Evaluation of performance of parallel processors in a real-time environment

by GREGORY R. LLOYD and RICHARD E. MERWIN

SAFEGUARD System Office
Washington, D.C.

INTRODUCTION

The use of parallelism to achieve greater processing throughput for computational problems exceeding the capability of present day large scale sequential pipelined data processing systems has been proposed and in some instances hardware employing these concepts has been built. Several approaches to hardware parallelism have been taken including multi-processors\(^3,4\) which share common storage and input-output facilities but carry out calculations with separate instruction and data streams; array processors\(^5\) used to augment a host sequential type machine which executes a common instruction stream on many processors; and associative processors which again require a host machine and vary from bit\(^5\) to word oriented\(^6\) processors which alternatively select and compute results for many data streams under control of correlation and arithmetic instruction streams. In addition, the concept of pipelining is used both in arithmetic processors\(^7\) and entire systems, i.e., vector machines\(^8\) to achieve parallelism by overlap of instruction interpretation and arithmetic processing.

Inherent in this approach to achieving greater data processing capability is the requirement that the data and algorithms to be processed must exhibit enough parallelism to be efficiently executed on multiple hardware ensembles. Algorithms which must be executed in a purely sequential fashion achieve no benefit from having two or more data processors available. Fortunately, a number of the problems requiring large amounts of computational resources do exhibit high degrees of parallelism and the proponents of the parallel hardware approach to satisfying this computational requirement have shown considerable ingenuity in fitting these problems into their proposed machines.

The advocates of sequential pipelined machines can look forward to another order of magnitude increase in basic computational capability before physical factors will provide barriers to further enhancement of machine speed. When this limit is reached and ever bigger computational problems remain to be solved, it seems likely that the parallel processing approach will be one of the main techniques used to satisfy the demand for greater processing capability.

Computational parallelism can occur in several forms. In the simplest case the identical calculation is carried out on a number of separate data sets (array processing). A more complex case involves different calculations on separate data sets (multiprocessing) and finally, the greatest challenge to the parallel processing approach occurs when a single calculation on a single data set must be decomposed to identify parallel computational paths within a single computational unit. A number of mathematical calculations are susceptible to this type of analysis, e.g., operations on matrices and linear arrays of data.

The computational support required for a phased array radar is representative of problems exhibiting a high degree of parallelism. These systems can transmit a radar beam in any direction within its field of view in a matter of microseconds and can provide information on up to hundreds of observed objects for a single transmission (often called a “look”). The amount of information represented in digital form which can be generated by this type of radar can exceed millions of bits per second and the analysis of this data provides a severe challenge to even the largest data processors. Applications of this radar system frequently call for periodic updates of position for objects in view which are being tracked. This cyclic behavior implies that a computation for all objects must be completed between observations. Since many objects may be in view at one time, these computations can be carried out for each object in parallel.

The above situation led quite naturally to the application of associative parallel processors to provide part of the computational requirements for phased array radars. A number of studies\(^9,10,11,12\) have been made of this approach including use of various degrees of parallelism going from one bit wide processing arrays to word oriented processors. As a point of reference this problem has also been analyzed for implementation on sequential pipelined machines.\(^13\) One of the main computational loads of a phased array radar involves the filtering and smoothing of object position data to both eliminate uninteresting objects and provide more accurate tracking information for objects of interest. A technique for elimination of uninteresting objects is referred to as bulk filtering and the smoothing of data on interesting objects is carried out with a Kalman filter.

The following presents an analysis of the results of the above studies of the application of associative parallel processors to both the bulk and Kalman filter problems.
criteria used to evaluate the application of parallel hardware to these problems are the degree of hardware utilization achieved and the increase in computational throughput achieved by introducing parallelism. The latter measure is simply the ratio of computational throughput achieved by the array of processing elements to the throughput possible with one element of the array. The Parallel Element Processing Ensemble (PEPE) considered as one of the four hardware configurations is the early IC model and is not the improved MSI PEPE currently under development by the Advanced Ballistic Missile Defense Agency.

Finally, a comparison of hardware in terms of number of logical gates is presented to provide a measure of computational throughput derived as a function of hardware complexity. The paper concludes with a number of observations relative to the application of the various associative parallel hardware approaches to this computational requirement.

FILTER COMPUTATIONS

The bulk and Kalman filters play complementary roles in support of a phased array radar. The task assigned to the radar is to detect objects and identify those with certain characteristics e.g. objects which will impact a specified location on the earth, and for those objects so identified, to provide an accurate track of the expected flight path. The bulk filter supports the selection process by eliminating from consideration all detected objects not impacting a specified area while the Kalman filter provides smoothed track data for all impacting objects. Both filters operate upon a predictive basis with respect to the physical laws of motion of objects moving in space near the earth. Starting with an observed position, i.e., detection by a radar search look, the bulk filter projects the position of the object forward in time, giving a maximum and minimum range at which an impacting object could be found in the next verification transmission. Based upon this prediction the radar is instructed to transmit additional verification looks to determine that this object continues to meet the selection criteria by appearing at the predicted spot in space following the specified time interval.

Those objects which pass the bulk filter selection criteria are candidates for precision tracking by the radar and in this case the Kalman filter provides data smoothing and more precise estimates of the object's flight path. Again a prediction is made of the object's position in space at some future time based upon previously measured positions. The radar is instructed to look for the object at its predicted position and determines an updated object position measurement. The difference between the measured and predicted position is weighted and added to the predicted position to obtain a smoothed position estimate. Both the bulk and Kalman filter are recursive in the sense that measurement data from one radar transmission is used to request future measurements based upon a prediction of a future spatial position of objects. The prediction step involves evaluation of several terms of a Taylor expansion of the equations of motion of spatial objects. Detailed discussion of the mathematical basis for these filters can be found in the literature on phased array radars.

The computations required to support the bulk filter are shown in Figure 1. The radar transmissions are designated as either search or verify and it is assumed that every other transmission is assigned to the search function. When an object is detected, the search function schedules a subsequent verification look typically after fifty milliseconds. If the verification look confirms the presence of an object at the predicted position another verification look is scheduled again after fifty milliseconds. When no object is detected on a verification look, another attempt can be made by predicting the object's position ahead two time intervals i.e., one hundred milliseconds, and scheduling another verification look. This procedure is continued until at least $M$ verifications have been made of an object's position out of $N$ attempts. If $N - M$ attempts at verification of an object's position result is no detection then the object is rejected. This type of filter is termed an $M$ out of $N$ look bulk filter.

Turning now to the Kalman filter the computational problem is much more complex. In this case a six or seven element state vector containing the three spatial coordinates, corresponding velocities, and optionally an atmospheric drag coefficient is maintained and updated periodically for each tracked object. A block diagram of this computation is shown in Figure 2. The radar measurements are input to state vector and weighting matrix update procedures. The weighting matrix update loop involves an internal update of a covariance matrix which along with the radar measurements is used to update a weighting matrix. The state vector update calculation generates a weighted estimate from the predicted and measured state vectors. The Kalman filter computation is susceptible to decomposition into parallel calculations and advantage can be taken of this in implementations for a parallel processor.

COMPUTATIONAL MODELS

Bulk filter

The bulk filter is designed to eliminate with a minimum expenditure of computational resources a large number of uninteresting objects which may appear in the field of view of a phased array radar. A model for this situation requires
assumptions for the number and type of objects to be handled, efficiency of the filter in eliminating uninteresting objects, and radar operational parameters. These assumptions must produce a realistic load for the filter which would be characteristic of a phased array radar in a cluttered environment. The assumptions, which are based upon the Advanced Ballistic Missile Agency's Preliminary Hardsite Defense study, are:

1. The radar transmits 3000 pulses, i.e. looks, per second and every other one of these is assigned to search.
2. New objects enter the system at a rate of 100 per 10 milliseconds (Ms) all of which are assumed to be detected on one search look.
3. Fifteen objects are classified as being of interest, i.e. impacting a designated area (must be precision tracked), and 85 of no interest (should be eliminated from track).
4. Following detection an attempt must be made to locate each object not rejected by the filter every 50 Ms.
5. The filter selection criteria is 5 (M) detections out of 7 (N) attempts. Failure to detect the object three times in the sequence of 7 looks results in rejection.
6. The filter is assumed to reduce the original 100 objects to 70 at the end of the third; 45 at the end of the fourth; 30 at the end of the fifth; 25 at the end of the sixth; and 20 at the end of the seventh look; thus failing to eliminate 5 uninteresting objects.

Based upon the above assumptions the bulk filter accepts 500 new objects every 50 Ms. When operational steady state is reached, the processing load becomes 100 search and 290 verify calculations every 10 Ms. Each object remains in the filter for a maximum of 350 Ms and for a 50 Ms interval 1950 filter calculations are required corresponding to 10,000 new objects being detected by the radar per second.

The above process can be divided into two basic steps. The first involves analysis of all radar returns. For search returns the new data is assigned to an available processor. For verify returns each processor must correlate the data with that being processed to determine if it represents new positional information for an object being tracked by that processor. For all objects in process, new data must be received every 50 Ms or it is considered to have not been redetected and hence subject to rejection by the filter. The associative processors studied were unable to carry out the required calculations within the pulse repetition rate of the radar (330 μsec). To achieve timely response, the processing was restructured into correlation and arithmetic cycles as shown in Figure 3. During the first 25 msec interval, the processors correlate returns from the radar with internal data (predicted positions). During the subsequent 25 msec interval, processors carry out the filter calculations and predict new object positions. This approach allowed all required processing to be completed in a 50 msec interval. Objects which fail the selection criteria more than two times are rejected and their processor resources are freed for reallocation.

**Kalman filter**

The Kalman filter computation requires many more arithmetic operations than the bulk filter. The radar becomes the limiting factor in this case since only one object is assumed for each look. Assuming a radar capable of 3000 transmissions per second and a 50 Ms update requirement for each precision track, a typical steady state assumption would be 600 search looks and 2400 tracking looks per second (corresponding to 120 objects in precision track). At this tracking load it must again be assumed that the 50 Ms update interval is divided into 25 Ms correlation and compute cycles as was done for the bulk filter and shown in Figure 3. This implies that 60 tracks are updated every 25 Ms along with the same number of verify looks being received and correlated.

**EVALUATION APPROACH**

The three quantities of interest in determining the relation between a parallel processor organization and a given problem are: resources required by the problem, resources available from the processor configuration, and time constraints (if any). A more precise definition of these quantities follows, but the general concept is that the processor capabilities and problem requirements should be as closely balanced as possible.

Quantitative resource measures and balance criteria are derived from Chen's1 analysis of parallel and pipelined computer architectures. Chen describes the parallelism inherent in a job by a graph with dimensions of parallelism
width (number of identical operations which may be performed in parallel) and execution time. The ratio $\phi$ is defined for a job as the area under the step(s) showing parallelism (width $W > 1$) divided by the total area swept out by the job. The hardware efficiency factor, $\eta$, is the total job work space (defined as the product of execution time and the corresponding job width $W$ summed over all computations) over the total hardware work space (defined as the product of total execution time and the number, $M$, of available parallel processors). This provides a measure of utilization of a particular hardware ensemble for each segment of a computation. A modification of Chen's $\eta$ allows consideration of time constraints. Hardware space will now be defined as a product of the total time available to carry out the required computation times $M$, the number of processors available. Call this ratio $\hat{\eta}$. The work space is as defined above except that periods of no processor activity may be included, i.e., job width $W = 0$. Figure 4 illustrates these concepts showing a computation involving several instruction widths carried out in an available computation time $T_a$. The stepwise value of $\eta$ varies during job execution and the average value for the whole job becomes: ($T_a$ is divided into $N$ equal time intervals $= \Delta T$, $W(T_i) > 0$ for $K$ steps).

$$\hat{\eta} = \frac{\sum_{i=0}^{N} W(T_i) \Delta T}{MT_a}$$

Note that under this interpretation, $\hat{\eta}$ measures the fit between this particular problem and a given configuration. If $\hat{\eta} = 1.0$ the configuration has precisely the resources required to solve the problem within time constraints, assuming that the load is completely uniform (with non-integral width in most cases). Although $\hat{\eta}$ will be much less than 1.0 in most cases, it is interesting to compare the values obtained for processors of different organizations and execution speeds, executing the same job (identical at least on a macroscopic scale). Implicit in the stepwise summation of the instruction time—processor width product are factors such as the suitability of the particular instruction repertoire to the problem (number of steps), hardware technology (execution time), and organizational approach (treated in the following section).

A criterion $\pi$ is expressed as the inverse ratio of time of execution of a given job with parallel processors to the execution time with only one such processor (speedup over the job). Expressing $\pi$ in terms of job width $W$ gives for any job step

$$\pi = \frac{\text{sequential processor execution time}}{\text{parallel processor execution time}} = W(T)$$

Similarly, averaging this quantity over an entire job during the available time gives:

$$\hat{\pi} = \frac{\sum_{i=0}^{N} W(T_i) \Delta T}{Ta}$$

or simply:

$$\hat{\pi} = \hat{\pi}M$$

which states that the speed of execution of a computation on parallel hardware as contrasted to a single processing element of that hardware is proportional to the efficiency of hardware utilization times the number of available processing elements. Again, $\hat{\pi}$ measures the equivalent number of parallel processors required assuming a uniform load (width $= \hat{\pi}$, duration $= T_a$).

**PROCESSOR ORGANIZATIONS**

**General observations**

In the analysis which follows, job parallelism is calculated on an instruction by instruction step basis. For the purposes of this discussion, consider a more macroscopic model of job parallelism. Sets of instructions with varying parallelism widths will be treated as phases ($\phi_i$), with phase width defined as the maximum instruction width within the phase. (see Figure 5, for a three phase job, with instructions indicated by dashed lines).

Given this model, it is possible to treat the parallel component in at least three distinct ways (see Figure 6). The simplest approach is to treat a parallel step of width $N$ as $N$ steps of width one which are executed serially. This would correspond to three loops (with conditional branches) of iteration counts $W_1, W_2, W_3$ or one loop with iteration count $\max[W_1, W_2, W_3]$. The worst case execution time (macroscopic model) for width $N$ would be $T = NT_a$.

Parallel processing, in its purest sense, devotes one processing element ($PE$) to each slice of width one, and executes the total phase in $T = ta$, where $ta$ is the total execution time for any element. Variations on this basic theme are possible. For example, STARAN, the Goodyear Aerospace associative
processor\textsuperscript{2,10,11} is actually an ensemble of bit-slice processors\textsuperscript{16} arranged in arrays of 256 each having access to 256 bits of storage. The ensemble is capable of bitwise operations on selected fields of storage. Since the bulk filter algorithm requires 768 bits of storage for the information associated with one filter calculation, i.e. track, a “black box” model devotes three PE’s to each object in track (generally one of the three is active at any instruction step).

The converse of the STARAN case is exemplified by the Parallel Element Processing Ensemble (PEPE),\textsuperscript{6} which devotes \( M \) PE’s to \( N \) tracks, \( M < N \). In this case, the total processing time for one phase would be \( T = \lceil N + M \rceil \delta_t \), since each PE may process up to \( \lceil N + M \rceil \) tracks sequentially. Note that for parallel correlation of \( K \) returns (associative operations such as “between limits search”), at least \( K \) PE’s must be available, since objects which are illuminated by a single beam must be handled by separate processors.

A third approach is analogous to the pipelining of instruction execution. Assuming that each phase has execution time \( \delta_P \), one could use one sequential processor to handle execution of each phase, buffering the input and output of contiguous phases to achieve a total execution time of \( T = (N - 1 + m) \delta_P \) for an \( M \) stage process. The Signal Processing Element (SPE) designed by the US Naval Research Laboratory\textsuperscript{17} can utilize this strategy of functional decomposition, linking fast micro-programmed arithmetic units under the control of a master control unit to achieve \( \delta_P \leq M^{-1} \delta_s \) for “sequential” machines of the CDC 7600, IBM 370/195 class \( (T \leq \lceil M^{-1}(N-1) + 1 \rceil \delta_s) \).

One other factor of considerable importance is the number of control streams active in each processor array. The simplest arrangement is a single control stream, broadcast to all elements from a central sequencing unit. Individual PE’s may be deactivated for part of a program sequence by central direction, or dependent upon some condition determined by each PE. Dual control units mean that arithmetic and correlative operation can proceed simultaneously, allowing the two phase strategy outlined earlier to work efficiently (one control stream would require an “interruptible arithmetic” strategy, or well defined, non-overlapping, search/verify and arithmetic intervals). These two control streams can act on different sets of PE’s (e.g. each PE has a mode register which determines the central stream accepted by that PE), or both control streams can share the same PE on a cycle stealing basis (PEPE IC model).

**Configurations considered**

Table I presents the basic data on the four hardware configurations considered for the bulk filter problem. Sizing estimates are based upon the assumptions described previously, i.e. 1950 tracks in processing at any given time (steady state). Over any 25 Ms interval, half of the tracks are being correlated, half are being processed arithmetically.

The Kalman filter results compare the performance of STARAN, PEPE (IC model), and the CDC 7600 in sustaining a precise track of 120 objects (1 observation each 50 Ms) using a basic model of the Kalman filter algorithm. The STARAN solution attempts to take advantage of the parallelism within the algorithm (matrix-vector operations). Twenty-one PE are devoted to each object being tracked. PEPE would handle one Kalman filter sequence in each of its PE’s, performing the computations serially within the PE.

**COMPARATIVE RESULTS**

**Bulk filter**

Table II presents the values for \( \eta, \bar{\eta}, \bar{\eta}, \text{and } \bar{\eta} \), and execution time for each of the 4 processor configurations. As has been explained earlier \( \eta \) and \( \bar{\eta} \) differ only in the definition of hardware space used in the denominator of the \( \eta \) expression. It is interesting to note that although the efficiency \( \bar{\eta} \) over the constrained interval is not large for any of the parallel processors, all three do utilize their hardware efficiency over the actual arithmetic computation (\( \eta \)). The implication is that some other task could be handled in the idle interval, a
TABLE I—Bulk Filter Processor Configuration Comparison

<table>
<thead>
<tr>
<th></th>
<th>STARAN</th>
<th>HONEYWELL</th>
<th>PEPE (IC model)</th>
<th>CDC 7600</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>3 PE/TRACK</td>
<td>1950</td>
<td>PE/TRACK</td>
<td>PE/20 TRACK</td>
</tr>
<tr>
<td>Number of PE's (1950 track load)</td>
<td>30 Arrays = 7680 PE's</td>
<td>1950</td>
<td>100</td>
<td>1</td>
</tr>
<tr>
<td>32 bit fixed point Add time/PE</td>
<td>18.0 μsec</td>
<td>.75 μsec</td>
<td>.25 μsec</td>
<td>27.5–55 n.s. (60 bit)</td>
</tr>
<tr>
<td>Control Streams</td>
<td>Single—(Standard option) all PE's correlate or perform arithmetic functions</td>
<td>Double—Each PE may be in correlation or arithmetic mode</td>
<td>Double—EACH PE may perform correlation and arithmetic functions simultaneously</td>
<td>Single pipelined</td>
</tr>
<tr>
<td>Approximate gate count/PE (not including storage)</td>
<td>82 (21,000/256 PE array)</td>
<td>2,400</td>
<td>9,000</td>
<td>170,000</td>
</tr>
<tr>
<td>Gate Count for configuration (PE's only)</td>
<td>630,000</td>
<td>4.68 x 10^9</td>
<td>900,000</td>
<td>170,000</td>
</tr>
<tr>
<td>Adds/sec x 10^9 (~MIPS)</td>
<td>437</td>
<td>2900</td>
<td>400</td>
<td>18</td>
</tr>
<tr>
<td>Gates/track</td>
<td>320*</td>
<td>2400</td>
<td>450</td>
<td>87</td>
</tr>
</tbody>
</table>

* Based on a 30 Array configuration—246 are required for the algorithm.

more sophisticated filter algorithm could be used, or the PE's could be built from slower, less expensive logic. It should be stressed that the gate counts given are strictly processing element gates, not including memory, unit control, or other functions.

Kalman filter

As noted above, 60 precision tracks must be updated in 25 Ms while 60 track looks are being correlated. Benchmark data for the CDC 7600 indicates that a Kalman filter calculation consisting of 371 multiplies, 313 add/subtracts, 2 divides, 6 square roots, and 1 exponentiation will require approximately 0.3 Ms (18 Ms for 60 tracks). This leaves a reserve of 7 Ms out of a 25 Ms interval for correlation. An analysis of the STARAN processor applied to the Kalman filter indicates that with 21 processing elements assigned to each precision track the calculation can be carried out in slightly less than 25 Ms. This performance is achieved by decomposing the filter calculation into 56 multiplies, 61 add/subtracts, 3 divides, 4 square roots and is achieved at the cost of 322 move operations. Figure 7 shows the processor activity for the first 15 instructions of the Kalman filter sequence. One bank of STARAN processing elements (5256 element arrays) containing 1280 processors is required to update tracks for 60 objects in one 25 Ms interval and correlate returns during the other. The PEPE configuration would require 60 processing elements (two track files per element) taking advantage of this hardware's ability to do arithmetic calculations and correlations simultaneously, achieving a 45 percent loading (11.3 Ms execution time per Kalman filter sequence) of each PEPE processing element. Table III summarizes the Kalman filter results.

OBSERVATIONS AND CONCLUSIONS

It should be emphasized that this study was not an attempt to perform a qualitative evaluation of the processor organizations described in the studies. Each of the proposed configurations is more than capable of handling the required calculations in the time available. System cost is really outside the scope of this paper. In particular, gate count is not a good indicator of system cost. The circuit technology (speed, level of integration) and chip partitioning (yield, number of unique chips) trade-offs possible within the current state of the art in LSI fabrication relegate gate count to at most an order of magnitude indicator of cost.

Each of the three parallel processor organizations represents a single point on a trade-off curve in several dimensions (i.e. processor execution speed, loading, and cost, control stream philosophy, etc.). Given an initial operating point, determined by the functional requirements of the problem, the system designer must define a set of algorithms in sufficient detail to

<table>
<thead>
<tr>
<th></th>
<th>STARAN</th>
<th>HONEYWELL</th>
<th>PEPE (IC model)</th>
<th>CDC 7600 SEQUENTIAL</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>3 PE/TRACK</td>
<td>PE/TRACK</td>
<td>PE/20 TRACK</td>
<td>PE/TRACK</td>
</tr>
<tr>
<td>Correlation time</td>
<td>1.8 msec</td>
<td>15.9 msec</td>
<td>2.1 msec</td>
<td></td>
</tr>
<tr>
<td>θ</td>
<td>.006</td>
<td>.035</td>
<td>.68</td>
<td></td>
</tr>
<tr>
<td>θ</td>
<td>139</td>
<td>34</td>
<td>68</td>
<td></td>
</tr>
<tr>
<td>θ</td>
<td>.0026</td>
<td>.022</td>
<td>.057</td>
<td></td>
</tr>
<tr>
<td>θ</td>
<td>9</td>
<td>22</td>
<td>5.7</td>
<td></td>
</tr>
<tr>
<td>Arithmetic time</td>
<td>14.5 msec</td>
<td>5.1 msec</td>
<td>3.85 msec</td>
<td></td>
</tr>
<tr>
<td>θ</td>
<td>16</td>
<td>80</td>
<td>79</td>
<td></td>
</tr>
<tr>
<td>θ</td>
<td>2450</td>
<td>780</td>
<td>79</td>
<td></td>
</tr>
<tr>
<td>θ</td>
<td>.38</td>
<td>.10</td>
<td>.122</td>
<td></td>
</tr>
<tr>
<td>Total time</td>
<td>16.3 msec</td>
<td>21 msec</td>
<td>5.95 msec</td>
<td></td>
</tr>
<tr>
<td>θ</td>
<td>.30</td>
<td>.11</td>
<td>.75</td>
<td></td>
</tr>
<tr>
<td>θ</td>
<td>2270</td>
<td>216</td>
<td>75</td>
<td></td>
</tr>
<tr>
<td>θ</td>
<td>.19</td>
<td>.093</td>
<td>.18</td>
<td></td>
</tr>
<tr>
<td>θ</td>
<td>1480</td>
<td>181</td>
<td>18</td>
<td></td>
</tr>
<tr>
<td>θ</td>
<td>128</td>
<td>286</td>
<td>300</td>
<td>18</td>
</tr>
</tbody>
</table>
convince himself that he can operate within any given constraints. Fine tuning of the system is accomplished by restructuring the algorithms, redefining the operating point, or both. In the two cases treated in this paper, elapsed time is the crucial measure of system performance (in a binary sense—it does or does not meet the requirement). The purpose of the $g$ and $f$ calculations, as well as the step by step processor activity diagrams is to provide some insight beyond the elapsed time criteria which might be helpful in restructuring algorithms, or modifying some aspect of the system's architecture such as control stream philosophy. The properties of the processor activity diagrams are of significant interest in determining the number of PE's that are required to handle the given load (uniform load implies fewer PE's and higher $g$). The measures used in this paper are of some interest because of the fact that they are functions of problem width and instruction execution time, allowing factors such as the selection of a particular instruction set to enter into the values of the resultant tuning parameters.

Several more specific observations are in order. First, for the particular bulk filter case considered, the CDC 7600 can easily handle the computational load. Proponents of the parallel processor approach would claim that quantity production of PE's, utilizing LSI technology, would enable them to produce equivalent ensembles at less than a CDC 7600's cost. In addition, computation time for the parallel ensembles is only a weak function of the number of objects in the correlation phase, and essentially independent of object load in the arithmetic phase. Therefore, it would be simple to scale up the capabilities of the parallel processors to handle loads well beyond the capability of a single, fast sequential processor. The functional pipelining approach advocated by the Naval Research Laboratory would appear to be the strongest challenger to the parallel approach in terms of capabilities and cost (and to a somewhat lesser extent, flexibility). Very rough estimates indicate that the bulk filter case presented here could be handled by no more than two arithmetic units (each with $\approx 10,000$ gates) and a single microprogrammed control unit ($\approx 5,000$ gates). Tasks which stress the correlative capabilities of parallel arrays rather than the parallel arithmetic capabilities should show the parallel array architecture to its greatest advantage.

ACKNOWLEDGMENTS

The authors wish to express their gratitude to Messrs. Brubaker and Gilmore, Goodyear Aerospace Corporation for providing STARAN logic gate counts and Kalman filter computation timing estimates; to Mr. W. Alexander, Honeywell Inc. for providing PEPE and Honeywell Associative Processor Logic gate counts; and to Mr. J. Burford, Control Data Corporation for providing the CDC 7600 logic gate counts. The study reports were prepared under the direction of the Advanced Ballistic Missile Defense Agency, whose cooperation is greatly appreciated.

REFERENCES


<table>
<thead>
<tr>
<th>TABLE III.—Results of Kalman Filter Analysis</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Time</strong></td>
</tr>
<tr>
<td><strong>ms</strong></td>
</tr>
<tr>
<td>25</td>
</tr>
<tr>
<td>$\tau$</td>
</tr>
<tr>
<td>$\tau$</td>
</tr>
<tr>
<td>$\tau$</td>
</tr>
<tr>
<td>$\tau$</td>
</tr>
<tr>
<td>$\mu$MIPS</td>
</tr>
<tr>
<td>gates/track</td>
</tr>
</tbody>
</table>

NOTE: Correlation time for the Kalman filter is not significant ($\sim 100 \mu s$) since each track is assigned a unique track number number (120 total). Accordingly, only total time figures are presented.

Figure 7—STARAN Kalman filter loading (one track, first 15 instructions)


