# Efficient Mapping Algorithms for a Class of Hierarchical Systems

Sotirios G. Ziavras, *Member, IEEE*

*Abstract*—**This paper proposes techniques for mapping application algorithms onto a class of hierarchically structured parallel computing systems. Multiprocessors of this type are capable of efficiently solving a variety of scientific problems because they can efficiently implement both local and global operations for data in a two-dimensional array format. Among the set of candidate application domains, low-level and intermediate-level image processing and computer vision (IPCV) are characterized by high-performance requirements. Emphasis is given in this paper to IPCV algorithms. The importance of the mapping techniques stems from the fact that the current technology cannot be used to build cost-effective and efficient systems composed of very large numbers of processors, so the performance of various systems of lower cost should be investigated. Both analytical and simulation results prove the effectiveness and efficiency of the proposed mapping techniques.**

*Index Terms*— **Hierarchical structures, image processing and computer vision, mapping techniques, process assignment, scheduling.**

## I. Introduction

MULTILEVEL systems consist of successive layers of mesh-connected arrays of PE's (processing elements) where the size of the arrays decreases with an increase of the level number (assuming that the finest array is at level 0) [26]. They support the efficient implementation of both local and global operations applied to data in a two-dimensional array format, and are also suitable for divide-and-conquer techniques [7]–[19]. The most common structure of such systems is the *standard pyramid*, or just pyramid for simplicity, which has reductions $2 \times 2$ between all pairs of neighboring levels and a single PE (the apex) at the topmost level [1], [12], [17], [19]. The following discussion uses the triplet $(i, x_i, y_i)$ to denote the PE at level $i$ that has Cartesian coordinates $(x_i, y_i)$. Each PE $(i, x_i, y_i)$ of the pyramid, where $i \neq 0$, has four children $(i-1, 2x_i + j_1, 2y_i + j_2)$, where $0 \leq j_1, j_2 \leq 1$; in addition, each nonapex PE $(i, x_i, y_i)$ has the single parent $(i+1, \lfloor x_i/2 \rfloor, \lfloor y_i/2 \rfloor)$. Multilevel systems are formally defined below.

Since low-level and intermediate-level IPCV require large amounts of both local and global operations, with the majority of them being local [19], multilevel systems are suitable for relevant problems. The allocation of a single pixel per PE is often assumed for level 0 of the pyramid. Pyramids comprising 10 or 11 levels of very powerful PE's, which should be appropriate for processing images of size $512 \times 512$ or $1024 \times 1024$ pixels, are difficult or impossible to build with the current technology so alternative hardware solutions should be investigated. The most important choices are to reduce the total number of levels by increasing the reductions between them, to use coarse granularity by allocating an image block per PE, and to use truncated systems where higher levels are missing. Higher performance sometimes results from systems that have small reductions at lower levels for the application of standard multiresolution techniques, while larger reductions at higher levels may allow for fast collection of information extracted at lower levels [24]. Based on the control structure, two major categories of pyramid machines may be identified: the SIMD (single-instruction stream, multiple-data stream) pyramid [12], [22] and the MSIMD (multiple SIMD) pyramid [9], [25]. For systems of the former category, active PE's execute simultaneously the same instruction on different data, while for systems of the latter category, each level only is of the SIMD type. This paper concentrates on MSIMD multilevel systems.

The pyramid topology can easily be embedded into the widely available mesh and hypercube topologies [15], [16], [21], [27]. Nevertheless, the pyramid is much more cost-effective and/or efficient than the hypercube or the mesh for the majority of algorithms [8], [11], [18], [25]. For example, if two or more levels of the pyramid need to be active simultaneously (e.g., pipelining [1] or concurrent multilevel processing [8], [23]), then an MSIMD pyramid will be more cost-effective and efficient than the hypercube. This is the consequence of the following two reasons. First, to allow distinct nodes of the pyramid to be mapped onto distinct nodes of the hypercube, one has to use a hypercube of one higher dimension than that needed for just the embedding of the finest array [8]. Second, the distance between siblings will then be one, while it will be at most two between parents and children. An accurate evaluation of cost-effectiveness follows. If the size of the finest array in the pyramid is $2^n \times 2^n$ nodes, then the pyramid has $\lfloor 2^{2(n+1)}/3 \rfloor$ PE's and $2^{n+2}(2^n - 1)$ bidirectional channels, while the hypercube has $2^{2n+1}$ PE's and $(2n+1)2^{2n}$ bidirectional channels. The tabulation of these parameters for important values of $n$ in Table I shows that pyramids have much lower cost than hypercubes.

This paper investigates the class of multilevel systems that comprises a wide variety of pyramid-like systems. The class of multilevel systems is defined as follows.

Manuscript received June 18, 1991; revised March 12, 1992. This paper is based upon work supported in part by the National Science Foundation under Grant CCR-9109084.

The author is with the Department of Electrical and Computer Engineering, New Jersey Institute of Technology, Newark, NJ 07102.

IEEE Log Number 9213474.

TABLE I
A COMPARISON OF PYRAMIDS WITH HYPERCUBES WHEN DISTINCT NODES OF THE PYRAMID ARE MAPPED ONTO DISTINCT NODES OF THE HYPERCUBE ($2^n \times 2^n$ IS THE SIZE OF THE FINEST ARRAY)

| | n | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|
| Pyramid | Nodes | 1,365 | 5,461 | 21,845 | 87,381 | 349,525 | 1,398,101 |
| | Edges | 3,968 | 16,128 | 65,024 | 261,120 | 1,046,528 | 4,190,208 |
| Hypercube | Nodes | 2,048 | 8,192 | 32,768 | 131,072 | 524,288 | 2,097,152 |
| | Edges | 11,264 | 53,248 | 245,760 | 1,114,112 | 4,980,736 | 22,020,096 |

*Definition 1:* The structure of *multilevel systems* is pyramid-like. That is, multilevel systems are composed of successive layers of mesh-connected arrays of PE's. The number of PE's in the arrays decreases with the increase of the level number. Only pairs of neighboring levels can communicate directly with each other, and all communication channels are bidirectional. There exists a buffer of "unlimited" size for each communication channel. For simplicity, each such buffer will be referred to as a *data transfer register* (DTR). All PE's are identical, and the highest level contains more than one PE for "truncated" systems. The reductions are $2^m \times 2^m$, where $m$ are natural numbers that may vary for different pairs of neighboring levels. Finally, multilevel systems operate in the MSIMD mode of computation. A simplified block diagram of multilevel systems is shown in Fig. 1. Each level controller issues instructions to the respective array of PE's. The level controllers receive programs and data from one (or more) main controller(s).

It is assumed in this paper that IPCV algorithms are developed for chosen *source multilevel architectures* which may be the ones that match well with the algorithms. The main objective is to develop techniques for mapping source multilevel architectures onto given *target multilevel architectures* so that high performance will be obtained for the corresponding IPCV algorithms. The mapping problem for multilevel systems, which is formally defined in Section III, is important for the following reasons: 1) a target architecture with a smaller number of levels than the source architecture may be considered because the former has lower cost, 2) the reductions in the source architecture differ from the reductions in the target architecture, and 3) the algorithms are desiged for the most efficient source architectures[1] and their portability to existing (target) multilevel architectures is required. Mapping techniques for nondeterministic and deterministic IPCV algorithms are presented in this paper. For the purposes of this research, the distinction between these two types of algorithms is made in reference to the total execution time. More specifically, *nondeterministic algorithms* are either pure nondeterministic or data-dependent; in both cases, their exact computation and/or communication requirements are not known *a priori*. Pure nondeterminisitc algorithms are represented by computation graphs that contain at least one computation node with two or more outgoing transition edges directed towards different nodes; the edge to be followed at such a node may be data-dependent or time-dependent. Data-dependent algorithms are represented by computation graphs that contain at least one computation node with two or more outgoing transition edges, and one of those edges is directed towards the node itself; the number of times the operations represented by such a node are performed is data-dependent. *Deterministic algorithms* are pure deterministic as well as data-independent (i.e., their exact computation and communication requirements are known *a priori*).

The paper is organized as follows. Section II presents an MSIMD model of parallel computation for multilevel systems. The mapping problem for multilevel systems is addressed in the remaining sections. In order to produce mappings of the maximum of efficiency, Section III proposes objective functions that measure the quality of mappings as dictated by predefined optimization goals. Sections IV and V propose mapping algorithms (one for each objective function) that yield high performance through attempts to minimize these objective functions. Some interesting examples of mapping, as well as collective performance results, are presented in Section VI. Section VI also presents performance measures which are appropriate for a tradeoff analysis between performance and cost. Conclusions are presented in Section VII.

## II. AN MSIMD MODEL OF COMPUTATION

This section expands upon the model of parallel computation presented in [10] to derive an MSIMD model for multilevel systems. The new model describes algorithms as sequences of computations, communications, and idle states, with one sequence per level. Two operations are indispensable for the communication of a datum between any two PE's: a write operation executed by the sender and a read operation executed by the receiver. A write operation is implemented as a sequence of two primitive operations: an operation that stores the datum into an outgoing DTR, and one that transmits the datum from the outgoing DTR to a neighboring PE. A read operation always follows its corresponding write operation; the arrival time of incoming data is known *a priori* for deterministic algorithms, while for nondeterministic algorithms, hardware interrupts involving the level controllers do not require the implementation of producer–consumer protocols. Read operations deal with data being already in incoming DTR's. A PE either stores incoming data in memory, if it is the destination, or retransmits the data, if it serves as an intermediate node for the particular communication pattern. The PE's are personally involved in the exchange of data, so they are not capable of simultaneously transmitting data and also performing internal operations. This is justified by the very high cost of massively parallel systems with separate communication processors. It is also assumed that at any instance of time, a PE may send a single value to a subset or all of its neighbors, and/or may receive one or more values on different channels.

The proposed MSIMD model of computation uniquely describes a given multilevel system for a given IPCV algo-

[1] Such "ideal source architectures" directly support all of the communication patterns dictated by the respective problems and correspond to the smallest possible amount of operations. However, finding the ideal architecture for an algorithm is not always an easy task. An example where such an architecture obviously exists is a two-level system with reduction $2^m \times 2^m$ for convolution of an image with a $2^m \times 2^m$ window.

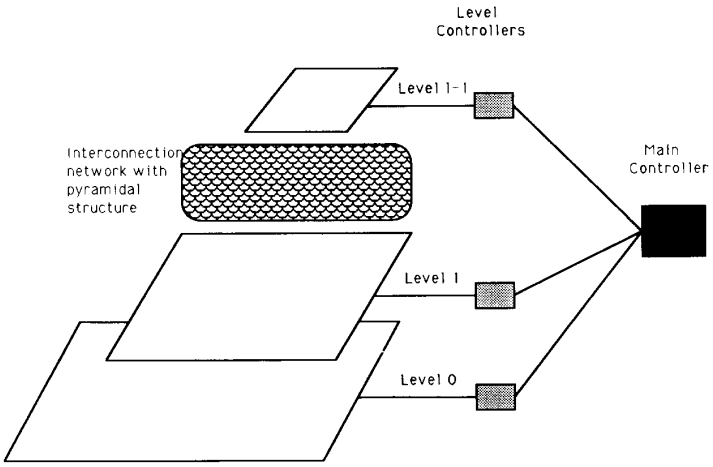



Fig. 1. A block diagram of multilevel systems.

rithm by a set of sequences $\{V_0, V_1, \cdots, V_{l-1}\}$, where $l$ is the total number of levels in the system and each sequence $V_k = (v_k^1, v_k^2, \cdots)$ describes the behavior of a single level $k$ (where $0 \leq k \leq l-1$). $v_k^j$ represents the operation performed in the $j$th step; it may be an internal operation for all active PE's at level $k$ (e.g., the addition of two values) or an inter-PE data transfer operation (e.g., the transmission of a value to a selected subset of neighbors). The execution of the system is described by two sequences $(C^1, C^2, \cdots)$ and $(D^1, D^2, \cdots)$. $C^j = (c_0^j, c_1^j, \cdots, c_{l-1}^j)$ denotes the set of operations attempted in the $j$th execution step; thus, $C^j$ is an $l$-vector containing program counter values for all levels (those values refer to the programs stored in the level controllers). $D^j = (d_0^j, d_1^j, \cdots, d_{l-1}^j)$ is an $l$-vector containing timer values, expressed in machine cycles, for the execution of the latter set of operations.

Algorithms are designed for source multilevel architectures by using a two-dimensional representation. The first dimension represents time, while the second dimension represents the levels of the source architecture. The former dimension is divided into time slots that do not correspond to any particular amount of time. Starting with the first time slot and continuing in sequence with the higher order ones, operations are written for each time slot, one for each level (idle states are represented by empty slots). To allow for efficient mapping, communication operations are described by using either functional expressions or relative addressing, that is, independently of the exact structure of any underlying network. More specifically, a write operation for a lateral communication pattern in the $j$th step of the behavioral sequence $V_k$ is denoted $v_k^j = w_{f,e}$, where $f$ is the set of Cartesian displacements of the PE's which will receive a value and $e$ is the value to be written.[2] One read operation per each incoming value is then applied by receiving PE's. A read operation for a lateral communication pattern in the $j$th step of the behavioral sequence $V_k$ is denoted $v_k^j = r_{f,e}$, where $f$ represents the Cartesian displacement of the sender in reference to the receiver. A write operation for a vertical communication pattern in the $j$th step of the behavioral sequence $V_k$ is denoted $v_k^j = w_{f(x_k,y_k),e}; f(x_k, y_k)$, which is a multivalued function, gives the destination addresses for the one-to-many communication pattern whose sender is the PE at level $k$ with Cartesian coordinates $(x_k, y_k)$ and $e$ is the value to be written. For an upward (downward) transfer, $f$ defines a two-dimensional window at level $k+1(k-1)$. Due to the SIMD mode of parallelism per level, such a communication pattern is valid for all active PE's at level $k$. "$w^+$" and "$w^-$" represent upward and downward transfers, respectively. A read operation for a vertical communication pattern in the $j$th step of the behavioral sequence $V_k$ is denoted $v_k^j = r_{f(x_k,y_k),e}$, where $f(x_k, y_k)$ gives the address of the PE from which to read the value of $e$. "$r^+$" and "$r^-$" represent the reading of a value from a parent and a child, respectively.

[2] Relative addressing for lateral communication operations is also used for image processing operations in the Apply programming language, which was initially developed for the Warp systolic array [13]; the problem of writing an algorithm is reduced to the task of writing the functions to be applied to windows around a single pixel. The experience with the Apply language has shown that the programming task becomes much easier.

## III. The Mapping Problem for Multilevel Systems

### A. Formulation of the Mapping Problem for Multilevel Systems

The central issue in the mapping problem is to assign a program's modules or processes to the PE's of a given parallel computing system and schedule the operations so as to minimize the total cost of executing the program [3]. Since the problem of finding the best mapping is generally NP-complete, research in this area concentrates on the development of efficient heuristics that yield good solutions for restricted classes of systems and application algorithms [4], [5], [20]. Like the majority of the mapping techniques, our solution to the mapping problem for multilevel systems consists of two phases: the *process assignment phase* and the *scheduling phase*. The major objective of the process assignment phase is to map the nodes (i.e., processes) of a source architecture that has been chosen to design the algorithm under consideration onto the PE's of a given target architecture so that the distances

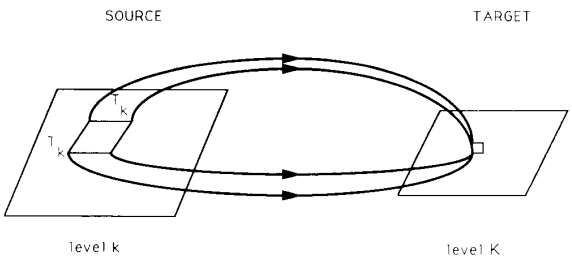



Fig. 2. The shrinking factor of the source level $k$.

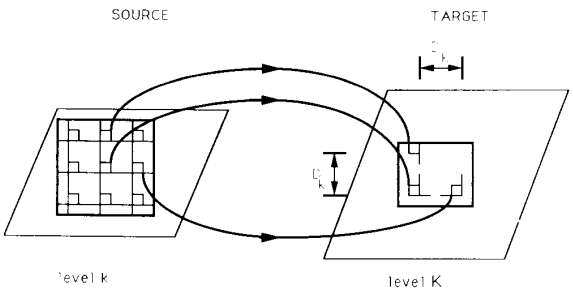



Fig. 3. The dilation of the source level $k$.

between communicating PE's are minimized and the workload is uniformly distributed among as many target PE's as possible [3], [4]. Based on the already determined process assignments, the problem of finding appropriate schedules of operations that yield high performance on the target architecture is solved during the scheduling phase.

Since individual levels of multilevel systems operate in the pure SIMD mode of computation, the process assignment phase is concerned with the problem of finding an appropriate set of mappings $g$ of source levels onto target levels, where

$$g = \{k \rightarrow K, (k, x_k, y_k) \rightarrow (K, x'_K, y'_K): \quad \text{with} \quad x'_K = g_k(x_k) \quad \text{and} \quad y'_K = g_k(y_k)\} \quad (1)$$

and

- "$k \rightarrow K$" represents the mapping of the source level $k$ onto the target level $K$; and
- "$(k, x_k, y_k) \rightarrow (K, x'_K, y'_K)$" represents the mapping of the node with Cartesian coordinates $(x_k, y_k)$ at the source level $k$ onto the PE with Cartesian coordinates $(x'_K, y'_K)$ at the target level $K$. The meaning of $g_k$ is given in the next paragraph.

It is assumed here that the total number of PE's at the target level 0 is never greater than the total number of nodes at the source level 0. Two parameters are used to quantify the quality of mapping for each source level $k$. These parameters are as follows.

- $T_k^2$: the maximum number of nodes from the source level $k$ which are mapped onto a single PE of the target architecture. $T_k^2$ will be called the *shrinking factor of level k*; $T_k$ represents the shrinking factor of level $k$ for the corresponding one-dimensional system. The value of this parameter is greater than 1 if and only if the size of the target level is smaller than the size of the source level. In the latter case, each PE at the target level is assigned the workload of $T_k^2$ nodes from the source level $k$ (this uniform distribution of the workload becomes necessary because of the per-level SIMD mode of computation). Fig. 2 illustrates the meaning of this parameter.
- $D_k$: the maximum distance between two PE's in the target onto which two (4-connected) neighboring nodes from the source level $k$ are mapped; if $T_k > 1$, then $D_k = 1$. $D_k$ will be called the *dilation of level k*. The distances between PE's onto which parents and children of the source architecture are mapped are kept small as follows. If the size of the source level is smaller than the size of the target level, then the distance between any two chosen PE's at the target level is equal to $(\alpha_1 D_k, \alpha_2 D_k)$, where $0 \le \alpha_1, \alpha_2 \le \sqrt{Z_k} - 1$, with $Z_k$ being the total number of nodes at level $k$ of the source architecture. Fig. 3 illustrates the meaning of this parameter. The component $g_k$ of the mapping function in (1) is then given by $g_k(z_k) = \lfloor z_k D_k / T_k \rfloor$, where $z_k$ is either $x_k$ or $y_k$.

Since the mapping problem in general is NP-complete, high performance may be obtained at relatively low cost through the introduction of process assignment and scheduling algorithms tailored to the requirements of the IPCV algorithms. As a consequence, this research distinguishes between two classes of important IPCV algorithms based on their performance requirements with respect to the utilization of levels on multilevel systems. Algorithms in the first class, say *class A*, process a single image without applying simultaneous concurrency at different levels (i.e., one level of the source architecture is active at a time). The second class of algorithms, say *class B*, corresponds to single-image concurrent multilevel processing and pipelining (in the latter case, sequences of bottom-up operations are applied to either a single image or multiple images, e.g., histogramming, finding the perimeter of objects in more than one image). The minimization of the total image turnaround time, irrespective of the utilization of levels on the target architecture, is the optimization goal for class A algorithms. However, the maximization of the throughput for multiple image processing, and the minimization of the total image turnaround time for single-image concurrent multilevel processing and pipelining with a single image (i.e., for class B algorithms), require the minimizatin of the workload, corresponding to the processing of a single image, for each target level; this is because the level with the highest utilization will be the performance bottleneck. Two objective functions are developed for each class of IPCV algorithms in the next subsection. They correspond to the deterministic and nondeterministic cases; the distinction between deterministic and nondeterministic algorithms was made in the Introduction. The objective functions will be used to measure the quality of mappings with respect to the corresponding optimization goals.

Process assignment algorithms, one for each objective function, that produce good quality solutions by evaluating the proposed objective functions are proposed in Section IV. The procedures to evaluate the objective functions are also presented. This includes the introduction of routing rules for the implementation of communication operations. After the completion of the process assignment phase, the scheduling of operations is treated. Three scheduling algorithms of low computation complexity are proposed, one for each of the following categories of IPCV algorithms: those that process a

single image without applying multilevel concurrency (sometimes a small portion of multilevel concurrency may be present), those that process a single image by applying concurrent multilevel processing, and those that apply pipelining. Not only is the justification of the mapping algorithms based on theoretical issues, but also on their performance with examples. Results show that the mapping algorithms produce very good mappings, which are most of the times optimal under the proposed routing rule.

The evaluation of the objective functions requires the knowledge of some parameters. Two sets of parameters that identify the structures of the source and target architectures should be available.

The *set of parameters for the source architecture* are

- $n$: $2^n \times 2^n$ is the total number of nodes at the lowest level;
- $l$: the total number of levels; and
- $r_{k,k+1}$, for $0 \le k \le l-2$: $r_{k,k+1} \times r_{k,k+1}$ is the reduction between levels $k$ and $k+1$.

The *set of parameters for the target architecture* are

- $N$: $2^N \times 2^N$ is the total number of PE's at the lowest level;
- $L$: the total number of levels; and
- $R_{K,K+1}$, for $0 \le K \le L-2$: $R_{K,K+1} \times R_{K,K+1}$ is the reduction between levels $K$ and $K+1$.

The computation and communication requirements of deterministic algorithms are also input parameters. This requires that the execution time of all their instructions be known.

### B. Objective Functions

Objective functions for deterministic IPCV algorithms are first presented. They are followed by objective functions for nondeterministic IPCV algorithms. The ultimate goal for class A algorithms is the minimization of the total image turnaround time. The total image turnaround time is equal to the sum of the total computation, lateral communication, and vertical communication times. These are the components of the proposed objective function for deterministic class A algorithms:

$$OF_1 = \sum_{k=0}^{l-1} (comp_k + comm_k + comm_{k-1,k})$$

where

- $comp_k$: the new (i.e., after the mapping) total computation time for the source level $k$.
- $comm_k$: the new total lateral communication time for the source level $k$, and
- $comm_{k-1,k}$: the new total time for vertical communication operations between the pair of source levels $k-1$ and $k$ (assume that $comm_{-1,0} = 0$).

Normally, the parallel implementation of operations, as indicated by the application algorithm, should be considered for the evaluation of the objective function. In addition, uniformity of communication operations is assumed for all of the levels, due to the per-level SIMD mode of computation. The shortest paths are found for the implementation of data transfers. For multiple shortest paths, one of them is chosen at random. The traffic corresponding to a single communication operation between two nodes may not be split and routed along different paths. The calculation of communication times is done in a way similar to that used for the quadratic assignment problem [14]. For a communication pattern $q$ involved in the transmission of data from node $r$ to node $s$ of the source architecture, the communication cost on the target architecture is

$$c_{rs}(q) = S^1_{r'} + \sum_{c \in \{I(r,s) \cup r'\}} CM_c(r,s,q) + \sum_{c \in I(r,s)} (S^2_c + DL_c(r,s,q)) + DL_{s'}(r,s,q) + S^3_{s'}$$

where

- $r'$ and $s'$: the PE's of the target architecture onto which nodes $r$ and $s$, respectively, of the source architecture are mapped;
- $S^1_{r'}$: the setup time for the transmission of the data from $r'$;
- $I(r,s)$: the set of intermediate PE's for the transmission of the data (the cardinality of this set is $d(r',s') - 1$, where $d(r',s')$ is the shortest distance between $r'$ and $s'$);
- $CM_c(r,s,q)$: the time it takes PE $c$ to transmit the data, which is equal to the ratio of the size of the data to the bandwidth of the outgoing channel;
- $S^2_c$: the time it takes PE $c$ to copy the data from an incoming DTR into an outgoing DTR;
- $DL_c(r,s,q)$: the delay at PE $c$, which is the sum of the times it takes to transmit (or store) data of higher priority (starting at the time the data arrives at the PE); the priority may depend on the chosen routing algorihtm, as well as on the order of operations in the application algorithm; and
- $S^3_{s'}$: the setup time to receive the data at $s'$.

The objective function to be minimized for deterministic class B algorithms is

$$OF_2 = \max_{K \in F} \left\{ \sum_{k \in S_K} (comp_k + comm_k + comm^K_{k \pm 1,k}) \right\}$$

where

- $comm^K_{k \pm 1,k}$: the part of $comm_{k \pm 1,k}$ that keeps the target level $K$ busy,
- set $F = \{K /$ a target level onto which one or more source levels are mapped$\}$, and
- set $S_K = \{k /$ a source level mapped onto level $K \in F\}$.

This objective function represents the workload of the target level(s) which is(are) the performance bottleneck(s) for the corresponding IPCV algorithm.

To achieve the objective of small total image turnaround time for nondeterministic class A algorithms, neighboring nodes (i.e., processes) of the source architecture should be mapped onto PE's of the target architecture which are close enough, while the workload (expressed in number of assigned processes) of target PE's should be kept as low as possible. Thus, the minimization of the following objective function

becomes the goal for nondeterministic class A algorithms:

$$OF_3 = \sum_{k=0}^{l-1} T_k^2 D_k.$$

This objective function is a qualitative measure of the total image turnaround time on the target architecture because the computation times and the lateral communication times are proportional to $T_k^2$ and/or $D_k$ while the proposed process assignment algorithms always yield very small vertical distances. The following definition is pertinent for the justification of the latter.

*Definition 2:* A set of PE's $\Omega_\phi$ at a level $\phi$ are said to be *directly visible* from PE $\xi_\tau$ of a higher level $\tau$ if they are located within a subpyramid whose apex is $\xi_\tau$.

If there exist appropriate reductions in the target architecture, then a particular node from level $\sigma$ and its children from level $\sigma - 1$ of the source architecture are mapped onto PE $\xi_\tau$ of level $\tau$ and the set of PE's $\Omega_\phi$ of level $\phi$, respectively, in the target architecture, where $\phi < \tau$, so that the set of PE's $\Omega_\phi$ are directly visible from $\xi_\tau$. Otherwise, the node at level $\sigma$ is mapped in the middle of a square outlined by a collection of PE's; the set of PE's $\Omega_\phi$ are then directly visible from the PE's of that collection.

The objective function to be minimized for nondeterministic class B algorithms is

$$OF_4 = \max_{K \in F} \left\{ \sum_{k \in S_K} T_k^2 D_k \right\}$$

where the sets $F$ and $S_K$ are defined as for $OF_2$. This objective function represents a qualitative measure of the maximum workload for target levels. Since no weights are incorporated into the individual terms of this objective function, very high performance is derived when the workload is well balanced among the levels of the source architecture.

## IV. Process Assignment Algorithms

Four process assignment algorithms are proposed here, one for each objective function of the previous subsection. Each process assignment algorithm yields high performance through attempts to minimize the corresponding objective function. The next definition is essential in understanding the process assignment algorithms.

*Definition 3:* An *optimal mapping* for a source level $k$ is a mapping with parameters $T_k = 1$ and $D_k = 1$. In this case, if the source level $k$ is mapped onto the target level $K$, then each PE at the target level $K$ will be assigned the workload of a single node from level $k$ of the source architecture.

The process assignment algorithm that attempts the minimization of $OF_1$ for deterministic class A algorithms is first presented. It is followed by modifications that render the algorithm appropriate for deterministic class B algorithms and nondeterministic class A and class B algorithms. The variables $k$ and $K$ in the following process assignment algorithm represent a source and a target level, respectively.

*Step 1:* Map the source level 0 onto the target level 0. According to the previous section, we have $N \leq n$. If $N = n$, then the mapping of level 0 is optimal. If $N < n$, then every PE at the target level 0 will simulate an array of $2^{(n-N)} \times 2^{(n-N)}$ nodes from the source level 0, so set $T_0 = 2^{n-N}$ and $D_0 = 1$. In addition, initialize the variables $k$ and $K$ to 1.

*Step 2:* The *optimal reduction* in one dimension that yields optimal mapping for the source level $k$ is $OR_{k-1,k} = \lceil (D_{k-1}/T_{k-1}) r_{k-1,k} \rceil$, while the reduction between the pair of target levels $K - 1$ and $K$ is $R_{K-1,K}$. A distinction is made between the following two cases.

*Step 2a:* If $OR_{k-1,k} \leq R_{K-1,K}$ (i.e., the reduction in the target architecture is greater than or equal to the optimal reduction), then choose one of the following two candidate solutions, as described below.

*Step 2ai:* If the current source level $k$ is mapped onto the current target level $K$, then we get $T_{k,1} = R_{K-1,K}/OR_{k-1,k}$ and $D_{k,1} = 1$, where $T_{k,1}$ and $D_{k,1}$ represent the values of $T_k$ and $D_k$ if this assignment is chosen.

*Step 2aii:* However, if the current source level $k$ is mapped onto the target level $K - 1$, then we get $T_{k,2} = \lceil T_{k-1}/r_{k-1,k} \rceil$ and $D_{k,2} = OR_{k-1,k}$. PE's at the target level $K - 1$ which are assigned nodes from the source level $k$ in this case have Cartesian coordinates $(i_1 D_{k,2} + \lceil D_{k,2}/2 \rceil - 1, i_2 D_{k,2} + \lceil D_{k,2}/2 \rceil - 1)$, where $i_1$ and $i_2$ are positive integers; the Cartesian coordinates of the PE in the upper-left corner of each level are (0, 0). Thus, the minimization of average distances between pairs of parent–child nodes is guaranteed by allocating a parent in the middle of the square outlined by its four outermost children. For example, Fig. 4 shows the mapping of levels 4–6 from a source architecture onto a single target level.

For each of these two candidate solutions, find the value of $\alpha_k = comp_k + comm_k + comm_{k-1,k}$, and choose the one that assigns the smaller value to $\alpha_k$. If the first solution is chosen, then set $T_k = T_{k,1}, D_k = D_{k,1}, k = k + 1$, and $K = K + 1$; otherwise, set $T_k = T_{k,2}, D_k = D_{k,2}$, and $k = k + 1$.

*Step 2b:* If $OR_{k-1,k} > R_{K-1,K}$ (i.e., the optimal reduction is greater than the reduction in the target architecture), then map the source level $k$ onto the target level $\lambda$, where $\lambda$ is found as follows.

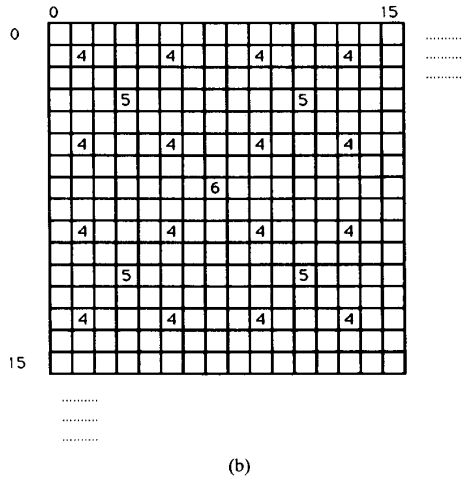
*Step 2bi:* If there exists a value $v$ (where $v < L$) such that $OR_{k-1,k} = \Pi_{i=K}^{v} R_{i-1,i}$, then set $\lambda = v, T_k = 1, D_k = 1, k = k + 1$, and $K = \lambda + 1$. ($\Pi_{i=K}^{v} R_{i-1,i}$ represents the total reduction in one dimension between the target levels $K-1$ and $v$, where $K$ assumes its previous value.) Optimal mapping results for level $k$ because of $T_k = 1$ and $D_k = 1$. The source levels $k - 1$ and $k$, where $k$ assumes its previous value, are mapped $\lambda - K_1 - 1$ levels apart in the target architecture, where $K_1$ is the target level onto which the source level $k - 1$ is mapped.

*Step 2bii:* However, if $OR_{k-1,k} > \Pi_{i=K}^{L-1} R_{i-1,i}$ (i.e., the total reduction produced by combining the reductions of the remaining target levels is smaller than the optimal reduction for level $k$), then set $\lambda = L - 1$ (that is, the source level $k$ is mapped onto the highest target level), $T_k = 1, D_k = OR_{k-1,k}/\Pi_{i=K}^{L-1} R_{i-1,i}, k = k + 1$, and $K = L$.

*Step 2biii:* Otherwise, find the largest value $\lambda$ that satisfies the inequality $OR_{k-1,k} > \Pi_{i=K}^{\lambda} R_{i-1,i}$, and then go to Step 2a with $r_{k-1,k} = OR_{k-1,k}/\Pi_{i=K}^{\lambda} R_{i-1,i}, T_{k-1} =$

| Level k of the source | Level K of the target with k→K | $T_k$ | $D_k$ |
|---|---|---|---|
| 0 | 0 | 1 | 1 |
| 1 | 1 | 1 | 1 |
| 2 | 1 | 1 | 4 |
| 3 | 2 | 1 | 1 |
| 4 | 2 | 1 | 4 |
| 5 | 2 | 1 | 8 |
| 6 | 2 | 1 | 16 |

(a)




(b)

Fig. 4. Mapping levels 4–6 of a source architecture onto a single target level.

$1, D_{k-1} = 1$, and $K = \lambda+1$ to choose from the two candidate solutions that map the current source level onto the target levels $\lambda$ and $\lambda + 1$, respectively, the solution that assigns the smaller value to $\alpha_k$ ($\alpha_k$ is found as in Step 2aii).

*Step 3:*

*Step 3a:* If $k = l$ (i.e., there are no more source levels), then go to Step 4.

*Step 3b:* If $K = L$ (i.e., there are no more target levels), then map the remaining source levels onto the highest target level. Apply the formulas of Step 2aii to find the parameters of these mappings.

*Step 3c:* Otherwise (if there are more levels in both the source and target architectures), go to Step 2 to decide about the mapping of the next source level.

*Step 4:* Stop. □

The process assignment algorithm that attempts the minimization of $OF_2$ is derived by modifying the above algorithm as follows.

*Change 1:* The mapping presented in Step 2bii is performed "temporarily." Processing resumes as follows. 1) Create a set $Q$ of source levels obtained by picking out from all source levels that have been mapped onto each target level (except for level 0) the one with the smallest number. 2) Choose from $Q$ the source level $\beta$ with the smallest difference in the two $\alpha_\beta$'s for the mappings onto the current and the immediately lower target levels; if more than one source level in $Q$ satisfy this condition, then choose the highest one. 3) If the source level $\beta$ is currently mapped onto the target level $\gamma$, then map level $\beta$ onto level $\gamma - 1$ (the parameters of this mapping are obtained by applying the formulas of Step 2aii). 4) Map the remaining levels, up to level $k - 1$, according to the current mapping of level $\beta$. 5) Finally, choose from the solutions obtained by the two mappings (i.e., the new solution and the "temporary" solution that existed before) the one that assigns the smaller value to the quantity $\max_{\gamma-1\le j\le\lambda}\{\Sigma_{i\in S_j}(comp_i + comm_i + comm^j_{i\pm1,i})\}$. This quantity is the part of $OF_2$ that corresponds to different mappings of levels in the two candidate solutions. Repeat the above five steps if the new solution is better than the old one. This procedure produces good mappings at low cost with minimal changes of previously existing (i.e., temporary) mappings.

*Change 2:* Step 3b is modified as follows. If $K = L$, then for each remaining source level, compare the following two candidate solutions by evaluating their $\alpha_k$'s: the solution that maps the source level onto the highest target level and the one obtained by applying the same technique as in Step 2bii. □

The process assignment algorithm proposed for deterministic class A algorithms is modified as follows for nondeterministic class A IPCV algorithms. In Step 2a, find the *relative speedup* $RS_{1,2}$ of the mapping onto level $K$ versus the mapping onto level $K - 1$ as the ratio $RS_{1,2} = T^2_{k,2}D_{k,2}/(T^2_{k,1}D_{k,1})$. If $RS_{1,2} > 1$, then choose the mapping onto level $K$; otherwise, choose the mapping onto level $K-1$. The process assignment algorithm for deterministic class B algorithms is modified as follows for nondetermministic class B algorithms. The relative speedup in Step 2a is given by $RS_{1,2} = (\Sigma_{i\in S_{K-1}}T^2_iD_i + T^2_{k,2}D_{k,2})/(T^2_{k,1}D_{k,1})$. Also, the quantity to be checked in the fifth step of Change 1 is $\max_{\gamma-1\le j\le\lambda}\{\Sigma_{i\in S_j}T^2_iD_i\}$.

For deterministic IPCV algorithms, the proposed process assignment algorithms do not always minimize the corresponding objective functions. Nevertheless, the minimization of $OF_1$ should most often be expected for class A algorithms because of the following two reasons. First, if an optimal mapping exists for a particular source level, it will be chosen by the process assignment algorithm. This choice should be expected to almost always minimize the computation and communication times associated with that level. Second, if no such mapping exists, then from the two mappings that have the smaller distances from the optimal mapping, the one with the smaller contribution to $OF_1$ is chosen. For similar reasons, the minimization of $OF_2$ should most often be expected for class B algorithms. Finally, the effectiveness of the process assignment techniques for nondeterministic IPCV algorithms is shown by the following theorem.

*Theorem 1:* The process assignment algorithms for nondeterministic IPCV algorithms always yield optimal assignments (i.e., they minimize the corresponding objective function $OF_3$ or $OF_4$).

*Proof:* Let us start with the proof for nondeterministic class A algorithms. The proof is based on mathematical induction. The best possible mapping is always chosen for level 0 of the source architecture (i.e., $T^2_0D_0$ is always minimized in Step 1a). Assume that the best possible mapping results for level $k - 1$, where $k < l$, with $k - 1 \mapsto K - 1$

(i.e., $T_{k-1}^2 D_{k-1}$ is minimal). We must prove that the process assignment algorithm then minimizes $T_k^2 D_k$. If an optimal mapping exists for level $k$, it is obviously chosen in Step 2a. The minimization of $T_k^2 D_k$ is then guaranteed because of $T_k = 1$ and $D_k = 1$. If the reduction between the target levels $K-1$ and $K$ is greater than the optimal reduction, then from the two candidate solutions in Steps 2ai (i.e., $k \mapsto K$) and 2aii (i.e., $k \mapsto K-1$), the one with the smaller value for $T_k^2 D_k$ is chosen. This minimizes $T_k^2 D_k$ because of the following two reasons. 1) Any mapping $k \mapsto K'$, where $K' > K$, will yield parameters $T_k'$ and $D_k'$ such that $T_k' > T_k$ and $D_k' = 1$, while $D_k = 1$. Thus, $T_k^2 D_k < T_k'^2 D_k'$. 2) Finally, any mapping $k \mapsto K''$, where $K'' < K-1$, will not obviously yield a better solution since $T_{k-1}^2 D_{k-1}$ is minimal. If the reduction between the target levels $K-1$ and $K$ is smaller than the optimal reduction for level $k$, then one of the Steps 2bi, 2bii, and 2biii is followed. Step 2bi yields $T_k^2 D_k = 1$, while the minimization of $T_k^2 D_k$ in Steps 2bii and 2biii could be proven by combining parts of the above discussion. Thus, the process assignment algorithm for nondeterministic class A algorithms always yields optimal assignments by minimizing $OF_3$. In addition, the process assignment algorithm for nondeterministic class B algorithms always yields optimal assignments by minimizing $OF_4$ because of the following two reasons. 1) With the assumption that the "shrinking operation" in Step 2bii is not performed, a proof similar to the previous one is applicable. 2) Finally, the procedure in Step 2bii guarantees minimal changes in all other cases, so the value of $OF_4$ is kept minimum. □

The following two theorems assume that the total computation time $CM_k$ is known for each source level $k$ ($comp_k$ is then equal to $T_k^2 CM_k$), and that it takes constant time to find the cost of a communication operation.

*Theorem 2:* The best case computation time of the process assignment algorithm that uses $OF_1$ is proportional to $l$. Its worst case computation time is proportional to the total number of communication operations in the algorithm.

*Proof:* For each source level in a bottom-up fashion (except for level 0), either an optimal mapping is found in constant time, it if exists, or two candidate mappings are compared. The best case appears when an optimal mapping is found for each level. Since $l$ is the total number of source levels, the best case computation time of the algorithm is proportional to $l$. The worst case is encountered when the quality of two candidate mappings is evaluated for each level. To evaluate the quality of each mapping, it takes time proportional to the total number of communication operations for the particular level. Thus, the worst case computation time is proportional to the total number of communication operations in the algorithm. □

*Theorem 3:* The best case computation time of the process assignment algorithm that uses $OF_2$ is proportional to $l$; its worst case computation time is proportional to the squared value of the total number of communication operations in the algorithm.

*Proof:* Changes 1 and 2 will not be carried out in the best case (i.e., the execution of the algorithm will not reach the corresponding steps); as in Theorem 2, the computation time will then be proportional to $l$. The worst case computation time is proportional to the squared value of the total number of communication operations in the algorithm because for each higher source level during the bottom-up mapping of levels, reassignment for all lower levels (except for level 0) is attempted. □

*Theorem 4:* The computation time of the process assignment algorithm that minimizes $OF_3$ is proportional to $l$.

*Proof:* For each source level in a bottom-up fashion, either an optimal mapping is chosen, if it exists, or two candidate mappings are evaluated in constant time. Thus, the computation time is proportional to $l$. □

*Theorem 5:* The best case computation time of the process assignment algorithm that minimizes $OF_4$ is proportional to $l$, while its worst case computation time is proportional to $l^2$.

*Proof:* Changes 2 and 3 will not be carried out in the best case, and the computation complexity of the algorithm will be given by Theorem 4. The worst case computation time is proportional to $l^2$ because for each higher source level during the bottom-up mapping of levels, reassignment for all lower levels (except for level 0) is attempted. □

## V. Scheduling Algorithms

As discussed in Section III, the process assignment phase is followed by the scheduling phase; the latter phase finds an appropriate schedule of the algorithm's operations that yields high performance on the given target architecture. Scheduling algorithms are proposed for three categories of IPCV algorithms: those that process a single image and either do not apply concurrent multilevel processing or utilize a small portion of multilevel concurrency, those that apply pipelining, and those that process a single image by applying concurrent multilevel processing. The first category deviates a little from the pure class A of single-image processing algorithms which are not involved in multilevel concurrency. This extended category is studied here because some small degree of multilevel concurrency can often be extracted from algorithms that in principle belong to class A. The last two categories constitute the class B of IPCV algorithms.

For the presentation in this section, graphs are used to represent IPCV algorithms. The following definition is pertinent.

*Definition 4:* The *time–space dependence graph (TS-DG)* of an algorithm for an $l$-level architecture is a weighted directed graph $(V, E)$. The set of nodes (vertices) $V$ form $l$ sequences of nodes distributed in time, with one sequence per level. Each node represents a group of arithmetic and/or logical functions, and/or lateral communication operations, and its weight is the corresponding execution time. The set of edges $E$ shows interlevel data dependencies between nodes at neighboring levels and their weights represent communication times.

Fig. 5 shows the most general structure of a TS-DG; TS-DG's should never contain cycles that lead to deadlocks. In the following discussion, the terms source TS-DG and target TS-DG stand for a TS-DG for a source and a target architecture, respectively.

Some parts of the scheduling sequences for nondeterministic algorithms may be decided upon at run time, while decisions

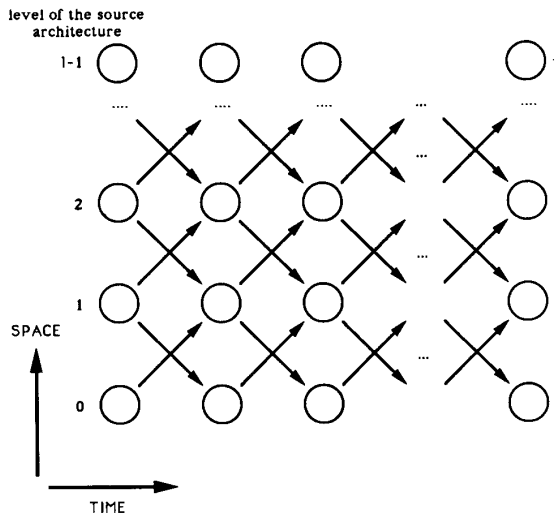



Fig. 5. The most general structure of a TS-DG.

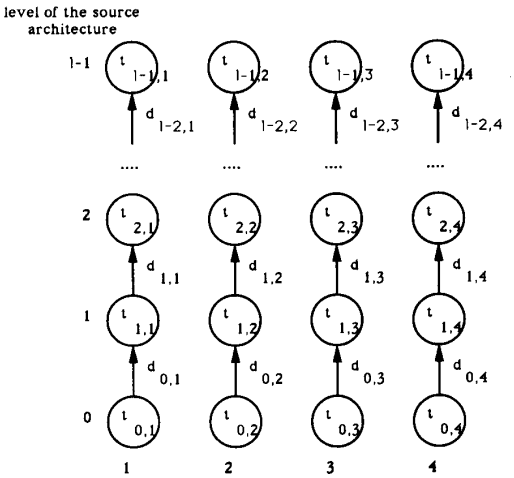



Fig. 6. The first four stages of a TS-DG for an algorithm that applies pipelining.

about other parts may be made at static time. It is assumed that the main controller initiates schedules by first sending identification numbers to the level controllers for the routines to be executed and then sending a "start signal" simultaneously to all of the level controllers. The main controller decides about the new state for nondeterministic algorithms after receiving interrupts from the level controllers. The rest of this section is devoted exclusively to deterministic algorithms.

### A. Pipelining

The scheduling problem for IPCV algorithms that apply pipelining is investigated here. This discussion will also simplify the presentation of the other scheduling algorithms in the next two subsections. A distinction is made here between algorithms that operate on a single image or multiple images. The same sequence of bottom-up operations is repeatedly applied to incoming images in the latter class of algorithms, while multiple sequences of bottom-up operations are applied to a single image in the former class of algorithms.

The first problem to be addressed is the transformation of a given source TS-DG after the mapping of source levels onto the levels of a given target architecture has been determined. Since pipelining implies the application of only bottom-up processes, it is adequate to study only the transformation of a single *space dependence graph (S-DG)* from the source TS-DG. The transformation algorithm applies a bottom-up process with respect to the levels of the source and target architectures. Assume that $t_{k,j}$ and $d_{k,j}$ (where $0 \leq k < l$ and $j = 1, 2, \cdots$) represent the weights of nodes and edges, respectively, of the source TS-DG (as shown in Fig. 6), while $t'_{k,j}$ and $d'_{k,j}$ represent the corresponding new weights that result from the chosen mapping of levels. Also, the following discussion focuses on the $j$th S-DG. Starting with level $k = 0$ of the source TS-DG and moving upward towards higher levels, the new weight $t'_{k,j}$ is found for each node with initial weight $t_{k,j}$. If there is not any node yet at level $K$, where $k \mapsto K$, of the target TS-DG which is initially empty, then $t'_{k,j}$ is assigned to a new node created for level $K$; otherwise, $t'_{k,j}$ is added to the current weight of the node at level $K$. Then, the following cases are investigated separately for an existing edge with weight $d_{k-1,j}$ (this discussion is for any $k > 0$).

1) If the source levels $k-1$ and $k$ are mapped onto the same target level $K$, then the weight of the node at level $K$ of the target TS-DG is incremented by $d'_{k-1,j}$.

2) If the source levels $k-1$ and $k$ are mapped onto the target levels $K-1$ and $K$, respectively, then detailed information about the individual communication pattern is required because the implementation of a data transfer between the source levels $k-1$ and $k$ may also involve lateral data transfers at the target levels $K-1$ and $K$. The communication times of these lateral data transfers (i.e., portions of $d'_{k-1,j}$) are then added to the weights of the nodes located at the corresponding levels of the target TS-DG. The rest of $d'_{k-1,j}$ will be the weight of a new edge that connects the nodes at levels $K-1$ and $K$ of the target TS-DG.

3) Finally, if the source levels $k-1$ and $k$ are mapped onto the target levels $K-\lambda$ and $K$, respectively (where $\lambda > 1$), then the implementation of the data transfer represented by the edge whose weight is $d_{k-1,j}$ contributes to the weights of new nodes and edges created for levels $K-\lambda$ through $K$ of the target TS-DG. The above procedure is also applicable to the remaining S-DG's of the source TS-DG.

For a chosen mapping of source levels and a routing rule, optimal schedules can result for single image processing algorithms that use pipelining by applying the FCFS (first-come, first-served) priority policy because of the continuous bottom-up information flow. For multiple image processing algorithms, the FCFS priority policy should use the ID number of incoming images (starting with 0 and incrementing the ID number with each incoming image) in order to always derive the best possible performance.

### B. Single-Image Processing Without Major Multilevel Concurrency

The scheduling of operations for single-image processing algorithms that are not involved in multilevel concurrency is

straightforward. The ease of scheduling stems from the fact that at most one operation will compete for any target level at any time. There also exist some single-image processing algorithms composed of both sequential and short concurrent multilevel processing stages. Such an algorithm finds the brightest spot in an image, which may be used as a seed point for region segmentation techniques or for tracking moving objects in dynamic scenes [2]. The TS-DG of this algorithm for a six-level source architecture that has reductions $2 \times 2$ between all pairs of neighboring levels is shown in Fig. 7(a) (the weights in the graph represent machine cycles). The algorithm is composed of a bottom-up processing stage, followed by a short concurrent multilevel processing stage, which is finally followed by a top-down processing stage. An example illustrates the scheduling algorithm. The target architecture for this example has three levels, PE's similar to those of the source architecture, and reductions $2 \times 2$ between all pairs of neighboring levels, while its lowest level contains the same number of nodes as level 0 of the source architecture. The invocation of the process assignment algorithm yields $0 \mapsto 0$ (with $T_0 = 1$ and $D_0 = 1), 1 \mapsto 0$ (with $T_1 = 1$ and $D_1 = 2), 2 \mapsto 1$ (with $T_2 = 1$ and $D_2 = 1), 3 \mapsto 1$ (with $T_3 = 1$ and $D_3 = 2), 4 \mapsto 1$ (with $T_4 = 1$ and $D_4 = 4$), and $5 \mapsto 2$ (with $T_5 = 1$ and $D_5 = 1$).

The weights of the source TS-DG in Fig. 7(a) are transformed as shown in Fig. 7(b). Each set of edges and nodes for source levels mapped onto the same target level that form a path of maximum length are then merged into a single node (the weight of this resultant node is the sum of the weights of those nodes and the in-between edges). On the contrary, an edge representing a data transfer between a pair of neighboring source levels which are mapped onto two distant target levels is expanded into nodes and edges (with a single node and a single outgoing edge per intermediate level to represent a retransmission operation). The application of this procedure to the TS-TG in Fig. 7(b) yields the target TS-DG of Fig. 7(c).

For a given mapping of source levels and a routing rule, the schedule that delivers the highest possible performance is found as follows. The dominant bottom-up and top-down processing stages suggest that the operations represented by nodes that precede an outgoing data transfer operation from any target level be first accommodated if the required data are available. Otherwise, if for such an implementation data need to arrive from a neighboring level, we schedule the operations for nodes, if any, that precede the node with the incoming arc and/or introduce idle states, if necessary. Preemptive scheduling is also used when data arrive while a level is busy, in order to give high priority to operations that lead to outgoing data transfers. The resultant schedules are always optimal. The application of this algorithm to the TS-DG in Fig. 7(c) yields the schedules shown in Fig. 7(d).

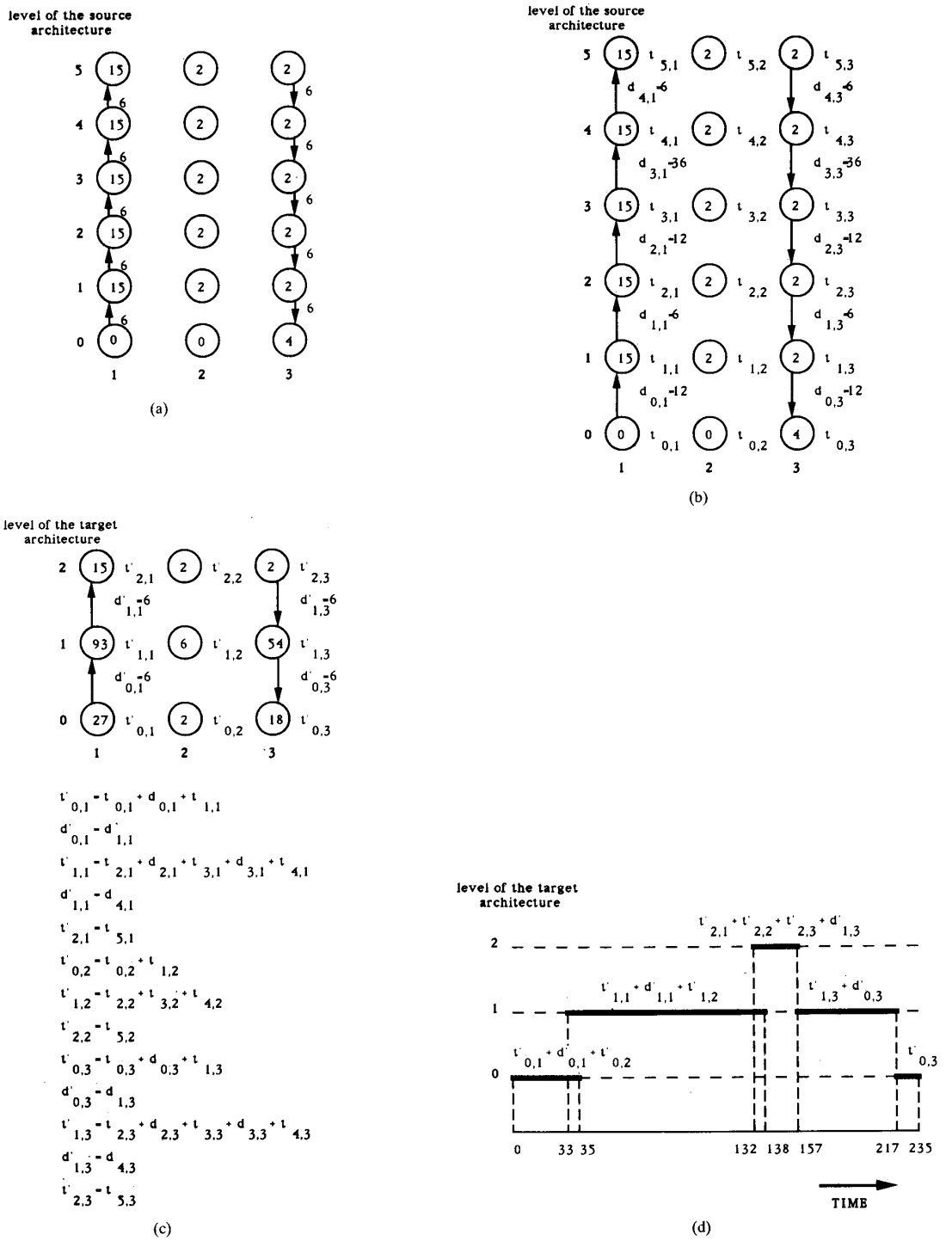
### C. Single-Image Concurrent Multilevel Processing (SICMLP)

This subsection studies the scheduling problem for algorithms that process a single image by applying multilevel concurrency. One of the most representative algorithms applies a concurrent multilevel relaxation technique to derive visible surface representations [23]. The objective of multilevel relaxation is to efficiently compute discrete, multiresolution approximations to the solutions of given continuous problems. A dynamic multilevel technique initially applies a strong coarse-to-fine interaction which accelerates the convergence rate of the finer relaxation processes during the early iterations when the approximation is rather poor. This interaction decays as the approximation improves and is replaced by a strengthening fine-to-course interaction, which eventually enables accurate approximations computed at finer levels to dominate and improve the accuracy of coarser approximations.[3] The TS-DG of such an algorithm is similar to that shown in Fig. 5. For the sake of efficiency, it is assumed that the iterative process is terminated when a predefined convergence criterion is satisfied. This calls upon an efficient schedule for the operations of a single S-DG from the TS-DG; the operations to check the condition of convergence are also included. This is because the minimization of the execution time for the operations within a single S-DG guarantees the minimization of the total image turnaround time. A single S-DG is called the *kernel* of the TS-DG in the rest of the discussion.

Thus, the kernel of the algorithm's source TS-DG should first be transformed based on the characteristics of the target architecture and the chosen mapping of source levels. The stages of this transformation are best illustrated in Fig. 8 with an example that involves a source architecture composed of eight levels and a target architecture composed of three levels, with $0, 1, 2 \mapsto 0; 3, 4, 5 \mapsto 1$; and $6, 7 \mapsto 2$. After the calculation of the new weights for nodes and edges of the TS-DG's kernel [see Fig. 8(a)], any nodes and their incoming edge(s) that represent operations to be performed at the same target level are merged into a single node [see Fig. 8(b)]. If two neighboring source levels $\sigma$ and $\sigma + 1$ are mapped onto two distant target levels $\sigma'$ and $\sigma' + \lambda$, respectively (i.e., $\lambda > 1$), then the pair of edges that represent interlevel data transfers between these two source levels will be transformed into two edges of opposite direction for each of the intermediate $\lambda - 1$ levels.

The scheduling algorithm operates as follows. To avoid unnecessary idle states, each level initially transmits all required data to its neighboring level(s). It then proceeds with the execution of other existing ready operations. As shown in Fig. 8(c), there may exist at most three nodes at any level of the target TS-DG's kernel. Although the structure of this graph does not conform to Definition 4, it will be referred to as a target TS-DG; the actual target TS-DG will be derived after the schedules are determined. Hence, the first decision to be made is about the order of the two outgoing data transfers for each target level. As shown in Fig. 9, each target level $m$ will be busy during the first $\tau_m = c_{m,0} + c_{m,1} + t_{m,1}$ time units (of course, $c_{0,1} = 0$ and $c_{m,0} = 0$ if $m$ is the highest level) because of first transmitting to neighboring levels and then performing internal operations (the implementation of the operations with weight $t_{m,1}$ does not depend on any incoming data). The appropriate times at which level $m$ should receive the data corresponding to the transmissions

[3] Similar techniques are also of interest to other application domains [6].




(a) (b) (c) (d)

Fig. 7. The scheduling of operations for the bright-spot detection algorithm. (a) The source TS-DG. (b) The transformation of weights after the mapping of levels. (c) The target TS-DG. (d) The scheduling of operations.

represented by the edges whose weights are $c_{m-1,0}$ and $c_{m+1,1}$ are found as follows. If $c_{m-1,0} + c_{m-1,1} \leq \tau_m$, then the transmission corresponding to $c_{m-1,0}$ could follow the transmission corresponding to $c_{m-1,1}$ without introducing any idle states in the scheduling of operations for level $m$. In a similar manner, if $c_{m+1,0} + c_{m+1,1} \leq \tau_m$, then

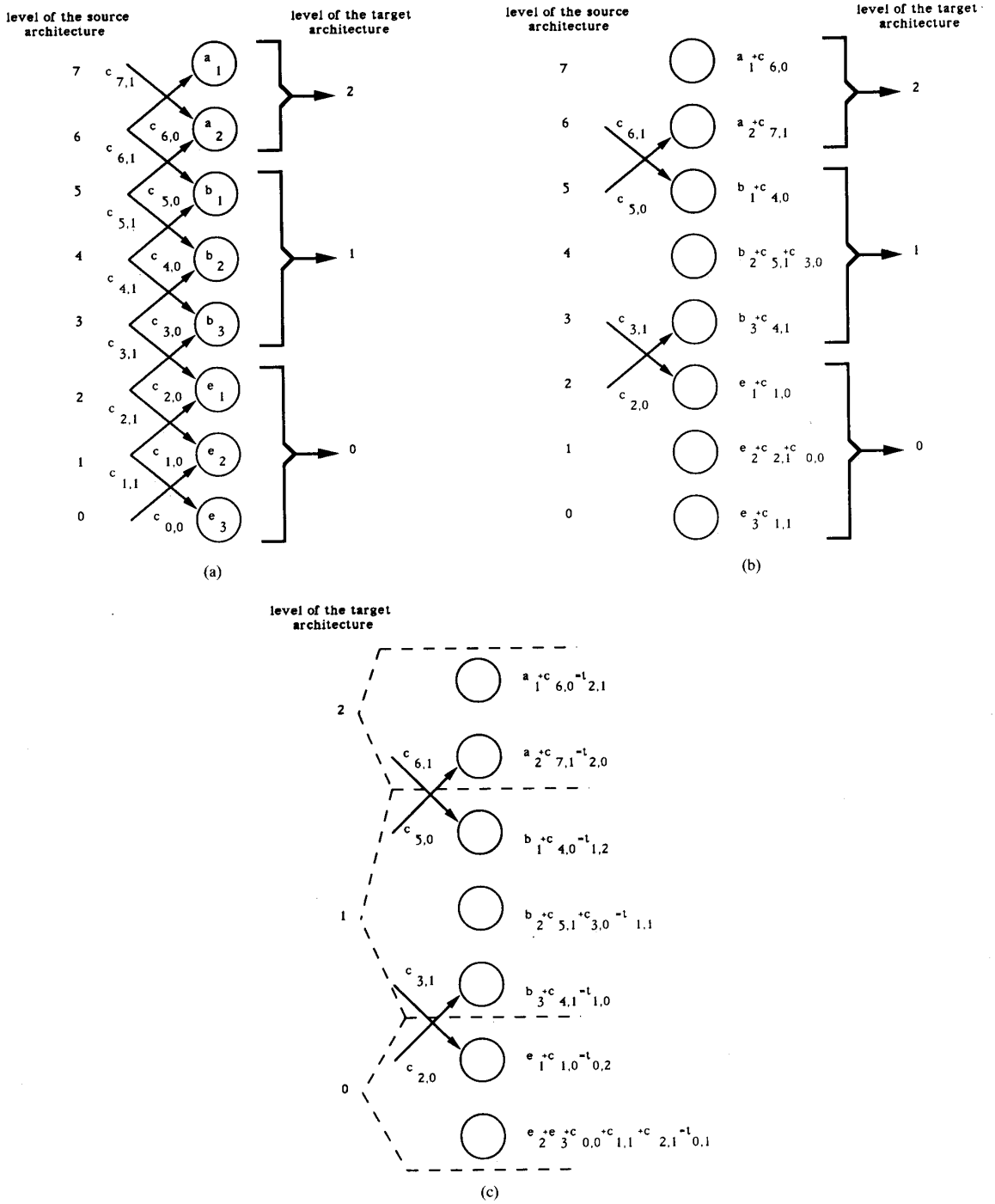



Fig. 8. The stages of transformation for the kernel of a source TS-DG that represents an SICMLP algorithm. (a) The new weights of the TS-DG's kernel after the mapping of source levels. (b) Merging interdependent operations that will be performed at the same target level. (c) The kernel of the "target" TS-DG.

the transmission corresponding to $c_{m+1,1}$ could follow the transmission corresponding to $c_{m+1,0}$ without introducing any idle states in the scheduling of operations for level $m$. Further processing is often required, so the scheduling algorithm may implement multiple bottom-up passes through the levels of the target TS-DG. In the first pass, all of the above conditions

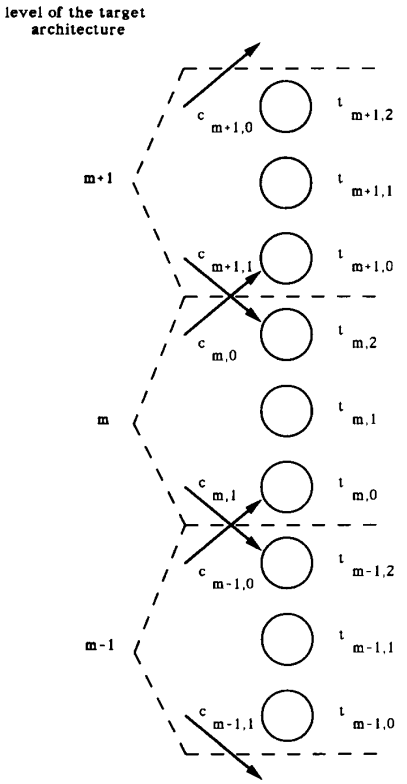



Fig. 9. Scheduling the operations for the target level $m$.

are checked for each target level $m$. If the first of the above conditions is met, then part of the scheduling sequence for level $m-1$ is determined by scheduling (if it was not scheduled earlier, when the current level was $m-2$) the transmission corresponding to $c_{m-1,1}$, then the transmission corresponding to $c_{m-1,0}$, and finally the set of operations corresponding to $t_{m-1,1}$. In addition, the set of operations for level $m$ represented by $t_{m,0}$ is scheduled to start at time $\tau_m$. If the second of the above conditions is met, then part of the scheduling sequence for level $m+1$ is determined by scheduling the transmission corresponding to $c_{m+1,0}$, then the transmission corresponding to $c_{m+1,1}$, and finally the set of operations corresponding to $t_{m+1,1}$. Also, if $t_{m,0}$ was not scheduled earlier, then the set of operations corresponding to $t_{m,2}$ is scheduled to start at $\tau_m$; otherwise, it is scheduled to start at $\tau_m + t_{m,0}$. New values are assigned to $\tau_{m-1}, \tau_m$, and $\tau_{m+1}$ to represent the total active times for the target levels $m-1, m$, and $m+1$, respectively, as dictated by the already determined scheduling sequences. The above procedure is then repeated until no further decisions about the scheduling of any operations can be made.

The next pass deals with levels, if any, for which the order of the implementation for the two outgoing data transfers has not been determined yet. Let $m$ be such a level. A decision is made about the order of the two transmissions for which $m$ is the source level by choosing the one that yields the smaller maximum of the total execution times for the two levels $m-1$ and $m+1$. In a further pass (to be performed if there still exist some unscheduled operations), the maximum execution time of target levels is first derived from the already determined schedules. For each target level, lower priority is assigned temporarily to the implementation of any remaining incoming transmission(s) to get the total execution time for the level (including idle states); if this time is smaller than the above maximum, then this temporary schedule is chosen. Finally, all possible implementations of any remaining transmissions are investigated.

If two consecutive source levels are mapped onto two distant target levels, then the interlevel transfers between these target levels will be scheduled by investigating the efficiency of all possible schedules for intermediate levels. To conclude, the above scheduling algorithm always yields optimal schedules at low cost (since the complexity of a TS-DG's kernel cannot be very high).

## VI. Performance Results

Examples of mapping important IPCV algorithms onto multilevel systems, pertinent performance results, and finally collective performance results are presented in this section. The performance of the proposed mapping algorithms is compared with the best possible performance in each case; the latter is obtained through exhaustive search (that is, through the investigation of all possible mappings of source levels onto target levels and the application of the routing rule of Section III). Code was developed to find the mapping(s) that yield the best possible performance. The results show that the proposed mapping algorithms yield the best possible performance in most of the cases. Details follow.

### A. Perimeter of an Object

This subsection deals with an algorithm that finds the perimeter of an object in a binary image. Hypothetical PE's with characteristics very close to those of the IMS T800 Transputer of Inmos are considered. The source and target architectures have the same total number of nodes at level 0. The source architecture is a pyramid with eight levels, while eight target architectures composed of one–eight levels with reductions $2 \times 2$ between all pairs of neighboring levels are considered. The algorithm operates as follows. Assuming allocation of a single pixel per PE at level 0, boundary pixels are first found at this level. Those PE's that contain a boundary pixel send the value of 1 to their parent. PE's at level 1 sum up the values they receive from their children and send the result to their parent. In a bottom-up fashion, PE's at higher levels perform the same operations as their children. This way, the apex finds the perimeter of the object by summing up the values it receives from its children. The results of applying our mapping algorithms are shown in Fig. 10 (the values of $OF_1$ and $OF_2$ are expressed in numbers of machine cycles). The performance obtained through the application of these mapping algorithms is the best possible in all cases.

### B. Convolution

$8 \times 8$ convolution of an image is the subject of this subsection. The source architecture has two levels, reduction $8 \times 8$, and the same number of nodes at level 0 as the

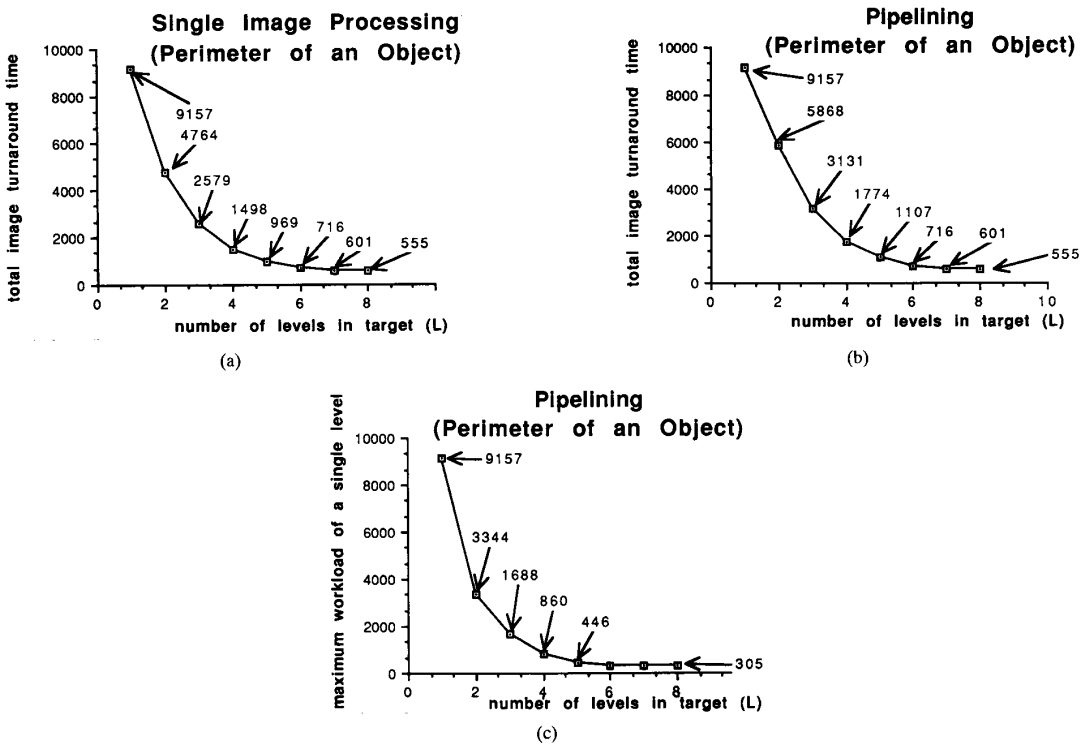



Fig. 10. Performance results for the algorithm that finds the perimeter of an object. (a) The value of $OF_1$ (i.e., total image turnaround time) for single image processing. (b) The value of $OF_1$ for pipelining. (c) The value of $OF_2$ (i.e., maximum workload of a single level) for pipelining.

TABLE II
PERFORMANCE RESULTS FOR THE CONVOLUTION ALGORITHM

| Reductions in the target architecture | Total number of PE's | Total number of channels | Total image turnaround time | $U_K$ (for K=0,1,...,L-1) | U |
|---|---|---|---|---|---|
| 8 | 266,240 | 793,472 | 57,580 | 0.40, 0.60 | 0.50 |
| 2, 4 | 331,776 | 989,568 | 64,940 | 0.35, 0, 0.53 | 0.29 |
| 4, 2 | 282,624 | 842,368 | 82,604 | 0.28, 0, 0.42 | 0.23 |
| 2, 2, 2 | 348,160 | 1,038,464 | 85,548 | 0.27, 0, 0, 0.40 | 0.17 |
| 2, 2 | 344,064 | 1,014,016 | 140,012 | 0.16, 0, 0.25 | 0.14 |
| 16 | 263,168 | 787,392 | 150,316 | 0.14, 0.86 | 0.50 |

target architectures. PE's with characteristics similar to those in the preceding subsection are considered. The convolution algorithm applies 64 times some internal operations and lateral shifts at level 0, a sequence of bottom-up operations, and a sequence of top-down operations. Performance results are shown in Table II (the reductions are shown for one dimension). The performance obtained through the application of the mapping algorithm for class A algorithms is the best possible in all cases.

Table II also shows the utilization of target levels. The *average utilization of PE's at the target level K* is defined as

$$U_K = \frac{1}{E} \frac{\sum_{i=1}^{H_K} t_{K,i} A_{K,i}}{(2^N / \prod_{i=1}^{K} R_{i-1,i})^2}$$

where

- $E$: the total image turnaround time,
- $H_K$: the total number of steps performed at the target level $K$,
- $t_{K,i}$: the time it takes to perform the $i$th step at the target level $K$ (it does not include the overheads of retransmission and extra masking operations), and
- $A_{K,i}$: the total number of active PE's at the target level $K$ during the execution of the $i$th step.

The denominator of the second fraction represents the total number of PE's at the target level $K$. The *average utilization of PE's at a single target level* is given by $U = \Sigma_{K=0}^{L-1} U_K / L$ ($L$ is the total number of levels in the target architecture).

More performance measures could be introduced to represent the overheads of retransmission and extra masking operations, etc. However, their importance is much lower

compared to the importance of the above measures.[4] The results in Table II may be used in a tradeoff analysis between performance and cost. The first architecture in the table is the best one because it offers high performance with high utilization of PE's, while it maintains a relatively low cost. The second–fifth architectures do not match well with the convolution algorithm because the average utilization of their PE's is low. Furthermore, the fifth architecture seems to be the worst choice because it has very high cost, yields a very large total image turnaround time, and the average utilization of its PE's is very low. The third architecture is preferred to the fourth architecture because it has lower cost, provides a smaller total image turnaround time, and the average utilization of its PE's is higher. The sixth architecture provides the worst total image turnaround time, but has the lowest cost of all six architectures and the average utilization of its PE's is high. Finally, the second and third architectures provide medium performance and have relatively low cost.

### C. Collective Results

The computation and communication requirements of some important IPCV algorithms were calculated in order to design random number generators that produce reasonable computation and communication requirements for hypothetical algorithms. More specifically, for each IPCV algorithm, these requirements were allowed to vary between 0 and 50% from the calculated values. Diverse sets of source and target architectures were also considered. The performance of the proposed mapping algorithms was compared with the best possible performance in each case (when the routing rule of Section III is applied). The results show that the mapping algorithms always yield very high performance, which is very often the best possible. Thus, the mapping techniques for nondeterministic IPCV algorithms were also proven to be very effective. More specifically, very high performance, which was most often the best possible, was obtained for class A algorithms. Even though the mapping techniques perform well for nondeterministic class B algorithms, the highest performance was basically obtained when the workload was well balanced among the set of source levels.

## VII. Conclusions

It is well known, primarily from analytical results, that pyramid systems could very efficiently implement a wide variety of low-level and intermediate-level IPCV algorithms. However, either the current technology may not allow the construction of very powerful pyramid systems with a large number of PE's, or their cost may be prohibitively high. Hence, the performance and cost of alternative systems must be investigated. This paper provided the necessary tools for investigating the performance of multilevel (i.e., pyramid-like) systems. More specifically, its major objective was to propose algorithms for mapping IPCV algorithms onto multilevel systems. The general optimization goal of the mapping algorithms is to satisfy the performance requirements of the IPCV algorithms. This is accomplished by first identifying two important classes of IPCV algorithms for multilevel systems; the distinction between the two classes was made based on the required utilization of levels that yields high performance. Four objective functions, two for each class of IPCV algorithms, that correspond to the deterministic and nondeterministic cases were proposed in order to measure the quality of given mappings with respect to the corresponding optimization goal. Mapping algorithms, one for each objective function, were also proposed to derive high-performance mappings through attempts to minimize the corresponding objective functions. Performance results for a wide variety of IPCV algorithms and multilevel systems show the effectiveness of the proposed mapping techniques. These techniques can also be used to identify multilevel structures for the efficient implementation of algorithms on hypercube parallel computers [28].

[4] Had we not distinguished between operations in the algorithm and extra retransmission and masking operations for the computation of $U_K$, the knowledge of such overheads should have been absolutely essential.

## References


[1] N. Ahuja and S. Swamy, "Multiprocessor pyramid architectures for bottom-up image analysis," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. PAMI-6, pp. 463–474, July 1984.

[2] R. P. Blanford and S. L. Tanimoto, "Bright-spot detection in pyramids," *Comput. Vision, Graphics, Image Processing,* vol. CVGIP-43, pp. 133–149, Aug. 1988.

[3] S. H. Bokhari, *Assignment Problems in Parallel and Distributed Computing.* Boston, MA: Kluwer Academic, 1987.

[4] ——, "On the mapping problem," *IEEE Trans. Comput.*, vol. C-30, pp. 207–214, Mar. 1981.

[5] S. W. Bollinger and S. F. Midkiff, "Heuristic technique for processor and link assignment in multicomputers," *IEEE Trans. Comput.*, vol. 40, pp. 325–332, Mar. 1991.

[6] A. Brandt, "Multilevel computations: Review and recent developments," in *Multigrid Methods—Theory, Applications, and Supercomputing,* S. F. McCormick, Ed. New York: Marcel Dekker, 1988.

[7] V. Cantoni and S. Levialdi, "Multiprocessor computing for images," *Proc. IEEE,* Special Issue on Computer Vision, vol. 76, pp. 959–969, Aug. 1988.

[8] T. F. Chan and Y. Saad, "Multigrid algorithms on the hypercube multiprocessor," *IEEE Trans. Comput.*, vol. C-35, pp. 969–977, Nov. 1986.

[9] P. Clermont and A. Merigot, "Real time synchronization in a multi-SIMD massively parallel machine," in *Proc. Conf. Architecture Pattern Anal. Machine Intell.*, 1987, pp. 131–136.

[10] J. E. Cuny and L. Snyder, "A model for analyzing generalized interprocessor communication systems," in *Algorithmically Specialized Parallel Computers,* L. Snyder, L. H. Jamieson, D. B. Gannon, and H. J. Siegel, Eds. Orlando, FL: Academic, 1985, pp. 7–16.

[11] M. J. B. Duff, "The limits of computing in arrays of processors," in *From Pixels to Features,* J. C. Simon, Ed. The Netherlands: North-Holland, 1989.

[12] C. R. Dyer, "A VLSI pyramid machine for hierarchical parallel image processing," in *Proc. Conf. Pattern Recognition Image Processing,* 1981, pp. 381–386.

[13] L. G. C. Hamey, J. A. Webb, and I.-C. Wu, "Apply, A programming language for low-level vision on diverse parallel architectures," in *Parallel Computation and Computers for Artificial Intelligence,* J. Kowalik, Ed. Boston, MA: Kluwer Academic, 1987.

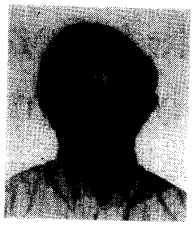
[14] M. Hanan and J. M. Kurtzberg, "A review of the placement and quadratic assignment problems," *SIAM Rev.*, vol. 14, pp. 324–342, Apr. 1972.

[15] W. D. Hillis, *The Connection Machine.* Cambridge, MA: M.I.T. Press, 1985.

[16] T.-H. Lai and W. White, "Mapping pyramid algorithms into hypercubes," *J. Parallel Distributed Computing,* vol. 9, pp. 42–54, 1990.

[17] M. Mareska, M. A. Lavin, and H. Li, "Parallel architectures for vision," *Proc. IEEE,* Special Issue Comput. Vision, vol. 76, pp. 959–969, Aug. 1988.

[18] A. P. Reeves, "Pyramid algorithms on processor arrays," in *Pyramidal Systems for Computer Vision,* V. Cantoni and S. Levialdi, Eds. New York: Springer-Verlag, NATO ASI Series F, vol. 25, 1986, pp. 195–214.
[19] A. Rosenfeld, Ed., *Multiresolution Image Processing and Analysis.* New York: Springer-Verlag, 1984.
[20] C.-C. Shen and W.-H. Tsai, "A graph matching approach to optimal task assignment in distributed computing systems using a minimax criterion," *IEEE Trans. Comput.,* vol. C-34, pp. 197–203, Mar. 1985.
[21] Q. F. Stout, "Hypercubes and pyramids," in *Pyramidal Systems for Computer Vision,* V. Cantoni and S. Levialdi, Eds. New York: Springer-Verlag, NATO ASI Series F, vol. 25, 1986, pp. 75–89.
[22] S. L. Tanimoto, "A pyramidal approach to parallel processing," in *Proc. 10th Int. Conf. Comput. Architecture,* 1983, pp. 372–378.
[23] D. Terzopoulos, "Concurrent multilevel relaxation," in *Proc. Image Underst. Workshop,* L. S. Baumann, Ed., Miami Beach, FL, 1985.
[24] L. Uhr, Ed., *Parallel Computer Vision.* New York: Academic, 1987.
[25] S. G. Ziavras, "On the mapping problem for multi-level systems," in *Proc. Supercomputing '89,* IEEE Computer Society and ACM SIGARCH, Reno, NV, Nov. 1989, pp. 399–408.
[26] ——, "Techniques for mapping deterministic algorithms onto multi-level systems," in *Proc. Int. Conf. Parallel Processing,* St. Charles, IL, Aug. 1990, pp. 226–233.
[27] S. G. Ziavras and L. S. Davis, "Fast addition on the fat pyramid and its simulation on the connection machine," in *From Pixels to Features,* J. C. Simon, Ed. The Netherlands: North-Holland, 1989, pp. 373–382.
[28] S. G. Ziavras and D. P. Shah, "High performance emulation of hierarchical structures on hypercube supercomputers," *Concurrency: Practice and Experience,* vol. 6, no. 1, 1994.



**Sotirios G. Ziavras** (S'84–M'91) received the Diploma in electrical engineering from the National Technical University of Athens, Greece, in 1984, the M.Sc. degree in electrical and computer engineering from Ohio University in 1985, and the D.Sc. degree in computer science from George Washington University in 1990, where he was a Distinguished Graduate Teaching Assistant and a Graduate Research Assistant.

From 1988 to 1989, he was with the Center for Automation Research at the University of Maryland, College Park. He was a Visiting Assistant Professor in the Department of Electrical and Computer Engineering, George Mason University, during the Spring semester of 1990. He is currently an Assistant Professor in the Department of Electrical and Computer Engineering, New Jersey Institute of Technology.

Dr. Ziavras has received several fellowships and awards, including an award from the Greek Government (IKY) in 1984 and the GWU Richard E. Merwin Doctoral Fellowship in 1986. He was the recipient of a National Science Foundation Research Initiation Award in 1991. He has authored or coauthored more than 40 referred journal and conference research papers. His research interests are parallel computing systems, parallel algorithms, processor allocation techniques, computer architecture, performance evaluation, and computer vision. He is a member of the IEEE Computer Society, ACM, Hellenic Chamber of Electrical Engineers, SIAM, New York Academy of Sciences, and Eta Kappa Nu.