Decomposition of data flow graphs on multiprocessors*

by W. C. BRANTLEY, JR., G. W. LEIVE and D. P. SIEWIOREK
Carnegie-Mellon University
Pittsburgh, Pennsylvania

ABSTRACT

Methodologies are presented for decomposing algorithms on a classical multiprocessor. The class of algorithms considered are those that can be represented as data flow graphs without decision elements. Two passive sonar signal processing modes are used as detailed examples. Decompositions are performed by modeling data memory bandwidth and data memory interference as the primary constraints on execution speed. Algorithms with low interference between data flow graph nodes require only simple models of memory interference to be successfully decomposed. For high interference algorithms the memory interference can be (1) modeled using a linear model of interference, (2) modeled by a queuing network of exponential servers which is solved computationally, or (3) modeled exactly and then simulated. These three techniques give estimates of the aggregate effect of memory interference to be factors of 10.9, 2.11, and 1.82, respectively (the latter being the most accurate).

INTRODUCTION

In recent years interest has heightened in multiprocessor structures constructed from low cost mini and microprocessors. While such multiprocessors offer a significant cost/performance improvement over traditional uniprocessors, that advantage cannot be obtained unless effective methodologies exist for decomposing problems to execute in parallel. Effective decompositions have been made for specific applications and small grain parallelism; however, no methodology has been developed for a general problem. This paper presents a methodology for decomposing problems represented as data flow graphs on a classical multiprocessor. The methodology requires only minimal restrictions on the data flow graph. These restrictions are met by at least one important class of applications, passive sonar processing, which will be used as illustration. The restrictions on the data flow graphs are:

- The system must operate within realtime on the multiprocessor. Realtime operation means that in a given repetitive cycle the system data processing rate exceeds the data input rate.
- The data flow graphs are compositions of macros between which data flows. In the case of signal processing these are convolutions, digital filters, FFTs, and transforms.
- Macros are considered to be indivisible and non-preemptable; once started they run to completion.
- The processing times for macros are modeled as constants.
- Macros are initiated by the availability of data.
- There are no data dependencies in control flow; data dependent decisions must be made within the macros.

Constraints on the methodology and the example decompositions are:

- The decomposition techniques discussed are only valid for a predictable, repetitive data flow process operating in steady state; system start-up transients are ignored.
- The methodology only applies to multiprocessor architectures with identical processors.
- The decomposition methodology assumes static binding of macros to processors.

The second section of this paper depicts the architecture of the multiprocessor assumed in the example decompositions. General methodology and methodology for low interaction data flow graphs is given in the third section. The fourth section illustrates the methodology on a passive sonar algorithm: Constant Percentage Resolution LOFAR (Low Frequency Analysis Recording). Another class of data flow graphs, those with high interaction, is treated in the fifth section followed by an example decomposition of Constant Resolution LOFAR Beamforming (CRBF) in the sixth section. The last section summarizes the results and indicates some areas for future research.

* This investigation was partially supported by the Naval Research Laboratory, Washington, D.C. under Contract Number N00173-76-C-0046.
MULTIPROCESSOR ARCHITECTURE

The decompositions are performed for a classical multiprocessor typified by C.mmp. C.mmp (Figure 1) consists of 16 Digital Equipment Corporation PDP-11 processors communicating through a central crosspoint switch (Smp) to 16 memory modules. The major features of C.mmp are:

Mp: The primary memory consists of 16 modules of 65k 16 bit words. The memory access or cycle time (including switch and bus delay) is approximated as 1.0 microseconds.

Smp: The memory/processor crossbar switch connects any of the 16 processors to any of the 16 memory modules with a maximum concurrency of 16.

Pc: The processing elements may be any member of the PDP-11 family. The current implementation actually has a mixture of PDP-Ill 20s and PDP-I1140s.

Dmap: The address mapping component is the Dmap which intercepts 18 bit addresses generated by the processor and converts them to 21 bit physical memory addresses.

Functionally specialized processor definition

With PDP-11/40 processors, C.mmp is unable to execute any of the passive sonar signal processing modes in real-time. Since the primary goal was to evaluate the memory-switch architecture, each C.mmp processor must be replaced with a functionally specialized processor (P.fs). P.fs is defined such that memory bandwidth and memory interference are the architectural constraints on the decompositions. Memory access time will be the limiting constraint on execution speed if P.fs executes primitive data operations in one main memory cycle time. To satisfy this requirement, instructions for executing the signal processing functions would be microcoded and stored in a high speed ROM within each P.fs. All accesses to main storage are for data. A P.fs will directly replace each C.mmp processor for the example decompositions.

The specification for P.fs may be put into perspective with other signal processors by considering the time to do an FFT. A butterfly of the FFT would take 10 microseconds or 10 memory cycles on the C.mmp architecture using P.fs processors. Therefore, a 1024 complex point FFT could be executed in 51.2 msec on the augmented C.mmp. The commercially available SPS-41 is a functionally specialized, high performance processor which takes 8.3 msec for a 1024 complex point FFT. In comparison with the SPS-41, the augmented C.mmp FFT with the P.fs specification is quite conservative.

GENERAL METHODOLOGY AND LOW INTERACTION DATA FLOW GRAPHS

In this section, general methodologies for decomposing algorithms are discussed. First, an outline is presented of a preliminary analysis that can be used for general decompositions. Subsequently, the methodologies for low memory interference and high memory interference data flow graphs are discussed. The goal of the methodology is to maximize the number of identical instantiations of a given data flow graph running on the multiprocessor concurrently and in realtime. Each instantiation of the data flow graph is called a channel.

Preliminary analysis

The following items are determined before trial decompositions are started:

- A data flow graph for a single channel must be available or must be developed.
- The processing time for each node of the data flow graph must be determined or estimated.
- The node processing times must be normalized to a convenient time interval (one second was used throughout the examples).
- The number of memory references (per unit of time) on each arc of the data flow graph must be determined (estimated) and then normalized to the same interval used for the processing times.
- The input data rate must be determined and normalized.

The following preliminary analyses are performed to detect degenerate cases:

- If any node requires more than one normalized unit of
time to execute, there is no feasible decomposition (nodes are not decomposable).
• If the sum of all the node execution times for all the channels is less than one normalized unit of time, the processing requirements are met by a uniprocessor.
• Ignoring memory interference, determine the processing required for one channel.
• An upper bound on the number of channels that can be supported is determined by dividing the number of available processors by the total normalized processing time required for one channel.

Initial decompositions—Low interaction data flow graph

The following steps are performed in a trial decomposition. The example is keyed to the methodology identifiers below.

ML-1 All special nodes which are not part of the cyclic processing defined by the data flow graph are bound to the minimum number of processors and memories that satisfy processing time requirements. This class of nodes would include input and output functions.
ML-2 Normalize node times. If the normalized execution time for any node exceeds one, no possible decomposition exists.
ML-3 Decompose one channel by assigning the largest number of adjacent nodes to processors such that the sum of all normalized node times assigned to any processor does not exceed one.
ML-4 Replicate the decomposed channel until available processors are exhausted.
ML-5 To assign overflow channels, determine if any processor has enough slack time to execute at least the largest node of a channel. If there is not enough time available, no overflow channels can be allocated. Determine if the sum of the available slack times is large enough to process at least one channel.
ML-6 Decompose overflow channels until slack is exhausted or until a complete channel will not fit in the remaining slack time.
ML-7 Determine the effect of memory interference on the trial decomposition by using a linear model of memory interference. If the normalized execution time requirement (including memory interference) for any processor exceeds one, remove nodes from that processor until the requirement drops below one. Remove any nodes that do not belong to integral channels.

Memory interference determination—Low interaction data flow graph

Memory interference encountered in tree structured data flow graphs is caused by several processors attempting simultaneous block transfers involving a single memory bank. Memory interference is seen as an apparent increase in memory access time. The apparent increase in memory access time is linear with the number of processors in conflict. To illustrate this concept, consider the following logical connection of processors and memories in Figure 2.

In Figure 2 memory M1 is accessed by processors P1, P2, and P3 for a total of b, c, and e accesses, respectively. Memory M2 is accessed by P1 and P3 for a total of a and d accesses, respectively. Interference results if the processors attempt to reference a memory simultaneously. For example, consider transfers b, c, and e to maximally interfere (i.e., they start simultaneously) and c < e < b. From the point of view of P1, b accesses now takes b + min(b, c) + min(b, e) = b + c + e time units. Similarly, from the point of view of P2, its c accesses to M1 will require c + min(c, b) + min(c, e) or 3c time units.

DECOMPOSITION EXAMPLE—LOW INTERACTION DATA FLOW GRAPH

Constant Percentage Resolution (CPR) LOFAR is an example of a passive sonar mode whose data flow graph has low interaction. CPR LOFAR computes the power spectral density by octaves for each channel. Figure 3 is the data flow graph for one channel of CPR LOFAR. Typically, as many as 16 channels of data are processed concurrently.

The CPR LOFAR modes are described below. The node execution times given were determined by scaling similar node times for a specific military signal processor. The scale factor was determined by taking the ratio of the PIFS FFT butterfly time to the butterfly time for the military signal processor. Exact algorithms for the nodes are given in References 6-8.

INPUT: Receives the digitized transducer data and provides blocked data to DEMOD.
DEMOD: Divides the real input data into two parts: a low pass filtered part which is decimated by a factor of two and passed to the next DEMOD stage, and a high pass filtered part which is transformed

Figure 2—Block transfer memory interference example
FFf:

WEIGHT:

Figure 3—CPR LOFAR data flow graph

into the real and imaginary components of complex data points. The complex data points are bandshifted to baseband and the resulting complex array is passed to the FFT. From 2048 real points as input (2048 R in Figure 3), DEMOD produces 512 complex points for FFT (512 C) and 1024 real points for the next DEMOD. The execution time for DEMOD is 97.5 msec. on P.fs.

FFT:

Enhances certain aspects of the data by computing weighted sums of four complex terms. From each 512 complex point buffer from the FFT, WEIGHT produces 256 complex points. The execution time for WEIGHT is 9.1 msec.

DETECT: Calculates an approximation to the magnitude of each complex intensity. DETECT produces 256 real points from 256 complex points. The execution time for DETECT is 3.3 msec.

STI:

Short Term Integration averages a number of spectra to smooth data and to eliminate false alarms due to non-recurring noise in a sample. STI takes the block of data from DETECT and accumulates to a real vector of size 256 real points. The execution time for STI is 3.3 msec.

POST:

Is a summary term for various calculating, formatting, and display processing to interface the data to the display. POST is not shown in Figure 3.

The sample rate is 8192 Hz on each channel. Real data are represented with one 16 bit word and complex data with two such words. More details of the decomposition can be found in Reference 4.

Preliminary analysis

The initial step of the decomposition determines if it is possible to decompose CPR LOFAR on the modified C.mmp architecture. The execution times given show that the node requiring the most time is DEMOD. The ratio of required execution time to available time (97.5 ms/250 ms) is less than one, so a feasible decomposition is possible.

Decomposition

By method step ML-1, one processor/memory pair is dedicated to data acquisition and display storage. The remaining 15 processors are available for CPR LOFAR computations. The I/O processor receives data samples from all transducers and distributes the data samples to the appropriate channel processes. The I/O memory bank stores the processed data from all octaves of all channels. The I/O processor is available for any post processing of the data and for distribution of the processed data to the display.

The normalized execution time of node \( i \) in octave \( j \) for channel \( k \) is represented as \( t_{i,j,k} \) for the remainder of the CPR LOFAR decomposition. The normalized execution time for octave \( j \) of channel \( k \) is \( T_{j,k} = \sum_{i=1}^{m} t_{i,j,k} \), where the limit \( m \) is the maximum number of nodes in an octave (\( m=5 \) in this example).

CPR LOFAR (refer to Figure 3) is a tree data flow graph where octaves depend on preceding octaves for data input. The input data buffer to each octave is separated into two groups: half of the input is processed by the receiving octave and the other half of the data is filtered then passed to the next octave. An octave can initiate when its input buffer is full. Input buffers for all octaves are the same size.
An octave executes twice for each execution of its successor. Octave eight executes once for each data buffer supplied by INPUT. The normalized octave execution (method step ML-2) times are given in seconds per second:

\[
\begin{align*}
T_{8,k} &= 0.5456 \\
T_{7,k} &= 0.2728 \\
T_{6,k} &= 0.1364 \\
T_{5,k} &= 0.0682 \\
T_{4,k} &= 0.0341 \\
T_{3,k} &= 0.0171 \\
T_{2,k} &= 0.0085 \\
T_{1,k} &= 0.0043
\end{align*}
\]

Step ML-3 of the methodology is used next. Since the largest normalized octave execution time (T_{8,0}) is less than one, CPR LOFAR is decomposed by octaves rather than by nodes. To minimize memory interference, octave groups are bound to processors starting with octave eight. Successive octaves are bound to a processor until unity normalized time would be exceeded by adding another octave. In this example, octaves six, seven and eight fit on one processor (T_{6,k}+T_{7,k}+T_{8,k}=0.9548<1). The execution time for the other five octaves is \( \sum_{i=1}^{5} T_{i,k} = 0.1312 \) seconds/second. Thus one processor can support the lower five octave processing requirements for \( 1/0.1312 = 7.56 \) channels.

The decomposition on 16 processors for 12 channels of CPR LOFAR is as proceeds as follows using steps ML-4, ML-5, and ML-6. One processor is allocated to I/O processing. The remaining 15 processors are allocated in two groups of seven processor/memory pairs. Within a group, six processors execute octaves eight, seven, and six for each of six channels. The seventh processor within a group executes octaves five through one for the six channels. One processor on C.mmp is not needed. Figure 4 is a diagram of a portion of the decomposition for CPR LOFAR.

The single I/O processor (P.io) and I/O memory (M.io) are shown in the diagram. A processor and memory dedicated to the upper three octaves of one of the twelve channels (P.o86 and M.o86) is shown. A processor and a memory dedicated to the lower five octaves of six channels (P.o51 and M.o51) is shown. The data access variables (a through f) represent the accesses for only one channel. The variables are normalized to data accesses per second so that memory interference can be determined (method step ML-7). The values associated with the data access variables are:

\[
\begin{align*}
a &= 248 \text{ words per second.} \\
b &= 9984 \text{ words per second.} \\
c &= \text{an unspecified (large) number of accesses available for display data processing.} \\
d &= 9548000 \text{ words per second.} \\
e &= 132138 \text{ words per second.} \\
f &= 1024 \text{ words per second.}
\end{align*}
\]

The maximum memory access rate for C.mmp is \( 10^6 \) words per second. The maximum normalized execution times are computed as:

\[
\begin{align*}
\text{(Number of channels) (apparent accesses) + (max accesses)}
\end{align*}
\]

\[
\begin{align*}
P_{io}: &\quad (12) (21448)/10^6 = 0.257 \quad \text{(does not include ''c''} \\
P_{o86}: &\quad (1) (965808)/10^6 = 0.966 \\
P_{o51}: &\quad (6) (135458)/10^6 = 0.813
\end{align*}
\]

The normalized execution times for all processors are less than one for the best and worst cases of memory interference. All processes are able to complete execution in sufficient time to maintain realtime operation of the system. When a processor is finished processing a block of data, it will execute a WAIT instruction to wait for the I/O processor to interrupt it with the next block of data.

METHODOLOGY FOR HIGH INTERACTION DATA FLOW GRAPHS

Data flow graphs having a high degree of interaction between some nodes cannot utilize the decomposition technique described for low interaction data flow graphs. The linear model of memory interference used for low interaction data flow graphs assigns the more processors and memories than are necessary when memory interference is intensive but complex enough to be modeled as a random process. The example is keyed to the methodology identifiers below. The decomposition technique with a more sophisticated model of memory interference is:

MH-1 Allocate the low interaction nodes to memory-processor pairs using the procedure described for low interaction data flow graphs.

MH-2 Assign the data shared by the highly interactive nodes to the remaining memories and assign the nodes to the remaining processors. Formulate a queuing model of the resulting network of processors and memories. The queuing model is a fully connected network of servers, each with a queue, where servers represent memory modules.
and customers represent processors. Associated with each edge of the network is a probability that the edge will be traversed when the source server finishes. These probabilities are determined by the assignment of nodes and their data.

MH-3
For each server in the queuing network determine the average time between the arrival time to the queue and the departure time of the customer (called the time in system). Time in system for a server corresponds to the effective memory cycle time of the memory as seen by a processor. Approximate the memory cycle to be exponentially distributed with mean equal to the real memory cycle time (a constant). To estimate the time in system, use Buzen's method. Using the effective cycle time for each memory and the number of memory references made by processors memories, calculate normalized finish times. Increase normalized finish times to include interaction with processors in the low interaction group. From finish times calculate slack times (i.e., one minus the finish times). The slack times must be positive.

MH-4
The methodology for high interaction data flow graphs is illustrated by CRBF in which memory interference is a dominant part of the processing load. Unlike CPR LOFAR, CRBF cannot be broken into memory-processor pairs that have little interaction. The processors performing the BF nodes uniformly access all of the memories containing FFT results and all of the memories for the resulting beams (Figure 5).

DECOMPOSITION EXAMPLE—HIGH INTERACTION DATA FLOW GRAPH

Constant resolution LOFAR beamforming (CRBF) calculates the power spectrum of the acoustical energy received along an azimuth. Data for beamforming is acquired by sampling the output of an array of transducers in a known geometry. The beamforming algorithm is derived from the geometry of the transducers. Beamforming may be performed in the time domain or in the frequency domain; frequency domain beamforming is considered here. The equation for frequency domain beamforming is

\[ R(\omega) = \sum_{n=0}^{2^{m-1}} S(\omega,n) \exp \left( -2\pi i n d \sin \theta / c \right) \]

Where \( S(\omega,n) \) is the Fourier transform of the signal from the \( n \)th channel, \( i \) is \((-1)^{1/2}\), and \( \omega \) is 2\( \pi f \).

Figure 5 is the data flow graph of the CRBF mode which forms B beams from C transducers. The various nodes in the CRBF data flow graph are:

INPUT: Is identical to the input process of CPR LOFAR.

LPF: The Low Pass Filter limits the high frequency components of the sampled data and separates samples into inphase and quadrature components. From a block of 2048 real points, LPF produces a block of 1024 complex points. The execution time for LPF, \( T_{LPF} \), is 61.7 msec.

FFT: The Fast Fourier Transform (FFT) is a complex FFT that is used to convert the complex time domain data to complex frequency domain data. The execution time for the 1024 complex point FFT, \( T_{FFT} \), is 51.2 msec.

BF: Forms B beams for the \( j \)th frequency bin from FFTs of the C channels using Gortzels method. The execution time for BF is 56.3BC msec.

WEIGHT: Is identical to the WEIGHT of CPR LOFAR. From each 1024 complex point buffer from the FFT, WEIGHT produces 1024 complex points. The execution time for WEIGHT, \( T_{WEIGHT} \), is 25.7 msec.

DETECT: Calculates an approximation to the magnitude of each complex intensity (same as CPR LOFAR). From 1024 complex points, DETECT produces 1024 real points. The execution time for DETECT, \( T_{DETECT} \), is 12.9 msec.
STI: Is the same as STI of CPR LOFAR. The size of STI is 1024. The execution time for STI, \( T_{STI} \), is 12.8 msec.

POST: Is identical to POST of CPR LOFAR.

The first order analysis (method step MH-1) of the capacity of C.mmp for CRBF ignores memory interference within the pre-BF or post-BF nodes. The decomposition proceeds by first allocating \( N_{PRE} = \lceil C \times (T_{LPF} + T_{FFT} + T_{O}) \rceil \) processors and memories for pre-BF processing (\( \lceil x \rceil \) symbolizes the least integer greater than or equal to \( x \)). Next, \( N_{POST} = \lceil B \times (T_{WEIGHT} + T_{DETECT} + T_{STI} + T_{OUT}) \rceil \) processors and memories are assigned for the post-BF nodes. The remaining \( N_{BF} = 16 - N_{POST} - N_{PRE} - 1 \) processors and \( B + C \) of the memories are allocated for the BF nodes. The maximum number of BF nodes any processor must compute is \( 1024 \). Slack time, \( S_{BF}(C,B) \), is the minimum idle time remaining in each one second period for any of the BF processors. If \( S_{BF} \) is positive then, to first order, \( B \) beams can be formed from \( C \) channels of data on C.mmp using P.fs.

\[
S_{BF}(C,B) = 1000 - 4 \times (T_{IO} + C \times B \times T_{BF}) \times \left( \frac{1}{1024 + Up(N_{BF}, B+C)} \right) \text{msec.}
\]

Where \( C \times B \times T_{BF} \) is the beam forming time given for one bin, \( Up \) is the processor utilization from Table 1, and \( T_{IO} \) is the time to move the data and results in the BF memories to other memories. To simplify computations, \( T_{IO} \) is calculated assuming no memory interference, but this assumption must be verified. Figure 6 is a graph of the capacity of C.mmp in beams (B) and channels (C).

BF with five channels and four beams will now be decomposed in detail. The computation time to reduce 1000 msec of data for the pre-BF and post-BF nodes are \( T_{PRE} \) and \( T_{POST} \), respectively, giving:

\[
T_{PRE} = 4 \times (T_{LPF} + T_{FFT}) = 4 \times (16.1 + 51.4) = 452.4 \text{ msec.}
\]

\[
T_{POST} = 4 \times (T_{WEIGHT} + T_{DETECT} + T_{STI}) = 4 \times (25.7 + 12.9 + 12.8) = 205.6 \text{ msec.}
\]

Therefore, \( N_{PRE} = 3 \) and \( N_{POST} = 1 \). With one processor for I/O, eleven processors are left for BF nodes so that one BF processor will compute 94 BF nodes and the other ten will compute 93. Memory is allocated as follows: one bank for I/O, one bank for the post-BF processor, three banks for the three pre-BF processors, and \( B + C = 9 \) banks for the eleven BF processors. The allocation of processors and memories is shown in Figure 7. The pre-BF nodes are divided among the three processor-memory pairs (PRE1, PRE2, PRE3): PRE1 has LPF1, LPF2, and FFT; PRE2 has LPF3, LPF4, and FFT2; and PRE3 has LPF5, FFT3, FFT4, and FFT5 (subscriber on a node indicates channel number). Both channel two and four have their LPFs in one processor and their FFTs in another processor, causing 8192 words to be written to memory PRE4 from both processors.

<table>
<thead>
<tr>
<th>TABLE 1—Up(N, M) in percent</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Memories, M</td>
</tr>
<tr>
<td>N 1 100.0 100.0 100.0 100.0 100.0 100.0 100.0 100.0 100.0 100.0</td>
</tr>
<tr>
<td>N 2 100.0 100.0 100.0 100.0 100.0 100.0 100.0 100.0 100.0 100.0</td>
</tr>
<tr>
<td>N 3 100.0 100.0 100.0 100.0 100.0 100.0 100.0 100.0 100.0 100.0</td>
</tr>
<tr>
<td>P 4 100.0 100.0 100.0 100.0 100.0 100.0 100.0 100.0 100.0 100.0</td>
</tr>
<tr>
<td>P 5 100.0 100.0 100.0 100.0 100.0 100.0 100.0 100.0 100.0 100.0</td>
</tr>
<tr>
<td>P 6 100.0 100.0 100.0 100.0 100.0 100.0 100.0 100.0 100.0 100.0</td>
</tr>
<tr>
<td>P 7 100.0 100.0 100.0 100.0 100.0 100.0 100.0 100.0 100.0 100.0</td>
</tr>
<tr>
<td>P 8 100.0 100.0 100.0 100.0 100.0 100.0 100.0 100.0 100.0 100.0</td>
</tr>
<tr>
<td>P 9 100.0 100.0 100.0 100.0 100.0 100.0 100.0 100.0 100.0 100.0</td>
</tr>
<tr>
<td>P 10 100.0 100.0 100.0 100.0 100.0 100.0 100.0 100.0 100.0 100.0</td>
</tr>
<tr>
<td>P 11 100.0 100.0 100.0 100.0 100.0 100.0 100.0 100.0 100.0 100.0</td>
</tr>
<tr>
<td>P 12 100.0 100.0 100.0 100.0 100.0 100.0 100.0 100.0 100.0 100.0</td>
</tr>
<tr>
<td>P 13 100.0 100.0 100.0 100.0 100.0 100.0 100.0 100.0 100.0 100.0</td>
</tr>
<tr>
<td>P 14 100.0 100.0 100.0 100.0 100.0 100.0 100.0 100.0 100.0 100.0</td>
</tr>
<tr>
<td>P 15 100.0 100.0 100.0 100.0 100.0 100.0 100.0 100.0 100.0 100.0</td>
</tr>
<tr>
<td>P 16 100.0 100.0 100.0 100.0 100.0 100.0 100.0 100.0 100.0 100.0</td>
</tr>
</tbody>
</table>

From the collection of the Computer History Museum (www.computerhistory.org)
Figure 7.—Thousands of memory cycles required by processors per second

PRE₁ and PRE₂. P.io deposits 16384, 16384, and 8192 words into the first, second, and third pre-BF memories, respectively. P.io also reads 16384 words from the post-BF memory. Figure 7 shows all data references between each processor and memory. Data is written to the BF memories at the end of each of the FFTs in the pre-BF processors and data is read from the BF memories by the WEIGHTs in the post-BF processors. More details of the decomposition are given in Reference 4.

Following step MH-2, a queuing model of memory interference model is developed. The linear model of memory interference developed for low interaction data flow graphs is too pessimistic since it assumes references to a bank of memory occur in bursts. In the case of BF nodes, the accesses are distributed in time. The interference between the BF processors is modeled by the queuing network of Figure 8 where servers and customers represent memories and processors, respectively. The queuing model operates as follows: when a customer (processor) enters a queue (initiates a memory request), service (a memory cycle) is begun immediately if the customer is alone in the queue, otherwise he awaits his turn. After service, the customer enters, with equal probability, one of the queues for his next memory cycle. The ratio of average service time to average time in system is the utilization of a processor, $U_p$. $U_p$ depends on the number of memories, $M$, and the number of processors, $N$. For C.mmp neither $N$ nor $M$ can exceed 16.
Memory cycle time is a constant one microsecond and corresponds to service time in the model. Since the service times are constant, the queuing network must be simulated to get Up. If the service times, however, are modeled as an exponential distribution then Up(N, M) can be determined analytically using the method of Buzen. From the table of Up(N, M), for eleven processors and nine memories, the processors are waiting \( 1 - \text{Up}(11, 9) = 52.6 \) percent of the time, leading to an apparent processor cycle time of \( 1/\text{Up}(11, 9) = 2.11 \) microseconds instead of one microsecond. For comparison, a simulation of eleven processors and nine memories having constant service time gives 55.5 percent processor utilization or an effective cycle time of 1.82 microseconds. In this case, the exponential assumption is a more conservative estimate of the effective processor cycle time. With the Up data, CRBF can be decomposed for a classical multiprocessor as follows: Isolate the BF nodes to minimize their interference with other nodes. Group the remaining nodes into two classes: those preceding the BFs (LPF and FFT) and those following the BFs (WEIGHT, DETECT, and STI).

Now the effect of memory interference on the decomposition is determined (steps MH-3 and MH-4). To operate in realtime, each processor must finish processing before the next block of data arrives. Finish times of the PRE-BF nodes are:

\[
4 + T_{\text{transfer}} + B + C + 94/(1024 + \text{Up}(N_{\text{BF}}, B + C)) + T_{\text{transfer}}
\]

where \( T_{\text{transfer}} \) is the time for the longest transfer involving any of the BF memories. The longest transfer is from P,post and takes 24.432 msec. The maximum time for any of the BF processors to finish is 896.8 msec, leaving 103.2 msec of slack from the 1000 msec interval. For post-BF processors, finish time is 822.4 + 16.4 + 4 + 4.1 = 855.2 msec. The finish times for the pre-BF processors are 756.6, 756.6, and 912.8 msec, respectively. These finish times leave a worst case slack time of 87.2 msec, thereby insuring realtime operation.

It is enlightening to compare the interference predicted by this method and the method used in the low interaction case. The finish time of BFp, predicted by the low interference model used by the decomposition is approximately (ignoring the interference from the PRE, POST, and I/O nodes): \( 9*(46.5 + 10 + 46.0) = 4558.5 \) msec. This far exceeds the one second allowed for the BF mode to run in realtime. That is, the linear estimate is far too conservative when used on the high interaction case.

CONCLUSIONS

The methodology of decomposition varied with the type of data flow graph. CPR LOFAR has little interaction between nodes of the data flow graph. A simple memory interference model for CPR LOFAR estimates interference as a linear function of the number of processors referencing a memory bank. When concurrent block transfers interfere with one another, this linear estimate of interference is appropriate and accurate. CR Beamforming has a highly interactive data flow graph which requires a more sophisticated queuing model of memory interference. To calculate effective processor cycle time from the queuing model, the memory cycle time is assumed to be exponentially distributed. This queuing model is solved using (1) simulation (constant access per unit time), (2) Buzen's method (exponentially distributed memory cycle time), and (3) the linear interference model. These three techniques lead to effective processor cycle times of 1.82, 2.11, and 10.9 microseconds, respectively.

The methodology presented may not lead to optimal decompositions. However, in the examples presented, the processing time required for each channel was large enough to preclude more economical decompositions (i.e., \([N/T]^*\) was the number of concurrent channels). This situation may often be true in practice.

All these methods are useful design tools. The Buzen method is useful when memory references are random; the linear interference model is useful when memory references are mainly block transfers; simulation is useful as a final validation of a decomposition. It is anticipated that more complicated signal processing modes will require more detailed queuing models and simulation to verify decomposition.

REFERENCES

4. Siewiorek, D. P., W. C. Brantley, and G. W. Leive, "Modeling Multi-


