A Survey of Heterogeneous Computing: Concepts and Systems

ILIJA EKMEČIĆ, IGOR TARTALJA, AND VELJKO MILUTINOVIĆ, SENIOR MEMBER, IEEE

This survey of heterogeneous computing concepts and systems is based on the recently proposed by the authors "EM^3" (Execution Modes/Machine Models) taxonomy of computer systems in general. The taxonomy is based on two criteria: the number of execution modes supported by the system and the number of machine models present in the system. Since these two criteria are orthogonal, four classes exist: Single Execution mode/Single machine Model (SESM), Single Execution mode/Multiple machine Models (SEMM), Multiple Execution modes/Single machine Model (MESM), and Multiple Execution modes/Multiple machine Models (MEMM).

In Section II, heterogeneous computing concepts are viewed through three phases of the compilation and execution of any heterogeneous application: parallelism detection, parallelism characterization, and resource allocation. Parallelism detection phase discovers fine-grain parallelism inside every task. This phase is not an exclusive feature of heterogeneous computing, so it will not be dealt with in greater detail. The assignment of parallelism characterization phase is to estimate the behavior of each task in the application on every architecture in the heterogeneous system. In the parallelism characterization domain, one original taxonomy is given. This taxonomy contains scheme classes such as vector and matrix, static and dynamic, implicit and explicit, algorithmic and heuristic, and numeric and symbolic. Resource allocation phase determines the place and the moment for execution of every task, to optimize certain performance measure related to some criteria. In the resource allocation domain, the existing Casavant–Kuhl taxonomy is extended and used. This well known taxonomy is supplemented with scheme classes such as noncooperative competitive, noncooperative noncompetitive, and load sharing.

In Section III, heterogeneous systems characterized with multiple execution modes ("fully" heterogeneous systems falling in the MESM and the MEMM class) are surveyed. The MESM class systems are described and illustrated with three case studies, two of which support SIMD/MIMD and one supports scalar/vector combination of execution modes. The MEMM class systems are described and illustrated with two representative examples of fully heterogeneous networks supporting multiple execution modes. The system software for heterogeneous computing systems is presented according to an original three-dimensional (3-D) taxonomy whose criteria rely on the level of heterogeneity support implementation, the programming approach, and the data access technique applied. In Section III, several representative heterogeneous applications are described with their computation requirements and the systems used for their execution. Each topic covered in the paper contains several concise examples.

Manuscript received October 24, 1995; revised February 22, 1996.

The authors are with the Department of Computer Engineering, School of Electrical Engineering, University of Belgrade, 11000 Belgrade, Yugoslavia (e-mail: eekmecic@etf.bg.ac.yu, etartalj@etf.bg.ac.yu, and emilutiv@etf.bg.ac.yu).

Publisher Item Identifier S 0018-9219(96)05770-2.

I. INTRODUCTION

Throughout the history of computing, applications were the main driving force for the development of new architectures. Faster and faster machines were built to achieve the computing power required by the new and more sophisticated applications. When technological advances had become insufficient to cope with the growing application needs, parallel architectures opened a new avenue for speedup. Generally speaking, different types of parallelism exist: vector, superscalar, dataflow, systolic, as well as other forms of SIMD and MIMD processing. It can be said that there are various *computation types*: scalar (traditional) computation and many different forms of parallel computation. A computation type most suited for the execution of a part of application is referred to as the *computation requirement* of that particular application part. A computation type supported by some architecture is referred to as the *execution mode* of that particular architecture. It is obvious that one-to-one correspondence between computation requirements and execution modes exist. For the execution of a *homogeneous workload*—one application or set of applications characterized exclusively by a single computational requirement—the problem could be solved relatively simply. All that needs to be done is to select a suitable *homogeneous system* (a system that supports a single execution mode) whose execution mode corresponds to the exhibited computational requirement. The problem arises when a *heterogeneous workload* (one application or set of applications composed of segments characterized by different computational requirements; for example, Grand Challenge applications [1]) has to be executed. Homogeneous systems are not suitable, because there are more than one computational requirement in the application and only one execution mode is supported by the architecture. Surely, the mismatch between the execution mode and at least one computational requirement could significantly slow down the whole application. We obviously have to employ some *heterogeneous system*—a

system that supports more than one execution mode. Now, we can effectively execute segments characterized by different computational requirements on resources characterized by corresponding execution modes and thus achieve the most effective computation.

A. EM^3 Taxonomy

So far, two ways of heterogeneous system constructing have been dominant: the first has been to support different execution modes by interconnecting several different machine models; the second one has been to support different execution modes by reconfigurable parallel architecture, obtained by interconnecting the same processors, that is the same machine models. Consequently, the most frequently used classification of heterogeneous systems [2] until now, had divided those systems into mixed-machine systems and mixed-mode systems. Mixed-machine systems exhibit spatial heterogeneity, while mixed-mode systems exhibit temporal heterogeneity. However, as a drawback of this classification appears impossibility to precisely characterize difference between a "heterogeneous" network constituted of different types of general purpose workstations supporting practically the same execution mode (for example, scalar computations) and a suite of machines where different machines supports different execution modes (for example, scalar, vector, and MIMD computations). Essentially, the number of execution modes (execution mode is defined by the type of parallelism supported by a machine; for example, vector, superscalar, dataflow, systolic, as well as other types of SIMD and MIMD processing represent different execution modes) and the number of machine models (machine model is defined by the machine architecture, that is programmer's view of the machine, and the machine performance, that is speed of the machine; for example, different architectures, such as Sun Sparc CY7C601 and Intel i860, or the same architectures driven by different clocks are considered to represent different machine models) are orthogonal features that could serve as the criteria for *Execution Modes/Machine Models-EMMM* (EM^3) taxonomy of computer systems [3]. These criteria yield four classes of computer systems.

1) *Single Execution Mode/Single Machine Model* or *SESM* (uniprocessors, majority of multiprocessors based on the architecturally unique processor element).
2) *Single Execution Mode/Multiple Machine Models* or *SEMM* (networks of different but similar workstations, certain distributed systems).
3) *Multiple Execution Modes/Single Machine Model* or *MESM* (reconfigurable parallel architectures such as PASM [4], TRAC [5], and Triton/1 [6]).
4) *Multiple Execution Modes, Multiple Machine Models/ MEMM* (Nectar [7], PVM [8], HBD [9], and SmartNet [10]).

In the proposed EM^3 classification the problem pointed at above is solved, because the border between network of different models of general purpose workstations (belonging to the SEMM class) and suite of actually heterogeneous machines (belonging to the MEMM class) is clear. In this paper, we only consider heterogeneous systems supporting different execution modes (MESM and MEMM class systems).

B. Heterogeneous Computing Phases

The processing of every heterogeneous application is essentially constituted of three phases: parallelism detection, parallelism characterization, and resource allocation.

Parallelism detection phase is responsible for discovering parallelism inside every task in a heterogeneous application. This phase is not an exclusive characteristic of heterogeneous computing.

Parallelism characterization primarily estimates computation parameters for each task of an application. These parameters describe computation requirements of each task and are used for the successful matching of tasks to adequate machines. Characterization scheme can be simple, giving computation parameters such as the most suitable execution mode for each task, or complex, giving computation parameters such as execution times for each task on each architecture in the system. Besides computation parameters, the amount and the cost of communication between tasks are also estimated in this phase.

Resource allocation determines the place and the moment for execution of every task, to optimize certain performance measure related to some criteria (e.g., maximization of system throughput, minimization of cost or execution time), under possible constraints (e.g., for minimization of execution time the overall machine cost can be constrained or vice versa). The problem of resource allocation is often referred to as task scheduling problem (resources are allocated to task and task are scheduled to resources) [11]. The assignment decision depends on various computation and communication parameters (mentioned above), as well as system state parameters (such as availability and load of resources).

C. Heterogeneous Systems: Current Status and Open Problems

Today, there are no widely available commercial heterogeneous systems. However, there are several dedicated systems used in scientific environments. MESM class systems appeared almost a decade before MEMM class systems, but in the following decade it is MEMM class systems which are expected to achieve their full power.

A number of complex problems should be solved in a process of heterogeneous system design and exploitation. In the paragraphs to follow, we will address the major requirements that should be satisfied, from the technological, architectural, and system software points of view.

First, for MEMM class systems capable to satisfy today's heterogeneous application needs it is necessary to provide powerful networks based on fast communication media and protocols. For example, some heterogeneous applications

include volume rendering. If volume renderer interacting with a user operates on a 3-D picture of $256 \times 256 \times 128$ elements/frame, each element represented by 8 b/s, and generates 10 frames/s, 640 Mb/s [12] are needed. To enable ten or more applications that impose similar requirements to run efficiently, the requested bandwidth is in gigabits/sec domain. Also, low latency scalable interconnection networks are necessary to connect processing elements in a MESM class systems.

Second, in MEMM class systems all the machines are fundamentally different, and typically have incompatible data formats [9]. Process of standardization of simple data formats tends to eliminate this problem in the near future. However, the problem of different representations of more complex structures, such as matrices and dynamically chained structures, will probably remain for a long time. Network interfaces supporting the necessary data conversions have to be designed. These interfaces have to be powerful to reduce the data exchange overhead costs. The less communication and data conversion costs are the finer granularity applications can be executed on the MEMM class systems.

Third, the proper programming environment and system software support should be provided. Full-strength heterogeneous systems have to provide the user with an adequate language level, compiler level, and operating system level support. Compilers and operating systems have to include automatic orchestration tools responsible for optimal or close to optimal utilization of heterogeneous systems. These tools should perform the jobs associated with heterogeneous computing phases: parallelism detection, parallelism characterization, and resource allocation.

1) Motivation for this Paper: So far, several introductory texts [13], [14] and survey papers on the issues of heterogeneous computing [15]–[19] have been published. Since the appearance time of the last survey paper, old ideas have been extended, several new approaches have been generated, several new systems have been developed, and a wide variety of new applications have appeared. Examples include, but are not limited to: new results in parallelism characterization phase [20], [21], new approaches to resource allocation [22], [23], improved version of the Virtual heterogeneous associative machine (VHAM) [24], SmartNet, the new MEMM class system [10], and new applications in meteorology [25] and multimedia [26]. All these advances, together with the ones not mentioned here, give a good impression of the heterogeneous computing development today and, we believe, justify our motivation for a new survey of the field based on originally developed taxonomies.

2) Paper Structure: Following the introductory section, this survey paper contains the concepts section, the systems section, the conclusion, and the rich list of of references. The concepts section contains three subsections, each one devoted to a separate phase of heterogeneous computing: parallelism detection, parallelism characterization, and resource allocation. The systems section contains subsections dealing with architectural issues on MESM class systems, architectural issues on MEMM class systems, system software issues, and heterogeneous applications. In this survey we have tried to reconcile two opposite approaches: extensively and exhaustivity. We have tried to achieve extensivity by covering first example (one thought to have special importance) in subsections more deeply. Exhaustivity is partly achieved by covering other examples, albeit in less detailed fashion. The first paragraph presents the research environment and establishes the place of the approach in appropriate taxonomy scheme. The second one gives the essence of the approach through a brief description. The third one contains a short discussion of the approach involving its advantages and disadvantages. This structure enables not just the ordinary "vertical" reading, but also the "horizontal" reading, that is a reader could pass through introductory "environment and taxonomy place" paragraphs only, "essence and short description" paragraphs only, or "advantages and disadvantages" discussion paragraphs only, depending on the level of information required.

II. Concepts

Basic issues of conventional parallel and distributed computing include, but are not limited to: expression or parallelism (explicit and implicit), communication and synchronization, mechanism of data exchange (message passing and data sharing), physically and logically shared address space, file service, data format conversion, naming and protection, resource allocation, fault tolerance, and various high level services (file transfer, e-mail, data retrieval, etc.).

When used in heterogeneous environment, some of the solutions could be inherited from homogeneous computing, some have to be slightly modified, while the others require radically different approaches. Completely new concepts, unknown in homogeneous computing, as parallelism characterization is, also have appeared.

Only the most important concepts characterizing separate phases of heterogeneous computing were selected for the presentation here. These are parallelism detection, parallelism characterization, and resource allocation. Appropriate issues appear sequentially (as phases) in the execution of every heterogeneous application. Parallelism detection is conceptually the same as in homogeneous computing, parallelism characterization is a completely new concept, while resource allocation is a known concept, but has to be modified to support one new class of allocation parameters, dealing with the behavior of different tasks on different architectures (computation parameters).

A. Parallelism Detection

We will assume heterogeneous applications to be composed of a number of tasks as semantic units explicitly defined by a programmer. In some applications, tasks compute their results, send them to their child tasks, and terminate. Such applications are naturally modeled by the task flow graph (TFG) [27], directed acyclic graph whose nodes represent tasks, while edges represent precedence

relations. On the other hand, some applications are composed of tasks which communicate with each other (in both directions) during their lifetimes. Such applications are naturally modeled by the Task Interaction Graph (TIG) [27]. TIG is not necessarily directed and not necessarily acyclic graph whose nodes represent tasks, while edges represent interaction between tasks rather than precedence relations. Regardless of the intertask relations in an application, each task could be decomposed into blocks that could be executed in parallel. A task is defined as a semantic unit of allocation to a machine (note that process will be assumed to represent the traditional notion of a single executable image on one processor which should not be confused with the term task used here). It is logical to assume that semantic unit components are going to exhibit similar architecture requirements and will not be executed on different machines. A block is considered to be a unit of allocation to a processor in a homogeneous parallel machine. Parallelism detection discovers fine-grain parallelism inside every task and extracts blocks, taking care of precedence relations (data dependencies) between instructions inside blocks. Besides the tasks-and-blocks application structure, some other formats, such as the Heterogeneous optimal selection theory (HOST) application format [28] (to be defined later), could also be assumed.

Parallelism detection is conceptually the same as in conventional parallel and distributed computing, so (as already indicated) will not analyzed here any further.

B. Parallelism Characterization

As already mentioned, the assignment of parallelism characterization is to estimate the behavior of each task in the application on every architecture in the heterogeneous suite. An input to parallelism characterization phase is the TFG, while an output is the TFG supplied with computation and communication parameters. This enhanced TFG is referred to as the code flow graph (CFG) [27]. It is also possible to input the TIG into parallelism characterization phase and transform it into code interaction graph (CIG). Resource allocation schemes presented in the Section II-C are, for the sake of simplicity, related to applications naturally characterized by TFG.

Parallelism characterization in MEMM class systems is performed by two techniques referred to as *code profiling* and *analytical benchmarking* [29]. Code profiling provides information that pertains to the computation requirements of a task, such as code type, code profiling vector showing the matching degree between a task and the present machines, or code structure given in terms of previously defined templates (regular parallel structures [20]) or parameters (high-level operations [21]). Analytical benchmarking shows information that pertains to the execution mode of some present machine, usually the machine speedup when executing code characterized by certain type, code fitting into some template, or certain parameter.

In the MESM class systems, the heterogeneity is exhibited at the instruction level [17]. The primary objective of the parallelism characterization in the MESM class environments is to recognize instruction-level heterogeneity and, for more precise schemes, to determine the places in a program where the mode is going to be changed.

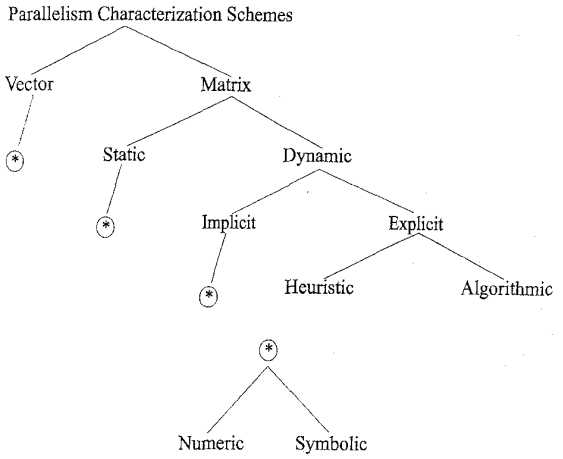


Fig. 1. *A taxonomy of parallelism characterization schemes.* All vector schemes are static, all static schemes are implicit, all implicit schemes are heuristic, and all explicit schemes are numeric. This taxonomy allows the existence of some classes, such as static/symbolic, regardless of the nonexistence of representative examples today.

An original taxonomy of parallelism characterization approaches is shown in Fig. 1.

According to the structure of results, parallelism characterization schemes are divided into vector and matrix schemes. Vector schemes, such as the one in [29], only establish the optimal code type for each task (so an application is characterized by a vector) and assume scalar speedup for other types. matrix schemes compute the absolute or relative execution time of each task on every architecture in the system (so an application is characterized by a matrix) [30], [27].

According to the utilization of input data sets size information, the matrix schemes are divided into *static* and *dynamic* schemes. In static schemes [30], input data sizes, usually not known in compile-time, are assumed not to influence characterization results. In dynamic schemes [27], [31], [20], [21] input data sizes are assumed to influence results. All vector schemes are static in their nature because input data sizes are not assumed to influence code types in vector schemes.

Dynamic schemes can be divided according to the type of computation parameters they provide into *explicit* and *implicit* schemes. Explicit schemes [31], [20], [21] combine the results from the code profiling and the analytical benchmarking to explicitly estimate the task execution times on various architectures. Rather than estimating execution times, the implicit approaches [30], [27], carry both the code profiling and the analytical benchmarking results (e.g., speedup factors) into the resource allocation phase where the task execution time on various architectures figures implicitly. All static schemes are implicit because the execution time metric, which figures in explicit schemes, is sensitive to the changes of input data sizes.

According to the availability of deterministic evaluation function, the explicit schemes can be divided into *algorithmic* and *heuristic* schemes. Algorithmic schemes, such as the one presented in [31], [21], assume the existence of such function. Heuristic schemes, such as the ones in [20] use embedded "experience rules" instead of an evaluation function. All implicit schemes are heuristic because implicit schemes yield compatibility factors between tasks and machines, which are not exact measures, and therefore cannot be computed by deterministic evaluation functions. Algorithmic explicit schemes are the most precise schemes, but even they cannot be optimal because of dynamic operations whose performance is dependent on the value of their operands (e.g., floating-point arithmetic), which are not known before run time.

All implicit schemes ("vector," "static," and "implicit") can be divided into *numeric* and *symbolic* schemes, according to the type of the parameters used. Both the static scheme in [30] and the dynamic scheme [27] are numeric. Symbolic schemes are often based on the artificial intelligence techniques (knowledge bases searching, etc.). No static symbolic approaches have been devised yet, although these schemes are potentially the simplest. Several dynamic approaches are devised and used in prototype systems, such as the one described in [32].

1) Parallel Assessment Windows System: Pease *et al.* from Syracuse University presented a matrix/dynamic/explicit/algorithmic scheme, called Parallel Assessment Windows System (PAWS) [31].

Besides parallelism characterization, PAWS is also responsible for parallelism detection. Every task is transformed into a dataflow graph, which represents data dependencies and parallelism. Each node in a dataflow graph represents an operation at architectural level. Every operation has to be evaluated for every architecture. This is performed either by low-level machine benchmarking, for static operations (execution time does not depend on operand values, or analytical modeling, for dynamic operations (execution time does depend on operand values). Now the time estimation process can be performed in two steps. First, a dataflow graph is mapped to the real architecture for which the evaluation is carried out. Second, critical path analysis of the mapped dataflow graph is performed using timing values of all operations for this particular architecture. These two steps are repeated to obtain a matrix of execution times of every task on every architecture. PAWS consists of four tools shown in Fig. 2: *Application Characterization (APCT), Architecture Characterization Tool (ARCT), Performance Assessment Tool (PAT),* and *Interactive Graphical Display Tool (IGDT).* APCT translates the application into a special language form referred to as the Intermediate Form one (IF1). This form shows data dependencies and parallelism at operation and procedure level. ARCT enables user to get timing information for any operation that can appear in IF1 graph for any architecture. Operations generally can be classified into: computation, communication, I/O, and control operations. These operation classes can be divided further. For example, communication operations

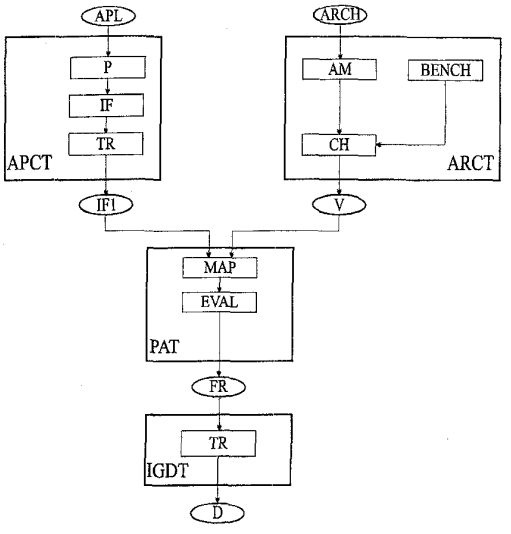


Fig. 2. *PAWS Evaluation Tool.* In APCT, applications are represented by data-flow graph language. Architectures are represented in ARCT by the special treelike machine representation scheme down to subsystems fine enough to be evaluated by low level benchmarking and analytical models. The dataflow graphs are mapped to real architecture models and evaluated in PAT. IGDT is responsible for display in a user-friendly fashion. The quality of obtained estimations are still not perfect, since they are influenced by the quality of mapping results (which are usually suboptimal).

can be divided into processor-to-processor, processor-to-memory, etc. Processor-to-processor communication operations differ according to the interconnection network between processors. In this fashion a tree whose leaves contain detailed characterization of each operation for the analyzed architecture is obtained. It is done by low-level benchmarking for static operations or analytical modeling for dynamic operations. Using the results from APCT and ARCT, PAT estimates the execution time of a task on some concrete architecture. PAT first has to perform mapping of ideal-case dataflow graph to some concrete machine. Often, user have to be satisfied with suboptimal heuristic solution for this nontrivial problem. The mapped data-flow graph for the analyzed architecture is traversed operation by operation. As soon as an operation is visited a query to ARCT is generated to get the execution time of that particular operation on the analyzed architecture. Since every node in the mapped dataflow graph is assigned with the timing value, critical path analysis can be conducted to return the execution time of the overall task. IGDT enables users to interactively access any information from APCT, ARCT, and PAT in a user-friendly fashion.

PAWS is a precise tool, but quite time consuming because execution time of every task has to be estimated for every architecture. However, the ideas behind PAWS served as the generator for two other schemes, which are presented in the following text.

2) Template-Based Characterization: Yang from Purdue University *et al.* presented template-based characterization, a matrix/dynamic/explicit/heuristic scheme [20].

Code profiles are represented in terms of regular parallel structures, referred to as templates. The idea behind this

approach is that one template is expected to yield similar speedup (speedup is considered to be a ratio of execution time on some parallel machine and the execution on a serial baseline machine) for any code fitting into them. Templates have to be described with the following parameters: number of blocks composing a template, template depth, block execution time on a supposed serial baseline system, etc. Such "template instances" are referred to as augmented templates. Obviously, there can be arbitrary many augmented templates characterized by more or less different speedups. Few of them are analyzed via benchmarking, while the others are submitted to regression analysis. A task can now be represented in the form of a graph composed of templates, for which the critical path analysis can be carried out for every architecture. Authors have also left an option of detailed characterization of selected templates, as in PAWS.

This approach introduces imprecision due to regression analysis, but is much less expensive than PAWS, since detailed computation is avoided. High degree of irregular parallelism or excessive fine-grained characterization can corrupt speed.

3) Parametric Characterization: Yang *et al.* have also developed a matrix/dynamic/explicit/algorithmic characterization scheme called parametric characterization [21].

Code profiles are expressed in terms of *parameters*, similar to PAWS operations. Tasks are supposed to be composed of computation "supersteps" that are separated by communication units. Supersteps are composed of parallel segments. Since the longest running segment dominates the execution time of a superstep, it is the only segment for which the critical path analysis has to be carried out. The machine performance for the entire set of parameters is obtained via benchmarking. The execution time spent on internal communication is obtained using *Architecture Independent Model* [33] based on interconnection network topology, network diameter, packet length, setup time for sending and receiving, etc. The total execution time of a task on a supposed parallel machine is obtained as a sum of the supersteps' execution times plus the overall communication time. The described operation is repeated for every task on every architecture to obtain the characterization matrix, as in PAWS.

Characterization in terms of parameters is somewhat less precise, but also less expensive than the one presented in PAWS. It is also more precise and not much more expensive than template-based characterization. All this makes parametric characterization probably the best choice for explicit characterization.

4) Characterization in MESM Systems: Watson from the Parallel Processing Laboratory at Purdue University *et al.* have presented one approach to parallelism characterization in the SIMD/MIMD (actually *single program multiple data* (SPMD), a form of the MIMD) MESM class systems [34].

The approach assumes heterogeneous applications composed of blocks, if-then-else constructs, and for-loops. Block execution times are estimated first for both execution modes. If-then-else constructs generally execute faster in the MIMD mode, except in some extreme cases [35], [34]. If the probabilities of appearance of these cases and branch probabilities are known, execution time of if-then-else constructs can be estimated. Execution times of for-loops can also be estimated, if the number of loop iterations is known at compile time. The authors developed an algorithm for determination of the mode switching points to obtain the minimal execution time of the overall application. These points can be arranged in exponentially many combinations, but since parallel constructs are run one after the other, the problem can be reduced to the finding of the least cost path through a multistage graph (Moore's algorithm [36]), which requires linear complexity.

The approach is simple, but can yield imperfect results because it is based on the usage of probabilistic parameters.

C. Resource Allocation

The assignment of resource allocation is to determine the time and place of execution for every task in a heterogeneous application. The decision, based on computation, communication, and/or system state parameters, tends to optimize performance according to a certain criterion, such as the overall execution time (possibly with cost constraints), the overall cost (possibly with time constraints), the system throughput (possibly with response time constraints), etc.

One of the most complete taxonomies in the domain of general purpose distributed homogeneous processing known so far is the Casavant–Kuhl (C-K) taxonomy [11]. C-K taxonomy is modestly extended here (Fig. 3), to describe allocation approaches in heterogeneous computing, known so far. Classes that do not have their representatives yet are retained because such representatives are expected to appear.

Hierarchical C-K taxonomy divides schemes into *static* and *dynamic*, according to the time of decision making. According to the availability of the complete set of input parameters and the evaluation function, static schemes are divided into *optimal* and *suboptimal*. According to availability of determinsitic evaluation function, suboptimal schemes are divided into *approximate* [37] and *heuristic* [28]. Both optimal and suboptimal approximate schemes can be reduced to *enumerative*, *mathematical programming* [29], [30], [22], [23], *graph theory* [37], and *queuing theory* problems. Dynamic schemes can be *distributed* or *nondistributed* depending on the distribution of responsibility and authority for the allocation decisions. Distributed schemes can be *cooperative*, if decisions are targeted to the performance improvement of the overall system, or *noncooperative*, if every system tends to improve its own performance. Cooperative schemes are divided in the same fashion as static ones.

Noncooperative can be divided according to the degree of interaction of allocation processes into *competitive* and *noncompetitive* (the last division is an original extension of the C-K taxonomy). Competitive schemes [38], allow tasks to simultaneously allocate resources (i.e., compete for them). Noncompetitive schemes regulate the allocation order using

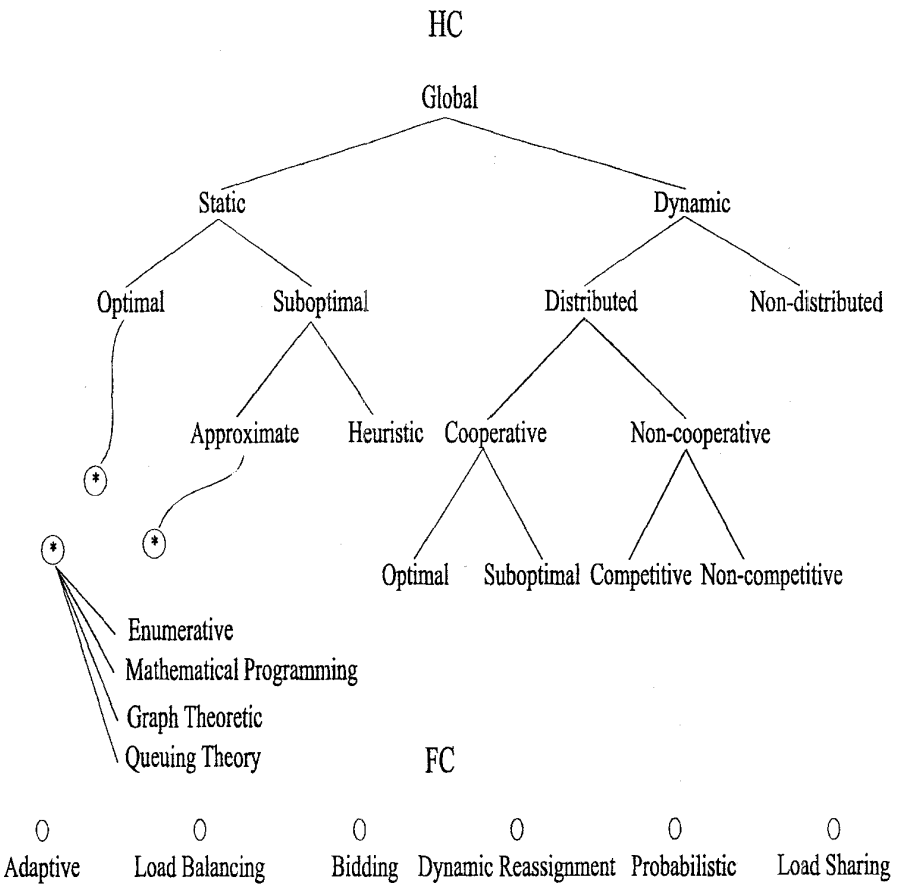


Fig. 3. *The modified C-K taxonomy of allocation schemes.* HC = hierarchical taxonomy, FC = flat taxonomy. The criteria in HC are organized to minimize the size of the taxonomy. The existence of FC taxonomy reduces the complexity of the taxonomy tree because FC classes could appear on more than one place in HC. two new subclasses of noncooperative schemes in HC are introduced: competitive and noncompetitive classes. Also, load sharing descriptor is added in FC.

a protocol (e.g., based on allocation token) which disallows simultaneous allocation (i.e., avoid competition).

To make the taxonomy more compact, classes characterized by relevant descriptors that can appear at several places in the described hierarchy are organized into the flat portion of the taxonomy scheme. These classes include *adaptive, load balancing, bidding, dynamic reassignment (task migration), probabilistic* [39], and *load sharing* [38] schemes. Adaptive schemes dynamically change allocation policies and mechanisms. Load balancing schemes increase the system throughput by equalizing the load across machines that support the same execution mode. Equalizing the load across the overall system would include assignments of tasks to unsuited machines and hence deteriorate the performance. Bidding is the form of negotiation between nodes which create tasks and those available for their execution. Dynamic reassignment allows tasks to migrate to a machine which in time became more appropriate for their execution. Probabilistic schemes tend to find satisfactory solution from a randomly selected subset of possible solutions, rather than to spend time on computing of evaluation functions and intelligent pruning of the entire solution space.

In attempt to classify the LOCO approach, we have concluded that load sharing descriptor should also be included into the flat taxonomy as a separate class. Load sharing means "hunting for a free processor" to decrease response time. In heterogeneous computing, it is not satisfactory to hunt for any processor, because some processors are not suitable for a specific task.

1) Optimal Selection Theory (OST): Freund from Naval Ocean System Center showed [29] how to reduce the problem of the optimal hardware selection (which is viewed as the special case of the static/optimal resource allocation problem) to the well known and formalized mathematical programming problem.

The selection of the optimal configuration for an application modeled by the TFG without any constraint is NP-complete problem [40]. OST assumes heterogeneous applications in the following restrictive format, rather than in TFG format. Tasks with the same computation requirements that can run in parallel are grouped into structures called code segments. Code segments are assumed to run sequentially. Code segments can be decomposable (composed of more than one task) or nondecomposable (composed of only one task). The problem can now be

reduced to the problem of selecting a machine per task, so the sum of code segment execution times is minimal, while the sum of machine costs is less than some given cost constraint. Freund shows this problem to be an instance of the well-known mathematical (actually, integer) programming problem. This problem has the optimal solution. Consequently, the optimal heterogeneous suite can be selected for arbitrary application (hence the name Optimal Selection Theory). However, the integer programming is still NP-complete [40] and optimal solutions are rarely sought. The task execution times are estimated using a vector parallelism characterization scheme based on code profiling and analytical benchmarking, techniques originally introduced in OST. Code segments are organized into equivalence classes with the respect to computational requirements they exhibit (vector code segments into one class, MIMD parallel code segments into another, etc.). To effectively calculate how long each code segment from one equivalence class execute on the appropriate machine model, two assumptions are made. First, the execution time of a code segment on some optimally suited machine is linearly dependent on the execution time of that code segment on a serial machine. This means that the speedup of machines optimally suited for the execution of one code segment is a constant (which can be obtained by analytical benchmarking). Consequently, the execution time of any code segment on any suited machine can be obtained by dividing its execution time on the serial baseline system by the speedup of the selected machine. Second, the execution time of a decomposable code segment on suited machines linearly decreases with the increase of the number of the same machines used. So the execution time of a decomposable code segment on more than one suitable machine is obtained by dividing the execution time needed on one machine by the number of same machines employed. These two assumptions hold when the execution mode of employed machines matches the computation requirements in a code segment. Otherwise, they do not hold.

This pioneering approach to hardward selection theory had the great impact on the following research in the field of heterogeneous computing. Its disadvantages, namely vector characterization and restrictive application format, were partly overcome in the OST successors, covered in the next two examples.

2) Augmented OST: Wang from Purdue University *et al.* reformulated OST to obtain another mathematical programming formulation of the static/optimal hardware selection, referred to as Augmented OST (AOST) [30].

OST was augmented in two ways. First, in AOST a matrix parallelism characterization scheme is assumed. The matrix containing relative execution times for each task on each machine is used if a task has to be scheduled for execution on a nonoptimal machine, due to unavailability or high-cost of the best suited machine. Second, a limited number of machines, opposed to OST, was assumed. Rather than in OST, AOST takes care of available parallelism in an application segment and utilization factor of a parallel machine, contributing to the precision of estimated task execution times. For example, AOST will recognize that a machine with 50% utilization factor will score only half of the maximum speedup obtained from analytical benchmarking, resulting in much more precise estimation. The authors also allowed the possibility of having different machine models of the same type (e.g., different vector machines). One algorithm for the allocation of tasks of the same type to such machines was also given.

Although AOST was an improvement over OST, several disadvantages still remained. The main two of them were that tasks of differing types were still unable to run concurrently and that communication parameters did not influence the allocation.

3) Heterogeneous OST: Chen from the New Jersey Institute of Technology *et al.* presented the new improvement over OST and AOST selection theories, referred to as Heterogeneous Optimal Selection Theory (HOST). The authors also suggested the Hierarchical Cluster-M (HCM) methodology for the static/suboptimal/heuristic allocation [28], which satisfies the specific conditions assumed in HOST.

HOST introduces an application format in which applications (called tasks in original paper) consist of sequentially sunning subapplications (called subtasks in original paper). Subapplications consist of segments (similarly defined as in OST), which run in parallel (opposed to OST) and are not characterized by the same computation requirement. Segments are composed of blocks, which also run in parallel and are characterized by the same computation requirement. The HCM allocation methodology takes applications in the HOST format as input and maps them to the special graph referred to as the HCM Application Graph. This hierarchical graph shows the parallelism at all granularity levels, the computation requirement of each block and the corresponding segment, as well as communication connections between components of an application. On the other hand, heterogeneous suite is represented by the HCM System Graph. This graph is also a hierarchical one and shows the execution mode of each machine in the suite as well as connections between hardware components at all levels. The HCM methodology is based on specific rules for level-by-level matching of application components, shown in HCM Application Graph, with appropriate hardware components from heterogeneous suite, shown in HCM System graph. This matching takes into consideration both the execution modes and connections from both of these two graphs.

This approach has overcome many of the shortcomings of the previous two approaches, but at the expense of its heuristic nature which can yield imperfect results.

4) Generalized OST: Narahari from George Washington University *et al.* introduced Generalized OST (GOST) and developed two static/optimal allocation schemes under the subset of GOST assumptions [22].

GOST assumes applications modeled by TFG, communication costs dependent on machines involved, various fine-grain mapping strategies, different networks in the system, and the existence of data reformating time. Both

proposed schemes assume certain matrix/dynamic/explicit parallelism characterization scheme. The first scheme maps applications modeled by in-trees (in-tree is defined as a tree in which each node has at most one immediate successor) to heterogeneous suites and is based on dynamic programming. Every element of the characterization vector of a graph node (the task execution time on the machine corresponding to that element) is summed up with the smallest time necessary for all children of the node to be completed. The assignment of every machine to every child has to be evaluated taking into consideration communication costs and execution time. The assignment yielding the minimal execution time should be chosen. The algorithm propagates from leaves to the root. The second scheme, based on graph theory, assumes applications modeled by series-parallel graph (a graph that can be obtained from initial single edge by repeated replacements of edges by series edges or parallel edges). Every node is described by the characterization vector of the corresponding task. Every edge is characterized by the set of weights denoting communication costs between every two machines allocated to the tasks in nodes connected by the edge. The scheme repeatedly merges two series or parallel edges into a new edge and the set of weights describing new edge is computed. The algorithm continues until the initial graph is reduced to one single edge connecting two nodes. The optimal allocation is determined by the minimal execution time of all combinations taking into consideration weights of two remaining nodes and one edge between them.

The first algorithm has $O(\omega^2 n)$, while the second one has $O(\omega^3 n)$ complexity, where ω is the number of machine models and n is the number of tasks. Since ω is usually a small number, both of these algorithms can be considered to have linear complexity.

5) Layered Graph Approach: Iqbal from Lahore University of Engineering and Technology presented a static/suboptimal/approximate/graph theoretic allocation scheme [37].

The approach works only for very strict conditions (chainlike applications specified in the HOST format, chainlike system topologies, restricted communication patterns, and restricted segment assignment policies). Under the listed conditions it is possible to reduce this problem to the problem of finding the least cost path through the special topology graph called the layered graph. First, time/cost combinations are computed for each subapplication. Second, it is decided which subapplication is going to be assigned more resources (and to run faster), to achieve the best possible execution time for the overall application. Both of these steps are reduced to a problem of finding the least cost path through a layered graph subject to cost constraints.

The results are approximate, but their precision can be controlled by varying the number of possible time/cost combinations. Even with the very strict conditions, this scheme can be useful for a wide class of applications, because the preprocessing schemes are designed to translate any application in the chainlike format, so it can be mapped onto the linear processor arrays.

6) LOCO Approach: Milutinović, at the time of publication with Purdue University *et al.* presented the earliest dynamic/distributed/noncooperative/competitive/load-sharing resource allocation approach, referred to as the LOCO approach [38].

Machines of the same type are supposed to be interconnected and organized into one cluster. Different clusters are also interconnected. The allocation control information, computed in hosts, propagates through the network in the form of a train. The LOCOmotive of the train contains information relevant for an application, while the wagons describe relevant information for task execution, such as the selected cluster, addresses of instruction and data blocks, data dependencies, and status information. The train is sent to the cluster indicated in the first wagon. This wagon is replaced with the imaginary copy of the wagon with the status indicating that the execution of the corresponding task is under way. If there are other tasks not data dependent of the first task, the train is redirected to the cluster indicated in the first wagon not data dependent on currently executing wagons. When execution of a task is finished, the wagon with the address of results is broadcast, accepted in the station in which the train is currently resident, and placed instead of the imaginary copy. The wagons not needed anymore are discarded. This scheme enables a train to concurrently wait in each machine queue by circulating around a cluster and allocating a machine which first becomes available.

The authors of the approach managed to avoid usage of allocation tables and make the approach simpler at the expense of broadcasting of wagon that contains the address of result block.

7) Probabilistic Heuristic Allocation: Tao from Concordia University *et al.* upgraded two allocation approaches related to homogeneous systems for the usage in heterogeneous systems, and presented one original approach to allocation problem [39]. All presented approaches are classified as static/suboptimal/heuristic/probabilistic schemes.

Presented approaches assume an arbitrary task distribution as the initial solution. New solutions are obtained by moves such as moving one task from one machine to another or switching loads of two machines. A newly obtained solution is evaluated by the function based on computation, communication, and task interference parameters. If the new solution is better than the old one, it is taken as the current solution. In order not to trap the searching process in a poor local optimum, worse solutions are sometimes taken as current ones. All three heuristic schemes are iterative in nature and stop when some criteria are satisfied. In the first heuristic scheme, referred to as *simulated annealing*, bad moves are taken with the probability which is getting smaller in every iteration. The second heuristic scheme, named *tabu search*, performs aggressive search in many directions. The third heuristic scheme, called *stochastic probe*, combines the features of the first two heuristic schemes.

It has been shown that stochastic probe always give slightly better results than the other two heuristic schemes, with a substantial cut in execution time of the allocation scheme.

III. Systems

In this paper, only systems that support different execution modes (the MESM and the MEMM class systems), and thus are able to score significant speedups on heterogeneous applications, are going to be considered as heterogeneous.

The MESM class systems are composed of replicated machine models that can together be reconfigured to support more than one execution mode. First MESM class systems appeared in late 1970's–early 1980's and supported switching between SIMD and MIMD modes, such as one presented in [5]. Besides systems supporting these two modes, systems supporting scalar and vector modes can also be thought of as the MESM class systems. The MESM class systems are considered to be an extreme form of heterogeneous computing systems [19].

The MEMM class systems are composed of several different computer models, out of which there are at least two that support different execution modes. These systems appeared in late 1980's [7]. They are expected to grow into the most promising systems to cope with requirements of Grand Challenge applications [1].

Although neither the MESM nor the MEMM class systems are commercially available at this moment, both are used in scientific environments as experimental systems. In the near future, commercial breakthrough could be expected in multimedia applications.

A. The MESM Class Systems

In the MESM class systems, heterogeneity is exhibited temporally [41]. The same set of processing elements at one moment operates in one mode, and at the other moment in some mode. This is usually achieved by dynamic reconfiguration of control logic. Also, it is possible that distinct subsets of processing elements are separately reconfigured to operate independently in different modes at different moments. In that case, we say that the MESM class system shows spatio-temporal heterogeneity. The most frequently supported mode combination is the SIMD/MIMD combination. The examples of this MESM class combination are PASM [4], TRAC [5], OPSILA [42], and Triton/1 [6]. The other combination, becoming more and more present, is the scalar/vector combination. One interesting example is the MultiTitan project [43]. The MESM class systems are particularly valuable for the execution of applications exhibiting the instruction level heterogeneity [17].

1) Partitionable SIMD/MIMD (PASM): Siegel from Purdue University *et al.* suggested the SIMD/MIMD MESM class architecture referred to as PASM [4]. The research efforts on explicit parallelism characterization [34] and dynamic resource allocation [44] related to PASM, were reported as well.

The objective was to develop the architecture which can efficiently execute tasks exhibiting both SIMD and MIMD type of parallelism intertwined on instruction level (image understanding, speech understanding, or biomedical signal processing). The PASM system is composed of a Parallel Computation Unit (PCU), Micro Controllers (MC's), and several units for memory management and control. PCU is composed of $N = 2^n$ Processing Elements (PE) interconnected by a multistage interconnection network. PE's are organized into $Q = 2^q$ partitions that could be utilized to execute different tasks at the same time. All processors from one partition operate in the same mode. MC's serve as the control units of partitions, that is, they broadcast instructions to PE's when executing in SIMD mode and synchronize the operation of PE's when executing in MIMD mode. Mode of a PE, and consequently of the whole partition, can be dynamically changed in a time comparable to the instruction execution time. Mode changing scheme is based on the existence of the special region in PE's memory referred to as the SIMD Instruction Space (SIMDIS). If program counter (PC) points to a location in SIMDIS, PE executes the instruction that comes from the partition control unit (MC), that is PE operates in the SIMD mode. Otherwise, PE executes the instruction pointed by PC, that is PE operates in MIMD mode. To cause PE's to switch to MIMD mode, MC broadcasts "jump to subroutine at A" instruction to each PE in the partition it controls, where A does not belong to the SIMDIS. The mode change from MIMD to SIMD is performed after all PE's in one partition execute "return" instruction which reloads program counter with some value pointing to the SIMDIS. Thus mode switch time is comparable to instruction execution time. "Jump to subroutine" and "return" instructions are inserted at points determined in the parallelism characterization phase. Explicit parallelism characterization scheme for MESM class systems given in [34] could be integrated in PASM. PASM partition sizes can adapt to the degree of parallelism resident in tasks. The smallest possible partition includes N/Q PE's, but R smaller partitions can be joined into a larger partition. The only constraint is that R has to be power of two. When R partitions are joined into one partition, the main question is how to make R MC's (originally, each MC was responsible for the control of one partition) to function as one MC. One way is to put the same SIMD program into each MC. Another way is to connect MC's by a reconfigurable bus, so that one MC sends instructions to other MC's that forward them to PE's. The first scheme consumes more space, while the second one imposes additional cost and time. Reconfigurable partitions represent resources, which are allocated to tasks by the operating system part referred to as the *Automatic Reconfiguration System* (ARS) [44].

Power-of-two sizes of partitions enable easy message routing. On the other hand, poor utilization of PE's could be scored due to inflexibility in the choosing of partition sizes (for example, if $5N/Q$ PE's are needed eight partitions have to be combines, leaving $3N/Q$ PE's unused).

2) Triton/1: Herter *et al.* from University of Karlsruhe introduced Triton/1, the MESM class heterogeneous system supporting both SIMD and MIMD execution modes [6].

Triton/1 parallel architecture was codesigned with Modula-2* parallel language and optimizing compiler. The system consists of the front-end and the back-end composed of 256 processing elements (PE) mutually interconnected by DeBruijn network. Each PE is supplied with a disk. The system supports parallel I/O based on the specific concept of vector files. The PE execution mode is determined by a special bit in a command register. To switch to MIMD mode, this bit has to be reset and program counter has to be loaded. SIMD mode is started by setting this bit, while program counter is not relevant since the instruction comes from the front-end. The mode switching time is obviously comparable to an ordinary instruction execution time. Barrier synchronization was achieved via special hardware. Modula-2* has two extensions over sequential Modula-2: 1) FORALL statement (synchronous or asynchronous), and 2) data allocators (hints for compiler to improve locality of array elements). Synchronous FORALL statement creates several blocks running in SIMD mode, while asynchronous FORALL creates several blocks running in MIMD mode. The mode switching is triggered by program structure, which means that parallelism characterization is manual. It is possible to have multiple independent tasks, but only one task at the time can run in SIMD mode. The allocation of resources to these tasks was not explained.

Triton/1 shows good programmability and scalability, supports instruction-level heterogeneity; however, it has limitations in the creation of parallel processes and the support for execution of multiple independent parallel tasks in SIMD mode.

3) MultiTitan: Jouppi *et al.* from Digital Equipment Corporation presented MultiTitan, the MESM class floating-point architecture supporting scalar and vector execution modes [43].

MultiTitan makes no difference between the hardware support for vector elements and scalars, because vector elements are also scalar values. Instead, MultiTitan uses the same architecture block acting as the scalar register file at one moment and as the set of dynamically sized vector registers at another moment. Vector instructions are emulated by reissuing and element-wise execution of a scalar instruction. Load-store operations can run in parallel with ALU operations enabling the maximal vector speedup of two (when there are no data dependency hazards). Since vector registers do not exist as a separate resource, reservation policy becomes a significant problem. Registers are dynamically reserved after the issuing of vector instruction. Parallelism characterization in MultiTitan is implicitly performed by the architecture (MultiTitan has different instructions for scalar and vector modes). Resource allocation problem was not elaborated in [43].

Rather than reaching the peak performance of vector machines, MultiTitan both optimizes more frequent scalar operations and speeds up vector operations two times. It is useful for the applications showing instruction-level heterogeneity and modest degree of vectorization.

B. The MEMM Class Systems

In MEMM class systems, heterogeneity is exhibited spatially. Different execution modes are supported on different places (different machines) at the same time. Although the known MESM class systems support two execution modes, MEMM class systems are more flexible and can support variety of execution modes by the appropriate machine models incorporated into the network. In this paper, we are going to present two gigabit network testbeds, Nectar [7], and VISTAnet [12]. Both of these testbeds, together with the projects Aurora, Blanca, and Casa, constitute a joined effort to develop the National Resource Education Network (NREN) [45].

1) Nectar: Arnould from Carnegie Mellon University *et al.* designed heterogeneous gigabit network testbed referred to as Nectar [7]. Nectar prototype was planned to provide general-purpose computing by powerful RISC machines such as Sun-3's and Sun-4's, as well as the specific-purpose computing by the Warp systolic array.

Nectar (Fig. 4) is composed of the Nectar-net and the set of Communication Accelerator Boards (CAB's). Nectar-net is a powerful network composed of hubs—modules that can be connected to form an arbitrary network topology. CAB's are powerful RISC processors responsible for connecting nodes to Nectar-net. CAB's execute protocol software that is responsible for heterogeneity support and effective communication. Parallelism characterization and resource allocation have to be performed manually. Hub can be connected to other hub or to CAB. Hub has several I/O ports, each with two fiber optic channels. One channel is connected to input queue, and the other one to output register. Input and output ports are connected by the crossbar switch. It is ensured that an output port cannot be connected to more than one input port. Hub supports hardware implemented commands for circuit switching as well as for packet switching. It is possible to implement more sophisticated protocols, like multicasting. It is well known that communication time is dominantly influenced by the time spent in software rather than the time spent in the hardware. Therefore, CAB software is very carefully designed to avoid unnecessary copying from layer to layer and unnecessary context switching. The CAB kernel is based on the concept of "lightweight processes" (similar to threads [46]) that have little state associated with them, making context switching time low. The communication protocol consists of data link protocol, transport protocol, and CAB-node interface. The programming interface (*Nectarine*) allows the programmer to manipulate with tasks, messages, and buffers. It is designed to hide heterogeneity from the programmer, but system software tools that would automatically perform heterogeneous computing phases were not supplied when the paper [7] was presented.

The advantages of Nectar include network flexibility (arbitrary topologies can be constructed, both circuit and packet switching are supported, both unicasting and multicasting are available) and system expandability (hub organization and sophisticated latency hiding techniques enable

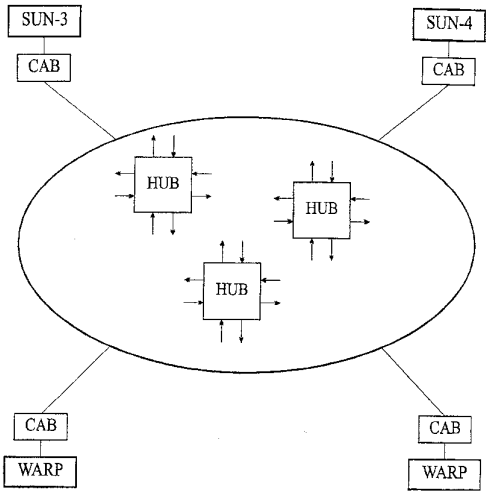


Fig. 4. *A heterogeneous system based on Nectar.* SUN-3 and SUN-4 = two scalar machines, WARP = a systolic array, CAB = the interface between the nodes and the Nectar network. Hub = building-block interface for a Nectar-net. CAB's are interfaces between Nectar-net and a node, while hubs are building block for an arbitrary network topology. A sophisticated system software support that automatically carries out heterogeneous computing phases is needed.

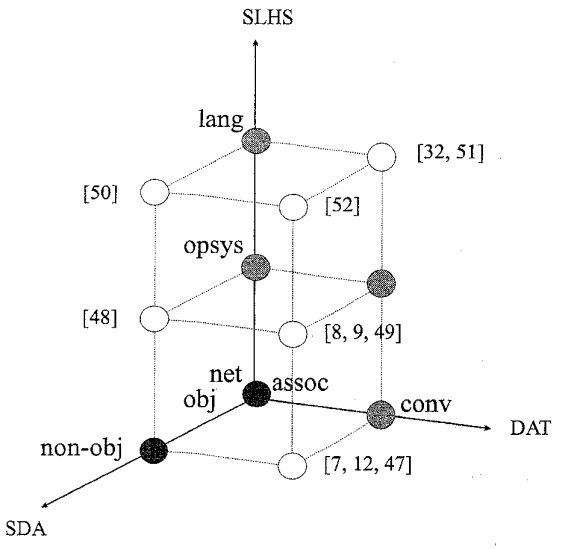


Fig. 5. *A taxonomy of the system software schemes.* SLHS = the system level responsible for heterogeneity support, SDA = the software development approach, DAT = the data access technique, *net* = network level, *opsys* = operating system level, *lang* = language level, *non-obj* = nonobject oriented, *obj*= object-oriented, *conv* = conventional addressing, *assoc* = associative addressing, white balls = implemented schemes, black balls = impossible schemes, and gray balls = possible, but not yet implemented schemes. All three criteria are orthogonal. Associative addressing on the network level is not possible since associativity represents direct support for the transparency achieving techniques (dealt with in the operating system domain). Gray balls enable the prediction of appearance of feasible systems not yet implemented.

scalable performance). The main disadvantage is the lack of tools for parallelism characterization and resource allocation.

2) VISTAnet: VISTAnet is another gigabit network testbed currently under development in the United States. In 1990, a large group of researchers began working on VISTAnet at the University of North Carolina at Chapel Hill. Their objective was to produce a MEMM class system capable of executing 3-D visualization tool for the evaluation of plans for radiation treatment of cancerous cells.

VISTAnet is a system composed on one workstation (Silicon Graphics 340 VGX), one graphics oriented machine (Pixel-Planes 5), one massive parallel machine (MasPar MP-1), and one vector supercomputer (CRAY Y-MP Model 464). Two interconnection node types are used: ATM switch and one experimental GTE node. These nodes are connected by powerful high-bandwidth links (622 Mbps OC-12 and SONET 2.448 Gbps OC-48). Machines, interconnection nodes, and links are selected to match the needs of dynamic radiation therapy planning (DRTP) application. DRTP application is a tool that enables a physician to view results of his/her manually designed plan of cancerous cells treatment by radiation beams. Results are given in the form of a 3-D picture that greatly increases physician's efficiency. The support for automatic execution of heterogeneous computing phases was not the objective of the VISTAnet prototype (since it is dedicated to only one application), so parallelism characterization and resource allocation are performed manually.

VISTAnet is a flexible system that could easily be plugged into a larger environments based on both ATM networks and HIgh Performance Parallel Interface (HIPPI). The prototype is used to evaluate networking issues. To transform VISTAnet into powerful heterogeneous computing system, it is necessary to design the tools that support automatization of parallelism characterization and resource allocation.

C. System Software

System software is responsible to provide users with the smooth and effective means of exploiting the computing power of various execution modes inherent to a heterogeneous system. Ideally, the computing power should be exploited by automatic tools that support heterogeneous computing phases. However, generally speaking, current practice is still a long way from that strategic goal. Since majority of the efforts in system software for heterogeneous computing area is devoted to MEMM class systems, the examples we present are also targeted to MEMM class systems.

We have selected three orthogonal criteria for the presentation of system software: the *system level responsible for heterogeneity support,* the *software development approach,* and the *data access techniques* (Fig. 5).

Heterogeneity can be supported on three levels: *network level, operating system level,* and *language level.* Network level provides services such as data transfer and remote task invocation. The examples fo systems supporting heterogeneity on the network level are [47], VISTAnet [12], Nectar [7], etc. For these systems parallelism characteriza-

tion and resource allocation have to be performed manually. These systems are important because employed networks (for MEMM class systems) or interconnection networks (for MESM class systems) are expected to serve as underlying networks for future heterogeneous systems (equipped with higher software layers). Operating system level provides transparency and support for dynamic parallelism characterization, automatic resource allocation, and various other services. The examples of systems supporting heterogeneity on the operating systems level are Parallel Virtual Machine (PVM) [8], HBD [9], VHAM [48], p4 [49], etc. Systems from this class are flexible because they allow usage of several programing languages. Language level provides special facilities to enable the programmer to express hints on parallelism characterization and resource allocation issues. The examples of systems supporting heterogeneity on the language level are Linda [50], Paralation Lisp [51], PCN [52], actor-based languages [32], etc. Systems from this class are less flexible but more comfortable for users.

Software can be developed as object-oriented or not. Software is object-oriented if and only if support data encapsulation, abstractions (class groups), and polymorphism (abstraction intersection and overlapping) [53]. The examples of heterogeneous object-oriented systems are presented in [32], [51]. If software is developed without supporting the above three requirements, the approach is referred to as nonobject-oriented. The examples of heterogeneous nonobject-oriented systems are presented in [47], [12], [7]–[9], [49], [52].

Data can be accessed through conventional or associative addressing techniques. Most of present heterogeneous computing systems use conventional addressing techniques. We are aware of only two heterogeneous systems that employ associative addressing techniques. The first one accesses data by content [50], while the second one use attributes such as data type, data size, etc. to access appropriate data [48].

1) The Parallel Virtual Machine (PVM): Sunderam from Emory University presented PVM, the nonobject-oriented and conventional addressing based extension to some distributed operating system. PVM enables a network of different machines to act as one virtual heterogeneous machine [8].

PVM is composed of PVM daemon routines and the PVM library of interface routines. PVM daemon routines provide communication by message passing, as well as synchronization by locks and barriers. PVM library routines interact with PVM demon routines to enable transparency of differences between architectures in the underlying heterogeneous suite. Parallelism characterization is manual and results are of matrix style. Resource allocation discipline is dynamic/distributed/cooperative with no task migration allowed. Recently, Beguelin *et al.* proposed the Xab tool for debugging of heterogeneous applications, as well as the Hence tool that supports development of applications and resource allocation [54]. Xab and Hence both run at the top of PVM. The Xab tool is composed of three parts: Xab library, PVM process abmon, and display process xab. Each routine from Xab library is based on one PVM library routine and performs the same thing as the PVM routine would plus generates appropriate Xab tracing message. These tracing messages are sent to the monitoring process, abmon, which has to be active at the time. Abmon translates these messages into human-readable form. Tracing messages contain information like event type, time stamps (in microseconds), and event-specific information. Xab does not rely on synchronized clocks, events are displayed as they arrive. Each message is appended with a serial number and the number of user message so each message can be associated with the process which initiated it. Also, time stamps can be used to determine how long it takes events to propagate from a user process to the monitor process. The Hence tool requires two inputs: high-level graphical representation of both an application and the heterogeneous network. High-level representation is a form of enhances TFG. Besides tasks and data dependencies, it provides control constructs, such as ones for looping of a subgraph or ones for conditional dependency constructs. The cost matrix is supplied by the user and represents the result of a manually performed parallelism characterization phase. This matrix is used at run time to determine the most effective machine on which a particular procedure is to going to be executed. Differently said, resource allocation discipline is dynamic. Once the resource allocation is performed the PVM program is automatically generated. PVM tasks are compiled for selected architectures and PVM routines are provided for the communication and synchronization. The goals of Hence, besides application composition and resource allocation, are tracing the execution flow within the application and localization of the application execution bottlenecks.

PVM system can be obtained via e-mail, so it has the potential of becoming a worldwide tool for heterogeneous parallel computing. Disadvantages of this system are manual parallelism characterization and disallowed task migration.

2) Heterogeneous By Design (HBD): LaRowe and Probert from Center for High Performance Computing at Worcester Polytechnic Institute presented a preliminary design of HBD, the nonobject-oriented conventional addressing based system which supports heterogeneity on the operating system level [9].

The system was planned to be based on the heterogeneous version of the Galactica Net mesh architecture. This architecture consists of electrical wires, fiber optics, Mesh Router Chips (MRC), and Calactica Net Interface Modules (GNIM). GNIM's are placed between the network and each type of computer present in the system to enable effective data conversion. HBD is based on Mach distributed operating system. System software tools are placed on the top of Mach, to support heterogeneous operation, using an analogy with a scientific team working on a multidisciplinary project: colleagues working on its specialties (colleagues are HBD terms for tasks), communicating via messages or shared files. Parallelism characterization is manual, and results are of matrix style (since several machines are

candidates for the execution of a colleague, similar to PVM). Regarding resource allocation, the authors describe it as associative matching scheme performed in the local HBD server and based on load, availability, and locality. From this, we can conclude that the allocation scheme is dynamic/distributed/cooperative.

HBD is a flexible system, supporting programming in many languages and enabling message passing as well as shared memory. HBD also provides access to knowledge bases used in application tuning, parallelism characterization, and resource allocation. Fully automatic orchestration tools are not developed yet.

3) Virtual Heterogeneous Associative Machine (VHAM): Potter from Kent University presented the nonobject oriented associative addressing based system with heterogeneity support managed on the operating system level [48]. This system is further improved in [24].

When data sets are very large it might be more efficient to move computation, rather than data, across the network [55]. In VHAM, data are initially placed on the most appropriate machines and not moved later. A pair of data set and appropriate machine constitutes the basic computation unit, referred to as a cell. Since data sets are identified by content rather than address, all instructions have to be broadcast. Broadcasting is an expensive operation, so instructions have to be powerful such as *matrix–multiply, convolve, histogram*, etc. If the *Node Monitor* module does not match resident data in a cell with incoming instruction, the instruction is rejected. Several machines could be eligible for the execution of the currently broadcast instruction. In that case the *Execution Arbitrator* module (in every node) performs an arbitration based on both the machine execution mode and system state parameters. The arbitration is implemented through the specific low-cost broadcast voting scheme. Fine-grain parallelism detection and parallelism characterization are implicitly present in the process of designing VHAM instructions for particular applications. Since data are initially placed on the most appropriate machine, the key issue is data placement, which influences task scheduling. Node Monitor is responsible for accepting/rejecting instructions, while Execution Arbitrator decides which of two or more suitable machines will execute the received instruction. Node Monitor and Execution Arbitrator together determine the place of the execution of VHAM instruction. Since VHAM instructions are actually tasks, operation that Node Monitor and Execution Arbitrator perform together represents resource allocation by definition. Consequently, we can conclude that allocation is dynamic/distributed/cooperative. Data are static, but intermediate results can be transferred as parameters.

The advantages of this system are easy programming by nonspecialists, dynamic adding and removing of instructions, expandability, etc. Employed instruction broadcasting should not be considered as disadvantage in environments characterized by very large data sets and high communication costs, because transfer of large data sets would be much more expensive than instruction broadcasting.

4) Actors: Agha and Panwar from University of Illinois at Urbana–Champaign suggested the object-oriented conventional addressing based system with heterogeneity support primitives being migrated to the language level [32].

Actors are active system components that control passive components such as objects. An actor implements a state machine where each state is described by content of state variables and a behavior (the set of operations performed in reaction to a received message). One actor can concurrently invoke actions on more than one object, so the coarse-grained concurrent object-oriented programming is efficiently supported. Each actor can communicate with other actors by placing messages in their mailboxes. Actor-based languages rely on user-supplied hints that could be used for parallelism characterization and resource allocation, for example, concurrency index, grain size, computation/communication ratio, isoefficiency functions, task priority, etc. A vector/symbolic parallelism characterization scheme is suggested. The latest versions include the support for dynamic/distributed/cooperative resource allocation with task migration [56]. Parallelism characterization and resource allocation are performed using constructs referred to as meta-level actors (actors defined on the system level). For example, when a message requiring task creation is detected by the meta-level actor *mailq*, the *eval–architecture* meta-level actor selects appropriate hardware with a help of the knowledge database resident in the *HCS–config* actor. Finally, a new actor is created on the selected machine by the *creation* meta-level actor.

The actor-based language presented here is able to express necessary requirements for heterogeneous computing in a modular fashion effectively hiding the specific implementation details. The most notable disadvantage of the system suggested here is that parallelism characterization and resource allocation are fully dependent on the user-supplied hints.

5) Linda: Carriero *et al.* from Yale University show how Linda, the associative addressing based coordination language for parallel processing, which has both object-oriented and nonobject-oriented versions, can be used in a heterogeneous environment [50].

Linda, as other coordination languages, provides mechanisms for process creation and interprocess communication, and thus extends certain sequential language into a parallel programming language. Parallelism characterization is performed manually. Tasks in an application developed in Linda are dynamically partitioned into as many blocks (each block requires the process creation on a processor at run time) as there are available processors in a suitable machine. Communication in Linda is based on the notion of the tuple space. Each datum (or program code) has to be placed to or taken from the tuple space. Tuples are associately addressed and atomically accessed. Both functionally parallel (various forms of MIMD) and data parallel (various forms of SIMD) processing can be expressed in Linda.

Transferring of each datum into the tuple space could result in significant communication overheads. Consequently,

Table 1 Summary of Systems' Features

	Parallelism Characterization	Resource Allocation	Data Conversion	Data Access	Development Approach	Heter. Support Level
Cray-2 + CM-2	manual (vector style)	manual	hw	Address	nonobject	net
Nectar	manual	*	sw	Address	nonobject	net
VISTAnet	manual (vector style)	manual	sw (hw assistance planned)	Address	nonobject	net
HBD	manual (matrix style)	dynamic/distributed/ cooperative	hw	Address	nonobject	opsys
PVM	manual (matrix style)	dynamic/distributed/ cooperative no reassignment	sw	Address	nonobject	opsys
VHAM	partly manual (matrix/dynamic/implicit style)	dynamic/distributed/ cooperative	*	Assoc.	nonobject	opsys
p4	manual	dynamic, load balancing	sw	Address	nonobject	opsys
actors	partly manual (vector/symbolic style)	dynamic/distributed/ cooperative	sw	Address	object	lang
Linda	manual	dynamic replication	sw	Assoc	object and nonobject	lang
Paralation Lisp	manual	dynamic, load balancing, no reassignment	sw	Address	object	lang

Linda is better suited for the computation-intensive coarse-grain processing.

6) Summary of Information: Besides the system features visible from the given 3-D taxonomy, there are some other very important characteristics mentioned in each of the examples. These characteristics are extracted from the presented examples, as well as from some examples not mentioned here, and organized n the Table 1. The asterisk (*) is used as a wildcard wherever no information is found in the open literature.

D. Applications

The class of applications which initiated the research on heterogeneous computing is very wide. Most of them are dominantly related to scientific environment. Typical applications from this class are image processing [57], particle tracing [58], simulation of mixing in turbulent convection [59], acoustic beam forming [60], climate modeling [25], etc. A more complete list can be found in [1]. Recently, a number of multimedia applications [26] have appeared. It could be expected that applications from this class will come out from laboratory environments and will achieve full commercial success. In this subsection, we describe several representative heterogeneous applications, their computational requirements, and systems used for their execution.

1) Image Processing: A typical image processing application is constituted of three tasks. in the first task, pixel-level operations, such as convolution, are carried out. The same instruction is issued for every pixel making SIMD machines suitable for this task. In the second task, operations such as matching, grouping, and splitting of objects are performed. There is substantial amount of data parallelism with lots of branching, which makes SPMD machines the most suitable for this task. The high-level parallelism is concerned with the knowledge-based processing. This kind of processing is truly MIMD in nature. Image processing had significant impact on development of several heterogeneous systems, such as PASM [4] and image understanding architecture (IUA) [57].

2) Simulation of Mixing in Turbulent Convection: This application is constituted of four tasks. Klietz *et al.* [59] show how these tasks are executed on the MEMM class system that consists of four different supercomputers connected by the HIPPI interface. The first task is responsible for the specification of parameters such as temperature and flow function in parts of the $128 \times 128 \times 64$ three–dimensional (3-D) space used for the modeling of the environment. The task requires the SIMD-type computation and is executed on a CM-5 machine. The second task is responsible for the solution of differential equations resulting in particle traces. This task is executed fast on vector supercomputers, such as a Cray 2. The third task makes the voxel file (voxel is a 3-D pixel) from the obtained particle traces. This task also requires the SIMD-type computation, and is executed on a CM-200 machine. Finally, the fourth task graphically shows the voxel file on the a graphical workstation.

3) Climate Modeling: This climate modeling application, referred to as Coupled General Circulation Model (CGCM), processes the time-averaged parameters like wind stress, heat, water fluxes, and sea surface temperature [25]. CGCM can be split into three components: Oceanic General Circulation Model (OGCM), Atmospheric General Circulation Model/Physics (AGCM/P), and Atmospheric General Circulation Model/Dynamics (AGCM/D). OGCM and AGCM/D run fast on vector machines, such as CrayC90 or Cray Y-MP, while AGCM/P is well suited for par-

allel machines, such as IBM SP1 or Thinking machines CM-5. Selected machines were connected by the Casa gigabit network. The authors show that the selection of the suitable computer architectures and the overlapping of computation and communication can provide superlinear speedup.

4) Multimedia Queries: The processing of multimedia queries [26] involves the extraction of video clips that show one particular group of persons (and/or objects), connected by particular relations, and doing particular actions in specified time intervals (i.e., show all home runs scored by St. Louis Cardinals in the 1982 World Series). This application is composed of two consecutive tasks. The first task analyzes high-level representations of faces, pictures, and sounds (present in knowledge bases and data dictionaries). Coarse-grained recognition of these data are performed to generate preliminary responses (hypotheses) to a query. The algorithms present in the first task can be efficiently executed on MIMD machines. The second task is responsible for the fine-grain recognition of faces and pictures as well as the intra-frame and inter-frame analysis of their relative position. These computations are performed on low-level (raw) multimedia data for the final verification of the result hypotheses obtained in the first task. The algorithms present in the second task exhibit the SIMD-type computation characteristics. An example of the heterogeneous system used for multimedia query processing is given in [26]. It is the MEMM class system constituted of one Mas-Par MP-1 machine (SIMD), one nCube machine (MIMD), two multimedia servers, a video camera, and display devices, all connected by a fast fiber-optic network.

IV. Conclusions

The applications exhibiting heterogeneous computational requirements, such as "Grand Challenges" applications, could achieve even superlinear speedup using the heterogeneous computing paradigm. This fact especially motivates recent research and development in the heterogeneous computing world. To design a heterogeneous system, a number of concepts and techniques has to be fully understood and implemented. Parallelism detection, parallelism characterization, and resource allocation as three most important concepts of the heterogeneous application processing have been selected for the presentation in this survey. Parallelism detection is present in homogeneous systems as well, so we did not discuss it in depth. On the other hand, parallelism characterization is the exclusive characteristic of heterogeneous computing. All the schemes for parallelism characterization are presented in the context of an original hierarchical taxonomy. Several resource allocation approaches were selected and presented according to the taxonomy scheme obtained by slight extension and modification of the existing C-K taxonomy. Existing heterogeneous systems are viewed in the context of the introduced EM^3 taxonomy of computer systems. The MESM and the MEMM class systems are considered to exhibit full heterogeneity. Consequently, they are covered in depth in this survey. However, significantly more space was devoted to the promising MEMM class systems. System software issues are presented according to an orginally developed 3-D taxonomy. This taxonomy scheme enables the prediction of appearance of systems in currently empty classes. Finally, the most important heterogeneous applications, which will dominate the future research in the field of heterogeneous computing, are discussed.

References

[1] *Grand Challenges: High Performance Computing and Communications,* Committee on Phys. Mathemat. and Eng. Sci., Nat. Sci. Found., 1991.

[2] D. W. Watson *et al.,* "A framework for compile-time selection of parallel modes in an SIMD/SPMD heterogeneous environment," *Proc. WHP '93,* Newport Beach, CA, Apr. 1993, pp. 57–64.

[3] I. Ekmečić, T. Tartalja, and V. Milutinović, "EM^3: A contribution to taxonomy of heterogeneous computing systems," *IEEE Computer,* vol. 28, pp. 68–70, Dec. 1995.

[4] H. H. Siegel, T. Schwederski, J. Kuehn, and N. J. Davis IV, "An overview of the PASM parallel processing system," *Computer Architecture, Tutorial,* D. D. Gajski *et al.*, Eds. Los Alamitos, CA: IEEE Comp. Sci. Press, May 1986, pp. 387–407.

[5] M. C. Sejnowski *et al.,* "An overview of the Texas reconfigurable array computer," in *Proc. 1980 AFIPS National Comput. Conf.,* AFIPS Press, Arlington, VA, May 1980, pp. 631–641.

[6] C. G. Herter, T. M. Warschko, W. F. Tichy, and M. Philippsen, "Triton/1: A massively-parallel computer designed to support high level languages," in *Proc. WHP '93,* Newport Beach, CA, Apr. 1993, pp. 65–70.

[7] E. A. Arnould *et al.,* "The design of Nectar: A network backplane for heterogeneous multicomputers," in *Proc. ASPLOS III,* Boston, MA, Apr. 1989, pp. 205–216.

[8] V. S. Sunderam, "PVM: A framework for parallel distributed computing," *Concurrency: Practice & Experience,* vol. 2, no. 4, pp. 315–339, Dec. 1990.

[9] R. P. LaRowe, Jr. and T. H. Probert, "Heterogeneous by design: An environment for exploiting heterogeneity," in *Proc. WHP '93,* Newport Beach, CA, Apr. 1993, pp. 84–91.

[10] R. F. Freund *et al.,* "SmartNet-A PVM based management system for near optimal scheduling of very heterogeneous HPC resources (machines and networks)," *Proc. Supercomputing '93,* Portland, OR, Nov., 1993, p. 617.

[11] T. Casavant and J. Kuhl, "A taxonomy of scheduling in general-purpose distributed computing systems," *IEEE Trans. Computers,* vol. 14, pp. 141–154, Feb. 1988.

[12] D. S. Stevenson and J. G. Rosenman, "VISTAnet: Gigabit network testbed," *IEEE J. Select. Areas Commun.,* vol. 10, Dec. 1992, pp. 1413–1420.

[13] R. F. Freund and H. J. Siegel, "Guest editors' introduction: Heterogeneous processing," *IEEE Computer,* vol. 26, Los Alamitos, CA, Apr. 1993, pp. 13–17.

[14] R. F. Freund and V. S. Sunderam, "Special issue on heterogeneous processing, Guest editors' introduction," *J. Parallel and Distrib. Comput.,* vol. 21, pp. 255–256, June 1994.

[15] M. Ercegovac, "Heterogeneity in supercomputing architectures," *Parallel Computing,* vol. 7, pp. 367–372, 1988.

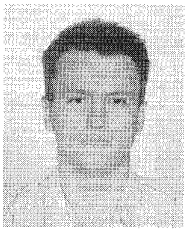
[16] A. A. Khokhar, V. Prasanna, M. E. Shaaban, and C.-E. Wang, "Heterogeneous supercomputing: Problems and issues," *Proc. WHP ;92,* Beverly Hills, CA, Mar. 1992, pp. 3–12.

[17] ——, "Heterogeneous computing: Challenges and opportunities," *IEEE Computer,* vol. 26, pp. 18–27, June 1993.

[18] C. C. Weems, G. E. Weaver, and S. G. Dropsho, "Linguistic support for heterogeneous parallel processing," in *Proc. HCW,* Cancún, Mexico, Apr. 1994, pp. 81–88.

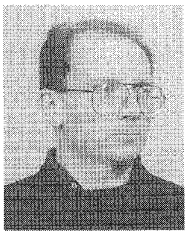
[19] H. J. Siegel *et al.,* "The goals of and open problems in high-performance heterogeneous computing," in *Proc. 23rd AIPR*

Workshop on Image and Inform. Syst. Applicat. and Opportunities, SPIE, Oct. 1994.
[20] J. Yang, I. Ahmad, and A. Ghafoor, "Estimation of execution times on heterogeneous supercomputer architectures," in *Proc. Int. Conf. on Parallel Proces.,* pp. I-219–I-226, 1993.
[21] J. Yang, A. A. Khokhar, H. Sheikh, and A. Ghafoor, "Estimating execution time for parallel tasks in heterogeneous processing (HP) environment," in *Proc. HCW '94,* IEEE CS Press, Los Alamitos, CA, Apr. 1994, pp. 23–28.
[22] B. Narahari, Y. Abdou, and H.-A. Choi, "Matching and scheduling in a generalized optimal selection theory," in *Proc. HCW '94,* Cancun, Mexico, Apr. 1994, pp. 3–8.
[23] DeSouza–Batista *et al.,* "A suboptimal assignment of application tasks onto heterogeneous systems," in *Proc. Heterogeneous Computing Workshop (HCW),* Cancun, Mexico, Apr. 1994, pp. 9–16.
[24] S. L. Scott and J. Potter, "A framework for the virtual heterogeneous associative machine," in *Proc. HCW '94,* Cancun, Mexico, Apr. 1994, pp. 97–102.
[25] C. R. Mechoso, J. D. Farrara, and J. D. Spahr, "Achieving superlinear speedup on a heterogeneous, distributed system," *Parallel and Distrib. Technol.,* vol. 2, no. 2, IEEE CS Press, Los Alamitos, CA, Summer '94, pp. 57–61.
[26] A. A. Khokhar and A. Ghafoor, "A heterogeneous processing (HP) framework for multimedia query processing," in *Proc. HCW '94,* Cancun, Mexico, Apr. 1994, pp. 51–57.
[27] A. Ghafoor and J. Yang, "A distributed heterogeneous supercomputing management system," *IEEE Computer,* vol. 26, no. 6, June 1993, pp. 78–87.
[28] S. Chen, M. M. Eshagian, A. A. Khokhar, and M. E. Shaaban, "A selection theory and methodology for heterogeneous supercomputing," in *Proc. Workshop on Heterogeneous Processing (WHP '93),* Newport Beach, CA. Apr. 1993, pp. 15–22.
[29] R. F. Freund, "Optimal selection theory for superconcurrency," *Supercomputing,* Nov. 1989.
[30] M. Wang *et al.,* "Augmenting the optimal selection theory," in *Proc. WHP '92,* Beverly Hills, CA, Mar. 1992, pp. 13–22.
[31] D. Pease *et al.,* "PAWS: A performance evaluation tool for parallel computing systems," *IEEE Computer,* vol. 24, no. 1, pp. 18–29, Jan. 1991.
[32] G. Agha and R. Panwar, "An actor-based framework for heterogeneous computing systems," in *Proc. Workshop on Heterogeneous Processing '92 (WHP '92),* Beverly Hills, CA, Mar. 1992, pp. 35–42.
[33] S. Hambrush and A. A. Khokhar, "An architecture-independent model for coarse-grained parallelism," Tech. Rep., Computer Sci. Dept., Purdue Univ., Oct. 1993.
[34] D. Watson *et al.,* "A block-based mode selection model for SIMD/SPMD parallel environments," *J. Parallel and Distrib. Computing,* vol. 21, no. 3, pp. 271–287, June 1994.
[35] T. Berg and H. J. Siegel, "Instruction execution trade-offs for SIMD vs. MIMD vs. mixed-mode parallelism," in *Proc. 5th Int. Parallel Process. Symp.,* IEEE CS Press, Los Alamitos, CA, order no. 2167, May 1991, pp. 301–308.
[36] E. Moore, "The shortest path through a maze," *Int. Symp. Theory of Switching,* 1957, pp. 285–292.
[37] M. A. Iqbal, "Partitioning problems in heterogeneous computer systems," in *Proc. WHP '93,* Newport Beach, CA, Apr. 1993, pp. 23–28.
[38] V. M. Milutinović, J. J. Crnković, L.-Y. Chang, and H. J. Siegel, "The LOCO approach to distributed task allocation in AIDA by VERDI," in *Proc. 5th IEEE Int. Conf. on Distrib. Computing Syst.,* Denver, CO, May 1985, pp. 359–368.
[39] L. Tao, B. Narahari, and Y. C. Zhao, "Heuristics for mapping parallel computations to heterogeneous parallel architectures," in *Proc. WHP '93,* Newport Beach, CA, Apr. 1993, pp. 36–41.
[40] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness.* New York: Freeman, 1979.
[41] W. B. Ligon III and U. Ramachandran, "Evaluating multigauge architectures for computer vision," *J. Parallel and Distrib. Computing,* vol. 21, no. 3, pp. 323–333, June 1994.
[42] M. Auguin and F. Boeri, "The OPSILA computer," *Parallel Languages and Architectures,* Consard, M., Ed. New York: Elsevier, 1986, pp. 143–153.
[43] N. P. Jouppi, J. Bertoni, and D. W. Wall, "A unified vector/scalar floating-point architecture," in *Proc. ASPLOS III,* Boston, MA, Apr. 1989, pp. 134–143.
[44] J. B. Armstrong, D. W. Watson, and H. J. Siegel, "Software issues for the PASM parallel processing system," in *Software for Parallel Computation,* J. Kowalik and L. Grandinetti, Eds. Berlin: Springer-Verlag, 1993, pp. 134–148.
[45] "Special Report: Gigabit network testbeds," *IEEE Computer,* vol. 23, no. 9, pp. 77–80, Sept. 1990.
[46] E. C. Cooper and R. P. Draves, "C Threads," Tech. Rep. CMU-CS-88-154, Computer Sci. Dep., Carnegie Mellon Univ., June 1988.
[47] R. J. Vetter, D. H. C. Du, and A. E. Klietz, "Network supercomputing experiments with a CRAY-2 to CM-2 HIPPI connection," *Proc. WHP '92,* Beverly Hills, CA, Mar. 1992, pp. 87–92.
[48] J. Potter, "Heterogeneous associative computing," in *Proc. WHP '93,* Newport Beach, CA, Apr. 1993, pp. 3–11.
[49] R. Butler, W. Gropp, and E. Lusk, "Developing applications for a heterogeneous computing environment," in *Proc. WHP '93,* Apr. 1993, pp. 77–83.
[50] N. Carriero, D. Gelernter, and T. G. Mattson, "Linda in Heterogeneous Computing Environments," in *Proc. WHP '92,* Beverly Hills, CA, Mar. 1992, pp. 43–46.
[51] D. J. Batey and J. A. Padget, "Toward a virtual multicomputer," in *Proc. Workshop on Heterogeneous Process.,* Newport Beach, CA, Apr. 1993, pp. 71–76.
[52] I. Foster, R. Olson, and S. Tuecke, "Productive parallel programming: The PCN approach," *Scientif. Programming,* 1993.
[53] J. R. Nicol, C. J. Wilkes, and F. A. Manola, "Object orientation in heterogeneous distributed computing systems," *IEEE Computer,* vol. 26, no. 6, pp. 57–67, June 1993.
[54] A. Beguelin, J. Dongarra, A. Geist, and V. S. Sunderam, "Visualization and debugging in a heterogeneous environment," *IEEE Computer,* vol. 26, no. 6, pp. 88–95, June 1993.
[55] R. Vetter and D. H. C. Du, "Distributed computing with high-speed optical networks," *IEEE Computer,* vol. 26, no. 2, pp. 8–18, Feb. 1993.
[56] R. Panwar and G. Agha, "A Methodology for programming scalable architectures," *J. Parallel and Distrib. Computing,* vol. 22, pp. 479–487, 1994.
[57] C. C. Weems, "Image understanding: A driving application for research in heterogeneous parallel processing," in *Proc. WHP '93,* Newport Beach, CA, Apr. 1993, pp. 119–126.
[58] F. J. Rinaldo and M. R. Fausey, "Event reconstruction in high-energy physics," *IEEE Computer,* vol. 26, no. 6, June 1993, pp. 68–77.
[59] A. E. Klietz, A. V. Malevsky, and K. Chin-Purcell, "A case study in metacomputing: Distributed simulations of mixing in turbulent convection," in *Proc. WHP '93,* Newport Beach, CA, Apr. 1993, pp. 101–106.
[60] C. E. Lee and D. Sullivan, "Design of a heterogeneous parallel processing system for beam forming," in *Proc. WHP '93,* Newport Beach, CA, Apr. 1993, pp. 113–118.



Ilija Ekmečić received the B.S.E.E. degree in 1991 from the University of Sarajevo, Sarajevo, Bosnia and Herzegovina.

Since 1992 he has been with the Department of Computer Engineering at the University of Belgrade. During 1991–1992 he was with the Laboratory for System Analyses (LASA), School of Electrical Engineering, University of Sarajevo, where he worked on the use of non-standard logical systems in compiler construction. His current research interests include resource allocation in heterogeneous computing systems (primarily MEMM class), reconfigurable architectures, and networking. He authored and/or coauthored ten conference and journal papers.



Igor Tartalja received the B.S.E.E. and M.S.E.E. degrees from the University of Belgrade, Yugoslavia, in 1984 and 1989, respectively. He is presently completing the Ph.D. degree at the same university.

He is presently with the Department of Computer Engineering at the University of Belgrade. From 1984 to 1989 he was with the Laboratory for Computer Engineering, Institute for Nuclear Sciences, Vinča, Serbia, Yugoslavia, where he worked on the development of a real-time computer for applications in biophysics as well as a distributed operating system for a special-purpose multicomputer. His current research interests include multiprocessor and multicomputer architectures, heterogeneous processing, and system software support for shared-memory multiprocessors and distributed systems. He has authored or co-authored about 20 conferences or journal papers and on book on shared memory systems.

Veljko Milutinović (Senior Member, IEEE) received the Ph.D. degree from the University of Belgrade, Yugoslavia, in 1982.

He is on the faculty of the School of Electrical Engineering, University of Belgrade, Serbia, Yugoslavia, since 1990. Prior to that, he was a faculty member of the School of Electrical Engineering at Purdue University, West Lafayette, IN. His R&D results include a commercial 16–node MISD machine for DFT processing developed at IMP, the architecture of an early 200 MHz RISC microprocessor for RCA, several multimedia PC-oriented multiprocessor concepts for NCR, and several DSM system level solutions for Encore. He has been actively researching heterogeneous computing since the early 1980's, when he introduced a dynamic/distributed/noncooperative/competitive/load-sharing algorithm for resource allocation. He has authored over 40 papers in IEEE journals and presented over 100 invited lectures worldwide.

Dr. Milutinović received several awards for his conference papers.