NASA Technical Memorandum 4103

Automating the Multiprocessing Environment

Dale J. Arpasi
Lewis Research Center
Cleveland, Ohio

National Aeronautics and
Space Administration
Office of Management
Scientific and Technical
Information Division

1989

Summary

An approach to automate the programming and operation of tree-structured networks of multiprocessor systems is discussed. A conceptual, knowledge-based operating environment is presented, and requirements for two major technology elements are identified as follows: (1) An intelligent information translator is proposed for implementing information transfer between dissimilar hardware and software, thereby enabling independent and modular development of future systems and promoting a language-independence of codes and information; (2) A resident system activity manager, which recognizes the systems capabilities and monitors the status of all systems within the environment, is proposed for integrating dissimilar systems into effective parallel processing resources to optimally meet user needs. Finally, key computational capabilities which must be provided before the environment can be realized are identified.

Introduction

Current operating system technology (e.g., DOS) provides only primitive user interfaces that are designed to furnish basic peripheral services for information and task management. More complex tasks are considered the responsibility of the user-selected application software package. Consequently, these packages tend to be autonomous, each with independent information formats (although some external integration may be possible through the use of standard file formats). This type of computing environment functions efficiently if the following conditions hold true:

(1) The application software packages make effective use of the available hardware.

(2) The applications (either standing alone or in conjunction with the DOS) provide sufficient integration of services and information to meet user requirements.

(3) The user is properly versed in the operation of the DOS and the applications to make efficient use of the environment.

These conditions are usually met by providing internally integrated, multifunction software packages, designed for a specific computer. These minimize the need to port information outside of the package and provide a common, structured, high-level interface to the functions. Unfortunately, the need to integrate functions and information between different packages cannot be completely avoided without imposing a shell of self-sufficiency around each user, thereby restricting technology utilization and the free interchange of information. Another problem with the internally integrated, multifunction approach is that it can stifle upward mobility to new technology software by imposing cost and effort penalties in the areas of porting existing data files to replacement software, and in the retraining of personnel. Similarly, upward mobility to new technology computing hardware is equally difficult if existing software packages cannot be easily transported.

The problems of functional integration and upward mobility are magnified when multiprocessing systems are considered. Achieving good performance on a multiprocessing system requires a good data-flow match between the computational activity and the computing hardware. In a multiuser, multitasking, multiprocessing environment, efficient use of the available processors could require different allocations of the activity's computations depending on the overall computational load. This would require an automated software transportation mechanism that insures maximum computational efficiency for whatever hardware is available. Since the computational activity may be performed on different processors and over different data paths at different times, functional integration is impossible without a good deal of intelligence built into the system. These additional complexities and requirements cannot be foisted on the internally integrated multifunction approach to a computational environment without increasing the costs associated with upward technical mobility to a great degree. Clearly a more modular and intelligent approach to the establishment of future computational environments is needed.

Knowledge-based system technology can be applied to the development of a multiprocessing environment to overcome these problems. For example, a learning capability can be provided within the environment to automatically adapt software to the available hardware. A communication mechanism, using software-specific knowledge bases, can be incorporated to allow functional integration of any computational activities. These technologies would consequently lead to lower cost, special purpose software packages and consequential improvements in upward mobility to new technology. With knowledge-based technologies, user interfaces can evolve to support programming languages of choice rather than having the system impose a common language on all users. Finally, by developing "smart" interfaces, information can be freely transmitted across utility and system boundaries. This approach can enable the integration of computing systems into a single, efficient problem solving unit.

Environment Overview

It is assumed that a universal sympathetic environment (USE) will be built around a tree-structured multiprocessor configuration as shown in figure 1. It is further assumed that each branch of the tree is a multiprocessor system containing a control processor (C) and a number of service processors (S). Each service processor may, in turn, be another branch (multiprocessor system) of the tree. The domain of the USE is therefore limited only by the communication paths provided to it. It is open ended, thereby allowing domain extension or retraction. Each control processor in the tree structure is assumed to have complete control over its local resources (i.e., all of the other processors and systems which branch out from it, where system refers to a control processor and all of its local resources).

Each control processor controls the computations of its local resources in response to the requests of control processors farther down the tree and any local user requests. Systems may request the services of other systems in other parts of the tree by passing these requests up through the chain of control until the control for the desired resource is encountered. This tree-structured multiprocessing concept is sufficiently general to be a superset of most practical configurations and will be the basis for USE design.

Before proceeding, certain terminology must be established. A utility is a component of the USE which provides a specific service. An activity is an application (e.g., word processing) that the user brings to the environment. The activity interfaces to the computing system as one or more computational jobs. These jobs consist of linked tasks which may preexist in a library. The tasks consist of parallel computational paths with data transfer between them. The paths translate furnished arguments into results. Each path contains operations which must be performed serially. The operations are generic (e.g., add and multiply) and must be translated into machine code for a particular processor. Thus, one task could have alternative path representations to ease translation to different processor types (e.g., vector and scalar).

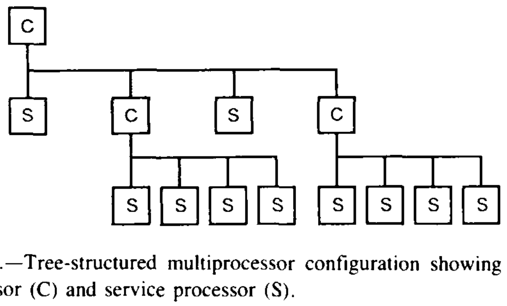


Figure 1.—Tree-structured multiprocessor configuration showing control processor (C) and service processor (S).

The USE must provide three major areas of service as follows: (1) computational services, to manage user activities; (2) information management services, to maintain and disseminate information; and (3) environment adaptation services, to build and maintain the knowledge base.

Computational Services

Figure 2 shows the major utilities in the computational services area. Each control processor in the USE will interface its resources to the others through these services. The services are targeted, through knowledge bases, to meet specific local requirements. The operational interface (OI) has the responsibility of controlling the flow of information between the operating system and the local peripherals (disks, keyboards, and terminal), which exist at each particular branch of the multiprocessing tree. If a user has direct access to a control processor then he/she may initiate and interact with computational activities on that processor through the OI. These activities may use the local peripherals and any or all of the multiprocessing resources of the USE to perform their functions.

Activities are always buffered by the information translator (IT). This utility provides translation or interpretation of information being transmitted among the local peripherals (including the user), the available computational resources, and the activity. As shown in figure 2, four relational data bases (translators A, B, C, and D) govern the translation of information on the four data paths of the activity. Translator

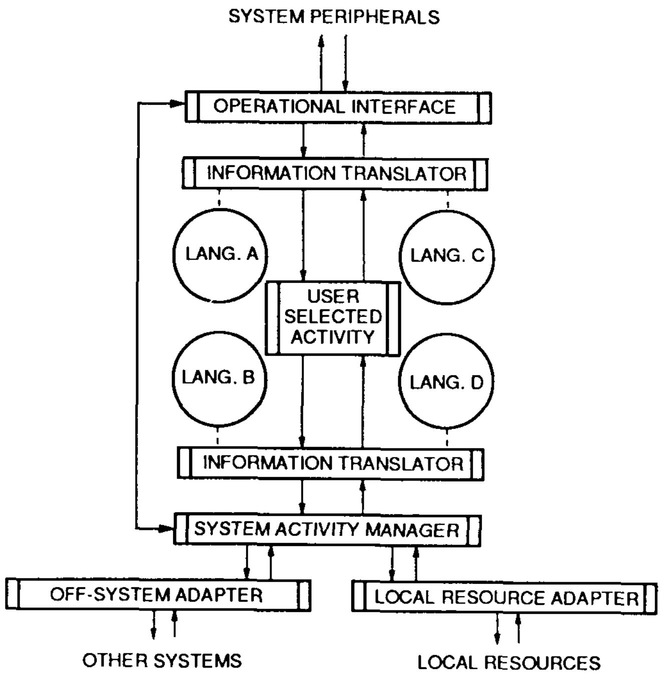


Figure 2.—USE computational services.

A is used to communicate from the user to the activity and is selected by the user to best meet his/her subjective purposes. The IT translates this information into the data structures of the activity. Translator B is used to translate activity data structures into a multiprocessing job language. Results from these jobs are translated back into activity data structures by using translator D. Activity results are sent back to the user in language C, which is selected to meet subjective format requirements. The establishment of all of these relational data bases is simplified by using environmental adaptation utilities provided in the USE. These IT interfaces allow detailed information transfer between activities running on the same or different processors without imposing these requirements on the activities themselves.

The multiprocessing resources of the USE are integrated by implementing the system activity manager (SAM) on each control processor in the environment. This utility performs the following three system integration functions:

(1) Resource management.—The SAM is responsible for optimally assigning multiprocessing resources to perform the computational tasks. This includes the partitioning of jobs and allocating tasks, and off loading those that can best be performed on other systems.

(2) Information distribution.—The SAM is responsible for identifying information transfer requirements necessary to support the computations. It is also responsible for insuring that these transfers, and those necessary to support off-loaded tasks, occur in a timely manner by using the best available information transfer paths.

(3) Operations.—The SAM provides operational control of its resources. It performs data-flow management of its computational tasks and initiates these tasks when their arguments are available. It also provides run-time interaction between the system peripherals (including the user) and these tasks via operational tasks embedded in the job and a direct link to the OI.

The off-system adapter and local resource adapter in figure 2 adapt the SAM to system characteristics. Their functions include data transfer, program loading and sequencing, computational monitoring, and message passing.

Information Management

There are three types of information stored in three libraries within the USE. The libraries are the user library, activity library, and system library.

(1) The user library contains descriptions and results of user applications that are of significance in the formulation of new technology applications. This library is organized according to scientific discipline.

(2) The activity library contains descriptions of the computational activities available to the user, and includes languages and optimized tasks and code modules that can be used with those activities. This library is organized according to multiprocessing system.

(3) The system library contains targeting information (knowledge bases) which define computational hardware, activities, peripherals, and others. The knowledge bases adapt the application of the utilities to specific USE implementations. This library is organized according to system with cross-reference to the activity library.

Each of the libraries are managed by a librarian. The librarian is an activity which functions within the computational services environment described above. It provides the typical electronic library services such as card catalogs, and cross-referencing. It also insures that new volumes of information conform to library standards, and provides translations of existing volumes to meet user, activity, or system needs.

Environment Adaptation

This operating system provides the facilities for developing the USE knowledge base. The following information has been identified as basic to USE operation and must be included in the knowledge base:

(1) Processors
  (a) Hardware: data formats, operations, structure, memory
  (b) Code generator: assembly language, macro definitions
(2) Multiprocessing systems
  (a) Configuration: information paths, control processors
  (b) Task generation: input and output formats, macros
  (c) Peripheral hardware: specifications
(3) Multisystem environment
  (a) Capabilities: information paths, system applications
  (b) Allocation rules: priorities, options, contingencies
  (c) Languages: vocabulary, syntax, semantics, grammar

This information is necessary for everything from the simplest uniprocessing environment to a multisystem environment accessible over a network. Each time the environment is changed or extended, at either the processor or system level, the knowledge base must be updated.

Because of the complexity of the information involved and the need for accuracy, the establishment of the knowledge base will be made as simple as possible. Most of the information required will have to be manually entered the first time. After that, information on similar components can be derived from the system library and edited if necessary. Manual entry of knowledge base information will follow an instructional handbook to ensure completeness. The IT will be used to monitor syntax and to establish the data structures of the knowledge base. Some information can be derived automatically. Execution and data transfer time can be measured by running benchmark programs on the hardware. A degree of adaptability can thereby be established in knowledge-base generation. Benchmarking utilities can also be valuable in establishing such activities as parametric studies of software and hardware, diagnostics, and system emulation. All of these utilities as well as those used in other operating systems will be available for activity development through the system library.

System Activity Manager

The potential for highest multiprocessing efficiency exists in a multi-user, multitasking, multiprocessing environment. Through high data rate networking, multiprocessing environments will eventually consist of loosely coupled networks of multiprocessing systems. The integration of a wide variety of individual multiprocessing systems into that environment, however, requires a new operational approach, in which all capabilities available within the environment are recognized and utilized to efficiently perform all tasks assigned to the network.

For example, let systems $A(1)$, ..., $A(n)$, be optimized to perform all tasks beginning with the letter A; let systems $B(1)$, ..., $B(m)$, be optimized to perform all tasks beginning with the letter B; and let system C be optimized to perform all tasks beginning with the letter C. Consider a job submitted to system C that consists of the following:

```
Begin job
    With the information in files 1, 2, and 3 do tasks Ax, Ay,
      and Bx, respectively;
    With the results of Ax and Bx, do task Cx;
    With the results of Bx, do task By;
    With the results of Ay, By, and Cx, do task Cy;
End job.
```

In a uniprocessing environment consisting only of system C, the A and B tasks would be performed suboptimally. If C could not take advantage of the parallelism in the job, the tasks would also be performed serially.

In an integrated multiprocessing environment, tasks *Ax, Ay*, and *Bx* might be assigned to systems $A(1)$, $A(2)$, and $B(1)$, respectively. If only one A-system was available, C could be assigned the shorter of the A-tasks. System C would then do task *Cx* once its arguments were available, with the $B(1)$ system picking up the *By* task after completing the *Bx* task. Finally, when *Ay, By*, and *Cx* were finished, system C could perform task *Cy* to complete the job. Obviously, the multiprocessing environment would be more efficient than the single C-system (assuming that data transfer times over the network are negligible). Integrating the computational capabilities of multiple systems can lead to other benefits as well. By distributing tasks among several systems, one allows for the development and use of highly efficient, special purpose systems and solvers. Large, general purpose systems can be replaced and new technology can more easily be incorporated into the environment. Maintenance, repair, and upgrade costs should be reduced. Utilization of the computing resources will be higher because of the sharing of workloads.

The challenges associated with developing an integrated, open-ended multiprocessing environment are not insignificant. Aside from the mechanics of high speed data transfers (which will eventually be solved through hardware technology), logic and software need to be developed to automate the prioritizing, scheduling, and other management tasks for arbitrary multiprocessor configurations. Fortunately, many of the concepts of data-flow parallelism, computational packing, and system definition have been developed and demonstrated in the Real-Time Multiprocessor Programming Language (RTMPL) project and are directly applicable to this effort. Using this technology as a starting point, the objectives are achievable.

In the USE, task scheduling is done by the system activity manager (SAM). This utility resides, to some extent, in each system within the environment. A functional diagram of the SAM is given in figure 3. The task level partitioner accepts jobs submitted from a user activity and partitions them into computational paths called S-tasks. (A path here is defined as a series of computations which contains no parallelism). The S-tasks are integrated with S-tasks from other systems and are placed in the S-task queue of pending computations. Embedded tasks, identified by the L-system run-time monitor, are also deposited in the S-task queue for assignment. Embedded tasks are those which are result dependent or which otherwise cannot be predicted from the job specification. The S-task queue is monitored by the intersystem allocator (ISA) facility. This facility considers the status and capabilities of all computational resources available within the USE, including those local resources under its control. Task requirements and resource specifications are available from

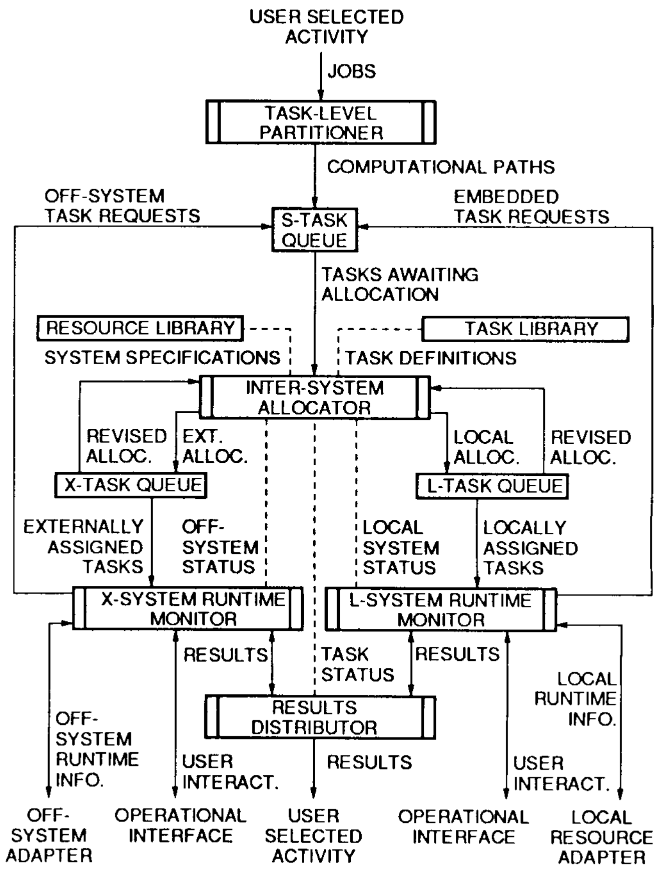


Figure 3.—System activity manager.

the libraries. On this basis, S-tasks are either designated as local tasks and placed in the L-task queue, or as external tasks and placed into the X-task queue. The ISA also conducts a continual review of the status of tasks in these queues (completed or not) and the status of local and external resources (available or not) and may shift S-tasks between the queues, accordingly.

When the tasks in the queues have their arguments available, they are marked as data-flow enabled, thus permitting access by their run-time monitors. The run-time monitors control the execution of assigned tasks on resources under their control. The external system run-time monitor (XRM) is responsible for off-system computations, and the L-system run-time monitor (LRM) is responsible for computations on the local resources. Enabled tasks are selected from the queues on a priority basis (priorities having been previously established during job formulation), and are assigned to available resources. The XRM sends its tasks to the off-system adapter for transmission to other systems. This adapter is the focal point of all intersystem information transfers. The LRM sends its tasks to the local resource adapter for local distribution. Both adapters must be targetable to specific hardware and formats. To this effect, they will use the facilities provided by the information translator. The run-time monitors provide other services as well. They are responsible for furnishing resource status information to the ISA. They also furnish computational results to the results distributor for transmission back to the activity, and integrate results to form arguments for other tasks to meet data-flow requirements. Finally, the run-time monitors provide interactive communication between the user activity (via the operational interface) and its tasks. This communication provides system status information and permits interaction with system operation. Operational interaction with the tasks, such as viewing of intermediate results and the changing of parameters, is accomplished by using predeveloped system tasks in the job specification.

The SAM is responsible for carrying out the efficient utilization of computational resources within its purview (including the control processor on which it resides). The purview of a SAM is based on the resources required to best perform an assigned job and is bounded by the communication paths available. A system cannot use the resources of another system unless a path to those resources exists. Of course, information transfer time must be considered. If that time is very large, then communication is essentially nonexistent. Purview must be subjectively established for each intended application and it must be reconfigurable. This requires that rules be established to optimally structure the environment for each application.

By proper design, the SAM can provide full computational integration and local management of systems within the USE. If the SAM is portable (able to be implemented on any control processor), it can provide a common multiuser, multitasking multiprocessor operating system for the entire environment.

Information Translator

In the USE environment, the passing of information between information processing tasks is automated, as is the interfacing of these tasks, to the SAM. The USE utility which will provide this automation is called the information translator (IT).

A primary requirement of the IT is that it provide the USE processors with a transparency to application programming languages. If it is to be useful to the scientific community, it should, at the very least, accept information in the most common programming languages of Fortran, Pascal, and Ada. Allowing the user to program in a language with which he/she is comfortable should maximize the effectiveness of the user and increase acceptance of the environment. Another requirement of the IT is that it provide extensive diagnostics and programming aids so that it can be a comfortable, error-free user interface. Finally, the IT should be easily retargetable to the informational requirements of any new activity. This feature of the IT could make future activity development simpler by providing a ready made, proven information management system.

The IT design includes a facility, called READ_IT, to convert textural information into the basic data structures of an information processor and a facility, called WRITE_IT, to perform the inverse function. Here, data structure is meant to describe the organization and identification of information that are required before processing can proceed.

Each of the above facilities uses a relational data base which defines the language of communication. The language definition contains vocabulary and usage rules. The vocabulary is linked to classify and otherwise establish semantic relationships. The rules establish syntax requirements and activity specific parsing and grammatical requirements which relate the language text to the data structures of the activity. Parsing rules are used by READ_IT and grammatical rules are used by WRITE_IT. Each time a language is defined, its knowledge base is established and each time it is used by an activity, the knowledge base is expanded.

To understand the functions of the READ_IT and WRITE_IT facilities, consider the elements of a conversation as depicted in figure 4. Information (an idea) is presented by a source processor in terms of its output data structures (mental images). The data structures are translated (spoken) by WRITE_IT into a text statement in the source language. The text statement is translated (heard) by READ_IT into the data structures (mental images) of an object processor (listener). The object processor operates on (thinks about) the data structures, forms a response, (also in terms of data structures) and the process is repeated with WRITE_IT translating data structures into a text statement in the object language.

Even though translation would be unnecessary if the parties were speaking the same language (source language equals object language), interpretation would be required since the "spoken" information would still have to be converted to/from

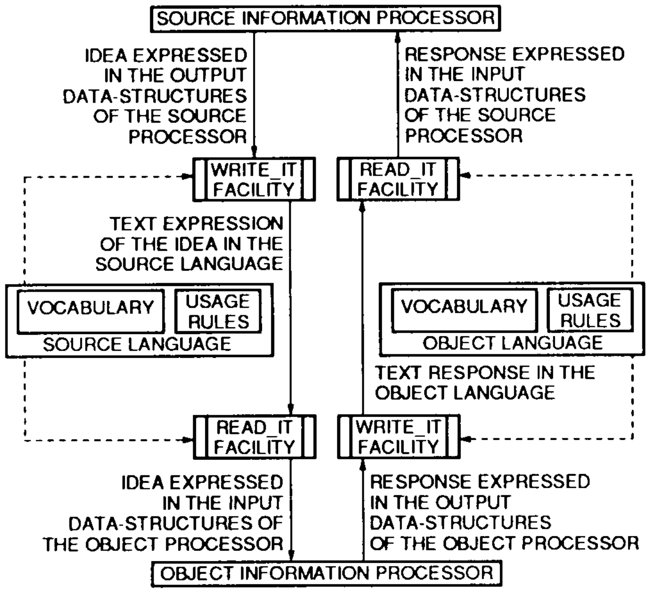
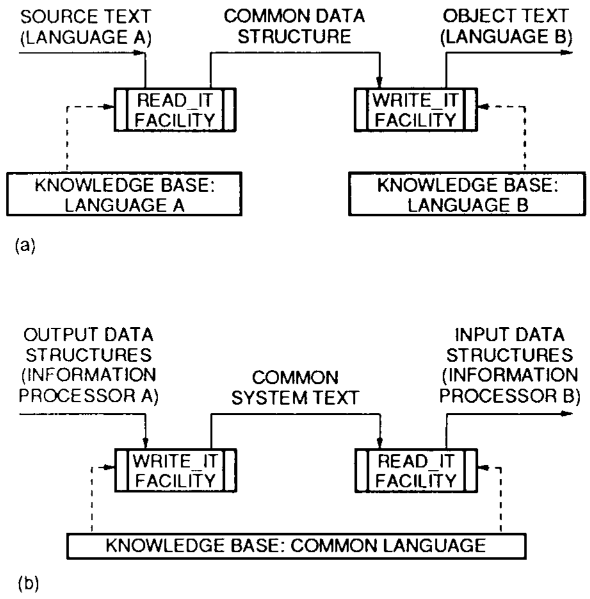

SOURCE INFORMATION PROCESSOR
IDEA EXPRESSED IN THE OUTPUT DATA-STRUCTURES OF THE SOURCE PROCESSOR
RESPONSE EXPRESSED IN THE INPUT DATA-STRUCTURES OF THE SOURCE PROCESSOR
WRITE_IT FACILITY
READ_IT FACILITY
TEXT EXPRESSION OF THE IDEA IN THE SOURCE LANGUAGE
VOCABULARY
USAGE RULES
SOURCE LANGUAGE
VOCABULARY
USAGE RULES
OBJECT LANGUAGE
TEXT RESPONSE IN THE OBJECT LANGUAGE
READ_IT FACILITY
WRITE_IT FACILITY
IDEA EXPRESSED IN THE INPUT DATA-STRUCTURES OF THE OBJECT PROCESSOR
RESPONSE EXPRESSED IN THE OUTPUT DATA-STRUCTURES OF THE OBJECT PROCESSOR
OBJECT INFORMATION PROCESSOR

Figure 4.—A conversation using the IT facilities.


SOURCE TEXT (LANGUAGE A)
COMMON DATA STRUCTURE
OBJECT TEXT (LANGUAGE B)
READ_IT FACILITY
WRITE_IT FACILITY
KNOWLEDGE BASE: LANGUAGE A
KNOWLEDGE BASE: LANGUAGE B
(a)
OUTPUT DATA STRUCTURES (INFORMATION PROCESSOR A)
COMMON SYSTEM TEXT
INPUT DATA STRUCTURES (INFORMATION PROCESSOR B)
WRITE_IT FACILITY
READ_IT FACILITY
KNOWLEDGE BASE: COMMON LANGUAGE
(b)

(a) As a language translator.
(b) As an interpreter.

Figure 5.—Information translator applications.

data structures for information processing. Of course, languages (and therefore interpretation) would be unnecessary and communication would be highly efficient if ideas could be shared directly through data structures (i.e., telepathy).

A form of telepathy is present in complex computer programs that are functionally modularized. There, modules are tied together through a main program and shared common data structures. Information generated by one module is directly usable by the other. In the USE, both telepathic and nontelepathic communication are utilized with groups of telepathic modules (called facilities) working together to form information processing services (called utilities). Tasks are not telepathic. Information translation is always required for communication between utilities and tasks, but not between facilities. The IT provides for information transfer among all nontelepathic information processors in the USE, including the users.

If the IT facilities are used with common data structures (as shown in fig. 5(a)), the IT becomes a translator from one language to another. If these facilities are used with a common language (as shown in fig. 5(b)), the IT becomes an interpreter among the data structures of two or more information processors. Using the latter approach, the IT can effectively interface the data structures of all information processors in a USE system to a single, common language, information channel. Interpretation to/from data structures would be accomplished by using a knowledge base (i.e., relational data base) between the system language and the information processors. If the establishment of this data base can be made simple and straight forward then the following advantages will be available:

(1) Communications between information processors can be carried out in languages that are best suited for that purpose.

(2) New information processors can be incorporated merely by establishing a relational data base between the system languages and the new processor.

(3) Individualized communication interfaces do not have to be developed for each new information processor.

(4) Information storage can be standardized and even minimized by using the IT as an interface to a system wide, minimum length, information storage language.

(5) The IT may be used to communicate between different computer systems, thereby expanding local processing capabilities, promoting information dissemination, and reducing the need for duplication of capabilities.

To summarize, the IT will provide a versatile communication link among information processors. Using the IT as an intermediary provides local independence when selecting a programming language. This should open the door to increased portability of codes while avoiding the need to standardize on software.

Core Activities

The USE is intended to carry out any user selected activity, on whatever computational resources are available, in an optimum fashion. As computational technology advances, users will conceive of more and more sophisticated activities that reflect their knowledge and new ideas.

New activities can be incorporated into the USE as they become available. However, there are certain core activities that must be developed and incorporated in the USE before the environment becomes operational. These include activities associated with knowledge-base development, library development, job specification, and run-time interaction. While current technology may not allow the "best" design of these activities, it is important to put an initial capability in place. Therefore, rudimentary versions will be incorporated into the prototype environment until better replacements are available. The USE design, being open ended and modular, makes this staged approach to activity development reasonable.

The following activities have been identified as being necessary to the establishment of the USE:

(1) Task development.—The development of computational tasks, including multilingual (Fortran and Ada) programming, automated loop decomposition, vectorizing, and path partitioning

(2) Job development.—Providing a job specification interface to the user, enabling direct access to the SAM, and the construction of interactive operational requests that can be integrated with other jobs

(3) Information management.—Controlling, accessing, and maintaining tasks and system information

(4) Environment adaptation.—Developing the knowledge base that can be used to adapt the USE to specific multiprocessing hardware and to meet specific user needs

The Information Management and Environment Adaptation activities are actually USE operating systems which function under the computational services operating system.

The following analytical activities also need to be included in initial USE development:

(1) System emulation.—The development of software simulations of candidate multiprocessing systems to support USE system design, benchmark applications, and to guide research and development

(2) Task evaluation.—Automated timing and measurement of task efficiency, while running on actual or simulated systems, to support development of multiprocessing algorithms

These activities will build on the existing utilities and features of the environment to simplify their development and enhance their operation. Activity development, itself, will provide a test and evaluation of the USE utilities, and will set standards for interfacing to the environment. These standards will then guide others in subsequent efforts to further develop multiprocessing technology.

Concluding Remarks

The multiprocessing environment presented in the previous paragraphs is intended to provide a sophisticated interface between the scientist and state-of-the-art computational technology. It is designed to automate many of the tasks that users of multiprocessor systems must now accomplish manually. The environment is versatile, portable, and upgradable, thereby allowing for improvements in both hardware and software.

It is hoped that the development and demonstration of prototype versions of the environment will encourage the use of multiprocessing and promote the integration of efforts and results from a variety of scientific disciplines. In the area of computational fluid dynamics, for example, the environment should stimulate the development and testing of highly parallel flow solver codes. The environment should also promote code validation by simplifying the exchange of information between analysis and experiment.

Although the technology is not yet available for complete development of this environment, it is hoped that the prototype effort will provide a focus for the development of needed technologies. As progress is made in the areas of parallel processing, knowledge-based systems, and artificial intelligence, the prototype environment will serve as a test bed for practical application and evaluation of these technologies.

The USE prototype design, described herein, is still being developed. Many design details are still to be worked out. The author hopes that this document will generate interest in the subject, and inquiries and technical exchanges that might contribute to the design of the environment are welcomed.

National Aeronautics and Space Administration
Lewis Research Center
Cleveland, Ohio, January 6, 1989

References

1. Arpasi, D.J.; and Cole, G.L.: Automating the Parallel Processing of Fluid and Structural Dynamics Calculations. NASA TM-89837, 1987.
2. Arpasi, D.J.; and Milner, E.J.: Partitioning and Packing Mathematical Simulation Models for Calculation on Parallel Computers. NASA TM-87170, 1986.
3. Arpasi, D.J.: Real-Time Multiprocessor Programming Language (TRMPL)—Users Manual. NASA TP-2422, 1985.
4. Cole, G.L.: Operating System for a Real-Time Multiprocessor Propulsion System Simulator—Users Manual. NASA TP-2426, 1984.

NASA
National Aeronautics and Space Administration

Report Documentation Page

1. Report No. NASA TM-4103	2. Government Accession No.	3. Recipient's Catalog No.

4. Title and Subtitle Automating the Multiprocessing Environment	5. Report Date March 1989
	6. Performing Organization Code
7. Author(s) Dale J. Arpasi	8. Performing Organization Report No. E-4426
	10. Work Unit No. 505-62-21
9. Performing Organization Name and Address National Aeronautics and Space Administration Lewis Research Center Cleveland, Ohio 44135-3191	11. Contract or Grant No.
	13. Type of Report and Period Covered Technical Memorandum
12. Sponsoring Agency Name and Address National Aeronautics and Space Administration Washington, D.C. 20546-0001	14. Sponsoring Agency Code

15. Supplementary Notes

16. Abstract

An approach to automate the programming and operation of tree-structured networks of multiprocessor systems is discussed. A conceptual, knowledge-based operating environment is presented, and requirements for two major technology elements are identified as follows: (1) An intelligent information translator is proposed for implementing information transfer between dissimilar hardware and software, thereby enabling independent and modular development of future systems and promoting a language-independence of codes and information; (2) A resident system activity manager, which recognizes the systems capabilities and monitors the status of all systems within the environment, is proposed for integrating dissimilar systems into effective parallel processing resources to optimally meet user needs. Finally, key computational capabilities which must be provided before the environment can be realized are identified.

17. Key Words (Suggested by Author(s)) Simulation Parallel processing Software	18. Distribution Statement Unclassified – Unlimited Subject Category 61

19. Security Classif. (of this report) Unclassified	20. Security Classif. (of this page) Unclassified	21. No of pages 12	22. Price* A03

NASA FORM 1626 OCT 86

*For sale by the National Technical Information Service, Springfield, Virginia 22161

NASA-Langley, 1989