution time, speed, parallel efficiency, overhead ratio, processor utilization, redundancy, cost effectiveness, speed-up of the parallel algorithm over the corresponding serial algorithm, and an additive measure called "price" which assigns a weighted value to computations and processors. To demonstrate one way in which the measures can be applied, an SIMD algorithm to calculate a global histogram of an image is presented in Section III. In this example algorithm evaluation, both the image size and the machine size are varied, permitting the performance of the algorithm to be examined and compared under a variety of conditions.

II. Performance Measures

In this section a number of different measures are defined by which the performance of a parallel algorithm can be evaluated. The performance measures can be divided into three groups. The first two measures, execution time and speed, deal with how fast the parallel algorithm is, i.e., how many data points it can process per unit time. The next five measures consider how "effectively" the parallel system is used. The speed-up and efficiency measures compare the time performance of a parallel algorithm to that of the corresponding serial algorithm. The overhead ratio considers the time introduced by operations required because the algorithm is to be executed on a system having more than one processor. Processor utilization quantifies how fully the available resources are used. Redundancy is a measure of how many redundant computations are performed in the parallel algorithm. The final two measures, cost effectiveness and price, incorporate information about the system cost into the performance criteria. The performance measures to be defined in this section are summarized in Table I.

A. Execution Time

The *execution time* $T_N(M)$ is a measure of the time involved in performing the algorithm for a problem of size M on a system having N PE's. In general, $T_N(M)$ will be a function of both M and N.

$T_N(M)$ can be expressed as the sum of two components: $c_N(M)$, which is the time spent performing computations which are actually a part of the task being performed, and $o_N(M)$, which is the "overhead" or time spent performing operations required to "manage" the parallelism. The two main sources of overhead are generally the interconnection network operations performed to transfer data from one PE to another and masking operations performed to control which PE's execute an instruction.

When evaluating the execution time of a parallel algorithm or program, the distinction between response time and throughput may become an issue. Response time of a job is defined to be the length of the interval between the request for its execution and the return of its results [5]. Throughput is defined to be the number of jobs executed per unit time [5]. Consider the case where A is an algorithm which can use one or N PE's, and A is to be executed on N different data sets, each of size M. By assigning a complete data set to each PE, N instances of the 1 PE algorithm can be performed in parallel. The total execution time will be $T_1(M)$. The response time will be $T_1(M)$, and the throughput will be $N/T_1(M)$. Alternatively, the N PE's together can process one data set at a time, performing the algorithm N times. If it is assumed that the algorithm can operate on one data set while another data set is being loaded into the PE memories (e.g., via double buffered memories [19]), then the total execution time will be $N * T_N(M)$. The expected value of the response time for processing of the jth data set, for arbitrary j, $1 \leqslant j \leqslant N$, will be

$$\frac{1}{N}\sum_{i=1}^{N} T_N(M)\, i = \frac{N+1}{2}\, T_N(M)$$

and the throughput will be $1/T_N(M)$. For an SIMD algorithm, $T_N(M) \geqslant T_1(M)/N + o_N(M)$. In some cases, the expected value of the response time may therefore be less for the N replications of the N PE algorithm than for the one-PE-per-data-set scheme. For example, for large N (so that $N \simeq N+1$), this will be the case when $c_N(M) = T_1(M)/N$ and $o_N(M) < T_1(M)/N$. On the other hand, the throughput rate will never be greater for the N PE algorithm, and its throughput rate will equal that of the one-PE-per-data-set implementation only when $T_N(M) = T_1(M)/N$ and $o_N(M) = 0$.

It should be noted that the 1 PE algorithm requires two conditions that, in general, may not be true. These are: 1) that there are N data sets to be processed, and 2) that each PE has sufficient memory space (and/or a memory management system) to allow it to handle an entire data set.

B. Speed

The *speed* $V_N(M)$ is defined to be the number of data points processed per unit time: $V_N(M) = M/T_N(M)$. Assuming that at least one instruction is needed to process a data point and that a unit of time is equal to the instruction cycle time of a PE, then the maximum attainable speed is N. The reciprocal of the speed, $T_N(M)/M$, is the execution time per data point.

C. Speed-Up

The *speed-up* $S_N(M)$ of an N PE algorithm over the corresponding 1 PE algorithm is the ratio $S_N(M) = T_1(M)/T_N(M)$ [9]. The speed-up is said to be *ideal* if $S_N(M) = N$. Since $T_N(M) \geqslant T_1(M)/N$, and $T_N(M) = c_N(M) + o_N(M)$, ideal speed-up implies that $o_N(M) = 0$. The speed-up measured only over the computation operations is the ratio $T_1(M)/c_N(M)$. The speed-up on computations is ideal if $T_1(M)/c_N(M) = N$.

An example of a task for which ideal speed-up can be obtained is maximum likelihood classification [26] of imagery data. In this task each pixel (picture element) in the image is classified independently of the others. Thus, the image can be subdivided among the processors, each processor classifying its own subimage. Neither interprocessor data transfers nor masking are required. Obviously, not all tasks are such that ideal speed-up can be obtained [22].

The speed-up $S_N(M)$ can be written as $V_N(M) * (T_1(M)/M)$, where for a fixed value of M, $T_1(M)/M$ will be a con-

TABLE I
SIMD ALGORITHM PERFORMANCE MEASURES
(N = NUMBER OF PE's; M = PROBLEM SIZE)

Name of Measure	Definition	Ideal	Comments
Execution Time	$T_N(M) = c_N(M) + o_N(M)$	$T_N(M) = T_1(M)/N$	$c_N(M)$ = computation time $o_N(M)$ = overhead time
Speed	$V_N(M) = M/T_N(M)$	$V_N(M) = MN/T_1(M)$	data points processed per unit time
Speedup	$S_N(M) = T_1(M)/T_N(M)$	$S_N(M) = N$	speedup on computations: $T_1(M)/c_N(M)$
Efficiency	$E_N(M) = S_N(M)/N$ $= T_1(M)/(N*T_N(M))$	$E_N(M) = 1$	$0 < E_N(M) \leq 1$
Overhead Ratio	$O_N(M) = o_N(M)/T_N(M)$	$O_N(M) = 0$	
Utilization	$U_N(M) = \sum_{x=0}^{X-1} t_x p_x/(N*T_N(M))$	$U_N(M) = 1$	t_x = time perform step x p_x = no. of PEs active for step x
Redundancy	$R_N(M) = \sum_{x=0}^{X-1} t_x p_x/T_1(M)$	$R_N(M) = 1$	$R_N(M) \geq 1$
Cost-effectiveness	$CE_N(M) = V_N(M)/C_N$		C_N = cost of N-PE system
Price	$P_N(M) = p_t*T_N(M) +$ $p_i*(Np_{PE} + \mu N\log_2 Np_S)$		p_t = cost of a unit of execution time p_{PE} = cost of a PE p_S = cost of a network switch p_i relates total implementation cost to hardware costs μ reflects number of switches
Generalized Price	$P'_N(M) = \frac{\alpha}{\alpha+1}*p_t*T_N(M)$ $+ \frac{1}{\alpha+1}*p_i*(Np_{PE} + \mu N\log_2 Np_S)$		α reflects relative importance of time and implementation costs

stant. For a given value of M, there will therefore be a direct relationship between $V_N(M)$ and $S_N(M)$, for all N.

D. Efficiency

The *efficiency* $E_N(M)$ of an N PE algorithm is defined to be $E_N(M) = S_N(M)/N = T_1(M)/(N * T_N(M))$ [9]. For SIMD algorithms, $E_N(M) \leq 1$ since it will necessarily be true that $T_N(M) \geq T_1(M)/N$. $E_N(M) = 1$ for $N = 1$; for $N > 1$, $E_N(M) = 1$ if and only if the speed-up on computations is ideal and the overhead $o_N(M)$ is 0. The efficiency therefore weights the N PE execution time by N and compares this weighted time to the serial time. Intuitively, the efficiency is a measure of how the achieved speed-up ($S_N(M)$) compares to the ideal speed-up (N).

The efficiency has an interesting form in the special case when the speed-up on computations is ideal. For this case $T_N(M)$ can be written as $T_1(M)/N + o_N(M)$, and $E_N(M) = T_1(M)/(T_1(M) + N * o_N(M))$. The efficiency can therefore be maximized by minimizing the product $N * o_N(M)$. In many cases, for fixed M, $N * o_N(M)$ will be a monotonically increasing function of N, so $E_N(M)$ will have its only maximum at $N = 1$. In such cases maximizing the efficiency may also maximize the execution time, since only one PE will be used. In these cases the execution time measure together with rate of change of E with respect to N, $dE_N(M)/dN$, may be of interest.

E. Overhead Ratio

The *overhead ratio* is defined here to be the ratio of overhead time to total execution time: $O_N(M) = o_N(M)/T_N(M)$. The overhead ratio will be 0 if and only if $o_N(M) = 0$. If the speed-up on computations is ideal (i.e., $T_1(M)/c_N(M) = N$), then $O_N(M) = 1 - E_N(M)$.

F. Utilization

The *utilization* $U_N(M)$ is a common measure which for an SIMD system refers to the fraction of time during which the PE's are busy executing computations of the algorithm. In general, this requires counting the number of PE's active for each computation step. Assume that for a problem of size M, there are X steps (sequential operations) in the N PE computation. With each step x, $0 \leq x < X$, can be associated a time t_x to perform the step and a count p_x of the number of PE's which are active. (The computation time $c_N(M) = \sum_{x=0}^{X-1} t_x$). The utilization is formally defined here as

$$U_N(M) = \sum_{x=0}^{X-1} t_x p_x/(N * T_N(M)).$$

In the special case when the speed-up on computations is ideal, $p_x = N$ for all x, so $U_N(M) = c_N(M)/T_N(M)$. In this case $U_N(M) = 1 - O_N(M) = E_N(M)$, where $O_N(M)$ is the overhead ratio and $E_N(M)$ is the efficiency. The utilization will be 1 if and only if the speed-up on computations is ideal and the overhead is 0.

G. Redundancy

The *redundancy* $R_N(M)$ is defined here to be the ratio

$$R_N(M) = \sum_{x=0}^{X-1} t_x p_x/T_1(M).$$

(If the simplifying assumption that all operations take one time unit to execute is made, then this is equivalent to the redundancy measure defined in [11].) If it assumed that comparable operations require the same amount of time whether executed on one PE or N PE's (e.g., that the time to perform an addition of two numbers does not depend on the number of PE's), and if it is assumed that all of the computation performed in the serial algorithm must also be performed in the N PE algorithm, then $\sum_{x=0}^{X-1} t_x p_x \geqslant T_1(M)$. Therefore, $R_N(M) \geqslant 1$, and $R_N(M) = 1$ if and only if $N = 1$ or the speed-up on computations is ideal, i.e.,

$$\sum_{x=0}^{X-1} t_x p_x / T_1(M) = N * c_N(M)/T_1(M) = 1.$$

The redundancy does not include any information about the overhead in the algorithm, but rather gives a measure of redundant computations that are performed. It therefore cannot, in general, be related to the other measures. However, if the overhead is 0, then $R_N(M) = 1$ if and only if $U_N(M) = 1$. If the overhead is 0 and the speed-up on computations is ideal, then $S_N(M) = N$, and $E_N(M) = U_N(M) = R_N(M) = 1$.

H. Cost Effectiveness

The *cost effectiveness* $CE_N(M)$ is a common notion which is defined here to be the ratio of the speed $V_N(M)$ to the "cost" C_N of the N PE system: $CE_N(M) = V_N(M)/C_N$. The cost can be computed approximately as the sum

$$C_N = \text{cost of control unit} + N * (\text{cost per PE}) + \text{cost of network.}$$

(It is assumed that PE memory costs are included in the cost of the PE.) The cost of the control unit will be a constant. The control unit will be more complex than a PE in some respects, e.g., it will have the ability to execute conditional and loop statements. In other respects, it may be simpler, e.g., it may not need floating point hardware, whereas if the system is to be used to perform operations such as FFT's, the PE's may have floating point hardware. Therefore, for discussion purposes, the control unit cost will be assumed to equal the cost of one PE. For large N, the control unit cost will therefore be negligible. For certain systems this simplifying assumption may not be true, in which case the control unit cost should be accounted for explicitly.

It may also be possible to further specify the network cost. For a multistage SIMD network such as the omega [10], indirect binary n-cube [14], generalized cube [20], or STARAN flip [1], [2], the network for an N PE machine will consist of $\log_2 N$ stages of switches, with $N/2$ switches per stage. The network cost will then be $(N/2)\log_2 N *$ (cost per switch). For the data manipulator [7] and augmented data manipulator [20], which have $\log_2 N$ stages of switches and N switches per stage, the cost will be $N \log_2 N *$ (cost per switch). Within a network, all switches will cost the same; switches for two different networks may differ in cost. In general, the cost for a multistage network can be represented as $\mu * N \log_2 N *$ (cost per switch), where μ is a constant depending on the number of switches in the particular network. For a nearest neighbor network, the cost will be $\sigma * N *$ (cost per switch), where σ is a constant reflecting the number of neighbors (e.g., 4 [4] versus 8 [15]).

To reflect the fact that the cost per hardware element may decrease as the number of elements increases due to economies of scale, it may be realistic to weight C_N by a scale factor which decreases as N increases. For simplicity, it is assumed here that the hardware cost is linear in the number of PE's and network switches.

It may also be of interest to consider the rate at which the cost effectiveness changes as a function of the number of processors: $dCE_N(M)/dN$. In general, the cost C_N will be a function of N, and the speed $V_N(M)$ will be a function of N and M. By a (possibly unwieldy) change of variables, for a given cost effectiveness, the speed could be written as a function of C_N and M. It would then be possible to obtain an expression for the derivative of the speed with respect to cost: $d(\text{speed})/d(\text{cost})$. From such a measure, two key types of questions could be answered.

1) To increase the speed by a factor of f, what will be the increase in cost?

2) If the cost is decreased by a factor of g, what will be the resulting decrease in speed?

I. Price

Finally, a figure of merit is introduced here which, like cost effectiveness, integrates considerations related to both speed of computation and system cost. In addition, this performance measure makes it possible to account for the diverse factors which contribute to the cost, in a generalized sense, of implementing and executing an algorithm on a parallel processing system. The following viewpoint is adopted. First, it is postulated that a processing task should be accomplished as fast as possible because a monetary charge may be levied for utilizing the processing system (in any case, computational resources are being consumed). Therefore, to quantify this notion, let p_t be the cost associated with using a unit of time to carry out the computation on the parallel processing system. Clearly, p_t will reflect a composite of factors related to the variable costs of operating the system. (It will be assumed that none of these factors depends on the size of the system, as another term will be introduced to account for such dependence.) The total cost of the computation based on these factors will be $p_t * T_N(M)$, and in general may be expected to decrease as the size of the system, N, increases, since the execution time should decrease with increasing N. The impetus is therefore to make N large.

On the other hand, increasing N increases the total cost of hardware required for the system and also the costs of design and implementation. For the sake of specificity and simplicity it will be assumed that the overall system cost can be linearly related to the number of PE's (N) and the number of interconnection network switches (say, $\mu N \log_2 N$) in the system. Let p_{PE} and p_s be, respectively, the cost of a PE and a switch, and let p_i be a proportionality constant relating the cost of the hardware components to the total cost of implementing

the system. Then the total implementation cost is $p_i * (Np_{PE} + \mu N \log_2 Np_s)$. To minimize implementation cost, the impetus here is to make N small.

The total cost associated with implementing and using a parallel processing system consisting of N PE's and a $\log_2 N$-stage network to perform the desired computation for a problem of size M will be called the *price* of the computation and, from the preceding discussion, is given by

$$P_N(M) = p_t * T_N(M) + p_i * (Np_{PE} + \mu N \log_2 Np_s).$$

The optimal size of the system to be used can be determined by minimizing the price with respect to N.

In some applications, e.g., weather forecasting and ballistic missile threat detection, the cost related to execution time and the cost due to system implementation may not be of equal importance. For example, the system designer may be prepared to pay a several-fold increase in implementation cost for even a modest improvement in speed. Therefore, it is useful to go one step further and define a *generalized price:*

$$P'_N(M) = \frac{\alpha}{\alpha + 1} * p_t * T_N(M) + \frac{1}{\alpha + 1} * p_i * (Np_{PE} + \mu N \log_2 Np_s)$$

where the parameter α ($\alpha \geq 0$) reflects the relative importance of speed and time-variable costs versus system costs. In many cases, α will be a subjectively chosen factor based on cost and speed constraints for a particular application.

Example: It is desired to implement a parallel processing computer system to perform feature extraction operations on large-area, high-resolution image data. The system must handle a very large volume of imagery.

Assume that it is known from past experience that to fabricate and program an N PE system, implementation costs generally run about 20 times the hardware cost. Thus, $p_i = 20$. Variable costs associated with power and cooling for the system, plus maintenance and operation might be on the order of \$100/h. This figure would be established based on the expected system lifetime, estimated percent utilization, reliability data, service contract costs, etc. Thus, $p_t = \$100$ and $T_N(M)$ is expressed in terms of hours. Then the expression for the price of the system is

$$P_N(M) = 100T_N(M) + 20(Np_{PE} + \mu N \log_2 Np_s)$$

where p_{PE} and p_s are the dollar costs of the hardware components. Once $T_N(M)$ is determined, the optimal system size is the value of N which minimizes $p_N(M)$.

If the application were such that execution time was a much more critical issue than the actual cost of the system, i.e., the user was willing to pay considerably (but not unboundedly) for increased system speed, then the generalized price concept could be invoked to help arrive at the appropriate system size. For this example, letting $\alpha = 4$, the "optimal" system size would be the value of N minimizing

$$P'_N(M) = 80T_N(M) + 4(Np_{PE} + \mu N \log_2 Np_s).$$

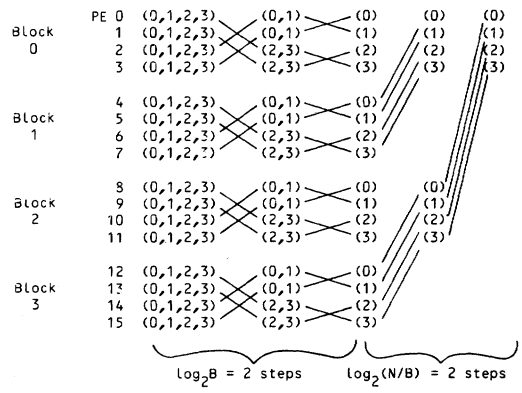


Fig. 1. Histogram calculation for $N = 16$ PE's, $B = 4$ bins. $(w, \cdots, z)$ denotes that bins w through z of the partial histogram are in the PE.

III. Performance of an Example Algorithm

In order to discuss and compare the performance measures, an example SIMD algorithm is examined. Each performance measure is applied to the algorithm. Conclusions which can be drawn from examination of the individual performance measures and comparisons among the performance measures are discussed.

As stated previously, the performance measures can be used to: 1) select from alternative SIMD algorithms for a given SIMD machine, 2) examine the effect of varying the size of the data set to be processed while the number of processors is unchanged, and 3) examine the effect of varying the number of processors in the system, while the size of the data set to be processed in unchanged. The example in this section will demonstrate uses 2) and 3).

1) Algorithm: The algorithm considered is an SIMD algorithm to compute the B-bin histogram of an $m \times m$ image using $N \geq B$ PE's [19], where $m^2 = M$. Each element of the image is represented by a "gray level," an integer from 0 to $B - 1$. The final histogram will contain a j in bin i if exactly j of the image elements have a gray level of i, $0 \leq i < B$.

The PE's are configured logically as a $\sqrt{N} \times \sqrt{N}$ grid, on which the $m \times m$ image is superimposed, so that each PE holds a $(m/\sqrt{N}) \times (m/\sqrt{N})$ subimage. The "local" histograms for the subimages are computed in parallel, so that each PE contains a B-bin histogram for the subimage which it holds. These local histograms are combined using a form of overlapped recursive doubling. This is shown for $N = 16$ and $B = 4$ in Fig. 1.

In the first $b = \log_2 B$ steps, each block of B PE's performs B simultaneous recursive doublings [25] to compute the histogram for the portion of the image contained in the block. At the end of the b steps, each PE has one bin of this partial histogram. This is accomplished by first dividing the B PE's of a block into two groups. Each group accumulates the sums for half of the bins, and sends the bins it is not accumulating to the group which is accumulating those bins. At each step of the algorithm, each group of PE's is divided in half such that the PE's with the lower addresses form one group, and the PE's with the higher addresses form another. The accumulated sums are similarly divided in half based on their indexes

in the histogram. The groups then exchange sums, so that each PE contains only sum terms which it is accumulating. The newly received sums are added to the sums already in the PE. After b steps, each PE has the total value for one bin from the portion of the image contained in the B PE's in its block.

The next $\log_2 (N/B)$ steps combine the results for these blocks to yield the histogram of the entire image, with the sum for bin i in processor i, $0 \leqslant i < B$. This is done by performing $\log_2 (N/B)$ steps of a recursive doubling algorithm to sum the partial histograms from the N/B blocks. This is shown by the last two steps of Fig. 1.

The exact data allocation scheme used for this particular histogram algorithm does not matter. The only requirement is that the allocation method assign $1/N$ of the image elements to each PE.

A sequential algorithm to compute the histogram of an $m \times m$ image requires $M = m^2$ additions. The SIMD algorithms uses M/N additions for each PE to compute its local histogram and $B - 1 + \log_2 (N/B)$ steps (transfer-and-add) to merge the histograms into the first B PE's. At step i in the merging of the partial histograms, $0 \leqslant i < \log_2 B$, the number of data transfers and additions required is $B/2^{i+1}$. A total of $B - 1$ transfers are therefore performed in the first $\log_2 B$ steps of the algorithm. $\text{Log}_2 (N/B)$ parallel transfers and additions are needed to combine the block histograms. This technique therefore requires $B - 1 + \log_2 (N/B)$ parallel transfer/add operations, plus the M/N additions needed to compute the local PE histograms. For each transfer in the first $\log_2 B$ steps, two masks are needed: one to control the sending of data from the low index groups and one for the high index groups. In each of the final $\log_2 (N/B)$ steps, two masks are needed: one to select the PE's which are to send data, and one to activate the PE's which received data for the addition operation. (Technically, the desired results will appear in PE's 0 to $B - 1$ even with no masking in the final $\log_2 (N/B)$ steps.) The total number of masking operations is therefore $2(B - 1 + \log_2 (N/B))$.

2) Performance: For application of the performance measures to the histogram algorithm, the following will be assumed:

t_a = time for 1 integer addition operation,
t_m = time for 1 masking operation,
$t_t(N)$ = time for 1 data transfer operation.

Depending on implementation, $t_t(N)$ may be a function of the number of PE's in the system. If a $\log_2 N$ multistage network is used, $t_t(N)$ can be written as $t_t(N) = t_s \log_2 N$, where t_s is the time to traverse one stage of the network. However, in order for the network to interface with the PE's, the effective transfer time must be an integer multiple of the PE cycle time. For simplicity, in the example that follows, it will be assumed that $t_a = t_t(N)/2 = 2t_m$. The actual relationships among t_a, $t_t(N)$, and t_m will be implementation dependent. Since the purpose of the example is to demonstrate performance measures rather than to evaluate the absolute performance of the algorithm, these simplifying assumptions can be made. The equations in the example are expressed in terms of t_a, $t_t(N)$, and t_m so that other relationships can be substituted if desired. In the example, it is assumed that the execution time is dominated by the time spent performing additions, masking operations, and data transfers. Time for loading and storing data items and for program control operations such as testing loop indexes has not been explicitly included in the analysis. If desired, the addition, mask, and transfer times could be assumed to include the time for memory accesses. The time required for program control will be negligible in comparison to the other times, and, in general, the program control operations can be overlapped with the PE and inter-PE transfer operations.

The performance of the histogram algorithm will be evaluated as a function of N for $m \times m$ images, with m ranging from $2^6 = 64$ to $2^{13} = 8192$, and $M = m^2$. The number of gray levels B has been fixed at $2^6 = 64$. The evaluation is limited to machines having between 2^6 and 2^{14} PE's. The histogram algorithm requires $N \geqslant B = 2^6$, so $N = 64$ is the smallest allowable SIMD machine. Based on proposed systems [3], [14], $N = 2^{14}$ represents a reasonable upper bound.

a) Execution Time:

$$T_N(M,B) = (M/N + B - 1 + \log_2 (N/B)) * t_a + (B - 1 + \log_2 (N/B)) * t_t(N) + 2(B - 1 + \log_2 (N/B)) * t_m. \quad (1)$$

The computation time is given by

$$c_N(M,B) = (M/N + B - 1 + \log_2 (N/B)) * t_a.$$

The overhead is given by

$$o_N(M,B) = (B - 1 + \log_2 (N/B)) * (t_t(N) + 2t_m).$$

The serial execution time is $T_1(M,B) = Mt_a$. Fig. 2 shows the $\log_2$ of the execution time as a function of N and M under the simplifying assumptions outlined above, i.e., $B = 64$ and $t_a = t_t(N)/2 = 2t_m$. The graph has been normalized to $t_a = 1$. For large images (e.g., $m \geqslant 2048$), it is clear that for given m, the execution time decreases as N increases. For such m, if N is doubled (corresponding to an increment of 1 on the horizontal axis), then the execution time is decreased by approximately a factor of two (corresponding to a decrement of 1 on the vertical axis). In these cases only the $(M/N) * t_a$ term in (1) contributes significantly to the complexity, and an increase in N yields a commensurate decrease in execution time. For small images, an increase in the number of PE's has little effect on the execution time. For such m, the terms $(\log_2 (N/B)) * (t_a + t_t(N) + 2t_m)$ in (1) dominate when N is large, and there is little or no advantage to using a large machine. For each m in the range 256 to 1024, the execution time decreases as N increases. However, the rate at which $T_N(M)$ decreases $(dT_N(M)/dN)$ also decreases as N increases. For example, for $m = 256$ the execution time criterion taken alone might dictate using as many PE's as possible. However, the decrease in execution time obtained by going from 2^{12} to 2^{13} PE's (adding 4096 PE's) is significantly less than the reduction in execution time achieved by increasing the number of PE's from 2^7 to 2^8 (adding 128 PE's). Therefore, for practical purposes it may be appropriate to consider $dT_N(M)/dN$ as well as $T_N(M)$.

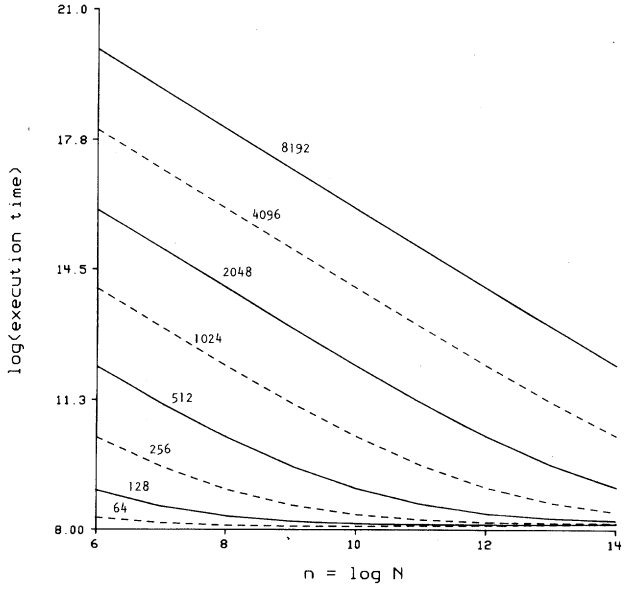


Fig. 2. Log_2 of the execution time as a function of $n = \log_2 N$, normalized to $t_a = 1$. Each curve is labeled with the associated value of m.

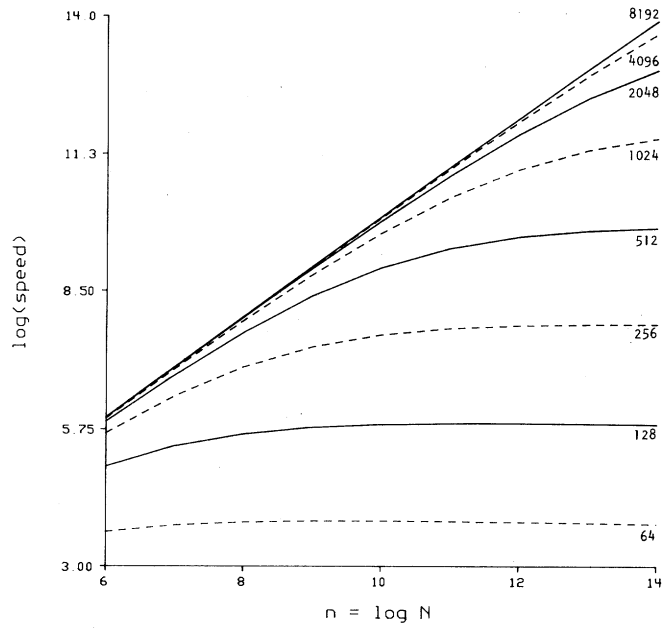


Fig. 3. Log_2 of speed as a function of $n = \log_2 N$, where t_a is the basic time unit (also $\log_2$ of speed-up). Each curve is labeled with the associated value of m.

b) Speed:

$$V_N(M,B) = \frac{M}{(M/N) * t_a + (B - 1 + \log_2 (N/B)) * (t_a + t_t(N) + 2t_m)}. \quad (2)$$

Fig. 3 shows the $\log_2$ of the speed as a function of N and M, under the simplifying assumptions. For given N, the speed increases as m increases. This happens because in this case the effect of the overhead is reduced. For given m, the impact of N on the speed depends on the value of m. For large m, the speed increases as N increases, and the rate of increase is approximately linear, so that, for example, doubling the number of PE's increases the speed by very nearly a factor of two. For $m = 8192$, 4096, and 2048, the speed, i.e., the number of data elements processed per unit time, is approximately equal to N, which is the maximum attainable speed. For small m, the speed increases to a maximum (between $n = 9$ and 10 for $m = 64$, and between $n = 11$ and 12 for $m = 128$), then decreases for larger N. However, the total range of the speed as a function of N is small for these m values, so the penalty is not great for choosing N to be a value other than the location of the maximum. In terms of (2), when m is large, $V_N(M)$ is dominated by the term $M/(M/N) = N$. For m in the middle range (e.g., $m = 256$, 512, 1024), for $6 \leqslant n \leqslant 14$, the speed increases at a rate that levels off as a function of N. In cases where it is possible to establish the number of execution cycles (time units) needed to process each data element t_p, an upper bound on the speed N/t_p can be calculated and compared with the speed attained.

c) Speed-Up:

$$S_N(M,B) = \frac{M * t_a}{(M/N) * t_a + (B - 1 + \log_2 (N/B)) * (t_a + t_t(N) + 2t_m)}.$$

As observed in the definition of speed-up, for fixed M, speed-up and speed are related by a constant factor $T_1(M)/M$. In the example, $T_1(M)/M = t_a = 1$, so the plots for speed-up are identical to the plots for speed (Fig. 3). For large m, $S_N(M) \simeq N$, i.e., the speed-up is almost ideal, for all N considered. Therefore, using $S_N(M)$ as the performance criterion would dictate using as many PE's as are available. For small m, $S_N(M) \ll N$, and the choice of N has little effect on the speed-up. For example, for $m = 64$, there is little advantage to using more than $2^6 = 64$ PE's. For each m in the middle range, there exists a value for N up to which increasing the number of PE's significantly increases the speed-up, but beyond which there is little advantage to increasing N. This can be determined by observing the rate of change of the speed-up with respect to N: $d(\text{speed-up})/dN$. Thus, it appears that using a combination of speed-up with $d(\text{speed-up})/dN$ (and/or a measure such as utilization or efficiency) will yield a more practical criterion than speed-up alone.

d) Efficiency:

$$E_N(M,B) = \frac{M * t_a}{M * t_a + N * (B - 1 + \log_2 (N/B)) * (t_a + t_t(N) + 2t_m)}.$$

Fig. 4 shows the efficiency for a range of values of M, under the simplifying assumptions. As required by the definition, $0 < E_N(M) \leqslant 1$. For all m, the efficiency decreases as N increases, but the rate of decrease is slower for large m (e.g., for $m = 4096$ and 8192). For large m, the efficiency is high for all N considered, and the choice of N does not significantly affect $E_N(M)$. This is markedly different from the information provided by the execution time measure, which indicates that for large m, N should be chosen to be as large as possible. For small m, especially for $m = 64$, $E_N(M)$ is small for all N

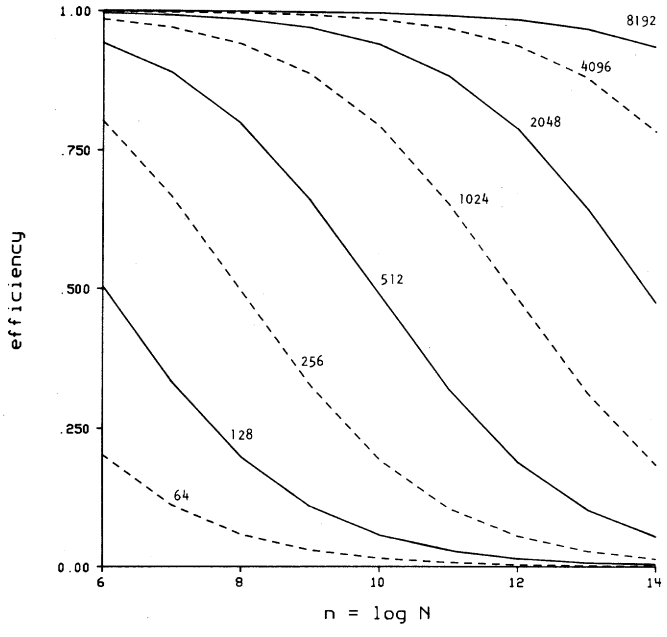


Fig. 4. Efficiency as a function of $n = \log_2 N$. Each curve is labeled with the associated value of m.

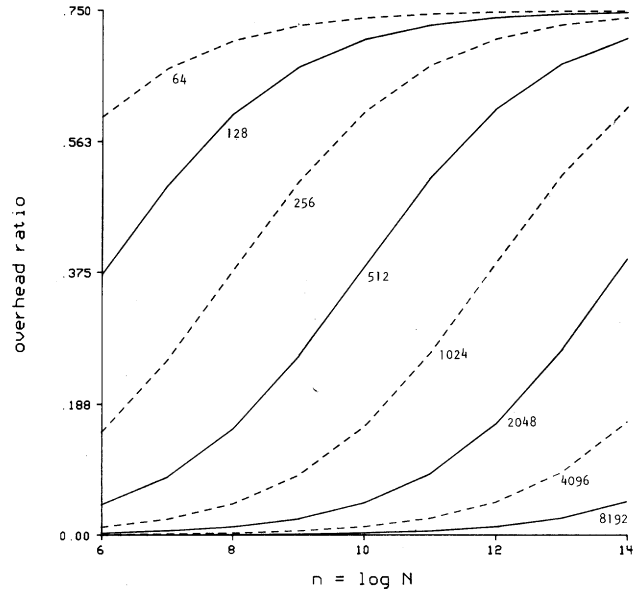


Fig. 5. Overhead ratio as a function of $n = \log_2 N$. Each curve is labeled with the associated value of m.

considered. Here, the conclusion that the efficiency is poor regardless of the choice of N is consistent with the information from the execution time measure. For m in the middle range (e.g., $512 \leqslant m \leqslant 2048$), the choice of N has a great impact on the efficiency. For each such m, there exists value(s) k such that the change from $N = 2^k$ to $N = 2^{k+1}$ (i.e., doubling the number of PE's) results in a decrease in efficiency of approximately 0.15. For example, for $m = 512$, a choice of $N = 2^9$ gives an efficiency of 0.66, whereas for $N = 2^{11}$, the efficiency has fallen to 0.32. For such m, the value of N which would be chosen to given high efficiency is smaller than the value which would be chosen to minimize execution time or maximize speed-up.

From the observation that $E_N(M)$ is higher for large m, it can be concluded that for a fixed N, there is no advantage to decomposing a size M problem into smaller subproblems, even if the result can later be recombined at low cost. However, given a reconfigurable system such as PASM [19], in which N may vary, a somewhat different conclusion can be drawn. For example, in such a system and using efficiency as the only measure, for $m = 256$ it would be better to process four images simultaneously each on machines of size 2^9 than four images sequentially, each on a machine of size 2^{11}. In other words, it would be more efficient to partition a system of size 2^{11} into four machines of size 2^9. The execution time for the processing of each individual image (i.e., the response time for each individual image as opposed to the set of four) would increase somewhat, but the total processing time for all four images would be less in the four-machine case. Thus, the choice of subsystem size would be best made as a task dependent balance between such measures as efficiency and execution time.

e) Overhead Ratio:

$$O_N(M,B) = \frac{(B - 1 + \log_2 (N/B)) * (t_t(N) + 2t_m)}{(M/N) * t_a + (B - 1 + \log_2 (N/B)) * (t_a + t_t(N) + 2t_m)}.$$

Fig. 5 shows the overhead ratio. As shown, $0 \leqslant O_N(M,B) < 0.75$, with $O_N(M,B)$ approaching 0 when M/N is very large, and $O_N(M,B)$ approaching 0.75 when M/N is very small. The conclusions obtainable from the overhead ratio are consistent with those from the efficiency measure. Specifically, for all m, the overhead ratio increases as N increases; for large and small m, the choice of N has little effect on the overhead ratio; and for m in the middle range, the choice of N significantly affects the overhead ratio. For this particular example, $O_N(M, B) = (\frac{3}{4})(1 - E_N(M,B))$. However, in general, the relationship between $O_N(M)$ and $E_N(M)$ will be a function of M and N.

f) Utilization: To derive the utilization requires counting the number of PE's active for each computation step. For this purpose the histogram algorithm can be viewed as having three stages. In the first stage, the local histograms are computed, requiring M/N addition steps, with all PE's active (i.e., $p_x = N$). In the second stage, the overlapped recursive doublings are performed. This stage consists of $B - 1$ addition steps, with all PE's active at each step. In the final stage, $\log_2 (N/B)$ steps of recursive doubling are used to merge the block histograms. At step i, $1 \leqslant i \leqslant \log_2 (N/B)$, the number of PE's performing the addition is $N/2^i$. Summing over the $\log_2 (N/B)$ steps gives $N - B$. The overall utilization is therefore

$$U_N(M,B) = \frac{(M + B * (N - 1)) * t_a}{M * t_a + N * (B - 1 + \log_2 (N/B)) * (t_a + t_t(N) + 2t_m)}.$$

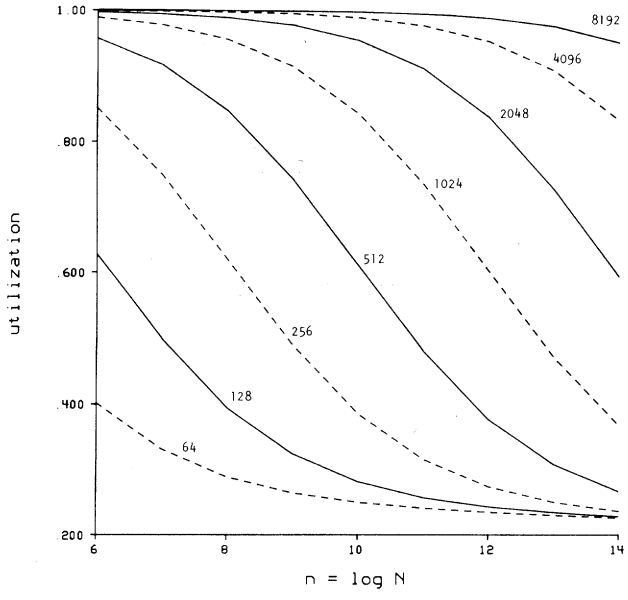


Fig. 6. Utilization as a function of $n = \log_2 N$. Each curve is labeled with the associated value of m.

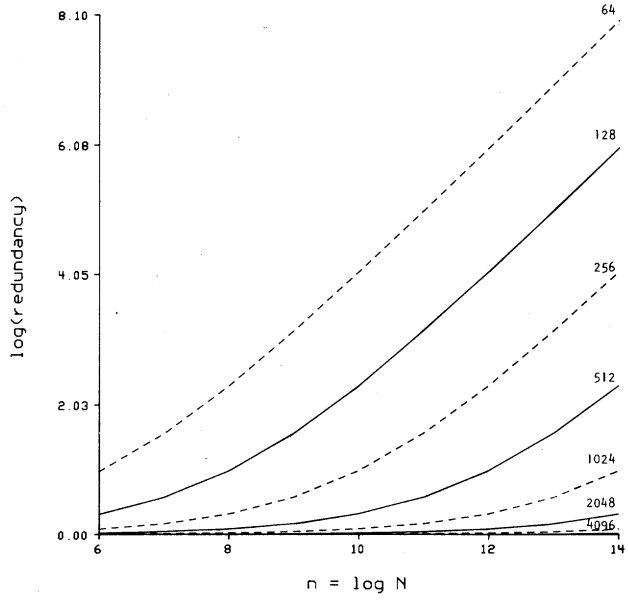


Fig. 7. Log_2 of the redundancy as a function $n = \log_2 N$. Each curve is labeled with the associated value of m. The values for $m = 8192$ coincide with the horizontal axis.

Fig. 6 shows the utilization under the simplifying assumptions. The patterns and conclusions obtainable are consistent with those for the efficiency and overhead ratio. In this example, because most computation steps use all N PE's,

$$\sum t_x p_x \simeq N * c_N(M).$$

Therefore, here $U_N(M, B) \simeq c_N(M)/T_N(M) = 1 - O_N(M, B)$.

Efficiency is directly affected by both overhead and utilization. As one may expect, efficiency will decrease as overhead increases and/or utilization decreases. If, for a given set of parameters, efficiency is low, then the overhead and utilization may be examined to determine factors contributing to the low efficiency. For the histogram, both overhead and utilization cause efficiency to decrease as N increases.

g) Redundancy: Using the operation count derived for the utilization

$$R_N(M, B) = \frac{(M + B * (N - 1)) * t_a}{M * t_a} = 1 + \frac{B * (N - 1)}{M}.$$

Fig. 7 shows the $\log_2$ of the redundancy under the simplifying assumptions. For all m, the redundancy increases as N increases, although the rate at which $R_N(M)$ increases is less for large m than for small m. The $B * (N - 1)$ redundant computations arise from the additions performed to combine the local and block histograms. When $M \gg BN$, i.e., the number of additions to compute the local histograms is much greater than the number of additions to combine the local histograms, $R_N(M)$ is small. For small images, however, the additions to combine the local histograms far outweigh the additions performed to compute the local histograms. As in the case of overhead, a high redundancy will be reflected as a decrease in the efficiency, since PE's are being used to perform redundant computations rather than essential computations. As would be expected, problem-size/machine-size combinations for which the redundancy is high have low efficiency values. From the efficiency alone, however, it is not possible to separate the losses due to overhead, utilization, and redundancy. For the histogram algorithm, all three of these measures contribute to the loss in efficiency.

In some cases, there may be a tradeoff between redundancy and utilization. For example, in the last $\log_2 (N/B)$ steps of the histogram example, is is not necessary to disable any PE's for the recursive doubling. The histogram would still appear in PE's 0 to $B - 1$, but may also appear elsewhere. The utilization would be increased, but at the expense of an increase in redundancy.

h) Cost Effectiveness: Assuming an N PE system having a $\log_2 N$ stage network

$$CE_N(M, B) = \frac{M/[(M/N) * t_a + (B - 1 + \log_2 (N/B)) * (t_a + t_t(N) + 2t_m)]}{(N + 1) * p_{\text{PE}} + \mu N \log_2 N * p_s}.$$

Fig. 8 shows the $\log_2$ of the cost effectiveness under the simplifying assumptions, plus the additional assumptions that $\mu = 1$ and $p_{\text{PE}} = 32 * p_s$, where p_{PE} is the cost of a PE, and p_s is the cost of a network switch. The cost effectiveness curves under the assumption that $p_{\text{PE}} = 128 * p_s$ are similar, with corresponding points having slightly higher values for the 128 factor case. Intuitively, the relation between cost effectiveness and speed can be considered by observing, for a fixed m, whether the speed increases with N. If it does, as for $m = 8192$, cost effectiveness stays high. If it does not, as for $m = 64$, cost effectiveness decreases. For each value of m, for this example,

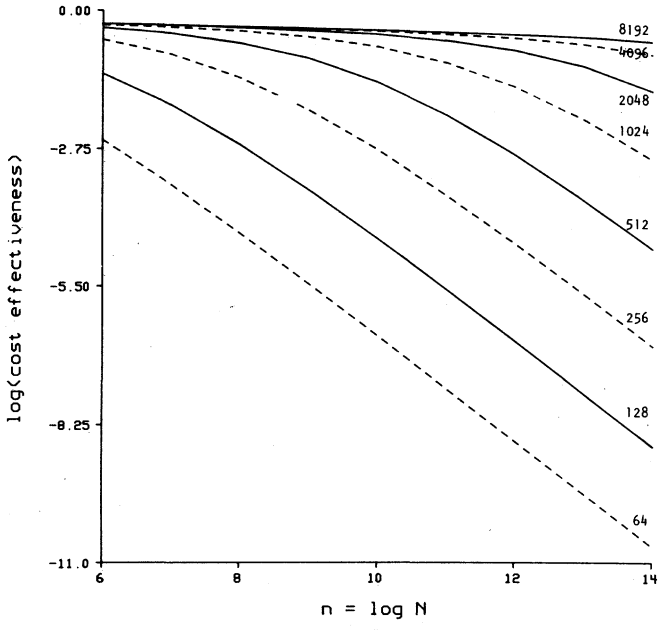


Fig. 8. Log_2 of the cost effectiveness as a function of $n = \log_2 N$, normalized to PEcost = 1. PEcost = 32 * switchcost and t_a is the basic time unit. Each curve is labeled with the associated value of m.

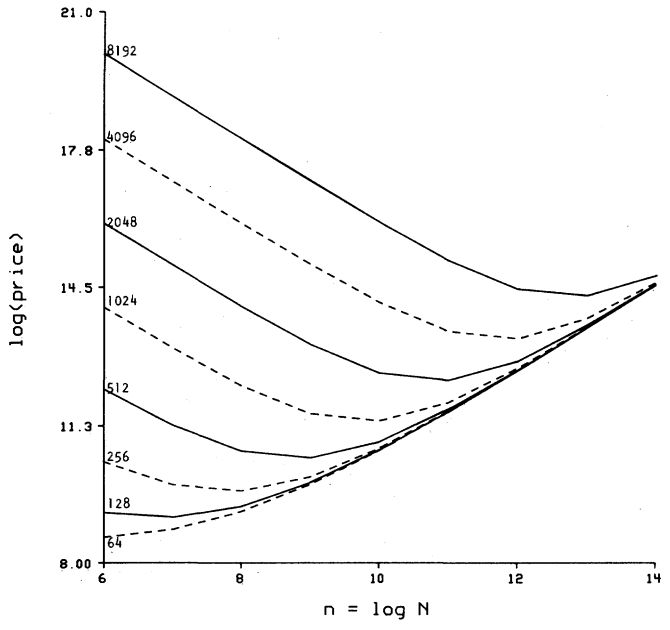


Fig. 9. Log_2 of the price as a function of $n = \log_2 N$, where $p_t = p_i = 1$, and $p_{PE} = 32 * p_s = 1$. Each curve is labeled with the associated value of m.

cost effectiveness decreases as n increases. This occurs because doubling N will increase the speed by less than a factor of two, but will more than double the cost.

i) Price: The price for the histogram example is

$$P_N(M,B) = p_t * [(M/N) * t_a + (B - 1 + \log_2 (N/B)) * (t_a + t_t(N) + 2t_m)] + p_i * [N * P_{PE} + \mu N \log_2 N * p_s].$$

Fig. 9 shows the $\log_2$ of the price under the simplifying assumptions, plus the assumptions that $p_t = p_i = 1$, $\mu = 1$, and $p_{PE} = \delta p_s = 1$, with $\delta = 32$. Except for the case of $m = 64$, each of the curves has a minimum in the range $2^6 < N \leqslant 2^{14}$. Furthermore, the optimal N is greater for large images than for small images. This occurs because for large images, execution time continues to decrease significantly as N increases, while for smaller images, the rate at which $T_N(M)$ decreases falls off for large N. For this histogram example under the simplifying assumptions, each of the minima occurs at $N = m$.

The generalized price for the histogram algorithm, under the same assumptions, is given by

$$P'_N(M,B) = \frac{\alpha}{\alpha+1} [(M/N) * t_a + (B - 1 + \log_2 (N/B)) * (t_a + t_t(N) + 2t_m)] + \frac{1}{\alpha+1} [N + (\mu N \log_2 N)/\delta].$$

Fig. 10 shows the generalized price as a function of N and α for a 1024 X 1024 image. As expected, the optimal value of N shifts to the right when execution time is more critical than system cost ($\alpha > 1$) and to the left when system cost dominates ($\alpha < 1$). The curves intersect at $n = 10$ ($N = 2^{10}$) because under the assumptions made, $T_{N=1024}(m = 1024, B = 64) \simeq C_{N=1024} = K$ so $P'_N(M, B)$ can be written as $(\alpha + 1) * K/(\alpha + 1) = K$, which is independent of α. In general, such a point of intersection will occur when for fixed M, there exists a value for N such that $T_N(M) \simeq C_N$.

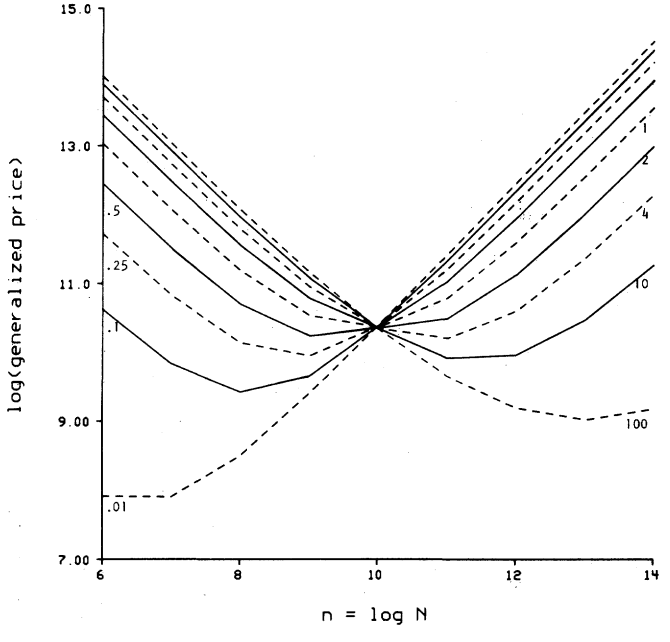


Fig. 10. Log_2 of the generalized price as a function of $n = \log_2 N$, where $p_t = p_i = 1$, and $p_{PE} = 32 * p_s = 1$. Shown for $m = 1024$. Each curve is labeled with the associated value of α.

3) Discussion: From the plots obtained for the example, the measures can be grouped according to which measures give similar answers to two types of questions.

1) Based only on the performance measure being examined, what value(s) for N are "good" for a given problem size?

2) What is the effect of choosing the "wrong" N?

Similar patterns are observed for the execution time, speed, and speed-up measures. For these three, general conclusions can be drawn based on problem size. For large m (e.g., $m \geqslant 2048$), N should be chosen as large as possible (over the range $2^6 \leqslant N \leqslant 2^{14}$), and there is no disadvantage to such a choice. Smaller N gives significantly poorer performance. For small m (e.g., $m \leqslant 128$), the choice of N has little effect on the performance. For m in the middle range, performance improves as N increases, but criteria specifying a desired rate of improvement may dictate choice of an intermediate value for N. Selecting N too small sacrifices performance; selecting N too large gives only minor improvement.

These three measures give fairly direct information as to how fast the N PE implementation of the histogram algorithm will be. They will be the most useful measures when the operating constraints are in terms of required execution time, as might be the case, for example, in systems aimed at real-time processing. With the possible exception of the speed-up, however, they do not directly address issues related to how effectively the system resources are being used.

As described in the individual analyses of the performance measures, the efficiency, overhead ratio, and utilization exhibit similar characteristics for the histogram example. For large m (e.g., $m \geqslant 4096$), the choice of N has little effect on these measures: performance is approximately the same for all N, degrading slightly as N increases. This conclusion differs significantly from that drawn from the first three measures. For small m (e.g., $m \leqslant 128$ or 256), N should be as small as possible. Increasing N even slightly over 2^6 seriously degrades the performance. For very large N, the performance is low, and remains low as N increases. For m in the middle range, there exist (small) N for which performance is very high, and (large) N for which performance is very low. Choice of N is therefore critical, and is smaller than would be dictated by the execution time, speed, and speed-up measures alone.

Together with the redundancy, these three measures quantify the various ways in which the system resources are used by the algorithm, and are therefore relevant when evaluating the resource utilization. An example of when such measures might profitably be applied is in deciding how to partition a reconfigurable SIMD system to perform the same algorithm on a number of different data sets. Assume that $K = 2^k$ data sets are to be processed, and consider two possibilities: running the K jobs sequentially, using N PE's per job, or running the K jobs simultaneously, using $N' = N/K$ PE's per job. The first configuration will require time $K * T_N(M)$ to complete the jobs; the second will require time $T_{N'}(M)$. For $N' < N$, $E_{N'}(M) > E_N(M)$ if and only if $T_{N'}(M) < K * T_N(M)$. Thus, the efficiency measure can be used directly to determine which configuration will yield higher throughput.

For the histogram example, the choices for N dictated by the cost effectiveness criterion, which takes account of both speed and system cost, are consistent with the choices indicated by the efficiency, overhead ratio, utilization, and redundancy. For the considered range of N, however, the choices appear to run counter to the choices suggested by the speed criterion. Over this entire range, the cost is increasing at a rate faster than the speed. Cost effectiveness can be related to the resource utilization measures by observing that high cost effectiveness requires efficient use of the resources.

A different picture is presented by the price and generalized price criteria in that these are the only measures which, for the particular system and cost parameters chosen, appear to indicate clear choices of system size in the range $2^6 \leqslant N \leqslant 2^{14}$. The N value indicated by the price appears in most cases to be a compromise between the values suggested by the execution time, speed, and speed-up measures and those indicated by the efficiency, overhead ratio, utilization, and cost effectiveness. The generalized price measure, as shown for the $m = 1024$ example, makes explicit the tradeoffs between system cost and execution time. For large α (execution time critical), the optimal values for N are consistent with the N values for high performance under the execution time, speed, and speed-up measures. Conversely, for small α (system cost critical), the optimal values of N are consistent with the N values for high performance under the efficiency, overhead ratio, utilization, and cost effectiveness measures. The identification of clearly optimal N values would appear to be an advantage; however, parameter values could easily be chosen which would make the price and generalized price curves monotonic throughout the given range of N values. Thus, it is important to recognize that the apparently optimal values of N are optimal in a meaningful sense only if the various parameters are determined in a meaningful and realistic way. The determination of these parameters will most likely be nontrivial. From a practical engineering viewpoint, if appropriate values can be assigned to the parameters, the price and generalized price criteria may be the most useful of the performance measures discussed here because they provide the opportunity to incorporate real cost factors and make explicit tradeoffs between system cost and raw computational speed.

Only one SIMD algorithm for computing histograms has been considered here. The effects of varying both the image size and machine size have been examined. The performance measures could, of course, be used to evaluate other SIMD algorithms for histogramming in order to choose one which best meets the desired criteria for a given machine.

IV. Conclusions

A number of performance measures for evaluating SIMD algorithms have been defined and applied to an example algorithm. The goal of this paper was to collect, create, and compare different criteria for determining the "goodness" of an SIMD-machine/algorithm/problem-size combination. As a result of the study, one may conclude that no single measure is sufficient for all processing environments. A number of the performance measures considered quantify computational speed but do not account for how effectively the resources of the parallel processing system are being used. For other measures the reverse is true. The cost effectiveness and price criteria attempt to provide an integrated picture of speed and resource utilization. The generalized price criterion offers the most flexible approach for "tuning" the system size or selecting an algorithm for a particular task based on the relative importance of processing speed versus system cost, but it also requires the most knowledgeable specification of system

and application parameters in order to obtain meaningful results.

The performance measures presented have been described in terms of SIMD machines. Another mode of parallel processing is represented by MIMD (multiple instruction stream–multiple data stream) machines, in which communicating processors have their own instruction streams as well as their own data streams [8]. The measures presented here can also be applied to algorithms for MIMD machines. However, due to the asynchronous nature of MIMD processing, algorithm analysis may be more difficult to perform. For example, network and memory contention would have to be accounted for in some way. In addition, computation of measures such as utilization and redundancy may require simulation if they are data dependent.

As the interest in large-scale parallel systems expands, measures such as those described in this paper will become increasingly important. Future work includes adapting these measures for MIMD machines and examining performance measures which deal with other system components, such as memory and input/output. The study and cataloging of performance measures will aid parallel machine designers and programmers in analyzing a system based on criteria relevant to their particular application area.

References

[1] K. E. Batcher, "STARAN parallel processor system hardware," in *Proc. Nat. Comput. Conf.*, AFIPS, vol. 43, May 1974, pp. 405–410.

[2] —, "The flip network in STARAN," in *Proc. 1976 Int. Conf. Parallel Processing*, Aug. 1976, pp. 65–71.

[3] —, "The design of a massively parallel processor," *IEEE Trans. Comput.*, vol. C-29, pp. 836–844, Sept. 1980.

[4] W. J. Bouknight *et al.*, "The Illiac IV system," *Proc. IEEE*, vol. 60, pp. 369–388, Apr. 1972.

[5] P. Brinch-Hansen, *Operating System Principles*. Englewood Cliffs, NJ: Prentice-Hall, 1973.

[6] M.J.B. Duff, "Parallel algorithms and their influence on the specification of application problems," in *Multi-Computers and Image Processing Algorithms and Programs*, K. Preston and L. Uhr, Eds. New York: Academic, 1982.

[7] T. Feng, "Data manipulating functions in parallel processors and their implementations," *IEEE Trans. Comput.*, vol. C-23, pp. 309–318, Mar. 1974.

[8] M. J. Flynn, "Very high-speed computing systems," *Proc. IEEE*, vol. 54, pp. 1901–1909, Dec. 1966.

[9] D. J. Kuck, "A survey of parallel machine organization and programming," *ACM Comput. Surveys*, vol. 9, pp. 29–59, Mar. 1977.

[10] D. H. Lawrie, "Access and alignment of data in an array processor," *IEEE Trans. Comput.*, vol. C-24, pp. 1145–1155, Dec. 1975.

[11] R. B-L. Lee, "Empirical results on the speed, efficiency, redundancy and quality of parallel computations," in *Proc. 1980 Int. Conf. Parallel Processing*, Aug. 1980, pp. 91–100.

[12] P. T. Mueller, Jr., L. J. Siegel, and H. J. Siegel, "Parallel algorithms for the two-dimensional FFT," in *Proc. 5th Int. Conf. Pattern Recognition*, Dec. 1980, pp. 497–502.

[13] G. J. Nutt, "Microprocessor implementation of a parallel processor," in *Proc. 4th Symp. Comput. Architecture*, Mar. 1977, pp. 147–152.

[14] M. C. Pease, "The indirect binary n-cube microprocessor array," *IEEE Trans. Comput.*, vol. C-26, pp. 458–473, May 1977.

[15] A. P. Reeves and R. Rindfuss, "The BASE 8 binary array processor," in *Proc. IEEE Conf. Pattern Recognition and Image Processing*, Aug. 1979, pp. 250–255.

[16] M. C. Sejnowski, E. T. Upchurch, R. N. Kapur, D.P.S. Charlu, and G. J. Lipovski, "An overview of the Texas reconfigurable array computer," in *Proc. 1980 Nat. Comput. Conf.*, May 1980, pp. 631–641.

[17] H. J. Siegel, "A model of SIMD machines and a comparison of various interconnection networks," *IEEE Trans. Comput.*, vol. C-28, pp. 907–917, Dec. 1979.

[18] —, "The theory underlying the partitioning of permutation networks," *IEEE Trans. Comput.*, vol. C-29, pp. 791–800, Sept. 1980.

[19] H. J. Siegel, L. J. Siegel, F. C. Kemmerer, P. T. Mueller, Jr., H. E. Smalley, Jr., and S. D. Smith, "PASM: A partitionable SIMD/MIMD system for image processing and pattern recognition," *IEEE Trans. Comput.*, vol. C-30, pp. 934–947, Dec. 1981.

[20] H. J. Siegel and S. D. Smith, "Study of multistage SIMD interconnection networks," in *Proc. 5th Symp. Comput. Architecture*, Apr. 1978, pp. 223–229.

[21] L. J. Siegel, "Parallel processing algorithms for linear predictive coding," in *Proc. 1980 IEEE Int. Conf. Acoust., Speech, Signal Processing*, Apr. 1980, pp. 960–963.

[22] —, "Image processing on a partitionable SIMD machine," in *Languages and Architectures for Image Processing*, M.J.B. Duff and S. Levialdi, Eds. London: Academic, 1981.

[23] L. J. Siegel, P. T. Mueller, Jr., and H. J. Siegel, "FFT algorithms for SIMD machines," in *Proc. 17th Allerton Conf. Commun., Contr., Comput.*, Oct. 1979, pp. 1006–1015.

[24] L. J. Siegel, H. J. Siegel, and A. E. Feather, "Parallel processing approaches to image correlation," *IEEE Trans. Comput.*, vol. C-31, pp. 208–218, Mar. 1982.

[25] H. S. Stone, "Parallel computers," in *Introduction to Computer Architecture*, H. S. Stone, Ed. Chicago, IL: S.R.A., 1975.

[26] P. H. Swain and S. Davis, *Remote Sensing: The Quantitative Approach*. New York: McGraw-Hill, 1978.

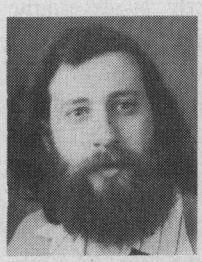
[27] P. H. Swain, H. J. Siegel, and J. El-Achkar, "Multiprocessor implementation of image pattern recognition: A general approach," in *Proc. 5th Int. Conf. Pattern Recognition*, Dec. 1980, pp. 309–317.

[28] C. R. Vick, S. P. Kartashev, and S. I. Kartashev, "Adaptable architectures for supersystems," *Computer*, vol. 13, pp. 17–35, Nov. 1980.

Leah J. Siegel (S'75–M'77) was born in Trenton, NJ, on August 27, 1949. She received the S.B. degree in mathematics in 1972 from the Massachusetts Institute of Technology, Cambridge, the M.A. and M.S.E. degrees in 1974, and the Ph.D. degree in 1977, all in electrical engineering and computer science from Princeton University, Princeton, NJ.

Since 1976 she has been on the faculty in the School of Electrical Engineering, Purdue University, West Lafayette, IN, where she is currently an Associate Professor. Her research interests include speech analysis and recognition, and the design of parallel processing algorithms for digital speech, signal, and image processing. At Purdue, she has been involved in the development of the Laboratory for One-Dimensional Signal Processing.

Dr. Siegel is a member of the Administrative Committee of the IEEE Acoustics, Speech, and Signal Processing Society. She is also a member of the Association for Computing Machinery, Eta Kappa Nu, and Sigma Xi.

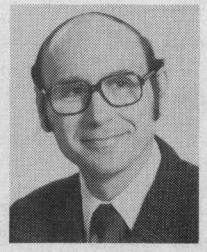


Howard Jay Siegel (M'77) was born in New Jersey on January 16, 1950. He received the S.B. degree in electrical engineering and the S.B. degree in management from the Massachusetts Institute of Technology, Cambridge, in 1972, the M.A. and M.S.E. degrees in 1974, and the Ph.D. degree in 1977, all in electrical engineering and computer science from Princeton University, Princeton, NJ.

In 1976 he joined the School of Electrical Engineering, Purdue University, West Lafayette,

IN, where he is currently an Associate Professor. Since January 1979 he has also been affiliated with Purdue's Laboratory for Applications of Remote Sensing. His research interests include parallel/distributed processing, multimicroprocessor systems, image processing, and speech processing.

Dr. Siegel has served as a Guest Editor of the IEEE TRANSACTIONS ON COMPUTERS and is on the Editorial Board of the *Journal of Digital Systems*. He is currently Chairman of the IEEE Computer Society Technical Committee on Computer Architecture, a Vice Chairman of the Technical Committee on Distributed Processing, the Vice Chairman of the ACM Special Interest Group on Computer Architecture, an IEEE Society Distinguished Visitor, and the General Chairman of the Third International Conference on Distributed Computing Systems, to be held October 1982. He is a member of Eta Kappa Nu and Sigma Xi.



Philip H. Swain (S'66–M'69–SM'81) received the B.S. degree in electrical engineering from Lehigh University, Bethlehem, PA, in 1963, and the M.S. and Ph.D. degrees from Purdue University, West Lafayette, IN, in 1964 and 1970, respectively.

Currently, he is an Associate Professor of Electrical Engineering at Purdue University, West Lafayette, IN. He is also Program Leader for Data Processing and Analysis Research at Purdue's Laboratory for Applications of Remote Sensing (LARS). Affiliated with LARS since its inception in 1966, he has developed methods and systems for the management and analysis of remote sensing data. Previously, he has been with the Philco-Ford Corporation and the Burroughs Corporation, and served as a Consultant to NASA and the Universities Space Research Association. His research interests include theoretical and applied pattern recognition, methods of artificial intelligence, and the application of advanced computer architectures to image processing. He is the Co-Editor and contributing author of *Remote Sensing: The Quantitative Approach* (New York: McGraw-Hill, 1978).

Dr. Swain is a member of the Pattern Recognition Society, Phi Beta Kappa, Sigma Xi, Tau Beta Pi, and Eta Kappa Nu.

Performance Adjustment of an APL Interpreter

MAMORU MAEKAWA AND YOJIRO MORIMOTO

Abstract—**An APL interpreter is analyzed to examine how microprogramming can improve its performance. Also examined is how the modularization method or program structure affects performance improvement. Two basic modularization methods, function and data modularizations, are investigated. We find that the performance gain may reach 100-time speed-up and that a proper selection of modules is very important to obtain the maximum performance gain under a limited microprogram memory. We also find that both the function and the data modularizations provide the same degree of performance improvement despite the finding that they tend to affect the complementary parts of a program.**

Index Terms—**APL interpreter, data modularization, function modularization, high-level language machine, microprogramming, modularization, performance modularization.**

I. INTRODUCTION

MICROPROGRAMMING is essential for the efficient implementation of an APL interpreter. Due to the microprogram memory limitation, it is important to properly select a set of modules that are converted into a microprogram. Since the selection is made based on the program structure, the modularization method will affect the performance. The performance is also, of course, affected by how much execution time is concentrated in particular portions of the interpreter. Although it is generally known that only a small portion of a program occupies most of its execution time, only a few actual measurement data are available [2].

This paper is an attempt to gather actual data and to investigate how the modularization method affects the performance. We consider two basic modularization methods, function modularization and data modularization, which will be explained later.

The investigated APL interpreter is developed for a microprogrammable computer system, called the experimental polyprocessor system (EPOS) [5], [6], and has about 9000 microprogram instructions and 770 APL statements. The investigation is extensive, involving a large amount of data gathering and manipulation.

Manuscript received February 13, 1980; revised January 12, 1981 and March 15, 1982. This work was supported by the Agency of Industrial Science and Technology, Ministry of International Trade and Industry, and was primarily performed at the Toshiba Research and Development Center for the Pattern Information Processing System Project.

M. Maekawa is with the Department of Information Science, Faculty of Science, University of Tokyo, Tokyo, Japan.

Y. Morimoto is with the Research and Development Center, Toshiba Corporation, Kawasaki, Kanagawa, Japan.

II. STRUCTURES OF THE APL INTERPRETER

Intermediate Language

It is not practical to implement the whole interpreter in microprogram. Therefore, some intermediate language is necessary between an APL and a microprogram language [2]. A strict hierarchical approach shown in Fig. 1(a) is not suitable

0098/5589/82/0700-0331$00.75