

Received December 5, 2019, accepted January 24, 2020, date of publication February 6, 2020, date of current version February 18, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.2972204

# A New Approach to Parallel Processing

WILLIAM C. CAVE<sup>1</sup>, ROBERT E. WASSMER<sup>1</sup>, HENRY F. LEDGARD<sup>2</sup>, ALAN B. SALISBURY<sup>3</sup>,  
KENNETH T. IRVINE<sup>1</sup>, (Member, IEEE), AND MICHAEL A. MULSHINE<sup>4</sup>

<sup>1</sup>Prediction Systems, Inc., Spring Lake, NJ 07762, USA

<sup>2</sup>College of Engineering, The University of Toledo, Toledo, OH 43606, USA (Retired)

<sup>3</sup>A. B. Salisbury & Co., McLean, VA 22101, USA

<sup>4</sup>Osprey Partners, Manchester, NJ 08759, USA

Corresponding author: Kenneth T. Irvine (ken@predictsys.com)

**ABSTRACT** The Application Space Architecture (ASA) defines parallel processor design as an adjunct to the Instruction Set Architectures (ISA) defined by Von Neumann for single processors. Before addressing hardware, one must understand applications requiring parallel processors, and software approaches required to meet speed constraints. Parallel processor applications use a single task to run  $N$  (number-of-processors) times faster than could be done on a single processor, where Processor Utilization Efficiency (PUE) is the ratio of single processor time to parallel processor time divided by  $N$ . Such applications require processing to be split into independent elements running in parallel on a large number of processors. This requires that processors do not sit idle (current PUEs are typically 2-10%). This sets the requirement for “independent” elements to communicate with each other without stopping. In special cases, clusters of PCs are used to perform parametric analyses running many copies of a simulation on separate computers, typically using different random numbers. Except for initial and terminal processes, these separate simulations hardly communicate while running. Thus they are easily divided into separate tasks that typically exchange information between computers passing data or messages through a top level management task. Such applications (e.g., LINPACK) are labeled “embarrassingly parallel.”

**INDEX TERMS** Application space architecture, data structures, engineering drawings, instruction set architecture, module independence, parallel processing, processor utilization efficiency, separation principle, simulation, software architecture, software engineering, software design, state-space methods.

## I. INTRODUCTION

Hennessy and Patterson describe the need for an equivalent single processor Instruction Set Architecture (ISA) for parallel processors, [1]. This requires more than a special hardware instruction set such as that for servers processing multiple independent tasks on multiple processors, see [2], [3]. Those working many parallel processor applications know that, although they appear different, they all share critical properties. Those listed in Table 1 are sufficiently representative to characterize common properties that directly affect the measures described below. Unlike LINPACK, none of these applications is considered “embarrassingly parallel.” As described by Stensrud, [4], weather models likely require 6 orders of magnitude above today’s speeds to achieve the desired accuracy.

Software people working in large financial and product distribution organizations understand the software problem in

The associate editor coordinating the review of this manuscript and approving it for publication was Mohammad Anwar Hossain.

general. Systems are comprised of hierarchies of many different computer centers spread across the U.S., interacting 24/7. Distribution Centers may support thousands of clients, taking many inputs and producing many outputs. Requirements for such systems are not simple. Just defining communications between nodes is complex. Detailed architectural design is critical to development and maintenance of such systems. The software development environment determines the productivity of people building such systems. Software people consider language the critical element. As described by people highly familiar with parallel processing, software people cannot keep up with the huge advances made in computer hardware using existing languages, see Section XI.

Can one imagine building an aircraft without engineering drawings? We cannot imagine designing software without engineering drawings. But the words “software architecture” and “software module” are used in the literature - without being defined! Yet these definitions are critical when designing huge software architectures using engineering drawings. But is it being taught?

**TABLE 1.** Representative list of parallel processor applications.

Real-Time Control of Large Groups of Autonomous Moving Platforms	Scanning, sorting, and correlating massive databases (Big Data)
Human Body Organ simulation	Weather prediction in mountainous terrain
Global Climate prediction	Power distribution simulation
Fluid Flow simulation	Electro-magnetic wave simulation
Biological Particle simulation	Global HF power transmission
Chemical - Molecular structure simulation	Global Military Planning - Simulation of many moving platforms, equipment, & environments

In the 1960s, engineers could not wait months for programming groups to generate simulations of electronic circuits, especially those requiring highly nonlinear waveforms used in computer design. Understanding computers, engineers built their own Computer-Aided Design (CAD) systems for circuit design. Multiple simulations are now built in a day. CAD quickly expanded to other engineering fields, e.g., mechanical, architectural, and aeronautical.

Having developed multiple CAD systems, the theoretical concepts described here were initiated to minimize run time as well as software development/support costs for real-time control and communication systems as well as the applications listed in Table 1. The goal of an early project was to reduce simulation run times from seven days to an hour. This appeared to imply the use of parallel processing, provoking out-of-the-box thinking about software development. The principle requirement is the ability to have multiple software modules running non-stop in parallel. This is similar to people driving cars on a one-way 3-lane highway, glancing at others while making decisions. Information is sampled - as needed - without stopping. This implies substantial independence of the operators.

This led to the underlying property of independence as defined mathematically, and particularly by Kalman for designing real-time control systems, [5]. This then led to the concept of software architectures that could be created using independent modules. But what is a software module? And what is software architecture?

## II. THE PROPERTY OF INDEPENDENCE

When designing applications to run on parallel processors, one must take maximum advantage of inherent parallelism in the application. This implies splitting applications into separate elements that run simultaneously while exchanging information as they run - without stopping - representing actual physical systems affected by their environment as they move ahead.

It is the software design that determines the implementation of independent elements. Although the words “Module” and “Architecture” are used *undefined* in the software literature, they are critical terms in other engineering fields when properly defined. This requires mathematically based definitions of *software architectures* and *independent modules* for placement on separate parallel processors. When using

the definitions provided below to support these requirements, the need for a new approach to software design becomes apparent.

Although the following explanation starts with an obvious mathematical relation to some of the applications listed in Table 1, it applies to all when considering general definitions of underlying spaces used to define the state of a system. Military planning and big data systems are example applications where the parameters within complex state spaces that represent these systems can be tested and changed with logical data equations. These are also defined mathematically.

When investigating properties of modules that run in parallel on separate processors, they must be independent, i.e., they do not improperly affect each others’ outcomes. This leads to definitions of the property of *module independence*. This requires two types of independence: *spatial* and *temporal*. Two modules may be *spatially independent* - independent for all time. Or, they may be *temporally independent* - independent over specified time periods.

To build fast parallel processor software, one must take maximum advantage of the inherent parallelism in an application. This requires a software environment that simplifies mapping the independence properties into independent modules. Fig. 1 depicts steps required to map application requirements into independent software spaces, and then into hardware spaces designed to support optimal mapping of modules to maximize operational speed.

## III. A CAD APPROACH TO PARALLEL PROCESSOR SOFTWARE DESIGN

To solve parallel processor software problems that are not “embarrassingly parallel”, one must map the inherent parallelism in an application onto the hardware. This requires a software environment that simplifies the design effort for application experts. Fig. 2 illustrates *VisiSoft*, a Computer-Aided Design (CAD) approach that supports ease of translation of software designs into a hardware space that maximizes understanding as well as speed.

Much of the burden of design/implementation is then shifted from the developer to highly sophisticated architecture and language translators - using the computer. As described by Peter van der Linden, [6-pg 64], this is where the burden should be.

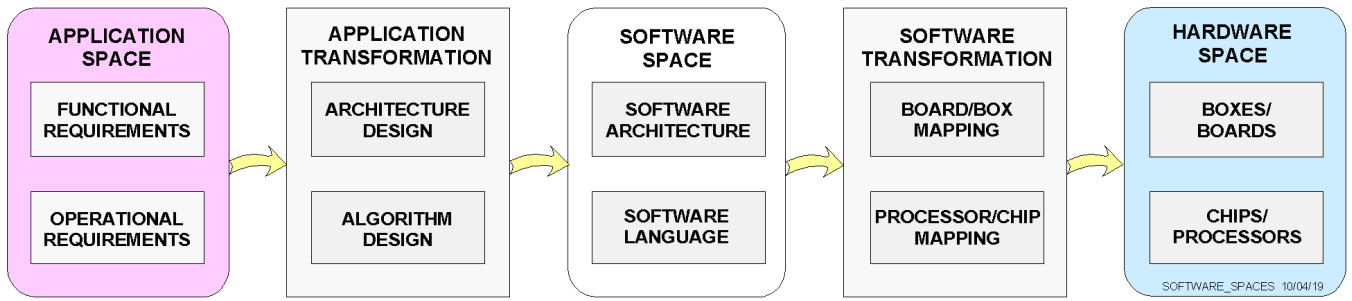


FIGURE 1. Spaces for translation of application requirements into software and then hardware.

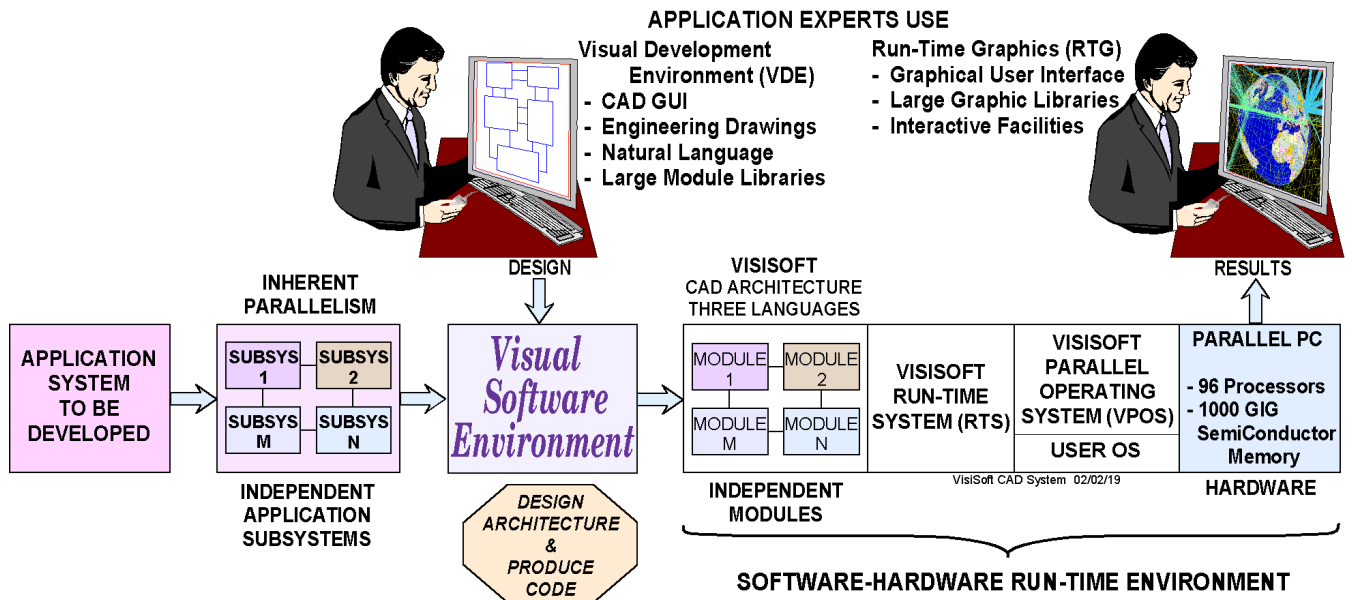


FIGURE 2. Using VisiSoft for translation of application requirements into software on parallel processors.

The transformations defined in Fig. 1 require special facilities illustrated in Fig. 2. They support selection of the “best spaces” to simplify the algorithms required to meet the application requirements. Selection of the best spaces implies that the resulting algorithms are:

- Maximally simplified while achieving desired accuracy
- Very fast
- Easy to build
- Easy to understand

These facilities must support the following requirements.

- Mapping the functional and operational application requirements into a software architecture and supporting algorithms requires: (1) Application experts with knowledge of the complex system and corresponding event spaces; and (2) Graphical facilities for visualization of software architecture - using engineering drawings.
- Mapping application space requirements into a software design for a parallel processor hardware space requires special architectural and language facilities. Graphical visualization of the architecture is critical to this process.

- Complex software spaces require the use of deep hierarchical data structures, organized in accordance with the application space - not by data type. These spaces must be easily organized and understood by application experts.
- Complex software algorithms require the use of deep hierarchical rule structures. These must be organized in accordance with the application space - using one-in one-out control structures that are easily understood by application experts, see Mills, [7].
- Application expertise is required to design Independent (IND) Modules. Special graphical visualization tools are critical to simplify understanding of both design of the modules and mapping them onto parallel processors.
- Mapping IND Modules over heterogeneous spaces provides substantial improvements in speed. Tests must be run to determine their relative speeds - for processor placement - to maximize Processor Utilization Efficiency (PUE).

- IND Module run-time speed measures are provided automatically (viewed as bar graphs of IND Module run-times on each processor). Speed is easily improved by reorganizing module assignments based on visual measures of PUE.
- Assignment of IND Modules across boxes, boards, chips and processors based on physical connectivity requires application experts and can substantially minimize delays when using DMA Channels.
- During initialization, architectural information derived from the development environment is provided to the VisiSoft Parallel OS (VPOS). This coupled with the proper allocation of IND Modules can eliminate the need for run-time memory management at the L3 cache and data level.
- Selection of cross-processor event states by application experts can save huge amounts of time by synchronizing the exchange of information between IND Modules that are temporally independent and can continue to run concurrently.

#### A. MEASURES OF COMPUTERS AND SOFTWARE

In the computer field, hardware products are generally measured in terms of speed and ease of use (both of which translate to time and cost):

- How much time does it take to run an application?
- How much time does it take to build/modify an application?

The first stored program computer - the Maniac - was designed by Eckert, Mauchley, and Von Neumann, [8]. Instead of wiring boards and worrying about timing and synchronization of calculations required by existing tabulators, the new architecture stored instructions in memory with the data. Using a binary instruction set - Von Neumann's Instruction Set Architecture (ISA) - the hardware design requirements were defined to simplify programming. Programmers could list the sequence of instructions to be executed in specified memory locations along with the data.

Since stored ISA instructions run sequentially, their operations are independent. The *property of independence* made computer programming simple - no more concerns about timing and race conditions. The field exploded. This software productivity breakthrough still holds today for single processors. But it does not apply to parallel processors, see The ASA, [2], and [3].

When building complex software applications to minimize run-time on parallel processors, run-time speed of different approaches is measured by comparing the time it takes to run an application. One can first compare the single processor time achieved by the different approaches, and then compare results on parallel processors using the best single processor time. To produce comparative measures of software speeds to evaluate the facilities described above, one can calculate the PUE, defined as:

$$PUE = \frac{1}{N_p} * \left( \frac{T_S}{T_p} \right)$$

In the above measure,  $N_p$  is number of parallel processors used,  $T_S$  is the fastest time to run on a single processor, and  $T_p$  is time to run on a parallel processor.

#### B. CAD SOFTWARE ENGINEERING

To meet speed constraints, mathematically sound definitions are required to support parallel processor software architectures. This implies that a large task can be decomposed into hierarchies of IND Modules that share data while running concurrently. To do this, application experts must easily understand the detailed design - down to the code. This requires a CAD software development system using engineering drawings to visualize detailed designs for parallel processors as illustrated in Fig. 2 and also described in [3].

The context of architecture used here is no different from the hierarchy of drawings required to support the design of skyscrapers. This is unnecessary when building dog houses or small software applications. Similarly, architectural drawings must be accompanied by a set of easily understood specifications. This raises the questions: What does it take to produce engineering drawings of software; and what language properties are required to support complex hierarchical data spaces that are easily understood by application experts?

#### IV. THE SEPARATION PRINCIPLE

The underlying principle characterizing module independence is the ability to determine which modules share what data. This is accomplished by separating data from instructions at the language level. Defined by Cave in 1982 while designing the General Simulation System (GSS), this has become known as the *Separation Principle*, [9]. GSS was designed to support huge simulations for the DoD, [10]–[12]. Using the Generalized State Space framework, the Separation Principle stores all data in *Resources* as illustrated in Fig. 3.

Deep hierarchies are required to design complex spaces. They must also allow large complex data structures to be moved with a single instruction, and with all of the individual fields directly available to instruction hierarchies in *Processes* as illustrated in Fig. 4. This provides ease of understanding as well as orders of magnitude improvement in single processor speeds. Experimental results of speed comparisons are described in [3], Chapter 17.

The process language supports hierarchies of rule structures, where looping and complex IF... THEN... ELSE statements are flattened - no nesting. *Waterfall* or *Fall through* code is gone - without GOTOs. These properties simplify design of algorithms while eliminating unintended recursion, leading to substantial increases in understanding and run time speed - on single as well as parallel processors. Global data is eliminated. All data is automatically referenced by pointer.

Note that subscripts are not used in Fig. 3 or Fig. 4. This is because the resource and process shown are part of an instanced module, where instance pointers (TRANSMITTER & RECEIVER) are automatically handled at the module level. These are set when a process within an instanced module is CALLED or SCHEDULED. Moving

RESOURCE NAME: TRANSCEIVER		INSTANCES: TRANSMITTER RECEIVER	
<b>GENERAL PARAMETERS</b>			
1	TRANSMITTER_POWER	REAL	INITIAL_VALUE 100
1	RECEIVER_THRESHOLD	REAL	INITIAL_VALUE 120
<b>RADIO</b>			
1	TRANSCEIVER	STATUS	TRANSMITTING RECEIVING IDLE OFF
1	LOCATION		
2	LATITUDE	REAL	
2	LONGITUDE	REAL	
2	ALTITUDE	REAL	
1	ANTENNA_HEIGHT	REAL	
1	ANTENNA_GAIN	REAL	
<b>RECEIVER_CONNECTIVITY_VECTOR</b>			
1	POWER_AT_RECEIVER		
2	SIGNAL_POWER	REAL	
2	TOTAL_NOISE_POWER	REAL	
1	CONNECTIVITY_MATRIX		
2	PROPAGATION_LOSSES		
3	FREE_SPACE_LOSS	REAL	
3	TERRAIN_LOSS	REAL	
3	FOLIAGE_LOSS	REAL	
3	TOTAL_LOSS	REAL	
2	SIGNAL_TO_NOISE_RATIO	REAL	
2	LINK_DELAY	REAL	
2	LINK	STATUS	GOOD FAIR POOR
<b>TRANSCEIVER_RULES</b>			
1	TRANSCEIVER_PROCESS	RULES	GOOD_RECEPTION CONFLICTING_RECEPTION CONFLICTING_BROADCAST

FIGURE 3. Example of a hierarchically structured state vector (RESOURCE).

instance implementation to the module level substantially enhances understanding of the code.

It is important to note that hardware design separates data memory from instruction memory. This is important since VisiSoft resources are stored in data memory while processes are stored in instruction memory. This provides additional speed advantages when mapping modules to processors, and assigning and managing memory.

Users assign sets of instanced modules to specified processors using the third language - the Control Specification. Assignments are aided by a graphical interface so application experts can use their knowledge of the application to make physical processor assignments. The third language provides many facilities, e.g., easily understood sections for defining files, libraries, graphics, initialization, and multiple runs for simulation and optimization, eliminating complex scripts.

These three languages support design of software spaces that simplify human translation of inherently parallel physical entities into an organization of independent mathematical functions (modules). Design of the resource and process languages are driven by factors akin to those motivating use of *tiling* in parallel versions of C. This minimizes memory management overhead for swapping instructions and paging data. It maximizes the work done on each processor while processes run concurrently with those on the other processors, maximizing the PUE. Fig. 3 and Fig. 4 provide examples of hierarchical resources and processes that help understandability of the design.

When building complex software, human translation is simplified if a language supports obvious representation of physical behavior. Redundancy supports ease of understanding, and the likelihood that information is communicated

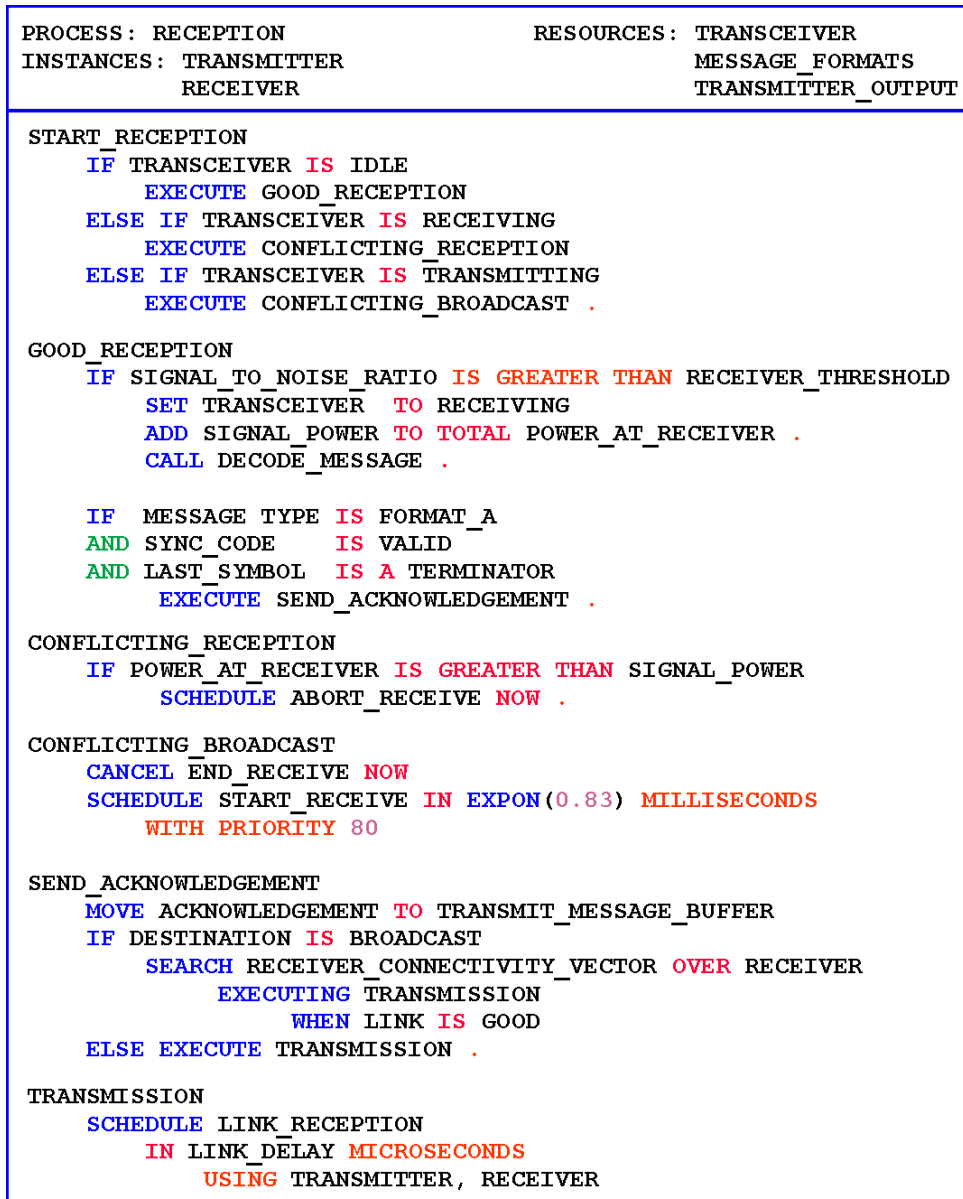


FIGURE 4. Example of a hierarchically structured transformation (Process).

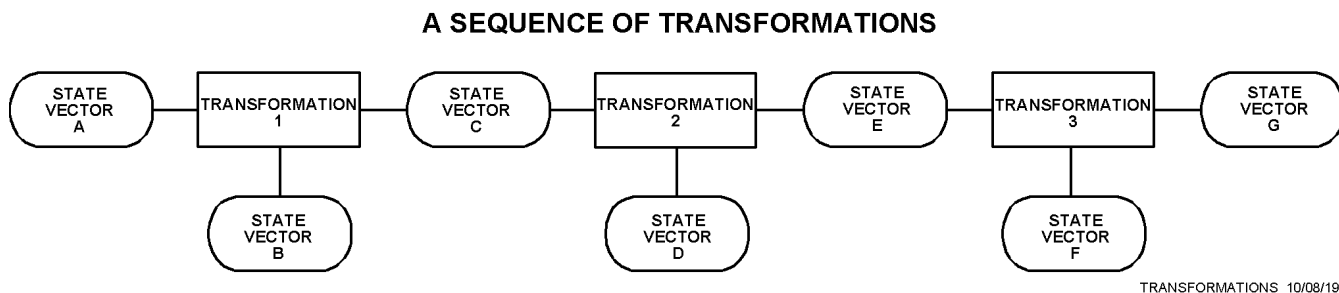
correctly to others, see [13] and [14]. The examples in Fig. 3 and Fig. 4 are taken directly from large simulations of Packet Radio networks. Using hierarchical data structures like those shown simplifies understanding of complex algorithms representing physical systems. As shown below, actual systems may entail more complex resources and processes than those above, but are easily understood by application experts.

## V. SOFTWARE ARCHITECTURE

The Separation Principle supports a precise definition of software architecture as well as engineering drawing visualization. *Resources* are depicted as ovals in architectural drawings as illustrated in Fig. 5. *Processes* containing instructions

that implement transformations are depicted as rectangles. The lines connecting them determine which processes have access to what resources. In this figure, each process has a dedicated resource and shared resources. Transformation 1 has state vector A as input, state vector B for dedicated use, and shares state vector C with transformation 2. Therefore, Transformations 1 and 2 are not *independent*.

As used here, the property of independence ensures that processes running on a parallel processor produce complete and consistent results for a given set of initial conditions. Consider that state vectors C, D, and E have initial values  $C_i$ ,  $D_i$ , and  $E_i$ . When run on a single processor (sequential machine), Transformation 2 will produce the same outputs:  $C_o$ ,  $D_o$ , and  $E_o$  for a given set of inputs every time it runs; i.e., the results



**FIGURE 5.** State vectors and transformations.

will be complete and consistent. If while it is running, one of the resources is changed from the outside, the results may not be complete and consistent. This is because the data being accessed is not consistent relative to Transformation 2.

If Transformations 1 and 2 run concurrently, shared state vector C could be changed by either, rendering the data as recognized by the other as potentially inconsistent. Therefore, in general, they cannot operate concurrently. Similarly, Transformation 2 is directly coupled to Transformation 3 by shared state vector E, is not independent of it, and thus cannot run concurrently with it. However, Transformations 1 and 3 can operate concurrently since they share no state vector directly and are therefore spatially independent. Transformation 2 can operate only when Transformations 1 and 3 are both idle; in that case they are temporally independent. Temporal independence is handled without idling using SCHEDULE statements between processors to schedule processes at the current or a future time, and memory copies as described below. We note that memory is abundant and easily traded for speed.

### A. VISUALIZATION

Fig. 6 illustrates how VisiSoft software architects can decompose a system into modules by grouping resources and processes into an *elementary module*. *Hierarchical modules* are created by grouping modules into higher level modules. Fig. 6 is a library module, PROPAGATION\_PREDICTION, that is sufficiently complex to warrant its own drawing. In general, modules are independent if they share no resources (i.e., they are not connected). Having designed an architecture, developers implement the data structures and rules using the *resource* and *process* languages.

The languages do not support declaration of scope rules. The architecture determines how data is shared, and the corresponding independence of modules. As an option, green arrow lines can be turned on to show CALLing tree control flow to eliminate unintended recursion. Arrows only point down ensuring the proper “chain of command.” Using this CAD approach, resources and processes may be edited directly on the drawing as illustrated in Fig. 7.

The languages are designed to provide for deep hierarchies in both data structures and rule structures to support

complex software spaces and algorithms. These mathematical properties are critical to simplifying the understandability of large complex software systems. For example, the engineering drawings are not flow charts. They define connectivity - explicitly - mathematically.

### B. PARALLELISM, ARCHITECTURE AND DECOMPOSITION

When striving to take advantage of inherent parallelism in a system, one must design an architecture of software modules that maximizes concurrency on a parallel processor. Designing the best set of state spaces is key to solving this problem. This translates into mapping the inherent parallelism of a system into independent modules that maximize run-time speed.

Having defined Generalized State Space as the VisiSoft framework, the mathematical analogy becomes one of selecting the best set of information tensors (Resources) to represent the system attributes. Depending upon how the resources are designed and structured, the rules (Processes) are much easier to build, understand, and modify. This is also determined by the independence properties of the architecture, i.e. the interconnection of resources and processes. This system also eliminates C++ type pointers and global data types. All these unnecessary complexities conflict with simplifying the design of software for parallel processing.

Instead of attempting to describe architectures at the language level, the problem is separated into the natural hierarchy of engineering drawings and language. To follow this concept further, there are specific types of modules and resources, also designated at the drawing level using different colors. These are described below. The simplifications that these visually distinct properties and hierarchies provide become obvious after using them.

### C. TYPES OF MODULES

The module types that make up the layers of a VisiSoft software task follow a design hierarchy described below and shown in Fig. 8. Module types provide different levels of protection with regard to their reuse in different hierarchies. Both elementary and hierarchical modules can reside within each type as well as in a VisiSoft Task.

- Modules - have a blue border. These are the basic building blocks in a task. In the CAD system described here,

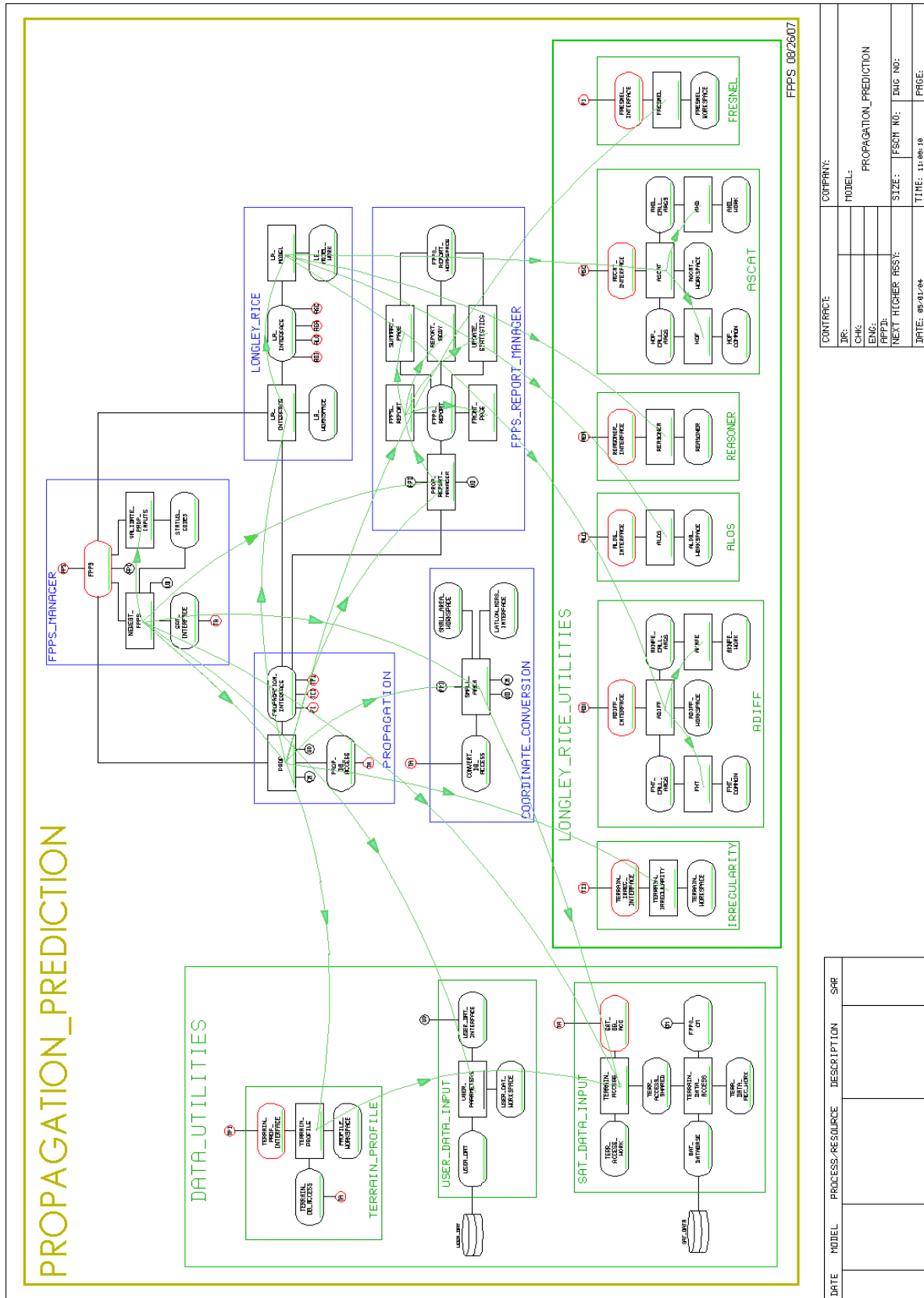


FIGURE 6. Illustration of an engineering drawing with the process CALL tree turned on.



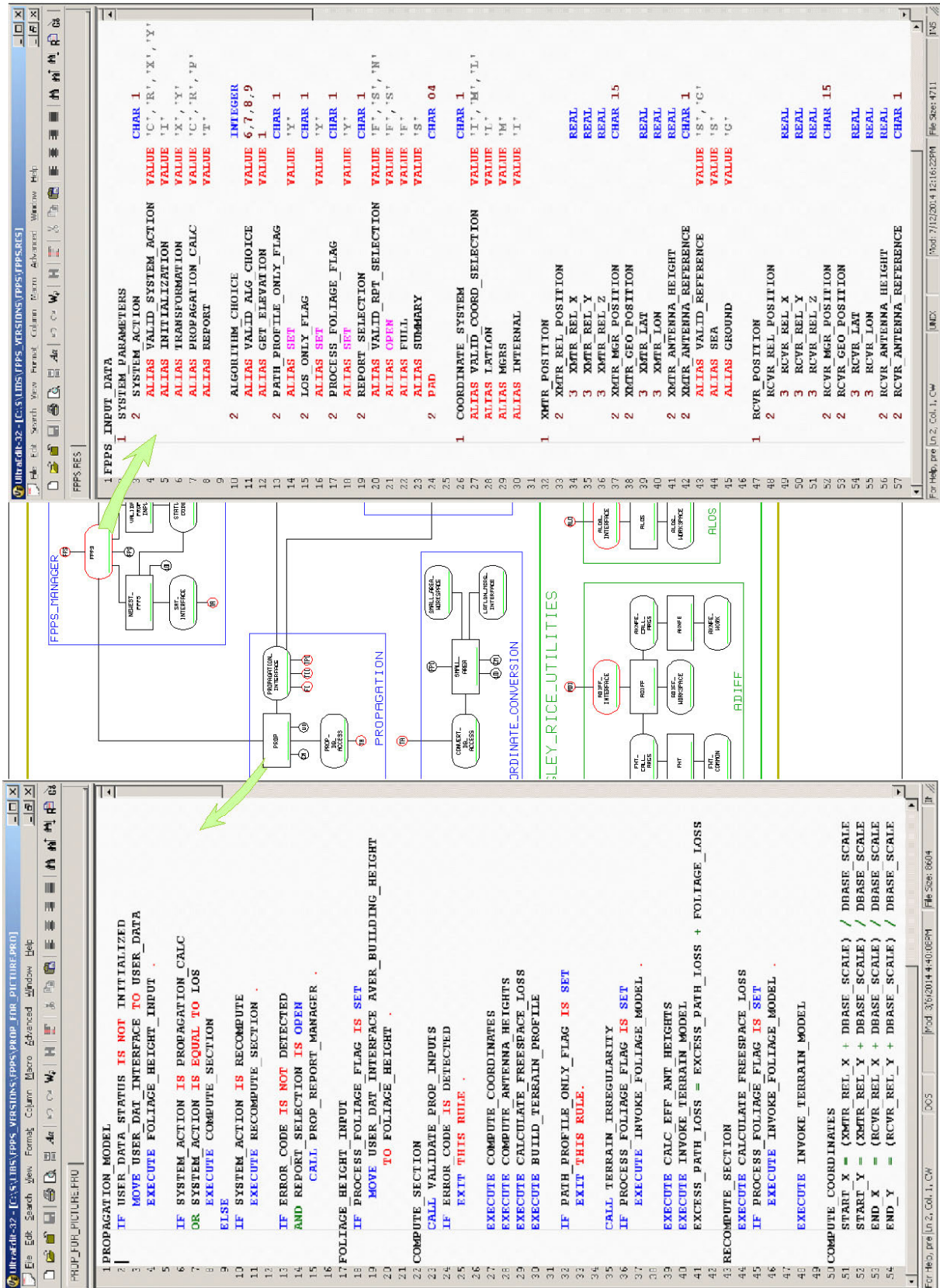


FIGURE 7. Illustration of editing resources and processes on the drawing.

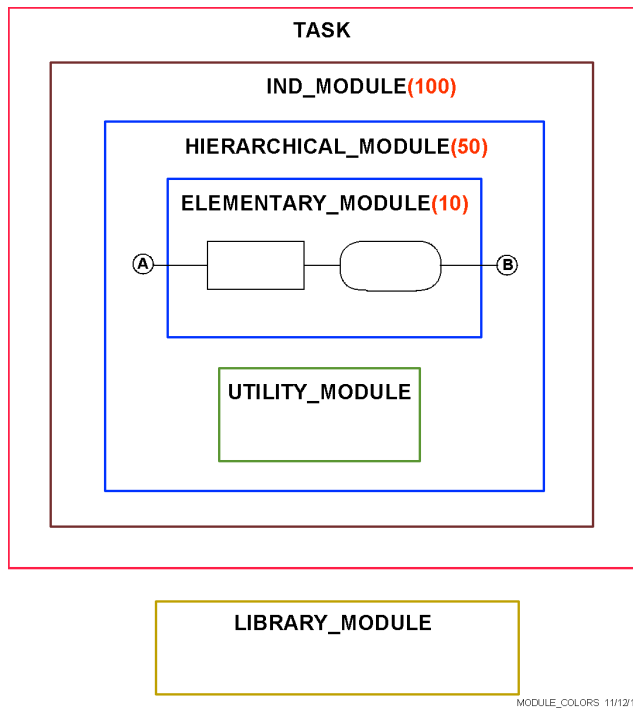


FIGURE 8. VisiSoft hierarchy of modules.

modules may be decomposed hierarchically, i.e., they may contain layers of submodules and sub-submodules, etc.

- Elementary Modules - contain resources and processes.
- Hierarchical Modules - contain elementary modules and hierarchical modules.
- IND (Independent) Modules - have a brown border. IND Modules only share Inter-Processor (IP) Resources externally - and only with other IND Modules. When using parallel processors, IND Modules must be the highest level modules on a processor. IND Modules may reside on the same or different processors.
- Utility Modules - have a green border. These are modules that are reused by processes in the same directory, and can appear in more than one hierarchy in different drawings. They are typically used to manage separate databases or perform utility type functions. The green color distinguishes them for change protection. They can only be changed in their own drawing. If they are changed to accommodate a different requirement, that change must be compatible with all processes that use them. Separate copies automatically reside on each processor that uses them.
- Library Modules - have a gold border. These are highly protected utility modules that can be shared from different directories and computers, being stored as object modules in special object library files. The source only appears in the directory where they are maintained. Library module Processes are called from an application

using their process name, module name, and library name. Since each of these names must be unique within the next level of hierarchy, there can be no duplicate names when linking to library modules in the VisiSoft CAD environment. Separate copies reside on each processor that uses them.

- The functions of a library module may be upgraded while at the same time preserving the original module in the library for prior users. Users can call the new function using the same process name within the same library by using a new module name. The existing CAD system has a large set of libraries that support various applications, including 3D graphics and various global coordinate system transformations. These are shared easily.

#### D. INSTANCED MODULES

Modules and IND Modules can be instanced (number of instances appear in red). This allows assignment of groups of instances to different processors. This facility is supported by the Control Specification language, providing the ability to assign specific sequences of IND Module instances to different processors to maximize PUE.

When building complex software systems on parallel processors, it is important to select the proper module types at the engineering drawing level. Module types are used to create hierarchies, and to differentiate how they are used and accessed. Depending upon the module type and what is required architecturally, modules must share specified resource types as dictated by design for concurrent operations. Equally important is selection of specific types of resources, also designated at the drawing level using different colors as described below.









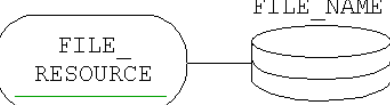
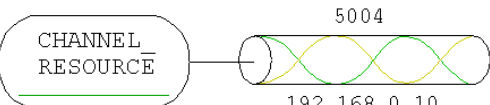
#### E. TYPES OF RESOURCES

A hierarchy of types of data resources is required to support different architectural requirements. It is the architecture - augmented by different resource types - that simplifies the design of complex software systems, particularly those designed to run on parallel processors. Resources also facilitate direct mapping of complex mathematical spaces into well organized data descriptions. Designers can easily create or modify resource types. The hierarchy of different resource types is shown in Table 2.

#### F. SELECTION OF RESOURCE AND MODULE TYPES

To assess parallelism, one must understand the property of module independence. Modules are *spatially independent* (independent of time) if they share no data (resources). From an application standpoint, these modules have no *direct* influence on each other while running. Modules are *temporally independent* if they only influence each other at specified points in time. A typical example is using differential equations to represent dynamic systems. Computer solutions typically assign a  $\Delta T$  time step that is small enough to ensure sufficient accuracy of solutions when compared to actual tests

TABLE 2. VisiSoft resource types.

	<p>Resources hold state data that may be organized in a hierarchical manner. They can be shared by several processes or “dedicated to” a single process in a single task. Connection to either a file or communications channel requires the resource to be dedicated to a single process (described below).</p>
	<p>A resource with a memory template, typically used in a utility or library module. Access to the actual data is provided automatically by a pointer to a resource attached by connector. Shared Alias resources are outlined in red.</p>
	<p>Local Inter-Task resources allow a family of tasks to share data, minimizing OS level memory management. Local Inter-Task resources are used when a task is responsible for “STARTing” another one that shares the same local Inter-Task resource. Local Inter-Task resources are outlined in green.</p>
	<p>Global Inter-Task resources are similar to Local Inter-Task resources, being used to allow two tasks to share data when they are RUN <i>independently</i> rather than when one task STARTs the other. Global Inter-Task resources are used for SYSTEM level EVENTS. They are outlined in purple.</p>
	<p>Inter-Processor (IP) resources are used to share data between Independent (IND) Modules running on different processors in the same task on a parallel processor. IP resources are outlined in blue.</p>
	<p>IP Access resources are used to ACCESS data from IP Resources in different IND modules on different processors in the same task running on a parallel processor. IP Access resources are outlined in gold.</p>
	<p>PANEL resources support graphical panel interfaces for input and output of information, which can include icons, check-boxes, scrolling lists, etc. The contents of a PANEL resource are created and modified using the Panel Library Manager (PLM). PANEL resources may be viewed, but not changed with a text editor. PANEL resources are labeled using Red text</p>
	<p>HLA resources support the use of High Level Architecture† for communications between disparate tasks in a multi-task environment. This resource and an associated HLA event handler enable easy use of HLA from within a VSE task. This resource type is used as an HLA Interface.</p>
	<p>A resource describing one or more records on the file to which it is attached. The FILE_NAME identifies the name of the file to be accessed.</p>
	<p>A resource describing the packets and TCP/IP channel to which it is attached. The number on top of the channel icon is the PORT number and that underneath is the SERVER ID.</p>

† Used in military simulations running together in physically separate network connected sites.

( $\Delta T$  may vary for nonlinear systems). This implies that equations are solved at successive time steps, ( $T + \Delta T$ ), so changes in one part of a system cause sufficiently accurate effects in

another part. Information on changes are exchanged at the beginning or ending of each time step, affecting calculations within the desired time step.

More generally, modules are *temporally independent* if they share information on a synchronized basis. This requires that only the module containing an IP Resource can change the information in that resource. Other modules can copy that IP Resource information into their own IP Access resource making the information available for follow-on calculations. Since this CAD system supports discrete events, where a process can SCHEDULE another on a different processor, system level states may be defined to ensure synchronization among all modules. For example, an IND Module changing an IP Resource can set a system state to indicate it has been updated. A module reading that resource can set a different system state showing that it has read the latest copy. Just as the real physical systems they represent, no module need stop to wait for the others, thus maximizing PUE. An example is people observing those around them as they drive.

Fig. 9 provides a stand-back illustration of the ability to visualize the level of complexity of the GLOBAL\_PLANNER simulation. This application requires parallel processors to meet speed requirements in support of multi-run planning. It uses many large library modules not shown on the drawing. It also has many different IND Module types (brown borders) and instances (red numbers). All these modules are connected using special types of resources. It can easily house additional modules within each of the platform modules as well as more platforms.

In addition, platforms must communicate while moving over mountainous terrain. As platforms move they may lose connectivity. This must be tracked to determine which ones can communicate. This requires an electromagnetic wave model (the library model in Fig. 6) to determine platform connectivity based on their position relative to terrain. With 153 platforms using the detailed propagation model, it runs 50 - 12 hour scenarios in less than 100 seconds to perform parametric analysis on an 18 processor PC size machine. The CAD system also contains graphical facilities described below that support the spread of instanced IND Modules evenly across processors to maximize the resulting PUE measure.

## VI. TIME SYNCHRONIZATION AND SPEED

Given the temporal independence condition, changes in one IND Module that affect another on a different processor will be reflected with sufficient accuracy as long as the effects are resolved within a user specified  $\Delta T$  time period. This provides the ability to synchronize information exchanges. It implies that the times when: (1) a sending process updates its IP resource; and, (2) the receiving process copies it into its IP Access resource - both fall within an allowed  $\Delta T$ . Synchronization is accomplished when the shared IP Resource information is exchanged within successive  $\Delta T$  time periods.

In typical applications, speed is most important, determining whether application run-time constraints are met. When applications impose a speed constraint, the optimal design problem is to minimize the number of processors required to meet that constraint. When time constraints can be met using

a smaller number of processors, speed will rise exponentially just due to the smaller footprint. This can dramatically reduce power requirements as well as floor space.

PUE depends upon the average amount of useful work done on each processor within a maximum ( $\Delta T_{max}$ ) window, [3]. When the amount of work done on each processor is highly varied, many processors will have large idle times, and the resulting PUE will be low. When the useful times spent on each processor are all close in size, the average idle time in a  $\Delta T_{max}$  window is typically much smaller rendering the PUE much higher. Fig. 10 provides a stand-back illustration of the ability to visualize measures of the PUE in three successive time steps for the GLOBAL\_PLANNER simulation in Fig. 9. Each line represents one of 14 processors, and each colored box on a line represents an IND Module. The IND Module processor assignments can be changed to maximize the PUE.

### A. PROCESSOR & Memory Assignment

Decisions to place modules on processors is a major factor in run-time speed. In the approach used here, processor placement can provide an order of magnitude increase. Here we must differentiate between stationary and non-stationary (stochastic) applications. In stationary cases, specific module instances can remain on the same processor throughout a run. This is true in applications where, when modeling the movement of particles from one cell to another, one is only interested in their combined nonstationary properties, e.g., the number of particles and total mass in a cell. In these cases, the individual particles are not labeled and their combined properties are stationary relative to cell placement.

In the GLOBAL\_PLANNER example used above, individual platforms have specific IDs, and are tracked as platforms move from cell to cell on different processors. However, in this simulation, sufficient copies of platform models exist in each cell that they may occupy. So the properties of a platform are copied to an unused model instance in a new cell to which they move, including its name and pointer when a cell crossing occurs. Memory assignments never change.

Even in non-stationary cases, once the memory for an IND Module is assigned to a processor, the memory need not be moved. In typical cases, sufficient memory exists at the L3 Cache level so that L3 Memory Management is virtually eliminated. Depending upon the size of the IND Modules and how they are designed, Cache management of L1 and L2 is minimized. Only the designer of such a system knows how to distribute the IND Modules at the L3 level. If the OS gets involved, huge amounts of time can be wasted moving memory that otherwise can remain stationary. The OS does not have the necessary information for this decision.

When looking at the overall architecture of large real-time control systems, and the speeds that can be achieved as memory sizes are increased at very low costs, it is clear that memory is a key to speed. Upon reviewing the use of stacks, cache coherency and other facilities, and the chip space required to support them, one must consider the potential increase of

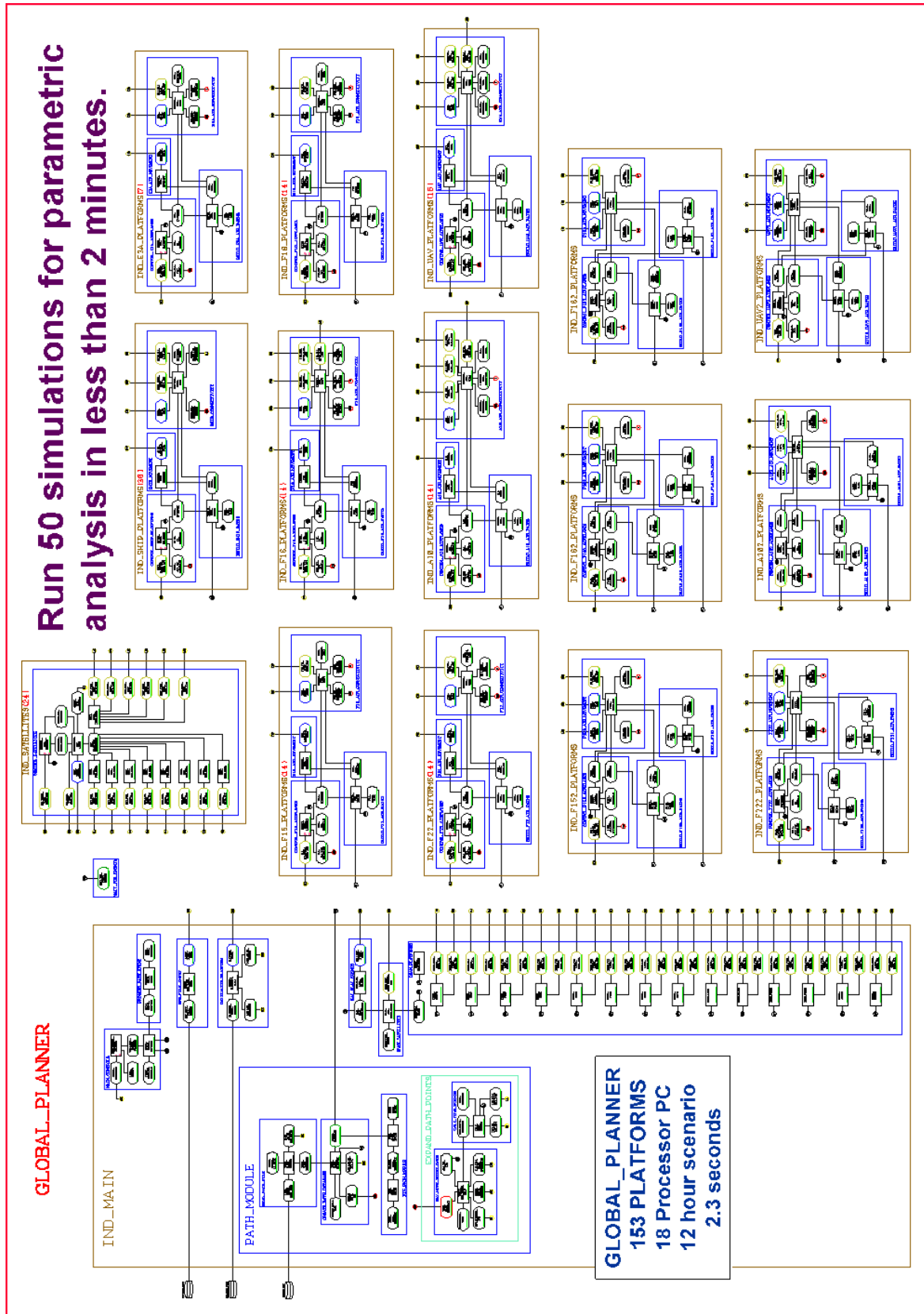


FIGURE 9. Standback illustration of an engineering drawing of the GLOBAL\_PLANNER-A parallel processor simulation.

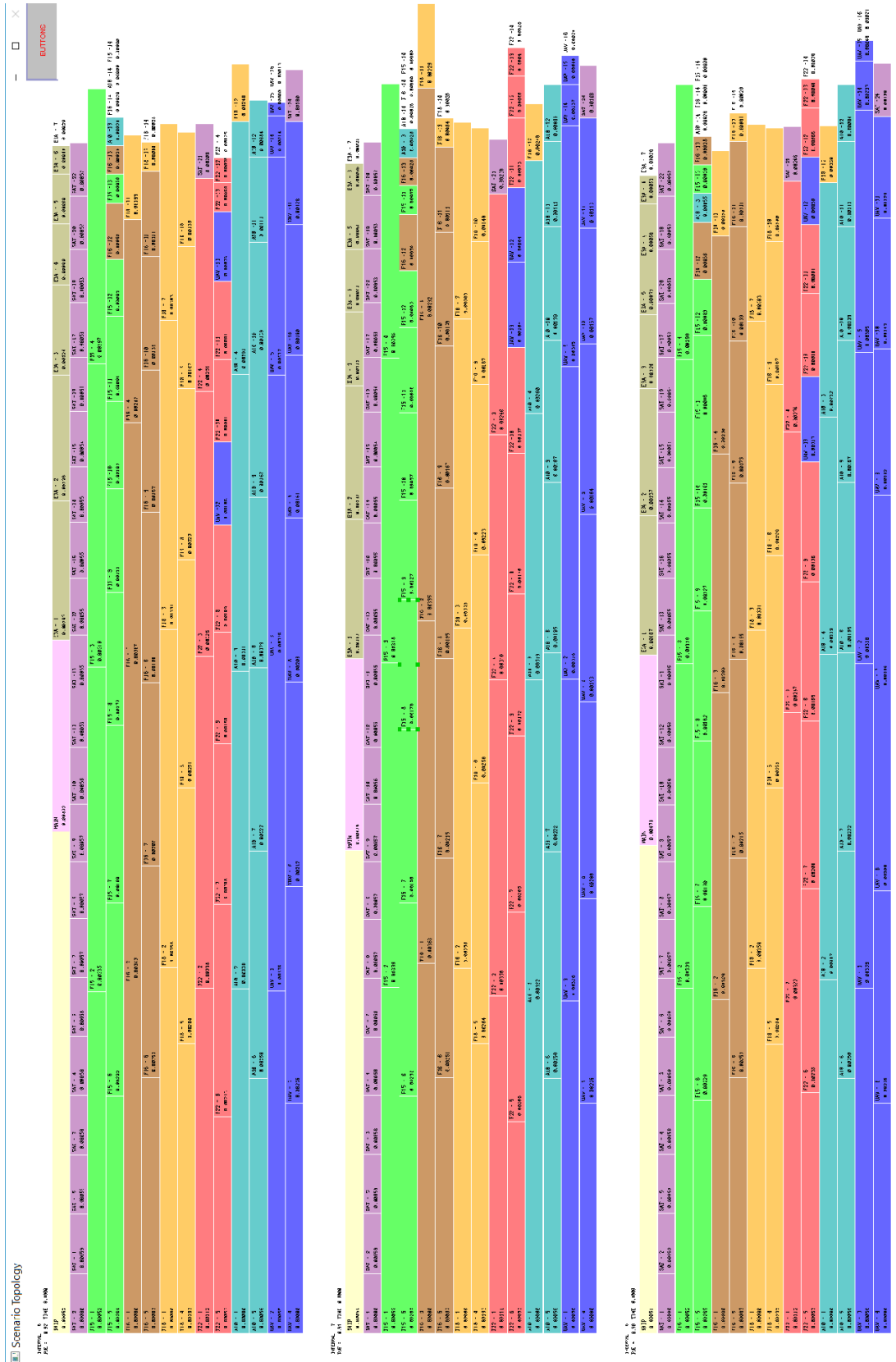


FIGURE 10. Snapshot of PUE for 3 Delta Ts, using 14 processors (one on each line) containing colored boxes representing IND modules.

memory that may be had by eliminating these unnecessary facilities.

## VII. SUPPORTING NONSTATIONARY & NONLINEAR CASES

As described above, connectivity between specific GLOBAL\_PLANNER platforms must be tracked, so platform IDs must be identified when changing cells. This requires a nonstationary solution. Weather molecule IDs are not tracked. Only changes in mass within a cell need be tracked, a stationary problem. This implies that the memory locations used need not change.

As described above, the OS *memory manager* knows little about the physical layout of an application and cannot map it so it does not move. This is the problem with cache coherency. Instead, using a little more memory, the designer can move a platform to a different cell by allocating sufficient memory in each cell to support the maximum number of platforms. This avoids an OS moving platforms and cells to processors that are likely to be far from the adjacent processors.

The upper left hand corner, ①, of Fig. 11 illustrates a breakout of the space in which platforms move. This space is used to calculate electromagnetic path loss between platforms based on their positions relative to terrain. This space is translated into cells for computational purposes. We note that this set of spaces is similar to that required for weather prediction as noted in the figure to the right of that containing the terrain, ②.

Weather models are currently inaccurate due to the parallel processing speed required to model nonlinear spaces. The linear approach, called parameterization, [4], uses cell sizes above 1Km that are much too large to support mountainous and other regions. Those familiar with this problem say it would take 6 to 8 orders of magnitude increase in speed to use the nonlinear models and 100 meter cell sizes needed to achieve the required accuracy. Many factors contribute to solving this problem. A critical factor is the use of heterogeneous spaces, e.g., using different cell sizes based on steepness of terrain. This requires the use of deep spatial hierarchies such as those described above. VisiSoft can meet the nonlinear weather model speed requirements.

## VIII. HARDWARE ARCHITECTURE

What becomes obvious when designing software architectures for parallel processors is the need to understand the complex spaces underlying an application so that their independence properties are easily represented in visualizations of the architectures. Now known as the Application Space Architecture (ASA), [2], it has been used for defining a new approach to parallel processor hardware design using directly interconnected chips, ④, and boxes, ⑤, shown in Fig. 11. Every processor, chip and box has Direct Memory Access (DMA) channel connections with its 26 adjacent neighbors. Given that some accesses may be slightly slower than others, this still eliminates the huge delays encountered when going through communication channels

in a server. Servers are not parallel processors; and parallel processors need no I/O facilities - they can be tied to a server for external access.

Being equivalent to the ISA for single processors, the ASA represents a huge improvement in parallel processor speed measures. The software CAD system used here to describe the GLOBAL\_PLANNER contains graphical facilities that support the mapping of multiple IND Module instances (e.g., groups of cells) evenly across processors to maximize the resulting Processor Utilization Efficiency (PUE) measure. It also provides additional facilities to ensure that the mapping maintains close proximity to adjacent processors, chips and boxes shown in Fig. 11.

The GLOBAL\_PLANNER simulation, with 153 platforms using detailed propagation models, runs 50 - 12 hour scenarios in less than 2 minutes to perform parametric analysis on an 18 processor PC size machine, achieving PUE measures of 95%.

## IX. HARDWARE DESIGN CHANGES

Based on extensive testing, existing chip, board, and box designs can be used “as is” to start. The current approach uses a single PC box with 28 processor chips and up to four chips on a single board – for a total of 112 processors. Initial tests indicate that this configuration can beat a typical large rack containing thousands of processors and using over 1000 times the energy.

### A. CHIP DESIGN

The initial chip effort is to analyze how to support the new board and box design described below. This must provide the ability to interconnect chips using DMA channels between shared memory on small boards in a single box (8” X 8”), and between different boxes (6” apart). It is expected that this may be done in a manner similar to that between multiple chips on a board and may not require any chip changes – only board and box circuit changes.

A major new chip design must remove unneeded instructions and logic that supports the following unnecessary facilities:

- I/O device facilities - Perform direct data transfers to a server box using DMA Channels
- Network Communications, e.g., bit modulation and routing - Use DMA Channels to adjacent boxes
- Cache Coherency - Application experts map memory directly
- Thread Synchronization - Application experts SCHEDULE processes
- Stack Facilities - Only needed for recursion which is unnecessary (can be done in software)

Automatic use of stacks is built into current popular language compilers to compensate for unintended recursion, a topic not covered in the literature. Languages not using explicit CALL statements are prone to unintended recursion, a difficult problem to uncover in complex systems. Good language translators ensure against unintended recursion using

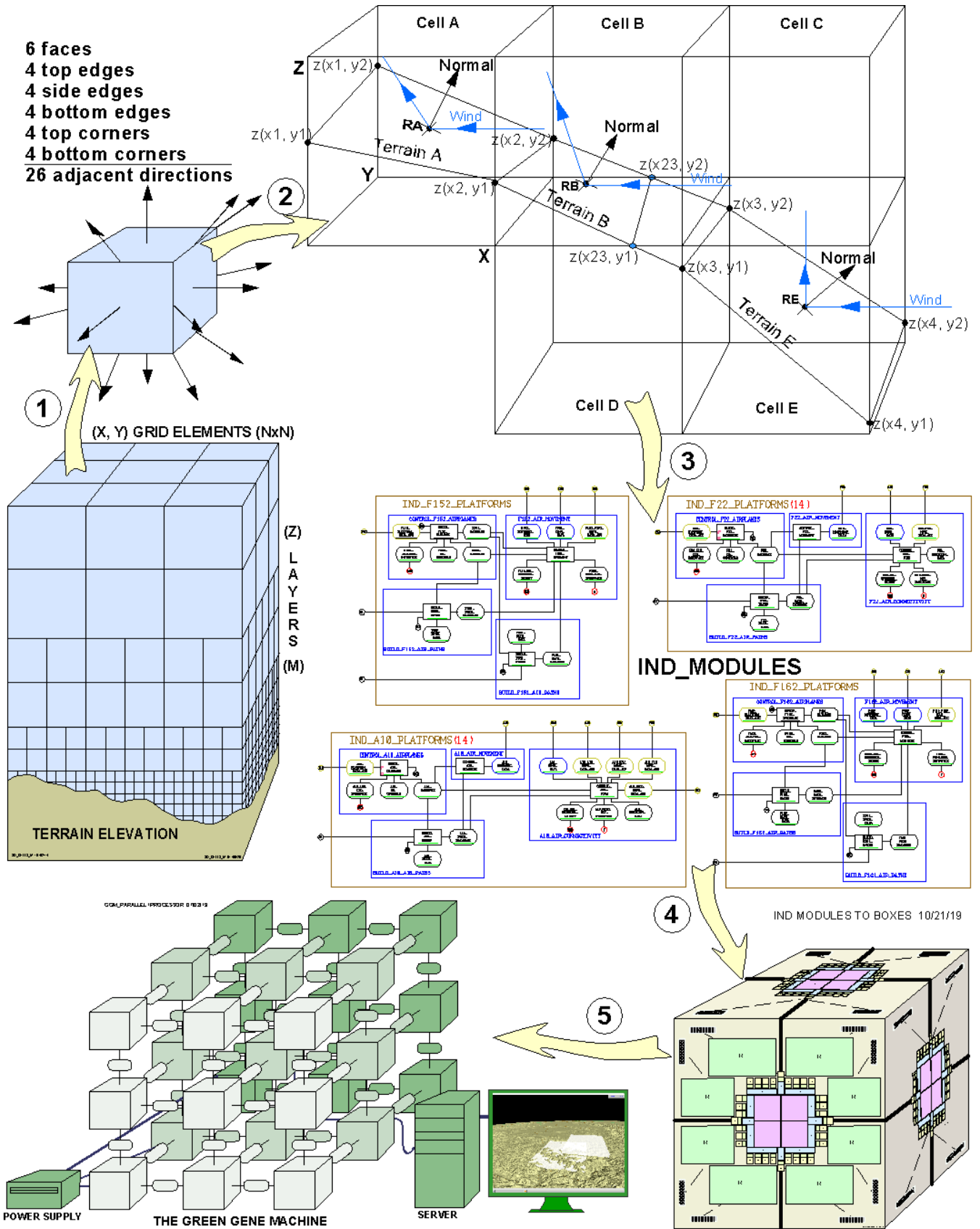


FIGURE 11. The ASA - mapping application cells into IND modules, processors, chips and boxes.



one-in one-out control structures as prescribed by Mills [7], and additional checks to ensure control sequences follow a proper “chain of command.” Unused chip space due to removal of automatic stack facilities and other bullets above can be used to expand cache memory, a major factor in speed.

### B. BOARD DESIGN

New boards must be designed to fit within 8” cubic boxes. Chips, memory, wires and other electronics will be on the inside surface of the boards. Boards will be hinged and attached with removable connectors where needed for both electrical and mechanical access. Connections between boards must be designed to support (possibly slower) DMA channels.

### C. BOX DESIGN

New boxes must be designed with connectors on faces allowing connections in all 26 directions. Connections between boxes must be designed to support (possibly slower) DMA channels.

### X. CONCLUSION

Hennessy and Patterson spelled out the need to rethink the ISA currently used for hardware designs and create a new approach for parallel processor designs. As stated by many top hardware designers, e.g., Chuck Moore, Justin Ratner, Craig Mundie, Gordon Bell, etc., a new approach is needed to build software that takes advantage of parallel processors, see SECTION XI below.

To solve these problems, one must start with an understanding of the underlying differences in applications that require parallel processing speeds to meet their time constraints versus those that can run on a single processor or cluster. Next, one must understand how the common properties of parallel applications lead to a new approach to the design of both software and hardware that takes maximum advantage of parallel processing. The first principle is that applications must have a sufficient number of independent elements that can run in parallel. The second principle is understandability of the architectural solution so that it can be designed and mapped effectively onto parallel processor hardware.

Because of the level of complexity, one must have the tools to separate the independent application elements into independent software modules. This is essential to designing the complex spaces required to represent parallel processor applications. It requires the ability to represent complex spatial hierarchies in a manner that is easily understood by application experts. It also requires the ability to easily understand and separate the complex algorithms that use the separate spatial hierarchies.

This leads to the Separation Principle, separating data from instructions at the language level. Once this is done, visualization of complex application architectures can be implemented using engineering drawings of software. Independent modules can be defined mathematically using the drawings. Independent modules can be mapped onto parallel processors

- based on their relative adjacent spatial positions - to maximize PUE and run-time speed.

The ASA follows from the optimal placement of adjacent modules on adjacent processors, chips and boxes to support direct memory access between the hardware elements that comprise the overall parallel processor. A simplified representation of this sequence is illustrated in Fig. 11. Actual hardware architectures must be based on designs that implement the ASA, particularly the use of DMA channels between processors, chips, boards and boxes that are in the 26 adjacent directions comprising the environment of each.

The approach and facilities described here have evolved and been tested on over 100 projects since 1982. Delivered systems include real-time planning, control, and simulation to support complex military and commercial operations and equipment. Speed comparisons with current software approaches have exceeded two orders of magnitude on single processors. Single box comparisons have exceeded four to six orders of magnitude on parallel processors. Just considering the reduction in footprint provides support for the increases in speed. Existing tests are easily repeated to understand and prove the theory.

### XI. ARTICLES BY INDUSTRY LEADERS ON PARALLEL PROCESSING

The problem of using parallel processors was addressed in the 1960s. Mathematical problems using vectors were approached using special hardware designs. Many efforts using parallel processors still continue down this path, with hardware designs making up for the lack of a software environment to deal with the underlying problem. Even when handling special problems, processor utilization efficiency is typically down around 2-10%.

The current state of affairs is best expressed in quotes from published articles below and references [15] through [27].

#### A. MULTI-CORES, SOFTWARE’S GORDIAN KNOT AND THE ALEXANDRIAN SOLUTION - SEP 13, 2007, SEE [29]

*To fully utilize the hardware parallelism inherent in embedded multi-core designs, they say, will require a shift to a more implicitly parallel programming language and methodology. However, many, including researchers at Microsoft, believe that it will take at least ten years for the industry to shift to a new parallel programming framework.*

#### B. EE TIMES: INTEL CTO PRESSES SOFTWARE DEVELOPERS TO KEEP PACE – JAN 17, 2008, SEE [30]

*Software development and delivery have failed to keep pace with advances in computer hardware, according to Intel Corp.’s CTO. — As hardware technology approaches the terascale level on the desktop, software has fallen further behind. — One result has been a lack of parallel programming applications to leverage dual-and multi-core processing technology. Intel is looking for “new languages for programming in parallel,” (Justin) Rattner told the India Semiconductor Association.*

### C. EE TIMES: INDUSTRY SEEKS A MODEL FOR NEXT-GEN MULTI-CORE CPUS - FEB 14, 2008, SEE [31]

“The industry is in a little bit of a panic about how to program multi-core processors, especially heterogeneous ones,” said Chuck Moore, a senior fellow at Advanced Micro Devices trying to rally support for work in the area. “To make effective use of multi-core hardware today you need a PhD in computer science. That can’t continue if we want to enable heterogeneous CPUs,” he said. The challenge in the parallel world is finding a dynamic and flexible approach to schedule parallel tasks from these modules across available hardware in complex heterogeneous multi-core CPUs.

### D. EE TIMES: MULTICORE PUTS SCREWS TO PARALLEL-PROGRAMMING MODELS - FEB 15, 2008, SEE [32]

Leaders in mainstream computing are intensifying efforts to find a parallel-programming model to feed the multicore processors already on chip makers’ drawing boards. — Developers need to expand the current software stack in fundamental ways to handle a coming crop of processors that use a variety of cores, accelerators and memory types, according to the company. — Both AMD and Intel have said they will ship processors using a mix of X86 and graphics cores as early as next year, with core counts quickly rising to eight or more per chip. But software developers are still stuck with a mainly serial programming model that cannot easily take advantage of the new hardware. — Thus, there’s little doubt the computer industry needs a new parallel-programming model to support these multicore processors. But just what that model will be, and when and how it will arrive, are still up in the air.

### E. REUTERS: MICROSOFT’S TOP VISIONARY SEES A PARALLEL WORLD - MAR 13, 2008, SEE [33]

Craig Mundie, Microsoft Corp’s Chief Research and Strategy Officer, is sure he has a good handle on where technology is going. When is another story. — The computer industry has taken its first steps toward parallel computing in recent years by using “multi-core” chips, but Mundie said this is the “tip of the iceberg.” — To maximize computing horsepower, software makers will need to change how software programmers work. Only a handful of programmers in the world know how to write software code to divide computing tasks into chunks that can be processed at the same time instead of a traditional, linear, one-job-at-a-time approach. — A new programming language would be required and could affect how almost every piece of software is written. — “This problem will be hard,” admitted Mundie, who worked on parallel computing as the head of supercomputer company Alliant Computer Systems before joining Microsoft. “This challenge looms large over the next 5 to 10 years.”

## REFERENCES

- [1] M. Martinez, “Nobel prize for computing, based on Hennessy-Patterson turing lecture,” presented at the Int. Symp. Comput. Architecture, Los Angeles, CA, USA, May 2018.
- [2] H. Ledgard, “The cave ASA versus the Von Neumann ISA,” Univ. Toledo, Toledo, OH, USA, Mar. 2019, [Online]. Available: [http://www.predictsys.com/LEDGARD\\_ASA.pdf](http://www.predictsys.com/LEDGARD_ASA.pdf)
- [3] W. Cave, “Software engineering for parallel processors,” Predict. Syst., Spring Lake, NJ, USA, Jul. 2019. [Online]. Available: <http://www.predictsys.com/SoftwareEngineeringForParallelProcessors.pdf>
- [4] D. Stensrud, *Parameterization Schemes*. New York, NY, USA: Cambridge Univ. Press, 2007.
- [5] R. E. Kalman, “A new approach to linear filtering and prediction problems,” *J. Basic Eng.*, vol. 82, no. 1, pp. 35–45, Mar. 1960.
- [6] P. van der Linden, *Expert C Programming: Deep C Secrets*. Englewood Cliffs, NJ, USA: Prentice-Hall, 1994.
- [7] H. D. Mills, “Mathematical foundations of structured programming,” IBM Federal Syst. Division, New York, NY, USA, Tech. Rep. FSC 72-6012, 1972.
- [8] G. Dyson, *Turing’s Cathedral*. New York, NY, USA: Pantheon Books, 2012.
- [9] Y. Kambayashi and H. Ledgard, “The separation principle: A programming paradigm,” *IEEE Softw.*, vol. 21, no. 2, pp. 78–87, Mar. 2004.
- [10] Prediction Systems, “Visual software development for parallel machines,” US Army CECOM Contract, Spring Lake, NJ, USA, Final Rep. DAAB07-97-C-H501, Mar. 1997.
- [11] Prediction Systems, “Multi-computer version of GSS,” DARPA BAA CONSORTIUM, Contract MHPCC, Spring Lake, NJ, USA, Final Rep., Sep. 1998.
- [12] Prediction Systems, “High Efficiency, Scalable, Parallel Processing,” DARPA Contract, Spring Lake, NJ, USA, Final Rep. SF022-035, Jun. 2003.
- [13] C. E. Shannon, “A mathematical theory of communication,” *Bell Syst. Tech. J.*, vol. 27, pp. 379–623, Jul./Oct. 1948.
- [14] Prediction Systems. *A Briefing on VisiSoft—A CAD System for Building Complex Software Systems*. Accessed: Sep. 19, 2019. [Online]. Available: [http://www.predictsys.com/PSI\\_CAD\\_SHOW.pptm](http://www.predictsys.com/PSI_CAD_SHOW.pptm)
- [15] D. H. Bailey, “Twelve ways to fool the masses when giving performance results on parallel computers,” *Supercomput. Rev.*, pp. 54–55, Aug. 1991.
- [16] A. Barr, “Interview by Stephen Cass, Why do good engineers write bad software? Adam Barr has some answers,” *IEEE Spectr.*, Jan. 2019. [Online]. Available: <https://spectrum.ieee.org/geek-life/reviews/when-good-engineers-write-bad-software>
- [17] E. Ogheneovo, “Software dysfunction: Why do software fail?” *J. Comput. Commun.*, vol. 2, pp. 25–35, Apr. 2014. [Online]. Available: <https://www.scirp.org/journal/paperinformation.aspx?paperid=45351>, doi: 10.4236/jcc.2014.26004.
- [18] G. P. Armour, “Software: Hard data,” *Commun. ACM*, vol. 49, no. 9, pp. 15–17, Sep. 2006.
- [19] M. A. Cusumano, “What road ahead for microsoft and windows?” *Commun. ACM*, vol. 49, no. 7, p. 21, Jul. 2006.
- [20] D. Anselmo and H. Ledgard, “Measuring productivity in the software industry,” *Commun. ACM*, vol. 46, no. 11, pp. 121–125, Nov. 2003.
- [21] D. Anselmo. (May 2004). Why Hasn’t Software Productivity Improved? presented at Software Summit, Washington, DC, USA. [Online]. Available: [http://www.predictsys.com/Software\\_Productivity\\_Anselmo.pdf](http://www.predictsys.com/Software_Productivity_Anselmo.pdf)
- [22] H. J. Poore, “A tale of three disciplines and a revolution,” *Computer*, vol. 37, no. 1, pp. 30–36, Jan. 2004.
- [23] R. Groth, “Is the software industry’s productivity declining?” *IEEE Softw.*, vol. 21, no. 6, pp. 92–94, Nov. 2004.
- [24] W. C. Cave and R. E. Wassmer, “The software survivors,” *Vis. Softw. Int.*, Spring Lake, NJ, USA, May 2014. [Online]. Available: <http://www.predictsys.com/SoftwareSurvivors.pdf>
- [25] R. Camford, “Software engineering?” *IEEE Spectr.*, pp. 62–65, Jan. 1995.
- [26] *Chaos, Charting the Seas of Information Technology*, Standish Group Int., Dennis, MA, USA, 1995.
- [27] *Battling Google, Microsoft Changes How it Builds Software*, Wall Street J., New York, NY, USA, Sep. 2005.
- [28] *Resistance References*. Accessed: Sep. 27, 2019. [Online]. Available: [https://www.predictsys.com/RESISTANCE\\_SP.pdf](https://www.predictsys.com/RESISTANCE_SP.pdf)
- [29] B. Cole. (Sep. 2007). Multi-Cores, Software’s Gordian Knot and the Alexandrian Solution. EE Times. [Online]. Available: <https://www.embedded.com/design/mcus-rocessors-and-socs/4026126/Multi-cores-software-s-Gordian-Knot-and-the-Alexandrian-Solution>
- [30] K. Krishnadas. (Jan. 17, 2008). Intel CTO Presses Software Developers to Keep Pace. EE Times. [Online]. Available: [https://www.eetimes.com/document.asp?doc\\_id=1167793](https://www.eetimes.com/document.asp?doc_id=1167793)

- [31] R. Merritt. (Feb. 13, 2008). Industry Seeks a Model for Next-Gen Multicore CPUs. EE Times. [Online]. Available: <https://www.eetimes.com/showArticle.jhtml?articleID=206503988>
- [32] R. Merritt. (Feb. 15, 2008). Multicore Puts Screws to Parallel Programming Models. EE Times. [Online]. Available: <https://www.embedded.com/design/mcus-processors-and-socs/4023225/Multicore-puts-screws-to-parallel-programming-models>
- [33] D. Wakabayashi. (Mar. 13, 2008). Microsoft's Top Visionary Sees a Parallel World. Reuters. [Online]. Available: <https://www.reuters.com/article/us-microsoft-mundie/microsofts-top-visionary-sees-a-parallel-world-idUKN1222563020080313>



**WILLIAM C. CAVE** received the B.S. degree in electrical engineering (computer option) from Pennsylvania State University, PA, USA, in 1960, the M.S. degree in electrical engineering/computer science from New York University, NY, USA, in 1963, and the Ph.D. degree in electrical engineering Fellowship from the Polytechnic Institute of Brooklyn, NY, USA, in 1965. He has completed PG courses in electrical engineering for optimal and stochastic control theory at the Stevens Institute of Technology, Hoboken, NJ, USA, from 1967 to 1968.

He worked on one of the first digital computers - PENNSTAC, Penn State, from 1958 to 1960. He worked on the first transistorized computer - BASICPAC at Philco, from 1960 to 1962. He was the Project Leader of the U.S. Army MINIPAC Computer, one of first transistor computers, from 1962 to 1965. He was the Chairman and the CEO of Optimal Systems Research, Inc., from 1967 to 1974. He was the U.S. Representative for NATO Panel XIII, Data Communications, Brussels, Belgium, from 1976 to 1978. He was the Chairman of the U.S. DoD Panel on Software System Development, from 1976 to 1978. He has been the Chairman and the CEO of Prediction Systems, Inc., Spring Lake, NJ, USA, since 1974. He has also been the Chairman and the CEO of Visual Software International, Inc., Spring Lake, since 2004. He has been the author of numerous articles and books for professional societies and publishers, including the book, *Software Lifecycle Management: The Incremental Approach* (Macmillan, 1984). He is a member of IEEE - Eta Kappa Nu.



**ROBERT E. WASSMER** received the B.E. degree in electrical engineering from Villanova University, PA, USA, in 1968. He has completed graduate courses in electrical engineering at the Stevens Institute of Technology, Hoboken, NJ, USA, from 1968 to 1969.

He was an Engineer with the U.S. Army Electronics Command, from 1968 to 1969. He was with the U.S. Air Force - Reserves from 1968 to 1974 receiving an Honorable Discharge Electronics. He was the Vice President of Optimal Systems Research, Inc., from 1969 to 1973. He was an Assistant Vice President with the Distributed Processing Division, Continental Corporation, from 1973 to 1982. He has been the Executive Vice President of Prediction Systems Inc., Spring Lake, NJ, USA, since 1982. He is a member of IEEE - Eta Kappa Nu.



**HENRY F. LEDGARD** received the B.S. degree (*magna cum laude*) in electrical engineering from Tufts University, MA, USA, in 1964, and the M.S. and Ph.D. degrees in electrical engineering from the Massachusetts Institute of Technology, MA, USA, in 1969. The author's thesis was Formal Specification of Programming Languages.

He is a retired Professor Emeritus of electrical engineering and computer science with The University of Toledo. He has been a Visiting Fellow with Oxford. He was a Professor with Johns Hopkins and the University of Massachusetts, spent two years in Paris on the Honeywell Design Team for the U.S. DoD Common Language Effort. He has consulted for large corporations on programming languages.

Dr. Ledgard has been the author and an Editor for numerous articles and books on software for professional societies and publishers.



**ALAN B. SALISBURY** received the B.S. degree from the U.S. Military Academy, West Point, NY, USA, in 1958, and the M.S. and Ph.D. degrees in electrical engineering/computer science from Stanford University, Stanford, CA, USA, in 1964 and 1973, respectively.

He retired with the rank Major General from the United States Army, in 1987. He was the President of CONTEL Technology Center, from 1987 to 1991, the Executive Vice President of Microelectronics and Computer Technology Corporation, from 1991 to 1993, the Chairman of the U.S. subsidiary of Learning Tree International, from 1998 to 1999, on the Board of Visitors of Software Engineering Institute, CMU, from 1990 to 2002, on the Board of Directors of Sybase, Inc., from 1993 to 2010, and the President of the Center for National Software Studies, from 1999 to 2005. He has been the Chief Executive Officer of A. B. Salisbury & Co., VA, USA, since 1999. He is the author of *Microprogrammable Computer Architectures* (Elsevier, 1976). This was the first book in the *Computer Design and Architecture Series*.



**KENNETH T. IRVINE** (Member, IEEE) received the B.E. degree in electrical engineering/computer science, the M.S. degree in computer science, and the M.E. degree in electrical engineering from the Stevens Institute of Technology, Hoboken, NJ, USA, in 1985, 1987, and 1990, respectively.

From 1985 to 2012 and 2016 to the present, he was with Prediction Systems, Inc., Spring Lake, NJ, USA, where he is currently the Director of Systems Engineering. From 2013 to 2016, he served as the Project Manager for developing Condition Based Maintenance (CBM+) architectures at LogTech, LLC. He is a member of IEEE - Eta Kappa Nu.



**MICHAEL A. MULSHINE** received the B.S. degree in electrical engineering from the NJ Institute of Technology, NJ, USA, in 1961.

He worked in engineering, sales/marketing at Electronic Associates, Inc., Analog/Digital computers, from 1961 to 1970. He was the Vice President and the General Manager of ATC Instrument Flight Simulator, from 1970 to 1976. He has been a Consultant for Osprey Partners - Corporate Governance & Business Development, since 1976. He has also been an Instrument Pilot - Private Pilot, since 1962, with Instrument Rating, since 1969.

...