CONCURRENT ARCHITECTURES and HIGH RATE DATA PROCESSING

Meemong Lee

Jet Propulsion Laboratory
California Institute of Technology
Pasadena Ca U.S.A.

ABSTRACT

High rate data processing is extremely challenging since it requires fast computation, large data storage, and fast data access simultaneously. In order to meet these challenges, computer technology has evolved toward concurrent architectures where multiple processing elements are integrated for parallel execution of data. Processors may communicate through a shared memory system or may communicate directly to neighboring processors via a communication network. The performance characteristics of a system depend on the processor connection topology, memory organization, and synchronization mechanism of the system.

Implementation of a data processing algorithm in a concurrent system requires decomposition and/or reorganization of the algorithm according to the characteristics of the concurrent system. It is critical to employ a concurrent system whose systematic characteristics are compatible with algorithmic properties in order to minimize the decomposition/reorganization effort. This paper presents a methodology of analyzing the algorithmic properties with respect to their implementational constraints in a concurrent architecture.

INTRODUCTION

The main objective of any concurrent architecture is to employ multiple processors for improved computational power. A concurrent system can be characterized based on the processing element (bit ALU, 32 bit CPU, etc.), memory organization (shared memory, local memory, data/program memory, etc.) interconnection topology (ring, grid, hypercube, crossbar, pyramid, etc), and control mechanism (synchronous, asynchronous). The system characteristics present a unique set of requirements and limitations to algorithm implementation.

A set of analogies between algorithmic properties and concurrent system architectures is developed. They are: 1) operational complexity vs. processing element's capacity, 2) level of data locality vs. interconnection topology, and 3) level of algorithmic homogeneity vs. system control mechanism. The operational complexity varies from a bit manipulation to a double precision number operation. The data locality implies a spatial connectivity of data and the level varies from a pixel level with no connection to a non-local level with multi-dimensional connection requirement. The homogeneity is an operational synchroneity and the level is determined based on the complexity of the operation.

A methodology that analyzes the properties of an algorithm with respect to the operational complexity, data locality, and homogeneity is presented based on various high rate data processing algorithms. Implementational constraints of the data processing algorithms on concurrent systems are examined employing two types of concurrent architectures. A GAPP (Geometric Array of Parallel Processors) system is employed to examine the applicability of a SIMD(Single Instruction on Multiple Data) architecture with a simple interconnection topology. A JPL/Caltech MARK III hypercube is employed to examine the applicability of a MIMD(Multiple Instruction on Multiple Data) architecture with a high dimensional interconnection topology.

DATA PROCESSING ALGORITHMS

There are parallel analogies between the algorithmic characteristics and concurrent computer architectures. They are operational complexity vs. processor's capacity, data locality vs. interconnection topology, and homogeneity vs. synchronization. Algorithms of three major high rate data processing areas, including, planetary data, multispectral data, and synthetic aperture radar data, are examined with respect to their operational complexity, data locality, and homogeneity.

1. Systematic Processing of Planetary Images

The planetary images taken by a camera (Vidicon or CCD) on a spacecraft have a set of distortions due to the camera system's response characteristics in amplitude, frequency, and spatial relations. The distortions are removed systematically via radiometric correction, MTF (Modulation Transfer Function) restoration, and geometric correction[1].

Radiometric correction is to calibrate gain and offset of a camera system. The Gain(G) and offset(δ) is a function of pixel location and the corrected pixel intensity (I') can be expressed as below.

$$I'(x,y) = (I(x,y) - \delta(x,y)) \,/\, G(x,y) \tag{1}$$

The response of the modulation transfer function (MTF) of a camera lens attenuates high frequency features. A high pass filter may be applied to compensate the attenuation in a frequency domain as well as in a space domain. The MTF restoration using a frequency domain high pass filter W on an image A is expressed in equation 2 where FFT and $CONJG$ imply a Fast Fourier Transform operation and conjugation of a complex array respectively and B is the restored image.

$$B = FFT^{-1}\,(\,FFT(A)\;CONJG(W)\,) \tag{2}$$

A spatial domain high pass filter applies a convolution kernel w directly to an image as shown in equation 3. The w is a truncated Fourier transform of W with the size of wl by ws.

$$B(x,y) = \sum_{1=1}^{i=wl} \sum_{j=1}^{j=ws} w(i,j)\; A(x-i+ws/2,\; y-j+wl/2) \tag{3}$$

The spatial response characteristics of a camera system introduce geometric distortion which affects the uniformity of the spatial relation among pixels. In order to detect the distortion characteristics, a set of reseau marks are placed on the view plane of a camera system in the form of a grid. The locations of the reseau marks in the image plane describes the geometric response of the camera system. The geometric distortion is removed by inverting the geometric relation between the locations of reseau marks in an image plane and the view plane.

A first order geometric transformation describing the distortion is computed based on four reseau marks as shown in equation 4.

$$x' = AM \quad \text{and} \quad y' = BM \tag{4}$$

where M, A, and B are defined to be

$$M = \left[1, x, y, xy \right] \quad A = \left[a_1, a_2, a_3, a_4 \right] \quad \text{and} \quad B = \left[b_1, b_2, b_3, b_4 \right]$$

The geometric coefficients, A and B, are computed by applying a set of four reseau mark locations in the view plane and the corresponding locations in the image plane for (x,y) and (x',y').

The geometric correction process is applied for an area formed by four reseau marks at a time. The correction process consists of pixel location computation using equation 4 and pixel intensity estimation. The intensity is estimated using bilinear interpolation based on four nearest neighbor pixels as described in equation 5.

$$I(x,y)= I(x'_l,y'_l) + d_y\left[d_x(I(x'_u,y'_l)-I(x'_u,y'_l)) - d_x(I(x'_l,y'_u)-I(x_u,y_u))\right] \tag{5}$$

$$d_y = y - y_l \quad \text{and} \quad d_x = x - x_l$$

where the subscript l and u imply lower and upper integer limits of a given real number.

2. Spectral Feature Classification of Multispectral Data

Multispectral data provide higher and wider wavelength resolution information than optical images, such as planetary images. Current imaging spectrometer technology (AVIRIS-Airborne Visible and Infrared Imaging Spectrometer) allows remote sensing of 224 wave length channels over an 18 km wide swath instantaneously. The systematic processing of multispectral data is very similar to the planetary image case and repeated for every spectral channel.

A significant portion of multispectral data analysis is devoted to "surface material identification" where each spectrum is classified by mineral class, soil type, vagetation growth, etc. . The material identification[2] is approached in three steps: 1) unique spectra sampling, 2) spectral feature classification and 3) geological classification.

Mineral identification process using imaging spectrometer data, is discussed here as an example case of "spectra classification". First, each spectrum is encoded for its amplitude and slope variation using the binary encoder described in equation 6.

$$F(\vec{S}) = \left\{ \vec{V}, \vec{U} \right\} \tag{6}$$

where

$$V_j = \begin{cases} 1 & \text{if } S_j > \bar{S}\ (average) \\ 0 & otherwise \end{cases}$$

$$U_j = \begin{cases} 1 & \text{if } S_j > S_{j-1} \\ 0 & otherwise \end{cases}$$

A set of unique spectra, $\{US\}$, is composed based on iterative grid sampling method where a regular grid is overlayed on an image and a sample is picked from each grid. The sample is inserted to the set if it is found to be unique as shown in equation 7. This process is repeated until the set is full.

$$X \rightarrow US_{n+1} \quad \text{if } Hd(\vec{X},\vec{US}_k) < th \quad \text{for } k{=}1,n \ \text{ and } \ n < N \tag{7}$$

where X is an encoded sample spectrum, Hd is Hamming distance, th is a similarity threshold, and n is the number of spectral members in the set. Initially, n is set to zero.

The Hamming distance is defined to be

$$Hd(\vec{X}, \vec{Y}) = \sum_{j=1}^{j=n} hv + \sum_{j=1}^{j=n} hu$$

$$hv = \begin{cases} 1 & \text{if } V_j = V_j \\ 0 & otherwise \end{cases}$$

$$hu = \begin{cases} 1 & \text{if } U_j = U_j \\ 0 & otherwise \end{cases}$$

Each feature encoded spectrum ($\vec{X}$) is compared against the sample spectra ($\vec{US}_i$)) using Hamm-ming distance (Hd) and its class is labeled as shown in equation 8.

$$X = k \quad \text{if } Hd(\vec{X}, \vec{US}_k) < Hd(\vec{X}, \vec{US}_i) \quad \text{for } all\ i \ \text{ and } \ i \neq k \tag{8}$$

The mineral content of each spectral feature class is identified by comparing the class average with a library of known mineral spectra.

3. Textural Scene Analysis of Synthetic Aperture Radar (SAR) Data

Active microwave sensors are playing a large role in remote sensing since they provide physical and dielectric properties as well as some subsurface properties. These sensors can operate independent of sun illumination and usually are not sensitive to weather conditions or cloud cover[3]. One of the physical properties provided by SAR data is surface texture characteristics. The texture information can be used to differentiate agricultural fields from city areas, to identify ice types, to analyze ocean wave characteristics, and various other applications.

A texture is a regular repetition of pixel variations which can be represented in a regressive filter[4]. In an interactive texture classification, a small area of an image representing a texture type is selected as a training area and a texture pattern is quantified as a regressive filter. The regressive filter algorithm assumes that a pixel is a linear combination of neighboring pixels. The combination coefficients are determined based on inter-pixel relations.

The regression is controlled within a half plane window and the filter coefficients are extracted from the autocovariance matrix of the training area. Equation 9 shows a 4 by 2 regression window filter(F) composed from a autocovariance matrix (C) of a training area(T).

$$F = \left[c_{11}, c_{12}, c_{21}, c_{31}, c_{32}, c_{41}, c_{42}\right] \tag{9}$$

$$C = T\,T^T$$

$$C = \begin{bmatrix} c_{11} & c_{21} & c_{31} & \cdot & \cdot \\ c_{12} & c_{22} & c_{32} & \cdot & \cdot \\ c_{13} & c_{23} & c_{33} & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot \end{bmatrix}$$

The above process is repeated for all texture classes in an image, then the regressive filters are applied to the entire image and residuals from each filter are analyzed for the maximum likelihood for texture classificaion. The probability of the i-th filter residual ($P(R_i)$) belonging the the i-th class is computed as below.

$$P(R_i(x)) = \sqrt{2\pi\sigma_i}\, \exp^{(R_i(x)-M_i)/\sigma_i} \tag{10}$$

$$R_i(x) = x - F_i W(x)$$

where M_i and σ_i are the mean and the varaiance of ith training area and W is a half plane window surrounding the pixel x.

4. Homogeneity

Concurrent systems with synchronous control mechanisms require all processing elements to perform simultaneously while asynchronously controlled systems allow each processor to perform individually. Thus, a synchronous system can apply a single instruction on multiple data while an asynchronous system can apply multiple instructions on mutiple data simultaneously. For this reason, concurrent systems are classified as SIMD architectures and MIMD architectures, in general.

A similar analogy applies to data processing algorithms. An algorithm that consists of a uniformly applicable instruction set is a homogeneous algorithm while an inhomogeneous algorithm consists of different sets of instructions applying to different parts of data. A data procesing procedure may be composed of both homogeneous and inhomogeneous algorithms. The homogeneous algorithms are divided into instruction level homogeneous algorithms and function level homeogeneous algorithms depending on the complexity of the operation. A SIMD architecture can be applied only to operations with the instruction level homogeneity while a MIMD architecture can be applyed to inhomogeneous algorithms as well as algorithms of both homogeneity levels.

The instruction level homogeneity implies that a simple arithmetic instruction, or a bit manipulation instruction is applied to all data without exception. Among the data processing operations, operations listed below have the instruction level homogeneity.

Radiometric Correction	: offset subtraction, gain division
MTF Restoration	: convolution kernel multiplication
Spectra Classification	: spectrum feature encoding Hamming distance measure

Homogeneous operations that contain data dependency, conditional branches, or location dependent parameter sets are classified as operations with function level homogeneity. These operations can not be implemented on a SIMD architecture easily. Example operations with function level homogeneity are listed below.

Geometric Correction	: bilinear Interpolation
Texture Classification	: residual and probability computation

Following operations are inhomogeneous operations and require more complicated concurrency analysis for load balancing and synchronization.

Geometric Correction	: geometric transformation computation
Spectra Classification	: unique spectra sampling
Texture Classification	: regression filter computation

5. Locality

The interconnection topologies of concurrent systems allow a limited inter-processor communication. Direct data communication channels provided by each system topology with N processors are: no interconnection - 0, a ring - 1, a grid - 2, a hypercube - $\log_2 N$, and a crossbar - N. The communication between two unconnected processors can be performed via several indirections. The indirect communication path length is inversely related to the dimensionality of the system topology. In order to achieve a global communication in a system with N processors, the number of processors involved in the indirect communication are: a ring - $N-1$, a grid - $2\sqrt{N}-1$, a hypercube - $\log_2 N-1$, and a crossbar - 0. The inter-processor communication slows down significantly as the indirect communication path length increases.

Similarly, data processing algorithms have various data locality characteristics that can be described in terms of topology. The topology of an algorithm's data locality can be classified as: pixel level locality - no interconnection, template level locality - a grid , Log_2N dimensional locality - a hypercube, and non-local or N dimensional locality - a cross bar. The N implies the number of data points. A system topology must have a higher dimensionality than the data locality in order to provide necessary inter-processor communication directly. When an algorithm's data locality has a higher dimensionality than the implemented system's topology, the concurrency is decreased since the communication must be performed indirectly.

Operations listed below have the pixel level locality, thus they can be implemented in a concurrent system with any interconnection topology including no interconnection. A spectrum is treated as a pixel with multiple intensity values in this analysis.

Radiometric processing	: offset subtraction, gain division
Spectra Classification	: spectrum feature encoding
	Hamming distance measure
Texture Classification	: probability computation

Following operations have the template level locality. Concurrent systems with a grid or a higher dimensional interconnection topology can provide necessary data communication for these operations.

Geometric Correction	: bilinear interpolation
MTF rectification	: spatial convolution
Texture Classification	: residual computation

The Fast Fourier Transform employed in the MTF restoration has Log_2N dimensional locality. Most non-homogeneous operations have no data locality either. Algorithms with an irregular data connectivity requirement can be classified as algorithms with no locality or N dimensional data locality since an arbitrary data connection can be met by an N dimensional spatial connection. Algorithms with a high dimensional data locality can not achieve the optimal concurrency of a system due to the increased data communication time.

Besides the interconnection topology of a concurrent system, the size of a local memory plays an important role in data communication. The frequency of data communication decreases as the local memory size increases since multiple data can be stored in one processor. When a processor has a very large local memory, the system can provide a wide range of data interconnection within a processor. Thus, the combination of the interconnection topology and the local memory size determines the system's compatibility to the algorithm's data locality characteristics.

IMPLEMENTATION ON CONCURRENT SYSTEMS

Based on the analysis of analogies between data processing algorithms and concurrent architectures, it is obvious that a concurrent system can offer its optimal concurrency when the system architecture characteristics and the properties of an implemented algorithm are compatible. The lack of compatibility may complicate the algorithm implementation process and affect the system performance.

For implementation of data processing algorithms on concurrent systems, the algorithms are divided into two classes. The first class algorithms have properties of an instruction level homogeneity and a pixel or a template level data locality which can be implememted in a SIMD architecture with a grid interconnection topology. The second class algorithms have properties of a function level or no homogeneity and a higher dimensional data locality which require a MIMD architecture with a high dimensional interconnection topology.

NCR's GAPP system is chosen to implement the first class algorithms and a JPL/Caltech MARK III hypercube system is chosen to implement the second class algorithms. The implementation process is discussed with respect to the algorithm and data decomposition requirements and the system performance is analyzed for the time involved in data transfer, interprocessor communication, and computation.

1. NCR GAPP(Geometric Arithmetic Parallel Processor) system

A GAPP system[5] is a SIMD architecture which consists of synchronously controlled 48 by 48 homogeneous processing elements. The processing elements are interconnected in a grid topology allowing direct communication among four neighbors. Each processing element consists of 1 bit ALU, 4 arithmetic registers, 4 communication registers, and 128 bit local data memory. The system provides a separate controller board equipped with a system clock, program storage area, a microsequencer, and a corner-turn buffer.

The main advantage of this system is in its massive concurrency achieved by the large number of processing elements. The two dimensional interconnection topology also allows an easy system upgrade by adding rows or columns to the existing system. However, the GAPP system's processing element is very primitive and is not suitable for complicated operations.

Data flow characteristics of this system are complicated since the data have to be reorganized from a word array into bit planes. First, a data segment is extracted from an image array. The segment is transfered to the corner-turn buffer a row at a time. The corner-turn buffer has 48 by 6 procesing elements with 128 bit of memory per each processing element. The data which were originaly oriented in a word by row by column array (6 by 48 by 48) are transfered to the GAPP memory and they are reoriented to a row by column by word array (48 by 48 by 6).

A column of data is simultaneously transfered from the corner-turn buffer to the GAPP memory as shown in figure 1. The west most registers of the corner-turn buffer are directly connected to the south most registers of the GAPP array. A detailed description of the data transfer is illustrated in the list of a 3 by 3 box filter program shown in figure 2. The data transfer requires two programs, one to send data and one to receive data. The synchronization between the two programs are performed as follow. Each corner-turn buffer processor loads a bit to the ew-register (ew:=ram(i)) and each GAPP processor picks up from the south communication register (cm:=cms). After the transfer process is repeated for 48 times for each bit, an entire bit plane is transfered over to the GAPP array. The process is then repeated for each bit by shifting the content of corner-turn buffer to west direction (ew:=w), thus completing the word to bit reorientation.

The data transfer/reorganization time of a GAPP system (T_{GAPP_DATA}) with G_r by G_c processing elements is sum of the data transfer time between the host and the corner-turn buffer ($T_{corner-turn}$) and the transfer time within the GAPP system (T_{GAPP}) as shown in equation 11.

$$T_{GAPP\_DATA} = T_{corner-turn} + T_{GAPP} \tag{11}$$

$$T_{corner-turn} = \frac{(G_r\, G_c)}{B_H} \quad \text{and} \quad T_{GAPP} = \frac{(G_r\, b)}{B_G}$$

where B_H and B_G are the data transfer rates of the host system interface and the GAPP system and b is the number of bits in a word.

A microprogram instruction is executed per clock cycle on 48 by 48 bit data. Though 48 by 48 data points can be executed simultaneously, each word is processed in a bit serial manner. Therefore, the concurrency factor decreases rapidly as the number of bits in a word increases. The concurrency factor of a GAPP system (C_{GAPP}) over a sequential system (T_{SEQ}) can be expressed as shown in equation 12 where N is the number of processing elements in the system and b is the number of bits in a word.

$$C_{GAPP} = T_{SEQ} \,/\, (T_{GAPP\_DATA} + T_{GAPP\_COMP}) \approx N \,/\, b \tag{12}$$

$$T_{GAPP\_COMP} \approx b \,/\, N$$

The main disadvantages of this system are difficulties involved in algorithm decomposition and bit plane data manipulation. An algorithm must be decomposed into homogeneous bit instructions and a bit data plane must be manipulated synchronously following two dimensional data communication rules. The architectural limitations of small local data memory and a bit serial processor reduce the applicability significantly.

2. JPL/Caltech Mark III Hypercube

A Mark III [6] hypercube achieves concurrency in two levels, a node level and a system level. The node level concurrency is from employing separate nodes for I/O, floating point operation, and general program execution. The system level concurrency is from multiple nodes interconnected in a hypercube topology for concurrent execution of multiple programs on multiple data sets.

Each node of a Mark III hypercube has 4 Mbyte of local memory and 7 communication channels allowing direct communication to seven neighboring nodes. Main advantages of this system are its expandability per additional communication channel, capability of handling non-homogeneous programs, and multidimensional connectivity.

The data processing algorithms are implemented by decomposing each algorithm into two modules; host program module and a cube program module. The host program handles the peripherral device I/O, such as file I/O, user interface, and display device interface, and the cube program manipulates the data. A homogeneous cube programming where each node has a copy of cube program is employed for the data processing algorithm implementation. The homogeneous programming achieves balances the computational load for nodes via a proper data distribution among nodes.

The hypercube topology offers a $\log_2 N$ dimensional data connectivity as shown in figure 3. The neighboring nodes are determined based on the fact that the Hamming distance of binary representation of nodes is 1. Also, utilizing the hypercube connection, data can be redistributed from one type of distribution to another by exchanging the data with neighboring nodes as illustrated in figure 4. The flexibility of data distribution allows implementational freedom of algorithms with various data localities. Also, the large local memory reduces inter-node data communication significantly by allowing redundant data distribution so that the data required by a node can be resident within its local memory.

The hypercube topolgy also allows fast propagation of data. Data may be downloaded and/or uploaded in $\log_2 N$ steps for a N-node system. Figure 5 describes the data download and upload path of a 8 node system. In step 1 of the data download path, node 0 passes data to node 1, in step 2, node 0 passes data to node 2 and the node 1 passes data to node 3, finally, in step 3, the nodes 0,1,2,3 passes data to nodes 4,5,6,7 simultaneously.

The time involved in transferring data with a size of D can be expressed in equation 13.

$$T_{CUBE\_DATA} = \frac{D}{B_H} + \frac{D}{B_C} \log_2 N \qquad (13)$$

where B_H and B_C are data transfer rates between host and the hypercube and within the hypercube respectively.

The concurrency of this system is determined based on computation time (T_{CUBE_COMP}) combined with communication time (T_{CUBE_COMM}) and data transfer time as shown in equation 14.

$$C_{CUBE} = T_{SEQ} \,/\, (T_{CUBE\_DATA} + T_{CUBE\_COMP} + T_{CUBE\_COMM}) \approx N \qquad (14)$$

$$T_{CUBE\_COMP} \approx \frac{1}{N}$$

The maximum concurrency factor achievable by this system approaches to the number of nodes in the system for applications with insignificant data transfer and inter-processor communication. When an algorithm is operationaly simple and is applied to a large data, the ratio between T_{CUBE_COMP} and the T_{CUBE_DATA} is low, thus the achivable concurrency is low. Also, when an algorithm requires data from non-neighboring node frequently, the communication time increases and consequently, the concurrency decreases.

Programming difficulty in orchestrating the complex inter-node communication, load balancing, and synchronization of multiple programs is a major disadvantage of a MIMD architecture. For data processing algorithm implementation, a single program is applied homogeneously, thus programming is somewhat simple. However, data distribution among nodes that have the Hamming distance oriented physical connectivity becomes very complicated for inter-node data sharing since the numerically adjacent nodes are not necessarily physical neighbors.

SUMMARY

The compatiblility analysis between concurrent system architecture for various data processing algorithms is presented by establishing a set of analogies between the architectural characteristics of concurrent systems and algorithmic properties. The analogies are discussed based on relationships of processor capacity vs. instruction complexity, system topology vs. data locality, and system synchronization mechanism vs. homogeneity. Each relation must be compatible in order to achieve an optimal concurrency.

The system concurrency is measured based on the time required for data transfer, inter-processor communication, and computation. A compatible concurrent system architecture can minimize the time involved in inter-processor communication and computation. However, most concurrent systems employ a sequential data I/O mechanism from/to outside of the system and the data transfer time becomes a dominant factor for high rate data processing applications. A concurrent system for a high rate data processing environment must provide concurrent data I/O to remove the data transfer bottle neck as well as computational concurrency.

ACKNOWLEDGEMENT

This research was carried out by the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration.

REFERENCES

1. K R Castleman 1979, Digital Image Processing, Prentice-Hall Signal Processing series

2. A Mazer, et all 1987, Image processing software for imaging spectrometry, vol 834, 136-139, Proceedings of SPIE-The International Society for Optical Engineering

3. C Elachi 1987, Introduction to the physics and techniques of remote sensing, Wiley interscience

4. W M Lawton, M Lee 1985, Random field models for use in scene segmentation, Final report for NASA fundamental research program (#677-24-41-04-36), Jet Propulsion Laboratory

5. NCR corporation, 1987, GAPP processor unit (Multibus compatible card set) functional specification

6. Tuazon J et all, 1988, The MARK III hypercube concurrent processor, Third conference on hypercube concurrent computers and applications, Pasadena, Ca. U.S.A

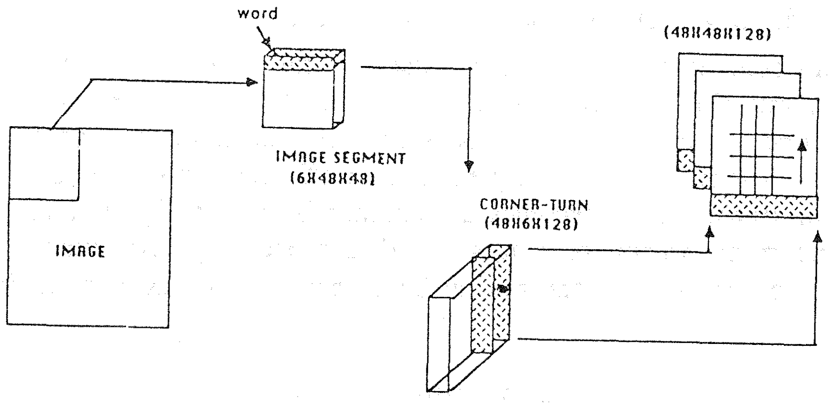


Figure 1. Dataflow characteristics of a GAPP system

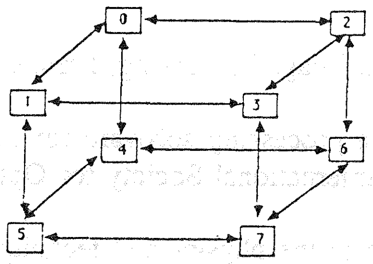
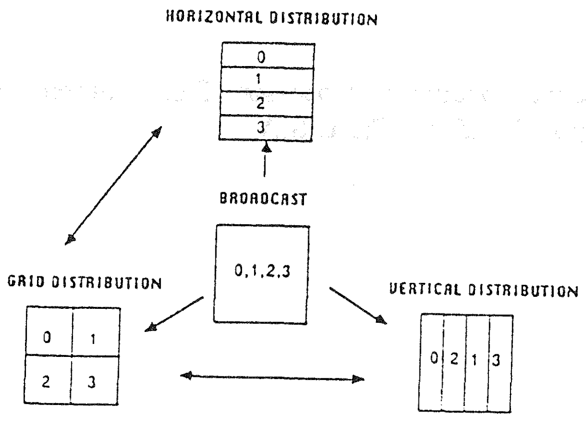


Figure 3. A 8-node hypercube connection



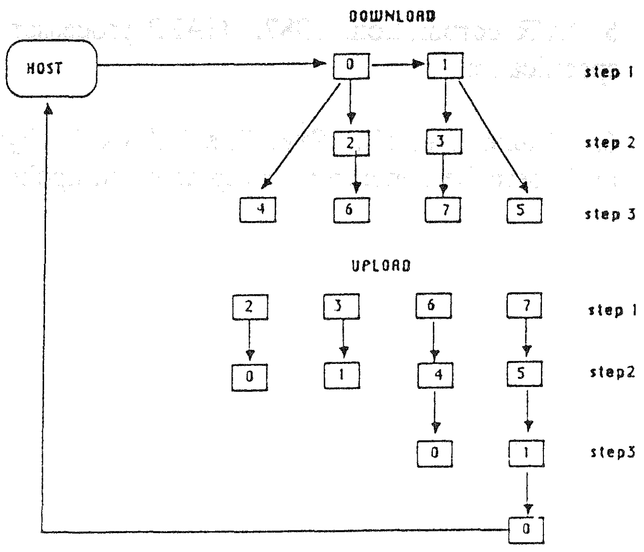


Figure 4. Data distribution/redistribution in a 4-node hypercube

Figure 5. Data download/upload in a 8-node hypercube

gapp.ct

```
/* Corner turn buffer program */
image     data: 0: 5;
int       xsize = 48,
            ysize = 48,
             ma_length=600;
main()
{

     int        i, j, k, b;

 /*            Shift data from Corner Turn line buffer
            to Main Array.                                  */

     b = 6;
     for ( k = 0; k < b; k++ ) {
          for ( i = 0; i < y; i++ ) {
               ew:=ram( :i );
               ew:=w c:=0 ns:=0;
               ram( :i ):=sm;
          }
          nop;
     }

 /*            Perform NOP's while Main Array is
            processing data.                                */

            for ( i = 0; i < ma_length; i++ ) nop;

 /*            Shift data from Main Array to Corner
            Turn line buffer.                               */

     for ( k = 0; k < b; k++ ) {
          nop;

          for ( i = 0; i < y; i++ ) {
               ew:=ram( :i );
               ew:=w c:=0 ns:=0;
               ram( :i ):=sm;
          }
     }
}
```

gapp.pgm

```
/* GAPP program */
image     data: 0: 5;
int       xsize = 48,
            ysize = 48;

main()
{
     image B: 6: 11;
     int       i, j, k, b;

 /*          Read data from Corner Turn line buffer
              to Main Array                                   */

     b = 6;
     for ( k = 0; k < b; k++ ) {
          for ( i = 0; i < y; i++ ) {
               nop;
               cm:=cms;
               nop;
          }
          ram( dest: b - 1 - k ):=cm;
     }

     convolve(A,B)    /* approx. 600 program steps */


 /*          Write data from the Main memory to
              Conrner turn line buffer                       */

     for ( k = 0; k < b; k++ ) {
          cm:=ram( source: b - 1 - k );

          for ( i = 0; i < y; i++ ) {
               nop;
               cm:=cms;
               nop;
          }
     }
}
```

Figure 2. 3x3 box filter gapp program example

gapp.sub

```
convolve(A,SUM)
image   A
image   SUM;
{

        image  scratch:8;
        int  x,y;
        int  scale;

        c  :=  0;
        for  (x=0;  x<size(SUM);  x++)  SUM:x  ram:=c;

        for(x=0;  x<3;  x++)  {
           for(y=0;  y<3;  y++)  {
                 move(A,x-1,y-1,scratch);
                 add(scratch,SUM,SUM);


        return;
```

```
add(A,B,C)
image  A,  B,  C;
{

          int  i,k,l,m,n;

          m  =  size(A);
          n  =  size(B);

          if  (m>n)  {k  =  m;  l  =  n;}
         else        {k  =  n;  l  =  m;}

          A:0  NS  :=  RAM       C  :=  0;
         B:0  EW  :=  RAM;
          C:0  RAM  :=  SM       C  :=  CY;

          for  (i=1;  i<k;  i++)
               {
                A:i    NS  :=  RAM;
               B:i   EW  :=  RAM;
               C:i   RAM:=  SM     C  :=  CY;
               }

          for  (i=k;  i<l;  i++)
               {
                if  (m<=  n)   A:(m-1)  NS  :=  RAM;
               else             A:i        NS  :=  RAM;
               if  (n<=  m)   B:(n-1)  EW  :=  RAM;
               else             B:i        EW  :=  RAM;
               C:i      RAM:=SM       C:=CY;
                }

                C:(l)   RAM:=  SM;
```

```
move(A,x,y,B)
image  A,B;
int  x,y;
{
 /*     no.  of  inst  =  size(A)*(|x|+|y|+4)

          int  i,m;

          for(m=0;  m<size(A);  m++)  {
               A:m   EW  :=  RAM;
               if  (x>0)
                  for  (i=0;  i<x;  i++)  EW  :=  W;
               else
                  for  (i=0;  i>x;  i--)  EW  :=  E;

               NS  :=  EW;

               if  (y>0)
                  for  (i=0;  i<y;  i++)  NS  :=  S;
               else
                  for  (i=0;  i>y;  i--)  NS  :=  N;

               c  :=  ns;
               B:m        RAM  :=  c;
```

Figure 2.   (continued)